Active Strategies for Coordination of Solitary Robots

by

Jordan Felicien Masakuna

Dissertation presented for the degree of Doctor of Philosophy in the Computer Science Division, Faculty of Science at Stellenbosch University

Supervisor: Dr. Simukai W. Utete
Co-supervisor: Prof. Steve Kroon

December 2020

The financial assistance of the African Institute for Mathematical Sciences (AIMS) and CSIR-SU Centre for Artificial Intelligence Research Group (CSIR-SU CAIR) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the AIMS and CSIR-SU CAIR.

Stellenbosch University https://scholar.sun.ac.za

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2020

Copyright © 2020 Stellenbosch University
All rights reserved.

Stellenbosch University https://scholar.sun.ac.za

Abstract

Active Strategies for Coordination of Solitary Robots

Jordan Felicien Masakuna

Computer Science Division,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Dissertation: Doctor of Philosophy

December 2020

This thesis considers the problem of search of an unknown environment by multiple solitary robots: self-interested robots without prior knowledge about each other, and with restricted perception and communication capacity. When solitary robots accidentally interact with each other, they can leverage each other's information to work more effectively. In this thesis, we consider three problems related to the treatment of solitary robots: coordination, construction of a view of the network formed when robots interact, and classifier fusion. Coordination is the key focus for search and rescue. The other two problems are related areas inspired by the problems we encountered while developing our coordination method. We propose a coordination strategy based on cellular decomposition of the search environment, which provides sustainable performance when a known available search time (bound) is insufficient to cover the entire search environment. A sustainable performance is achieved when robots that know about each other explore non-overlapping regions. For network construction, we propose modifications to a scalable decentralised method for constructing a model of network topology which reduces the number of messages exchanged between interacting nodes. The method has wider potential application than mobile robotics. For classifier fusion, we propose an iterative

Stellenbosch University https://scholar.sun.ac.za

method where outputs of classifiers are combined without using any further information about the behaviour of the individual classifiers. Our approaches for each of these problems are compared to state-of-the-art methods.

Stellenbosch University https://scholar.sun.ac.za

Uittreksel

Aktiewe Strategieë vir die Koördinasie van Alleenlopnde Robotte

Jordan Felicien Masakuna

Afdeling Rekenaarwetenskap,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Proefskrif: PhD

Desember 2020

Hierdie tesis beskou die probleem van soektog in ’n onbekende omgewing deur ’n aantal alleenstaande robotte: selfbelangstellende robotte sonder voorafgaande kennis van mekaar, en met beperkte persepsie- en kommunikasievermoëns. Wanneer alleenstaande robotte toevallig mekaar raakloop, kan hulle met mekaar inligting uitruil om meer effektief te werk. Hierdie tesis beskou drie probleme wat verband hou met die hantering van alleenstaande robotte: konstruksie van ’n blik van die netwerk gevorm deur interaksie tussen robotte, koördinasie en klassifiseerdersamesmelting. Koördinasie is die hoof fokuspunt vir soek en redding. Die ander twee probleme is uit verwante areas, gemotiveer deur uitdagings wat ons ervaar het tydens die ontwikkeling van ons koördineringsmetode. Ons stel ’n skaleerbare desentraliseerde metode voor om ’n model van netwerktopologie te bou wat minder boodskappe tussen wisselwerkende nodusse hoet te verruil. Die metode het wyer potensiële toepassings as mobiele robotika. Vir koördinasie, stel ons ’n strategie voor gebaseer op sellulêre ontbinding van die soekomgewing, wat volhoubare prestasie toon wanneer ’n bekende soektyd onvoldoende is om die hele soekomgewing te dek. Vir klassifiseerdersamesmelting, stel ons ’n iteratiewe metode voor, waar klassifiseerders se

Stellenbosch University https://scholar.sun.ac.za

voorspellings gekombineer word sonder om enige verdere inligting oor die gedrag van die individuele klassifiseerders te gebruik. Ons benaderings vir elkeen van hierdie probleme word vergelyk met stand-van-die-kuns metodes.

Stellenbosch University https://scholar.sun.ac.za

Acknowledgements

Writing a thesis not only requires motivation, but also a lot of time. First, I would like to thank my **God, the Lord ALMIGHTY**, the Lord of time. It is through **Him** that I could complete this work. A great thanks also to my supervisors, Dr. Simukai Utete and Prof. Steve Kroon, for giving me the means to write this work in complete serenity through their fruitful exchanges, sharp scientific opinions, always accompanied with friendly encouragement. I have not always been a good student to them, especially when things did not seem to work towards my advantage. Sometimes, I was very stubborn and impatient to them, but they always treated me well regardless of my stubbornness. Thanks to Prof. Jeff Sanders for his scientific comments, and to Prof. Manya Ndjadi and Prof. Kafunda Katalay for recommending AIMS to me. I would like also to thank my entire family, namely: Jean, Huguette, Apho, Wens, Leon, Auguysta, Gracia and Kerticia Masakuna. Thanks to my late sister Falonne Masakuna. Thanks to all my friends for their inducement. I thank ACGT[1] for approving this opportunity, Mr. Medard Ilunga and Mr. Georges Kazad in particular. Thank you to Pastor Dieudonne Kantu, Pastor Marcelo Tunasi, Apostle Roland Dalo and Pastor Athom's Mbuma for contributing to my spiritual growth through their divine conduct, preachings and teachings.

[1] Agence Congolaise des Grands Travaux

Stellenbosch University https://scholar.sun.ac.za

Dedications

To my dearest father and mother, Augustin Ngolpay Masakuna and Beatrice Muniongo Indwa

Stellenbosch University https://scholar.sun.ac.za

Contents

Declaration **i**

Abstract **ii**

Uittreksel **iv**

Acknowledgements **vi**

Dedications **vii**

Contents **viii**

List of Figures **xii**

List of Tables **xv**

List of Algorithms **xix**

Nomenclature and Notation **xx**

1 Introduction **1**
- 1.1 Objectives and background of this study 1
- 1.2 Contributions 4
- 1.3 Overview of this work 4

2 Background **6**
- 2.1 Introduction 6
- 2.2 Basic concepts 6
- 2.3 Autonomous robots 8
- 2.4 Multiagent systems 14
- 2.5 Conclusion 20

Stellenbosch University https://scholar.sun.ac.za

I Coordination 22

3 Background and literature 23
3.1 General background 23
3.2 Literature discussion and frontier-based exploration 24
3.3 Summary 33

4 Soft obstacle coordination strategy for solitary robots 35
4.1 Introduction 35
4.2 Considerations for the soft obstacle strategy 37
4.3 Soft obstacle coordination strategy 41
4.4 Conclusion 51

5 Distributed system of solitary robots 53
5.1 Introduction 53
5.2 Distributed methods for coordination of solitary robots 53
5.3 Leader election based on closeness centrality 57
5.4 Data fusion and task assignment 65
5.5 General algorithm 67
5.6 Conclusion 71

6 Metrics, results and discussion 78
6.1 Metrics 79
6.2 Simulation description and experimental setup 81
6.3 Results and discussion 87
6.4 Conclusion and avenues for future work 105

II Network view construction 108

7 Background and literature 109
7.1 General background 109
7.2 Literature discussion 110
7.3 Background of methods for failure detection 116
7.4 Summary 118

8 Decentralised view construction 119
8.1 Introduction 119
8.2 View construction 120

Stellenbosch University https://scholar.sun.ac.za

8.3 Communication analysis 129
8.4 Communication failure 131
8.5 Conclusion 139

9 Metrics, results and discussion 140
9.1 Metrics 140
9.2 Simulation description and experimental setup 142
9.3 Results and discussion 146
9.4 Conclusion and avenues for future work 156

III Classifier fusion 158

10 Background and literature 159
10.1 General background 159
10.2 Literature discussion 161
10.3 Summary 162

11 Classifier fusion method 163
11.1 Introduction 163
11.2 Yayambo method 164
11.3 Complexity 169
11.4 Conclusion 170

12 Metrics, results and discussion 171
12.1 Metrics 172
12.2 Simulation description and experimental setup 174
12.3 Results and discussion 178
12.4 Conclusion and avenues for future work 193

13 Conclusions and future work 196
13.1 Summary 196
13.2 Primary contributions 197
13.3 Future work 198

List of References 199

A Solitary search 213

Stellenbosch University https://scholar.sun.ac.za

B Obstacle avoidance techniques 220

C Toy example of pruning method 224

D Toy example of Yayambo 228

E Dataset characteristics 231

F Utilities 233

G Code of Python functions 235

G.1 Network view construction 235
G.2 Coordination 244
G.3 Classifier fusion 255

Stellenbosch University https://scholar.sun.ac.za

List of Figures

2.1 *Example of an autonomous mobile robot architecture.* 8
2.2 *Interaction paradigms in multiagent systems.* 16

3.1 *Illustration of cellular decomposition.* 25
3.2 *Illustration of assignment of sectors upon an accidental rendezvous.* . . 31

4.1 *Illustration of the proposed soft obstacle coordination.* 39
4.2 *Virtual world enlargement.* 45

5.1 *State transition diagram of a single robot during search.* 56
5.2 *Illustration of leader election.* 61
5.3 *Illustration of the area scanned per motion step by a robot.* 67
5.4 *Illustration of how a robot can get a new exploration region inside its actual exploration region.* 70

6.1 *A sample of the simulation environments used.* 85
6.2 *Illustration of some environments considered in this work.* 86
6.3 *The Victoria Park grid map [39].* 87
6.4 *Individual coverage progress for different numbers of robots.* 88
6.5 *Individual coverage performance for different teams of robots.* 89
6.6 *Individual robot trajectories for different teams of robots.* 90
6.7 *Box-and-whiskers plots of the performance of each coordination strategy.* 91
6.8 *Plots of* 50 *scenarios using SOS, ARS and Hybrid.* 93
6.9 *Cumulative distribution function (CDF) of* 30 *scenarios using SOS and ARS on the Victoria Park grid map.* 95
6.10 *The benefits of margins in SOS: unbalanced assignment.* 100
6.11 *The benefits of margins in SOS: balanced assignment.* 101
6.12 *SOS using the technique of frontiers as the search algorithm.* 102

Stellenbosch University https://scholar.sun.ac.za

6.13 *Histograms of interaction network sizes observed in coordination of solitary robots.* . . . 104

7.1 *An example of the construction of a view of a communication graph.* . . . 114
7.2 *An example of the benchmark method [118].* . . . 115
7.3 *Illustration of a communication graph.* . . . 115

8.1 *Example of discovery of prunable objects.* . . . 123
8.2 *Illustration of triangles (i.e. cycles of size* 3*) in pruning method.* . . . 123
8.3 *Illustration of failure of a node or edges.* . . . 134

9.1 *Histogram of diameters of random graphs.* . . . 145
9.2 *Illustration of a network of solitary nodes.* . . . 145
9.3 *Histogram of differences in average number of messages on the autonomous graphs.* . . . 146
9.4 *Histogram of differences in maximum number of messages on the autonomous graphs.* . . . 146
9.5 *Number of messages using benchmark and pruning methods on one random graph.* . . . 147
9.6 *Number of messages per node using both methods for view construction in the case of failures.* . . . 149
9.7 *Number of signals per node using both methods for view construction in the case of failures.* . . . 150
9.8 *Histograms of differences of shortest path distances between approximate central nodes obtained using the benchmark and our pruning methods with respect to the exact most central node on some random graphs for failure-free cases.* . . . 152
9.9 *Histograms of differences of shortest path distances between approximate central nodes obtained using the benchmark and our pruning methods with respect to the exact most central node on some random graphs for failure cases.* . . . 153
9.10 *Histogram of percentage reductions of maximum running time (in seconds) between the benchmark and our pruning methods for failure-free cases.* . . . 155
9.11 *Histogram of percentage reductions of maximum running time (in seconds) between the benchmark and our pruning methods for failure cases.* 155

Stellenbosch University https://scholar.sun.ac.za

9.12 *Histogram of percentage reductions of maximum memory (in MB) between the benchmark and our pruning methods for failure-free cases.* . . . 156
9.13 *Histogram of percentage reductions of maximum memory (in MB) between the benchmark and our pruning methods for failure cases.* 156

11.1 *Example of dissimilarities between probabilities using Equation 11.2.4.* 168

12.1 *Performances of classification and fusion techniques in active vision.* . . . 190

A.1 *Some examples of methods for exploration of a region.* 213
A.2 *Application of zigzag search in different situations.* 215
A.3 *Illustration of a zigzag move and obstacle detection.* 219

B.1 *Illustration of obstacle avoidance using Distance Bug.* 222

C.1 *Communication graph used to illustrate pruning methods.* 224
C.2 *Pruned nodes from the first pruning stage.* 226
C.3 *Pruned nodes after the second pruning stage.* 227

E.1 *Photographs of four of the objects in the Columbia dataset [78].* 232

Stellenbosch University https://scholar.sun.ac.za

List of Tables

5.1 *Nodes' initialisation of leader election using Algorithm 5.2 for the communication graph in Figure 5.2.* 61
5.2 *First iteration of leader election using Algorithm 5.2 for the communication graph in Figure 5.2.* 64
5.3 *Second iteration of leader election using Algorithm 5.2 for the communication graph in Figure 5.2.* 64

6.1 *Experimental setup for coordination.* 85
6.2 *The p-value and the effect size e for the coordination strategies for a team of ten robots.* 92
6.3 *Results of* 50 *scenarios using SOS, ARS and Hybrid.* 93
6.4 *Result for search and rescue by* 50 *solitary robots on a single scenario.* . 94
6.5 *Number of interactions that each robot participated in.* 94
6.6 *Results of* 30 *scenarios using SOS, ARS and Hybrid on the Victoria Park grid map.* 95
6.7 *The p-value and the effect size e for the coordination strategies with* 50 *solitary robots in the simulated search environment.* 96
6.8 *p-values and effect sizes e for the coordination strategies with up to* 3000 *solitary robots in the Victoria Park grid map.* 98
6.9 *The p-value and the effect size e for the coordination strategies with* 50 *solitary robots starting from separate locations in the simulated search environment.* 101
6.10 *The p-value and the effect size e for the coordination strategies with* 50 *solitary robots starting from separate locations in the simulated search environment.* 103
6.11 *The p-value and the effect size e for our coordination strategy (SOS) using two different leader election methods.* 104

Stellenbosch University https://scholar.sun.ac.za

7.1 *Illustration of the use of Algorithm 7.1 applied on the graph in Figure 7.3.* . . . 116

8.1 *Table indicating pruning using the graph in Figure 8.2.* . . . 126

9.1 *Results for a sample of* 5 *real-world networks (*1 *phenomenology collaboration network,* 1 *Gnutella peer-to-peer network and* 3 *autonomous networks) on the benchmark method and our proposed pruning method.* 147
9.2 *Shortest path distances between the exact most central node and approximate most central node for failure-free situations.* . . . 151
9.3 *Shortest path distances between the exact most central node and approximate most central node for failure situations.* . . . 152

11.1 *Evolution of probability distributions over iterations of the Yayambo algorithm.* . . . 169
11.2 *Evolution of supports of distributions over iterations of the Yayambo algorithm.* . . . 169

12.1 *Hyperparameters used to train classifiers on various data sets.* . . . 176
12.2 *Hyperparameters used to train similar classifiers on the subsets of data sets.* . . . 177
12.3 *Hyperparameters used to train similar classifiers on the Columbia data set.* . . . 177
12.4 *The characteristics of the data set used to train classifiers.* . . . 178
12.5 *The majority vote rule, Borda count rule, sum rule, product rule and Yayambo accuracies on test data with various values of* ε_0 *for Yayambo.* 180
12.6 *The majority vote rule, Borda count rule, sum rule, product rule and Yayambo cross-entropy losses on test data.* . . . 181
12.7 *Other performance metrics on classifier fusion algorithms.* . . . 182
12.8 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using similar classifiers with different hyperparameters trained on the Columbia data set.* . . . 183
12.9 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using classifiers trained on different subsets of data set.* . . . 184
12.10 *Data set sizes for training of classifiers on different training data set.* . . . 184

Stellenbosch University https://scholar.sun.ac.za

12.11 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using MLPs trained on different subsets of data set.* . . . 184
12.12 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using k-NN trained on different subsets of data set.* . . . 185
12.13 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using SVM trained on different subsets of data set.* . . . 185
12.14 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using logistic regression trained on different subsets of data set.* . . . 186
12.15 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using classifiers trained on different features of data set.* . . . 187
12.16 *The number of features considered to train classifiers on each data set.* . . . 187
12.17 *Measurement of disagreement between classifiers, trained using the Activity data set, from a prediction point of view.* . . . 187
12.18 *Measurement of disagreement between classifiers, trained using the Digits data set, from a prediction point of view.* . . . 188
12.19 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data.* . . . 189
12.20 *Expected performances of individual artificial classifiers.* . . . 192
12.21 *Expected agreement between individual artificial classifiers from a classification point of view.* . . . 193
12.22 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies using artificial classifiers.* . . . 193
12.23 *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies using artificial classifiers for different runs of the fourth and fifth classifiers as they are purely random.* . . . 193
12.24 *Measurement of disagreement between random classifiers from a prediction point of view.* . . . 194

C.1 *Initial situation before the use of pruning.* . . . 225
C.2 *The results of interaction after the first iteration.* . . . 225
C.3 *The results of the first pruning.* . . . 226

Stellenbosch University https://scholar.sun.ac.za

C.4 *The results of interaction after the second iteration.* 226
C.5 *The results of the second round of pruning.* 227
C.6 *Table showing the results of interaction after the third iteration.* 227

Stellenbosch University https://scholar.sun.ac.za

List of Algorithms

2.1 *Calculation of the inputs of the motion command of a robot* 9
3.1 *Our presentation of a frontier-based exploration method.* 27
4.1 *Choice of exploration regions.* 46
5.1 *Waiting for other interactions in its neighbourhood by a robot.* 55
5.2 *Leader election based on the work by Naz [77].* 62
5.3 *Our proposed soft obstacle strategy.* 72
7.1 *Our presentation of the benchmark method [118].* 112
8.1 *Our proposed pruning method in a failure-free scenario.* 127
8.2 *Our proposed pruning method with failure management.* 135
8.3 *Our presentation of the benchmark method [118] with failure management.* 138
11.1 *Our proposed classifier fusion algorithm.* 167
A.1 *Zigzag search used by a robot to explore its exploration region* $\mathcal{X}_i$. . 216
B.1 *A one-step move using the Distance Bug method for obstacle avoidance.* 223
F.1 *Enumeration of robot states.* 233
F.2 *Enumeration of states involved in interaction of robots.* 233
F.3 *Enumeration of search states for a robot.* 233
F.4 *Creation of a rectangle and related subroutines used for coordination of solitary robots.* 234

Stellenbosch University https://scholar.sun.ac.za

Nomenclature and Notation

Abbreviations		Page introduced
ARS	accidental rendezvous strategy	78
FD	failure detector	117
MB	megabytes	156
PRS	periodic rendezvous strategy	78
SOS	soft obstacle strategy	78

Symbols related to coordination		Page introduced
a_i	the closest corner of the exploration region of R_i.	49
$B_d(x)$	a closed ball centred at x of radius d.	10
$\mathcal{B}_i^{(t)}$	soft obstacle of R_i.	42
$\mathcal{C}_i^{(t)}$	the explored region known by R_i.	42
d	a robot's perception and communication range.	67
$\delta(x, y)$	the Euclidean distance between x and y.	49
P	the number of robots in an interaction.	44
R_i	the i-th robot.	38
τ	the total search time.	67
O	the set of obstacles.	39
v_i	the i-th node.	60

Stellenbosch University https://scholar.sun.ac.za

$\mathcal{V}_i$	the current virtual world of R_i.	40
W	the search environment.	39
x_i	current location of R_i.	215
$\mathcal{Y}_i$	the list of exploration regions of R_i.	48
$\mathcal{X}_i$	the current exploration region of R_i.	38
$\mathcal{U}_i^{(t)}$	the unexplored region of R_i.	47
$\mathcal{Z}_i$	the interference region known by R_i.	42

Symbols related to view construction		**Page introduced**
$\setminus$	set difference.	113
$\leftarrow$	assignment.	113
$\langle\texttt{mes}\rangle$	a message `mes` sent to or received by a node.	113
c_i	closeness centrality of node v_i.	19
D	a given number of iterations to construct a view.	113
ecc_i	the eccentricity of a node v_i.	125
e_{ij}	the edge between nodes v_i and v_j.	121
$\mathcal{E}_i^{\text{down}}$	the set of nodes in $\mathcal{N}_i$ incident to at least one edge in $\mathcal{S}_i^{\text{down}}$.	135
$\mathcal{F}_i^{(t)}$	the set of pruned nodes before the t-th pruning round known by node v_i.	123
$\mathcal{F}_{i,t}$	the set of pruned nodes until the t-th pruning round known by node v_i.	123
G	a communication graph.	19
$\mathcal{N}_i^{\text{down}}$	the set of nodes which have failed known by node v_i.	135
$\mathcal{N}_i$	the set of immediate neighbours of v_i.	113

Stellenbosch University https://scholar.sun.ac.za

$\mathcal{N}_{i,t}^{\mathrm{up}}$	the set of neighbours of v_i which are still involved in interaction at the beginning of iteration t.	124
$\mathcal{N}_i^{(t)}$	the set of $(t+1)$-hop neighbours of v_i.	113
$\mathcal{S}_i^{\mathrm{down}}$	the set of edges which have failed known by node v_i.	135
$\mathcal{S}_i^{(t)}$	the set of new edges discovered by v_i after iteration t.	121
$\mathcal{S}_{i,t}$	the set of edges discovered by v_i after iteration t.	122
v_i	the i-th node in a graph.	113

Symbols related to classifier fusion		**Page introduced**
$\beta_{ij}^{(t)}$	weight assigned to the i-th probability vector (i.e. distribution) from the j-th distribution at the end of iteration t.	165
c_k	k-th type of object of interest.	164
d_i	i-th classifier output.	164
f_i	i-th classifier.	164
l	the number of types of objects of interest.	164
m	the number of classifiers.	164
$\pi_i^{(t)}$	i-th distribution at the end of iteration t.	165

Stellenbosch University https://scholar.sun.ac.za

Chapter 1

Introduction

1.1 Objectives and background of this study

This thesis considers the problem of search by multiple solitary robots. We consider solitary robots to be self-interested robots without prior knowledge about each other, and with restricted communication capacity. Solitary robots are robots without a *common (social) goal*. Some robots can impersonate social signals and elicit human feelings [98]. An intention achieved by such a robot is viewed as a (social) goal. In other words, robots are solitary when individuals (not the team) are the key modelling units of a coordination strategy. Solitary robots need to be autonomous, and must also be equipped with an ability to amend their behaviour when they acquire new information. They do "the best they can" while not being aware of other robots, unless they accidentally meet another robot, whereupon they may collaborate. When robots interact with each other in this way, each can benefit from diverse information obtained from other robots to work more effectively.

This thesis considers search and rescue in an unknown static environment, i.e. finding static targets (i.e. objects of interest) by exploring the environment. Our proposed method might be applicable to search and rescue in various aspects of disaster management for which the choice of a static environment is reasonable [50]. When considering solitary robots for search of some static targets in an unknown environment, various problems could be considered such as formation control (where a group of robots aim to achieve some goal while preserving some spatial pattern between the robots

Stellenbosch University https://scholar.sun.ac.za

involved) and exploration. We tackle three problems: coordination, interaction and classifier fusion. We aim to propose strategies that solitary robots can apply to effectively explore an unknown environment when coordinating their exploration, interacting with each other, and recognising objects of interest. In this thesis, the problem of coordination is the key focus for search and rescue. The other two problems we consider are inspired by the problems we encountered while developing our coordination method.

We consider low power robots working under weak signal conditions [84, 103]. Weak signal conditions could be caused by the nature of the search environment or sensor conditions. Such robots can face problems of high propagation time in communication.

It was mentioned earlier that exploration is the problem we consider in this thesis for coordination of solitary robots. Many search strategies [17, 31, 63] have been proposed to tackle the exploration problem. In fact, a coordination strategy could be considered effective when individual robot exploration is maximised.

We motivate and propose a coordination strategy for solitary robots.

Since we consider search by solitary robots, we must treat their interactions carefully to optimize the benefit of the interaction. A natural way to manage interaction of robots is for the robots to elect a leader to coordinate their interaction. There are various approaches for leader election [77]. In this thesis, we consider the approach where, before electing a leader, each node must, in a distributed manner, construct a (limited) view of the network formed by their interaction. The process of leader election is distributed because solitary nodes have limited communication capacity and are autonomous. Also, because this work considers solitary robots, it is reasonable to develop distributed systems which are decentralised, as naturally solitary robots should not rely on a single robot.

In distributed networks, the topology of a network is very important. In packet-switched store-and-forward networks [62] for instance, messages are stored and processed at intermediary nodes when transmitted from a source node to a destination node. Korach et al. [55] claim that, in this context, it is essential for every node to have knowledge about the topology of their interaction (i.e. each node should know its tree parent and its tree children). Information about the network topology can be profitably employed to develop adequate distributed methods. For example, the propagation

Stellenbosch University https://scholar.sun.ac.za

time (or the amount of network traffic) required to synchronize the nodes of a network can be minimized if the most central node of the network is known [47, 82]. Here, we consider closeness centrality [8] to identify the most central node of a network. We consider closeness centrality because it is an appropriate centrality measure to use for identifying the node with the shortest distances to all other nodes in a network.

The elected leader has an essential role to play in the interaction of robots, which is application-dependent. For the case of search and rescue tasks as we consider here, the goal of each robot will be to maximise its exploration. However, robots maximise their joint exploration when they search non-overlapping regions. To coordinate interaction of robots, a leader has a two-fold role. First, a leader collects, fuses and distributes each robot's individual information, so that every robot can benefit from a merged view of the situation. Second, it allocates exploration regions to each robot, to enhance their combined search coverage. We refer to this latter role of a leader as coordination.

Coordination of robots requires exchange of robots' information, such as maps. To coordinate their future actions, the leader elected must receive individual information from other nodes. Since interactions are local, each receiving node combines the maps it receives with its own before it forwards the combined map to an immediate neighbour (specifically, its spanning tree parent, as will be discussed in Chapter 5) closer to the leader. This coordination method is nothing but a packet-switched store-and-forward method, as explained above.

This thesis also motivates and proposes a suitable decentralised method robots can apply to construct views of their interaction network for leader election.

Finally, when exploring, robots must determine the types of objects of interest they perceive in the search area. This is termed object recognition [107] in the machine learning community. An object recognition solution is a mathematical model that identifies to which type or class an object of interest belongs. Individual classifiers may provide different opinions on the same object of interest. One way to handle this situation is to combine the various classification results by a classifier fusion procedure. We motivate and design such a classifier fusion algorithm. Note that our proposed classifier fusion method has wider applicability.

Stellenbosch University https://scholar.sun.ac.za

1.2 Contributions

Above we presented three problems arising when employing solitary robots for search and rescue: interaction, coordination and classifier fusion. The following briefly describes the contributions of this thesis in these domains:

(1) **Coordination**: we propose a novel coordination strategy based on cellular decomposition [17] and the use of soft obstacles [70]. The proposed coordination method provides sustainable performance when a known search time precludes coverage of the entire search area.

(2) **Decentralised view construction**: we propose an enhancement to the method of You et al. [118] for distributed construction of a network. Our enhancement reduces the number of messages exchanged between nodes of the network. We also extend the algorithm to allow it to handle communication failures.

(3) **Classifier fusion**: we propose a fusion method [71] for classifiers that only report their outputs over class labels. Our proposed classifier fusion method weights classifiers' outputs unequally, even though only their outputs are reported.

While coordination of solitary robots was the inspiration for the techniques proposed in this thesis, the second and third contributions are more widely applicable.

1.3 Overview of this work

Apart from the next opening chapter (**Chapter 2**) which reviews background work with a focus on multiagent systems and autonomous robots, and the closing chapter (**Chapter 13**) which concludes, the rest of this thesis is organised in three parts.

Part I reviews background and literature for coordination of solitary robots (**Chapter 3**), describes our proposed coordination method (**Chapters 4** and **5**), and presents and discusses results of the proposed coordination strategy (**Chapter 6**).

Part II reviews background and literature for distributed view construction of a communication graph (**Chapter 7**), introduces our proposed dis-

Stellenbosch University https://scholar.sun.ac.za

tributed method for view construction (**Chapter 8**), and presents and discusses results of view construction (**Chapter 9**).

Part III reviews background and literature for classifier fusion (**Chapter 10**), introduces our proposed classifier fusion method (**Chapter 11**), and presents and discusses results of classifier fusion (**Chapter 12**).

Coordination of solitary robots for search and rescue in an unknown static environment is the overarching challenge of this thesis, and we aim to develop various distributed methods to answer the following main question: how much area can solitary robots interacting in a distributed manner explore within a given search time?

Stellenbosch University https://scholar.sun.ac.za

Chapter 2

Background

2.1 Introduction

This chapter provides background material on multiagent and distributed systems relevant for this thesis, i.e. this chapter reviews selected paradigms for modelling multiagent systems and how mobile robots interact with a search environment. A multiagent system is composed of multiple agents which can interact with each other in an environment. While the agents could be humans, human-robot teams, purely software agents or other machines, this thesis considers the typical case where multiagent systems refer to software systems of robots.

2.2 Basic concepts

We begin by defining various concepts that we will use to formalize the notion of a solitary robot. The aim of this study is to develop various decentralised methods that a solitary robot could apply to achieve some goal. The adjective solitary has been used previously in literature. For example, le Roux et al. [64] compared solitary and group foraging. They investigated when solitary foraging is more effective than group foraging, and vice-versa. Solitary foraging is a type of search where agents or animals decide to find their objects of interest alone [64, 104]. But the term solitary used for robots in this thesis has additional connotations. Here, not only is a solitary robot expected to find its objects of interest alone, it also has some limitations on its perception and communication capacity.

Stellenbosch University https://scholar.sun.ac.za

Before defining solitary robots, we need to introduce some other important terms. An *agent* is something that acts in an environment. When an agent is facing multiple alternatives, it can make an appropriate choice. An agent can be *autonomous*: an autonomous agent is *active* (i.e. it can learn from the environment) and does not require interaction with other agents to take action, although such interaction can be beneficial. An active agent is an online agent: it is called to take decisions as additional information becomes available. Self-driving cars [99] are an example of autonomous agents. A group of agents can be *homogeneous*, when they are equipped with the same abilities, or *heterogeneous* otherwise. An agent can also be *self-interested*. Agents are self-interested when individuals (not the team) are the key modelling units of a coordination strategy. *The solitary robots this work focuses on are homogeneous and autonomous self-interested agents.*

Different distributed methods that will be developed in this work are considered active. A method is considered active when the agents which apply the method are active, i.e. agents are called to take decisions as additional information becomes available—e.g. active vision [1]. Active strategies are considered due to the nature of the solitary robots in the system, the restricted communication between them [75], and the limited time available to achieve goals. Active strategies are important in persistence and lifelong learning problems [101] for autonomous robots. Lifelong learning problems consider the study of robots which learn in isolation. Such robots transfer knowledge between themselves when they encounter each other in order to work more effectively. The above discussion leads us to define our central objects of study, solitary robots:

Definition 2.2.1. *Solitary robots are self-interested (i.e. robots without a common social goal), autonomous and homogeneous robots without prior knowledge about each other, and with restricted perception and communication capacity.*

Although solitary robots do not have a common goal, it should be noted that some solitary robots can still start searching from nearby locations and when that happens, they are called to collaborate at that time. In some coordination approaches (including our proposed coordination strategy), each robot goes its own way with no intention of meeting the other robots again after collaboration.

Stellenbosch University https://scholar.sun.ac.za

Many autonomous agents, including self-driving cars, work in dynamic environments. However, this thesis considers solitary agents acting in static (but unknown) environments. For example, our results might be applicable to search and rescue in various aspects of disaster management (including housing and building damage, as well as disruption of roads, water supply, gas, and mining) where the choice of a static environment and static objects of interest are reasonable [50].

The following section describes autonomous robots, the agents for which the distributed algorithms in this thesis are proposed.

2.3 Autonomous robots

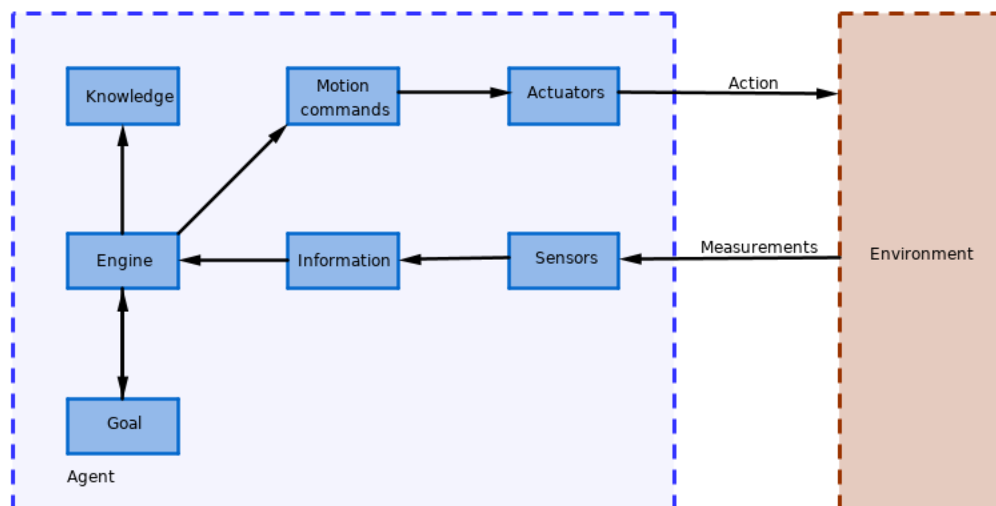


Figure 2.1: *Example of an autonomous mobile robot architecture.*

An example architecture for an autonomous robot is illustrated in Figure 2.1.

A robot obtains *information* as it interacts with its *environment*. A robot has *sensors*, which provide the robot with information about the environment, such as objects and their shapes, sizes, directions of motion, and so on. This information obtained is then processed by a programmable machine called the *engine*, and used in further interaction with the environment. A robot acquires *knowledge* by processing the information. The robot interacts with its environment by performing some actions to achieve some *goal*. Actions change robot state and interaction with environment can also change the

Stellenbosch University https://scholar.sun.ac.za

state of the environment. As shown in Figure 2.1, a robot also has multiple *actuators* that are responsible for controlling robot motion. Actuators execute instructions received from *motion commands*. Motion commands are determined by the engine; the information about the next location of the robot is provided by our search algorithm (Appendix A). Let us say a robot is at location $\boldsymbol{\mu}_t$ at time t. After time Δt, the search algorithm will provide to the robot the new location $\boldsymbol{\mu}_{t+\Delta t}$ (i.e. odometry information) to move to, at time $t + \Delta t$. From odometry information, the robot will work out its translation shift and rotational movement as described in Section 2.3.3. The robot will then change its internal motion command based on these two command values. The procedure for this task is given in Algorithm 2.1.

Algorithm 2.1 *Calculation of the inputs of the motion command of a robot.*

1: **function** MOTIONCOMMANDINPUTS($\boldsymbol{\mu}_t = (x_1, y_1, \theta_1), \boldsymbol{\mu}_{t+\Delta t} = (x_2, y_2, \theta_2)$)
2: $\quad\delta_1 \leftarrow \arctan 2(y_2 - y_1, x_2 - x_1) - \theta_1$
3: $\quad\alpha \leftarrow \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
4: $\quad\delta_2 \leftarrow \theta_2 - \theta_1 - \delta_1$
5: $\quad$**return** $\alpha, \delta_1, \delta_2$
6: **end function**

2.3.1 Physical construction of autonomous robots

Robot systems typically have requirements and specifications determined by the nature of their intended applications which must be considered when planning their design and construction. Some notable robotics applications are navigation and control (for mobile robots) or planning. Important considerations for robotics applications include: the types of robots to consider, e.g. autonomous underwater vehicles (AUVs), unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs); the types of information to handle, e.g. maps; and the types of actuation to perform, e.g. motion only or grasping. This thesis considers mobile autonomous robots with the ability to grasp objects of interest.

During exploration, robots acquire new information from their sensor nodes through their interaction with the environment. Describing the environment and the behaviour of sensors and actuators requires mathematical models, which can face limitations in terms of computation. These models also must take sensor noise and state uncertainty into account.

Stellenbosch University https://scholar.sun.ac.za

While we will abstract away most of the details on physical construction of autonomous robots in the thesis, we outline the main considerations in this regard in Subsections 2.3.2–2.3.4.

2.3.2 Configuration space

We mentioned earlier the environment in which agents work. In robot motion planning, an environment and robots are represented by a configuration space (C-space for short). In this thesis, a C-space, denoted by $\mathcal{W}$, is a 2D Euclidean space and is partitioned into free space and obstacle space. Free space does not contain obstacles that can hinder robot motion [100].

We do not consider 3D motion in this thesis: this abstraction allows us to focus on the main novelties introduced in this thesis.

Let $B_d(z)$ denote a closed ball centred at z of radius d. A robot has a perception range, say d: we define c_z as a region which the robot can perceive from its current location z, assuming obstacle-free space; $c_z = B_d(z)$.[1]

Many approaches can be used to represent environments, including occupancy grids and topological graphs [100]. We consider a (static) environment represented by an occupancy grid (a discrete representation).

An occupancy grid is composed of cells where each cell is assigned a probability that a corresponding environmental region is occupied—an occupancy probability [114]. A cell is considered occupied when it is likely that it contains (a portion of) an obstacle. Initially all of the cells are assigned a prior occupancy probability. Measurements are then used to update the occupancy probabilities [100, 114]. A cell is considered to be in one of three states

- **free**: prior probability $>$ occupancy probability;
- **occupied**: prior probability $<$ occupancy probability; or
- **unknown**: prior probability $=$ occupancy probability.

We use an occupancy grid to represent a search environment because we are concerned about arrangements of objects while exploring an environment. The presence and absence of objects in an unknown environment

[1]Other shapes for c_z can also be considered, such as rectangular shapes. The shape employed is guided by the robot's sensors.

Stellenbosch University https://scholar.sun.ac.za

are well represented by occupancy grids. Other advantages are that grid-based maps are extremely easy to construct, and that they capture essential features of the environment [100]. However, what we lose by using a discrete representation of an environment is the actual shapes of obstacles and continuous locations of landmarks. This is because discretization allows different maps to have the same representation—thus discretization is not invertible.

2.3.3 Kinematics and robot motion

Kinematics is the branch of mechanics concerned with the motion of objects without reference to the force that causes the motion [100]. For mobile robots which we consider in this thesis, the kinematic configuration of a robot includes the robot's position in 2D Euclidean coordinates and its orientation angle (or bearing or heading direction). The pose at time t is given by a vector

$$\boldsymbol{\mu}_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix}. \tag{2.3.1}$$

Motion is the change in displacement of a robot from $\boldsymbol{\mu}_t$ to $\boldsymbol{\mu}_{t+\Delta t}$, where Δt denotes a time step. Motion models can be deterministic or probabilistic [100]. For probabilistic models, there exist specific motion models based on velocity and odometry. We next describe a velocity model.

Let v_t and w_t denote the translational and rotational velocities controlling the robot at time t, respectively. The control command (i.e. combination of the translational and rotational velocities) is given by

$$\boldsymbol{u}_t = \begin{pmatrix} v_t \\ w_t \end{pmatrix}.$$

Under this control, after a time period Δt of robot motion, its pose is updated to

$$\boldsymbol{\mu}_{t+\Delta t} = \boldsymbol{\mu}_t + \begin{pmatrix} -\frac{v_t}{w_t}\sin\theta_t + \frac{v_t}{w_t}\sin(\theta_t + w_t\Delta t) \\ \frac{v_t}{w_t}\cos\theta_t - \frac{v_t}{w_t}\cos(\theta_t + w_t\Delta t) \\ w_t\Delta t \end{pmatrix} \text{ [100]}.$$

Unlike the velocity model above, which provides velocity commands to a robot's motors, an odometry model provides odometry information such as the distance between the robot and a nearby feature.

Stellenbosch University https://scholar.sun.ac.za

For the odometry model, each robot has its internal coordinate system represented by $\bar{\boldsymbol{x}}_t$, which corresponds to $\boldsymbol{\mu}_t$ in the global coordinate system. The robot measures $\bar{\boldsymbol{x}}_t$ by its sensors. We assume that a robot knows its original pose in global coordinates, but it should be noted that in some situations the relation between robot's internal coordinates and the global coordinates is unknown [100]. This means the transformation from robot's internal coordinates to the global coordinates is not straightforward. To estimate $\boldsymbol{\mu}_{t+\Delta t}$ after time of motion Δt, the robot represents $\boldsymbol{\mu}_t$ and $\boldsymbol{\mu}_{t+\Delta t}$ by $\bar{\boldsymbol{x}}_t$ and $\bar{\boldsymbol{x}}_{t+\Delta t}$ respectively. Let measurements $\bar{\boldsymbol{x}}_t$ and $\bar{\boldsymbol{x}}_{t+\Delta t}$ be represented as follows,

$$\bar{\boldsymbol{x}}_t = \begin{pmatrix} x_1 \\ y_1 \\ \theta_1 \end{pmatrix} \text{ and } \bar{\boldsymbol{x}}_{t+\Delta t} = \begin{pmatrix} x_2 \\ y_2 \\ \theta_2 \end{pmatrix} .$$

In our situation where the robot knows its initial pose in global coordinates, the new pose $\boldsymbol{\mu}_{t+\Delta t}$ is given by

$$\boldsymbol{\mu}_{t+\Delta t} = \boldsymbol{\mu}_t + (\boldsymbol{x}_{t+\Delta t} - \boldsymbol{x}_t) . \tag{2.3.2}$$

In real applications, both motion models are generally combined. For instance, when using the Kalman filter [100] to infer parameters of interest from inaccurate or uncertain measurements, there are two steps: prediction and correction. The algorithm predicts the robot pose using a velocity model and corrects the predictions using an odometry model.

In this thesis, an odometry model for motion commands is applied differently. The inputs of the motion commands of a robot are the current location of the robot and its next location, with the next location provided by a search algorithm (described later in Appendix A). From the command location $\boldsymbol{\mu}_{t+\Delta t}$ provided to the robot, the robot works out its translational shift and its rotation angle. The procedure for this task is given in Algorithm 2.1.

According to Belta et al. [9], the topic of motion planning for robots can be divided into two schools of thought: robot paths can be discrete (e.g. methods based on cellular decomposition [17]) or continuous (e.g. methods based on potential fields [43]). Our approach is in the family of discrete path planning methods. There are two reasons which we consider for the discrete nature of our approach. First, we assume that robots are fully actuated with no control bounds—this means that we do not focus on the detailed dynamics or kinematics of robots. Second (as will be discussed later in Part I),

Stellenbosch University https://scholar.sun.ac.za

we use cellular decomposition for exploration which builds discrete motion paths. It should be noted that integration of the strengths of the two families of motion planning methods is desirable. In fact, Belta et al. [9] claim that such integration is a very promising and challenging research avenue.

2.3.4 Robot measurement

Measurement models describe the process by which robots extract information about their environment from sensors. Sensors differ based on the type of information provided. Laser sensors and sonar sensors are examples [100], where measurements consist of detecting the presence or absence of an object, or discovering the range between two objects.

The internal representation of a map is usually topological or metric. But one can also consider hybrid (combination of topological and metric) or a semantic representation of a map [79]. In the topological representation, only relations between sensed objects are considered. With metric representations, a map can contain objects and their locations—(feature-based)—or simply a collection of locations—(location-based). The key difference between location-based and feature-based maps is that the former process every sensed location (i.e. the area of the grid has almost the same size as the area of the actual environment), whilst with the latter only locations of features/objects are saved (i.e. the area of the grid is much smaller than the area of the actual environment).

A feature is a description of an object. We denote the i-th feature in a map by

$$\boldsymbol{m}_i = \begin{pmatrix} m_{x,i} \\ m_{y,i} \\ m_{s,i} \end{pmatrix}, \tag{2.3.3}$$

where $(m_{x,i}, m_{y,i})$ denotes the i-th object's location and $m_{s,i}$ its label. For a feature-based approach, a map M is a collection of landmarks (features), $\boldsymbol{m}_i$. For a domestic autonomous vacuum cleaner for example, a map could represent landmarks with labels such as 'wall', 'table', 'bed' and so on. For the case of search and rescue by solitary robots, the maps we consider represent static objects of interest, static landmarks or obstacles (i.e. static environment) and explored areas.

Stellenbosch University https://scholar.sun.ac.za

Feature-based maps have lower computational cost and more advantages in adjusting for uncertain measurements than location-based maps [100].

When a robot is at location $\boldsymbol{\mu}_t$ and senses an object $\boldsymbol{m}_i$, there are different ways of relating $\boldsymbol{\mu}_t$ and $\boldsymbol{m}_i$. Using feature-based sensing, the sensor can return information in the form

$$z_i^t = \begin{pmatrix} d_i^t \\ \varphi_i^t \\ s_i \end{pmatrix},$$

where d_i^t denotes the range between $\boldsymbol{\mu}_t$ and $\boldsymbol{m}_i$, φ_i^t the relative bearing at time t, and s_i the description of the object sensed. Typically, a robot would be equipped with several sensor nodes. The problem of fusion of sensor measurements can then arise. Data fusion problems can also arise when two robots plan to combine their individual measurements. The process of data fusion consists of combining data to refine state estimates and predictions [95]. For example, Aragues et al. [4] studied a feature-based map fusion algorithm. They showed how several robots, each with a limited map, can dynamically reach consensus on the global map using the map increments between two consecutive time steps. This is an essential stage of map construction, since it allows robots to reduce uncertainty in the combined map.

In this thesis, we use a location-based map which seems to be a suitable choice for search and rescue by solitary robots.

2.4 Multiagent systems

This thesis aims to build a multiagent system [91]. Multiagent systems refer to the use of multiple agents to achieve some goal.

Before discussing details of multiagent systems, a relevant question regarding multiagent systems must be addressed. The question is derived from the perception that using multiple agents seems more costly than using a single agent. If this is so, why do multiagent systems matter? In other words, why not construct a single powerful agent with tremendous features such as speed, perception and processing ability to perform what multiple agents with lesser features can do? There are five situations where multiagent systems are essential [91], and one of these is relevant for our case: the

Stellenbosch University https://scholar.sun.ac.za

task is inherently distributed. The case of self-interested robots considered in this thesis is a realistic example of a multiagent system: although solitary robots may not initially consider themselves as part of a distributed system if they are not aware of others at the outset initially, they still ultimately form a distributed system as they can communicate and coordinate their actions by passing messages to one another when within communication range.

This section introduces relevant robotics terminology in the context of multiagent systems. We begin by describing three main paradigms for modelling multiagent systems.

2.4.1 Distributed, centralised and decentralised systems

A distributed system is defined as a system in which a collection of agents communicate with each other by passing messages [20]. The study of distributed systems is a broad area in computer science and engineering, often characterised by complex data structures and implementations [20]. In designing such systems, one typically considers three models: the physical model describes the types of devices to use; the architectural model specifies communication between devices; and the interaction model describes the solution that the system should provide. This thesis considers the interaction model only, i.e. we consider device coordination once the devices and their communication mechanisms have been determined.

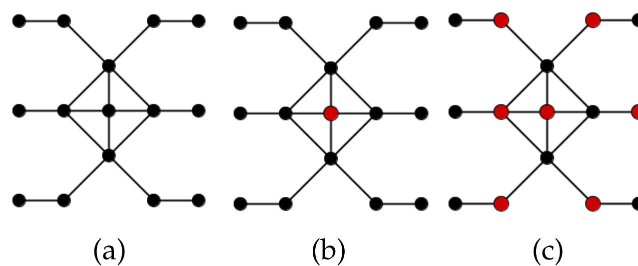
There are many applications of distributed systems. Sensor networks are one example [117]. Sensor networks are often used to monitor the physical environment and provide observations for various uses. Distributed behaviour is also found in insect colonies [12]: groups of insects achieve a common goal, such as lifting prey, by distributed coordination of their actions through local interactions. Bitcoin and other blockchain technologies [76] are other examples of distributed systems. Another potential application for distributed systems concerns intelligent traffic network control systems that, in a distributed manner, minimize time spent in queues caused by traffic conflicts or congestion to improve driver liveness and safety [26].

In distributed systems, agents are typically arranged in two main ways: client-server or peer-to-peer systems. (Other arrangements include multitier systems and pipeline systems.) In client-server systems, the clients request resources and the server provides those resources. Web servers and

Stellenbosch University https://scholar.sun.ac.za

web browsers function together as a client-server system. In peer-to-peer systems, which we consider here, agents are equal participants in communication and tasks are equitably distributed between all the agents. This means that agents both provide and use resources. Gnutella is a good example of a peer-to-peer network [66]. An illustration of a distributed system network is given in Figure 2.2a.

Distributed systems can be centralised (e.g. client-server systems with a single server) or decentralised (e.g. client-server systems with multiple servers). In centralised systems (see Figure 2.2b), there is a single agent that receives and fuses information from all other agents. This single agent can distribute information as required. On the other hand, decentralised systems are defined to be distributed systems in which processing is distributed to various agents [23, 24]. In other words, a decentralised system is a distributed system with multiple central agents (see Figure 2.2c). If all the agents are central, then the system is fully decentralised. Durrant-Whyte [23] gives an overview of decentralised systems.



(a) (b) (c)

Figure 2.2: *Interaction paradigms in multiagent systems.* ***2.2a****: Example of a distributed system where agents have local interactions.* ***2.2b****: Example of a centralised system with a central agent (the red agent).* ***2.2c****: Example of a decentralised system with seven central agents (the red agents).*

Unlike decentralised systems, in a centralised system the whole system fails when the central agent malfunctions. The failure of a node is easier to repair in a centralised system than in a decentralised system [20]. In this thesis, we consider decentralised distributed systems. We treat the system of solitary robots as a distributed system to model the limited communication capacity of solitary robots, and as a decentralised system to model the self-interest of solitary robots. We now briefly describe another relevant aspect of distributed systems: communication protocols.

Stellenbosch University https://scholar.sun.ac.za

2.4.2 Communication protocols

In interaction of robots, communication methods are required to facilitate the exchange of messages between agents. Communication protocols define rules for exchanging information. Methods for communication may differ depending on the paradigm of the system. In a centralised system for instance, methods such as semaphores [20] can be used.

To exchange data between robots, the following must be specified:

- nature of data: the type of information robots will exchange (e.g. maps);
- address format: how the sender and receiver are identified (e.g. robot identifier);
- routing mechanism: selection of a path to pass data along from the sender to the receiver when they are not directly connected (e.g. use of a spanning tree [24]); and
- optionally, a mechanism for detection and correction of transmission error, to ensure the success of data exchange.

For communication between two devices, there are two ways of transmitting signals from one device to the other: synchronous and asynchronous. For synchronous transmission, both devices use a common clock signal (typically the clock signal of the sender). The receiver uses the clock signal sent by the sender. Online conferencing systems such as Skype are a good example of synchronous transmission. For asynchronous transmission, there is no common clock signal between the sender and receiver, and information is relayed with some time lag. Electronic mail delivery is a good example of asynchronous communication. This thesis considers asynchronous transmission, because the time interval between messages exchanged by interacting robots is not constant and there is a need to distinguish messages exchanged between interacting robots at different times.

In a distributed system, each agent is an isolated component and agents use message passing methods to exchange information [20]. In message passing methods, multiple robots communicate locally using message queues. This thesis considers message passing methods for decentralised systems, because of the nature of solitary robots: robots with restricted communication capacity.

Stellenbosch University https://scholar.sun.ac.za

2.4.3 Leader election based on centrality measures

Solitary robots can interact with each other. In interaction of robots, a natural way of facilitating interaction of robots is to choose a leader that can coordinate the interaction. In our thesis, a leader is elected based on a closeness centrality because choosing a central node leader has the potential to reduce the propagation time of communication.

There are several algorithms that address the problem of leader election based on centrality measures. Naz [77] recently conducted a survey on the state-of-the-art leader election methods based on centrality measures. Before specifying some aspects considered in designing a leader election method, we note three relevant properties required of a leader election algorithm as specified in Coulouris et al. [20]:

1. termination: the process of choosing a leader should reach completion;
2. uniqueness: exactly one node must be elected as leader; and
3. agreement: the elected leader should be known and accepted by all participants on completion.

In the literature, some leader election methods require the construction of a spanning tree [34, 49, 86]. Spanning trees are important in interaction of robots because they are acyclic (i.e. there are no cycles). Cycles (also known as bridge loops) in networks, can cause repetitions of information during communication which can cause extreme amounts of communication traffic or require high processing time—which can significantly degrade network performance.

Methods to elect a leader differ, depending on the communication paradigms used (e.g. synchronous or asynchronous systems), how information is exchanged (e.g. centralized or decentralized systems), whether robots are uniquely identifiable or not, and the network topology [77]. We consider the problem of distributed leader election methods in networks of spatially separated robots.

There are various centrality measures; unlike in the method proposed by You et al. [118] where three centrality measures are considered, we only consider closeness centrality [8, 13] because we aim to know the node with the shortest distances to all other nodes in a network. In this thesis, we consider a decentralised leader election method based on closeness centrality

Stellenbosch University https://scholar.sun.ac.za

where each robot, in a distributed manner, is responsible for estimating its own closeness centrality [118].

Let δ_{ij} denote the path distance between the nodes v_i and v_j in an (unweighted) graph G with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. The distance between two nodes is the length of the shortest path between these nodes.

Definition 2.4.1. *The closeness centrality [8] of a node is the reciprocal of the average path distance from the node to other nodes. Mathematically, the closeness centrality c_i is given by*

$$c_i = \frac{|\mathcal{V}| - 1}{\sum_j \delta_{ij}} . \tag{2.4.1}$$

Nodes with high closeness centrality score have the shortest average path distances to all other nodes.

2.4.4 Dynamic networks

In the literature, a dynamic network is a network in which the topology changes over time, i.e., the set of vertices or the set of edges can change over time [57]. A typical example of such networks is a mobile ad-hoc network, where nodes can join or leave–thus, communication can appear or disappear at any time. There are three kinds of connectivity changes that describe the dynamics of a graph: node or edge deletion only (decremental connectivity), node or edge addition only (incremental connectivity), or both node or edge deletion and addition (fully dynamic connectivity). Solutions to problems such as the verification of the connectivity of a network after node deletion induced by the dynamics of a graph, exist for each type of connectivity. For instance, Shimon and Yossi [90] solved the case of decremental connectivity with edge deletion only. The solution proposed by Shimon and Yossi [90] answered the question: If we are given an undirected graph where edges are deleted one at a time, are two nodes v_i and v_j of the network in the same connected component? This may be important in applications where each node needs to know the size or the topology of the component to which it belongs.

In this thesis, the aspect of dynamic networks comes into play because a node can fail to communicate with another node. When a node fails to communicate with another node, the graph topology changes because it is the same as if an edge were deleted from the network. Failed nodes can be

Stellenbosch University https://scholar.sun.ac.za

restored. Restoration of nodes also defines some dynamics because addition of nodes changes the graph topology.

2.5 Conclusion

This thesis consists of proposing active strategies for coordination of solitary robots. A solitary robot is self-interested (i.e. a robot without a *common social goal*), autonomous, has a restricted communication capacity, and is equipped with an active ability to amend its behaviour when it acquires new information. A solitary robot does "the best it can" while not being aware of other robots, which it can accidentally meet with. When robots accidentally encounter each other, they can collaborate to work more effectively. This chapter reviewed and discussed relevant literature in the context of distributed systems.

When a system is composed of multiple robots, there are three approaches to model the interaction of robots, namely: distributed, centralised and decentralised systems. This thesis considers distributed systems which are decentralised because each robot does processing, and robots have restricted communication ability.

Four other important topics were also described. First, the idea of a configuration space for robots to search is required. Then, models of robot motion were discussed. We also considered how robots obtain and represent information from the environment.

For the case of search by solitary robots which is the topic of this thesis, maps are represented by grids. For motion commands, we provide to the robot information about its destination which it considers to determine its translational and rotational velocities. We also discussed two types of maps: location-based and feature-based. In the case of this thesis, we are not concerned about the type of features found in the environment except objects of interest. So a location-based map is suitable for search and rescue.

In what follows, we describe our three proposed methods in Parts II–III respectively:

- Part I reviews existing methods for coordination, describes our proposed method and discusses the results of coordination of solitary robots.

Stellenbosch University https://scholar.sun.ac.za

- Part II reviews existing methods for view construction, describes our proposed method and discusses the results of view construction.
- Part III reviews existing methods for classifier fusion, describes our proposed method and discusses the results of classifier fusion.

We now review background work and literature for the first problem: coordination.

Stellenbosch University https://scholar.sun.ac.za

Part I

Coordination

Stellenbosch University https://scholar.sun.ac.za

Chapter 3

Background and literature

Search and rescue by solitary robots is the overarching challenge of this thesis. This chapter presents background and reviews literature for coordination methods for solitary robots. We aim to design a coordinated search strategy for solitary robots that allows them to explore more effectively, where the robots have no prior information about each other or the search environment.

3.1 General background

Coordination is important in robotics applications where a group of robots must achieve some task [5, 17, 18, 29, 41, 53, 83, 85, 93, 113]. This work considers the case of multiple robots searching an unknown environment.

Mobile robots typically have constrained communication [75]. Amigoni et al. [3] recently conducted a survey of solutions to problems of limited communication. One of the main concerns arising when robots with limited communication capacity explore an unknown environment is *interference*: robots may obliviously search overlapping regions, which reduces their combined efficiency.

The only time that robots detect interference is when they detect each other. Under limited communication, the meeting of solitary robots is just a fortunate coincidence (also known as symmetric rendezvous [2]).

Traditionally, rendezvous of robots in a network refers to a control algorithm which brings together a group of robots that know about each other, typically at the centroid of their initial positions [44]. In this work, due to the

Stellenbosch University https://scholar.sun.ac.za

nature of solitary robots, we do not treat rendezvous as a control algorithm. Rather, we use the term rendezvous to refer to a meeting of solitary robots. We adapt the term rendezvous as robots must take advantage of their interaction, which means robots involved in a meeting behave as if they were from a team (i.e. formation control) to work more effectively. Once interaction is terminated, each robot again behaves as a solitary robot.

When search involves multiple robots, one would like a group of robots that know about each other to explore non-overlapping regions—this is termed *sustainability* [42]. We must design methods which ensure the effectiveness of robots at using information they possess. Consequently, each member of a group of robots that know about each other might explore a non-overlapping region with respect to other members. The more a robot is able to avoid exploring already explored areas, the more sustainable its performance becomes. In other words, low interference implies high sustainability.

We assume that when a robot starts searching, the only information it has about the environment is the region that it can perceive from its initial location. The scale and boundary locations of the environment are also unknown, although a boundary can be sensed when within perception range. It is also assumed that robots have a limited time for search: the environment is large and robots could not explore it entirely, even were the environment known [109]. The constraint on time could be, for example, due to limited battery power of robots.

This work applies a cellular decomposition strategy [17] to the search of an unknown environment by solitary robots under a time constraint. Cellular decomposition is a technique in which an environment is divided into non-overlapping regions (see an illustration of cellular decomposition in Figure 3.1) so that every region can be assigned to at least one robot for coverage [21].

3.2 Literature discussion and frontier-based exploration

In the literature, the problem of coordination of robots is addressed by various methods, including methods based on line of sight [5], frontiers [72] and

Stellenbosch University https://scholar.sun.ac.za

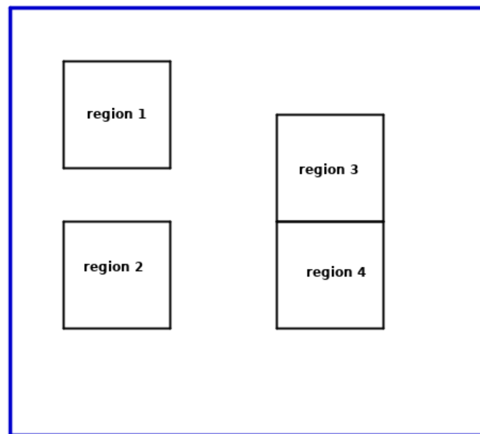


Figure 3.1: *Illustration of cellular decomposition with four regions. The blue lines denote boundaries of the search space and the black squares are regions.*

rendezvous [109]. The aim of coordination is to improve the robots' performance. For the coordination of solitary robots, two coordination methods are found to be appropriate, namely the accidental rendezvous [42] and periodic rendezvous strategies [109]. The accidental rendezvous strategy refers to a strategy where robots only meet accidentally, while the periodic rendezvous strategy refers to a strategy where robots which are initially aware of each other schedule future meetings.

The accidental and periodic rendezvous strategies apply a frontier-based approach for exploration of unknown environments [114, 115]. Frontier-based exploration is the process of selecting frontiers that yield the biggest contribution to a specific utility at an unknown environment. Frontiers are regions at the boundary between explored and unexplored regions. So moving to frontiers, a robot will gain as much information as possible. It should be noted that a frontier is selected based on single points which lie on borders.

For frontier-based exploration, various utility functions, including exploration time utility (minimize the search time) and localization utility (minimize the localization error), can be used to select the next frontier to visit. Gomez et al. [37] conducted a survey of frontier-based exploration algorithms. Utility functions are used for different purposes and subject to various assumptions. For example it does not make sense to use localization utility here because we assume that maps built by robots are accurate. Similarly, we can not sensibly use exploration time utility because exploration

Stellenbosch University https://scholar.sun.ac.za

time is assumed to be limited and common to all robots.

For our purposes, we consider two main utility functions for selection of frontiers, namely: navigation utility (minimize the displacement to frontiers) and information utility (maximize the amount of new information that can be acquired). First, for navigation utility, a robot only considers frontiers within its perception range. We use information utility because it suits the search and rescue task best. Second, to quantify the information utility, a robot considers balls centred at points at the edge of its perception range and selects the point for which the ball contains the maximum amount of unexplored region. There might not be frontiers within perception range of a robot. In that case, the robot will uniformly and randomly choose one point at the edge of its perception range to move to. It should be noted that it is possible that multiple frontiers yield the same contribution. When that happens, robots will choose the first frontier it has scanned among those which yield the maximum contribution. The robot selects the next point to move to based on these two utilities using a procedure GETNEXTTARGETPOINT in Algorithm 3.1. There are three conditions for a solitary robot to select a new target point: either after reaching its previous target point, when the robot realizes that it takes longer to reach its previous target point than initially planned or after an interaction with other robots. (The last condition will be discussed in the following paragraph). A frontier-based exploration method is illustrated using FRONTIERBASEDEXPLORATION in Algorithm 3.1. A robot avoids encountered obstacles using a Bug algorithm [80] our implementation of which is given in procedure DISTANCEBUG in Algorithm B.1 (see Appendix B).

The accidental rendezvous strategy is an interesting approach which provides a solution to mitigate interference: at each rendezvous, the search space is divided into sectors (i.e. unbounded regions) to which robots are assigned. For example, the westernmost part of the environment from a reference location (e.g. meeting point) might be assigned to a single robot (Figure 3.2a). An unbounded region is represented by a target point which we term the *coordination target point*.

The interacting robots use a procedure GETCOORDINATIONTARGETPOINTS (where the parameter h denotes how far the interacting robots should move in their respective directions) in Algorithm 3.1 to spread themselves to non-overlapping sectors. (We abstract away the details of interaction of robots in

Stellenbosch University https://scholar.sun.ac.za

Algorithm 3.1 *Our presentation of a frontier-based exploration method. The algorithm gives the code run on a robot R_i. The main method is* UPDATE *which is run by R_i once each step. An example of code to execute the class is given in the procedure* RUNARS*. Robots which are involved in an interaction are called to choose a leader which can facilitate their interaction. For the next step of coordination, after leader election, each member sends its personal information to the leader elected (using a variant of* DATAFUSION *described in Algorithm 5.3). Then after it has combined all individual information from other robots, the leader elected assigns coordination targets to other members (using a variant of* TASKASSIGNMENT *described in Algorithm 5.3). The variable h in this algorithm is a hyperparameter: we choose $h = 100$ after several attempts with various values.*

```
1:  procedure RUNARS()
2:      ARSrobot←ARS(τ, d, γ, Δt, T_w, D, η, N, h)
3:      while not ARSrobot.ISENDED() do
4:          ARSrobot.UPDATE()
5:      end while
6:  end procedure
7:
8:  class ARS
9:      Class variables
10:     a_i               its current target point
11:     Bug               object for obstacle avoidance
12:     C_i^(t)           its explored region
13:     γ                 motion velocity
14:     Γ_ij              hash table of exploration regions of tree descendants via its tree child R_j
15:     Δt                time step
16:     d                 robot perception range
17:     D                 maximum number of iterations for graph construction
18:     D_i               set of tree descendants of R_i
19:     D_ij              set of tree descendants of R_i via its tree child R_j
20:     ε                 a small threshold for stopping condition
21:     electionObject    object for leader election
22:     GO                a variable to control the steps of leader election
23:     G_i               set of previous meetings of R_i and their times
24:     h                 a variable indicating how far robots should move in their directions
25:     I_i               identifiers of robots currently in interaction known by R_i
26:     J_i               identifiers of interacting robots it knows about
27:     L_i               a hash table with robot identifiers as keys and their locations as values
28:     N                 sampling size
29:     N_i               immediate neighbours
30:     O                 set of obstacles
31:     planningState     a variable to indicate robot state during planning
32:     pruningObject     object for network view construction
33:     R_i               queue messages received by R_i during task assignment
34:     s                 number of steps required to check quantities of explored area
35:     robotState        a variable to indicate robot state
36:     S_i               queue messages received by R_i during data fusion
37:     stepB             a variable to control the steps of network view construction
38:     τ                 maximum search time
39:     treeChildren      its tree children
40:     treeParent        its tree parent
41:     t_w               time elapsed since robot has been waiting
42:     T_w               maximum time robots can wait for other robots before they start planning
43:     x_i               its current location
44:
45:     constructor (τ, d, γ, Δt, T_w, D, η, N, h)
46:         (τ, d, γ, Δt, h) ← (τ, d, γ, Δt, 100)
47:         (N, T_w, D, t, C_i^(0), J_i, G_i) ← (N, T_w, D, 0, B_d(x_i), ∅, ∅)
48:         (A, η, t_w) ← (∅, η, 0)
49:         (treeChildren, treeParent)←(∅, ⊥)
50:         Bug← BUG(d, γ)                                   ▷ using Algorithm B.1
51:         pruningObject ←⊥
52:         electionObject ←⊥
```

Stellenbosch University https://scholar.sun.ac.za


```
53:        robotState ← ROBOTSTATES.SEARCHING                          ▷ using Algorithm F.1
54:        planningState ← PLANNINGSTATES.NONE                         ▷ using Algorithm F.2
55:        searchState ← SEARCHSTATES.INREGION                         ▷ using Algorithm F.3
56:        a_i ← GETNEXTTARGETPOINT()
57:    end constructor
58:
59:    procedure UPDATE()
60:        if not ISENDED() then                                       ▷ using Algorithm 5.3
61:            t_0 ← currentTime()
62:            if robotState=ROBOTSTATES.SEARCHING or robotState=ROBOTSTATES.WAITING then
63:                (I_i, W_i) ← NEWINTERACTION()                       ▷ using Algorithm 5.3
64:                if |I_i| > 0 then
65:                    (𝓘_i, 𝓦_i) ← (I_i, W_i)
66:                    robotState ← ROBOTSTATES.STOP
67:                    t_w ← t
68:                else if |W_i| > 0 then
69:                    robotState←ROBOTSTATES.WAITING
70:                else
71:                    SEARCHING()
72:                end if
73:            else if robotState=ROBOTSTATES.STOP then
74:                if t − t_w ≤ T_w then
75:                    WAITING()                                       ▷ using Algorithm 5.1
76:                else
77:                    robotState←ROBOTSTATES.PLANNING
78:                end if
79:            else if robotState=ROBOTSTATES.PLANNING then
80:                if planningState=PLANNINGSTATES.NONE then
81:                    pruningObject ← PRUNING(𝓘_i, D)                 ▷ using Algorithm 8.1
82:                    pruningObject.INITIALONEHOP()
83:                    planningState ← PLANNINGSTATES.CONSTRUCTION
84:                    stepB ← "update"
85:                    GO ← false
86:                else if planningState=PLANNINGSTATES.CONSRUCTION then
87:                    TREECONSTRUCTION()                              ▷ using Algorithm 5.3
88:                else if planningState=PLANNINGSTATES.ELECTION then
89:                    LEADERELECTION()                                ▷ using Algorithm 5.3
90:                else if planningState=PLANNINGSTATES.FUSION then
91:                    DATAFUSION()
92:                else if planningState=PLANNINGSTATES.TASK then
93:                    TASKASSIGNMENT()
94:                end if
95:            end if
96:            t_1 ← currentTime()
97:            Δt_01 ← max(0, T_w − t_1 + t_0) ▷ R_i shall wait for some time T_w before it can increment its counter.
98:            DELAY(Δt_01)
99:            t ← t + Δt
100:           C_i^(t) ← C_i^(t−Δt) ∪ B_d(x_i)
101:           ΔT_i ← ΔT_i + Δt
102:       end if
103:   end procedure
104:
105:   function GETNEXTTARGETPOINT()
106:       α ← uniform(0, 2π)
107:       a ← x_i + d(cos α, sin α)
108:       (u, β) ← (|B_d(a) ∩ C_i^(t)|, 2π/N)        ▷ B_d(a) denotes a closed ball of radius d centred at a
109:       for j = 1 to N do
110:           α ← α + β
111:           b ← x_i + d(cos α, sin α)
112:           if |B_d(b) ∩ C_i^(t)| < u then
113:               a ← b
114:               u ← |B_d(b) ∩ C_i^(t)|
115:           end if
116:       end for
117:       return a
118:   end function
119:
120:   procedure REINITINTERACTIONINFO()
121:       (treeChildren, treeParent)←(∅, ⊥)
```

Stellenbosch University https://scholar.sun.ac.za

122: $(\mathcal{I}_i, \mathcal{W}_i, \mathcal{L}_i, \Gamma_i, \mathcal{D}_i) \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
123: $t_w \leftarrow 0$
124: $\Delta T_i \leftarrow 0$
125: $A \leftarrow \emptyset$
126: `planningState` $\leftarrow$ PLANNINGSTATES.NONE
127: `robotState` $\leftarrow$ ROBOTSTATES.SEARCHING
128: $y_0 \leftarrow x_i$
129: $\Delta y \leftarrow 0$
130: $A_B \leftarrow \emptyset$
131: **end procedure**
132:
133: **procedure** RELAYINFORMATION()
134: $F \leftarrow$ `treeChildren.clone()`
135: **while** $F \neq \emptyset$ **do**
136: R_i chooses an $R_j \in F$
137: $\Gamma_{ij} \leftarrow \emptyset$
138: **for** $R_k \in \mathcal{D}_{ij}$ **do**
139: $\Gamma_{ij}[R_k] \leftarrow a_k$
140: **end for**
141: R_i sends $\mathcal{C}_i^{(t)}$, $\mathcal{J}_i$, $\mathcal{G}_i$, and Γ_{ij} to R_j
142: $F \leftarrow F \setminus \{R_j\}$
143: **end while**
144: **end procedure**
145:
146: **procedure** TASKASSIGNMENT()
147: **if** `treeParent` $= \perp$ **then**
148: GETCOORDINATIONTARGETPOINTS()
149: **for** each tree child R_j of R_i **do**
150: **for** $R_k \in \Gamma_{ij}$`.keys()` **do**
151: $a_k \leftarrow \Gamma_{ij}[R_k]$
152: $\mathcal{G}_i[R_k] \leftarrow t$ ▷ $\mathcal{G}_i$ is a hash table
153: **end for**
154: **end for**
155: RELAYINFORMATION()
156: **else**
157: **while** $\mathcal{R}_i$`.size()` ≥ 1 **do**
158: $(j, \mathcal{C}_j^{(t)}, \mathcal{J}_j, \Gamma_{ji}, \mathcal{G}_j) \leftarrow \mathcal{R}_i$`.dequeue()`
159: $(\mathcal{C}_i^{(t)}, \mathcal{J}_i) \leftarrow (\mathcal{C}_j^{(t)}$`.clone()`, $\mathcal{J}_j$`.clone()`$)$
160: $\mathcal{G}_i \leftarrow \mathcal{G}_i \cup \mathcal{G}_j$
161: RELAYINFORMATION()
162: **end while**
163: **end if**
164: REINITINTERACTIONINFO()
165: **end procedure**
166:
167: **procedure** DATAFUSION()
168: $\mathcal{L}_i[R_i] \leftarrow x_i$ ▷ $\mathcal{L}_i$ is a hash table
169: $\mathcal{D}_i \leftarrow \emptyset$
170: **if** `treeChildren` $= \emptyset$ **then**
171: R_i sends $\mathcal{C}_i^{(t)}$, $\mathcal{I}_i$, $\mathcal{L}_i$ and $\mathcal{D}_i$ to `treeParent`
172: `planningState` $\leftarrow$ PLANNINGSTATES.TASK
173: **else**
174: **for** tree child R_j of R_i **do**
175: $\mathcal{D}_{ij} \leftarrow \emptyset$
176: **end for**
177: **while** $\mathcal{S}_i$`.size()` ≥ 1 **do**
178: $(j, \mathcal{C}_j^{(t)} j, \mathcal{I}_j, \mathcal{L}_j, \mathcal{D}_j) \leftarrow \mathcal{S}_i$`.dequeue()`
179: $\mathcal{N}_i^{(t)} \leftarrow \mathcal{N}_i^{(t)} \cup \mathcal{N}_j^{(t-1)}$
180: $(\mathcal{C}_i^{(t)}, \mathcal{I}_i) \leftarrow (\mathcal{C}_i^{(t)} \cup \mathcal{C}_j^{(t)}, \mathcal{I}_i \cup \mathcal{I}_j)$
181: $(\mathcal{J}, \mathcal{D}_{ij}) \leftarrow (\mathcal{J}_i \cup \mathcal{J}_j \cup \mathcal{I}_i, \mathcal{D}_{ij} \cup \mathcal{D}_j)$
182: $\mathcal{L}_i$`.add`$(\mathcal{L}_j)$ ▷ adding a hash table into another hash table
183: $\mathcal{D}_{ij} \leftarrow \mathcal{D}_{ij} \cup \mathcal{D}_j$
184: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \mathcal{D}_j$
185: **end while**
186: **if** `treeParent` $\neq \perp$ **then**
187: R_i sends (its combined information) $\mathcal{C}_i^{(t)}$, $\mathcal{I}_i$, $\mathcal{L}_i$ and $\mathcal{D}_i$ to `treeParent`

Stellenbosch University https://scholar.sun.ac.za


```
188:            planningState ← PLANNINGSTATES.TASK
189:        else
190:            planningState ← PLANNINGSTATES.TASK
191:        end if
192:      end if
193:    end procedure
194:
195:    procedure SEARCH()
196:      x_i ←Bug.DISTANCEBUG(A, x_i, a_i, C_i^(t), O)
197:      Δy ← Δy + γ
198:      if Δy > ηδ(a_i, y_0) or δ(a_i, x_i) < ε then
199:          a_i ← GETNEXTTARGET()
200:          y_0 ← x_i
201:          Δy ← 0
202:      end if
203:    end procedure
204:
205:    procedure GETCOORDINATIONTARGETPOINTS()
206:      P ← L_i.size()
207:      c ← GETCENTRALPOINT()                                   ▷ using Algorithm 5.3
208:      α ← uniform(0, 2π)
209:      β ← 2π/P
210:      for i = 1 to P do
211:          b_i ← c + h(cos α, sin α)
212:          α ← α + β
213:      end for
214:      B ← (b_1, b_2, ··· , b_P)
215:      robotsIDS, cost ← GETCOSTMATRIX(B)                     ▷ using Algorithm 5.3
216:      ids, indices ← HUNGARIAN(cost)     ▷ from SciPy [15] assuming that P robots
        are interacting. This subroutine returns identifiers corresponding to closest coordination target
        points for the P robots based on their locations
217:      for (j, k) ∈ zip(ids, indices) do
218:          r ← robotIDS(j)
219:          Γ_i(r) ← b_k
220:      end for
221:    end procedure
222: end class
```

Algorithm 3.1 because they will be discussed in Chapter 5.) Each robot will then resume the use of the procedure GETTARGETPOINT above mentioned once it has reached its coordination target.

However, dividing areas into sectors can result in unbalanced assignment, which promotes interference [42]—robots end up with different-sized areas to explore. For example, an unbalanced assignment arises when robots interact close to a boundary of the environment of which they are unaware while coordinating. In Figure 3.2b for instance, the areas assigned to two of the robots (the blue and orange trajectories indicate these robots) are partly outside the environment, something the robots are unaware of when coordinating, as boundaries of the environment are unknown until encountered. Thus, assigning sectors under the accidental rendezvous strategy may lead to interference [42] because sectors are unbounded. The key is-

Stellenbosch University https://scholar.sun.ac.za

sue in such scenarios is that robots have no fallback plan and can not communicate with the other robots again. To address unbalanced assignment,

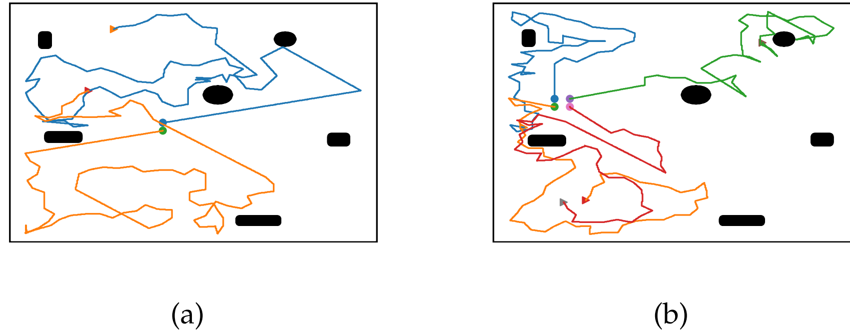


Figure 3.2: *Illustration of assignment of sectors upon an accidental rendezvous with four robots (coloured circles are robots, the lines show their respective trajectories).* ***(3.2a)****: Balanced assignment with two robots: robots start at the centre of the environment. The chance of searching in overlapping areas is small.* ***(3.2b)****: Unbalanced assignment with four robots: robots are unaware that they start close to a boundary and since sectors are unbounded, the chance of overlap is high (see crossed lines).*

robots could schedule further meetings during their rendezvous to share additional information—the periodic rendezvous strategy [109]. In the periodic rendezvous strategy, robots schedule further rendezvous whenever they meet and they all meet simultaneously. Rendezvous points are in already explored areas: Unexplored regions may be occupied by obstacles, so setting rendezvous points in unexplored regions may not work. This strategy reduces interference, as robots share additional knowledge on a regular basis. However, it suffers from the problem of *interruptibility*: robots cease new knowledge acquisition when they travel to the rendezvous point: robots travel through explored areas to avoid missing a rendezvous, especially when the environment is occupied by obstacles. It has been shown that interruptibility can consume up to half of the search time [42].

Coordination methods exist which mitigate interruptibility: one is the method proposed by de Hoog et al. [22], which uses two types of robots, explorers and relays, and a base station. The explorers intentionally meet and share knowledge with the relays. The relays then transfer the shared information to the base station. Interruptibility is mitigated by the fact that the explorers and relays meet closer to the frontiers of the explorers (i.e.

Stellenbosch University https://scholar.sun.ac.za

they meet at the boundaries between explored and unexplored areas of the explorers). This method requires a team hierarchy and typically aims to gather all information at a central location, whereas sector-based methods do not. In the approach taken in this thesis, all robots are explorers and no base station is necessary.

Another solution in the literature which mitigates interruptibility has robots plan to meet other robots during their exploration time before the rendezvous time arrives [42]. Two challenging aspects must be handled in this case: (i) forecasting the current positions of the robots involved; and (ii) forecasting the paths to be taken by the robots from their current positions. To address these challenges, the method proposed by Hourani et al. [42] is based on serendipity. Usually, when robots interrupt exploration to attend a rendezvous, new information might not be gained as they travel to the rendezvous point. To address this negative impact, some robots, which are referred to as serendip robots, plan to interact with their colleagues before the rendezvous time arrives by forecasting their paths. When serendip robots meet other robots, they can all coordinate before their scheduled rendezvous time and at some other location than their planned rendezvous point (the chance for serendip robots to meet all other robots before their rendezvous time is very small). Thus interruptibility might be mitigated. However, this approach has only been considered for obstacle-free environments: Hourani et al. [42] used graph-based maps, where nodes and edges of the graph were obtained from free-obstacle areas, to test the suitability of their approach.

The periodic rendezvous strategy was proposed for a team (not for solitary robots) with unlimited search time. Wellman et al. [109] treat situations where robots know about each other from the outset. The authors of the accidental and periodic rendezvous methods considered the question of how long a team of robots would take to explore some fraction (for example, 90%) of the search environment, given unlimited search time. Since coverage percentage is measured based on the size of the search environment, this is not a realistic stopping criterion if the size of the environment is unknown. Other realistic stopping criteria, for example exploration time, may be considered. Also, the periodic rendezvous method was originally tested in an environment with a very limited number and shape of obstacles, which is very unrealistic. In this thesis, we have a different research

Stellenbosch University https://scholar.sun.ac.za

question for coordination: how much area can robots cover within a given search time (i.e. limited search time).

This thesis proposes a coordination strategy that provides sustainable performance when the time for search is limited (specifically insufficient to cover the entire search area); it can also be used if only an upper bound on the available search time is known.

Since cellular decomposition allows robots to explore non-overlapping regions, we hope to mitigate interference. Cellular decomposition also allows robots to avoid the need to schedule further rendezvous, thus reducing the problem of interruptibility. Our proposed method is also in the class of accidental rendezvous strategies.

3.2.1 Optimal selection of coordination target points

Selection of coordination target points should be treated in depth as some coordination target points may be more optimally chosen than others—optimal selection leads to efficient exploration. Selection is optimal when it brings additional benefits with a favourable trade off. Burgard et al. [16] mentioned two criteria to make an optimal selection of coordination target points for exploration, which we can describe as follows.

- *Information utility*: selected coordination target points should lead robots to non-overlapping regions, which leads to efficient exploration.
- *Navigation utility*: costs (e.g. travelling distance or time) of the selected coordination target points should be relatively minimised. This means that robots involved in an interaction should choose coordination target points which are close to their meeting point.

3.3 Summary

This chapter reviewed background work and literature for coordination of solitary robots. To the best of our knowledge, most work done in this area considers coverage percentage as stopping criterion.

While research has been conducted in the area of coordination, there is still room for improvement in this area. Since coverage percentage is measured based on the size of the search environment, this is not a realistic

Stellenbosch University https://scholar.sun.ac.za

stopping criterion if the size of the environment is unknown. It should be noted that coverage percentage remains suitable for comparison to evaluate approaches. This thesis makes contributions by proposing a new approach based on exploration time, providing an improvement in terms of interference and interruptibility under certain conditions.

Stellenbosch University https://scholar.sun.ac.za

Chapter 4

Soft obstacle coordination strategy for solitary robots

Portions of Part I, and particularly this chapter, appear in Masakuna et al. [70].

4.1 Introduction

One of the main concerns encountered when robots with limited communication capacity explore an unknown environment is interference [109]. In Chapter 3, we identified a gap for effective maximization of coverage in situations where a known upper bound on the search time precludes search of the entire environment and robots are without a common (social) goal.

In this work, coverage is used as a proxy for finding targets: the exploration performance of a robot is taken to correspond to its area covered. This corresponds to optimizing the expected number of targets found assuming uniformly distributed independent target locations. We also assume that a robot can detect other robots in its perception range.

4.1.1 Contributions

Our contribution is to propose a coordination method with low interference and low interruptibility. Three novelties are considered in our proposed coordination method:

Stellenbosch University https://scholar.sun.ac.za

- use of bounded regions to robots for exploration (i.e. application of cellular decomposition);
- use of margins; and
- implementation of a hybrid approach (combination of accidental and periodic rendezvous methods).

We consider limited-time search. At coordination, robots use the remaining search time to bound the size of their exploration regions when applying cellular decomposition. In the case of assignment to an inaccessible region, the robot concerned must find a new region to explore. A key change from earlier approaches based on sectors [109] is that regions are bounded in size because of the known time bound. The robot considers regions assigned to other robots as soft obstacles. Thus, it eschews those regions—although a robot can move through soft obstacles if it would otherwise be surrounded by explored regions.

The bounded region assigned to a robot is termed an exploration region. Since the environment is unknown, a robot's exploration region may not be fully accessible: it may be occupied by one or more large obstacles, which limit the effective search space, or it might be smaller in size than first supposed because it is found (on exploration) to contain one or more boundary edges. To tackle this, we introduce the use of margins: robots leave some (believed to be) unexplored space around their assigned exploration regions. When a robot finds that it can not fully explore its assigned exploration region, it uses these margins to move to a new exploration region which it determines by itself. Our proposed method should thus have low interruptibility and interference.

The periodic rendezvous strategy requires an initial prearrangement of the rendezvous location for the robots. We implement a naive version of the periodic rendezvous strategy for solitary robots: all the robots are members of only one team and every member must be present at each rendezvous. We call the resulting approach a hybrid method because it essentially combines the accidental and periodic rendezvous strategies.

Stellenbosch University https://scholar.sun.ac.za

4.1.2 Limitations of our proposed coordination strategy

Our coordination strategy is not as widely applicable as other methods such as accidental rendezvous strategy and periodic rendezvous strategy. Some situations where our coordination method is not recommended or applicable are:

- when the search time is unlimited. With unlimited search time, there are suitable coordination methods such as in Wellman et al. [109], while our method can not be applied.
- when the robots are from a team. The purpose of a team is that robots keep collaborating while they explore. We consider robots with limited perception and communication capacity. There are many existing methods that robots with limited perception and communication capacity could apply so that they keep collaborating during the search process. Such methods include methods based on line of sight [5] and rendezvous [109].
- when the search environment is very constrained in terms of space (e.g. a long indoor corridor). Our proposed method assumes large open space, so it is recommended for outdoor search.

4.2 Considerations for the soft obstacle strategy

This section discusses two important aspects for coordination of solitary robots: the intuition underlying the proposed coordination strategy and a property guaranteeing satisfactory coordination of solitary robots. Before we discuss the intuition of our proposed method, we would like to formalise the problem of coordination.

Let $\mathcal{A}_i^{(\tau)}$ be the region explored by a robot R_i at the end of the search, where τ denotes a known upper bound on search time and N the number of solitary robots. The aim is to maximise the size of the combined $\mathcal{A}_i^{(\tau)}$. We do this approximatively using the following inclusion-exclusion principle,

$$\left| \bigcup_{i=1}^{N} \mathcal{A}_i^{(\tau)} \right| \geq \sum_{i=1}^{N} \left| \mathcal{A}_i^{(\tau)} \right| - \sum_{i=1}^{N} \sum_{\substack{j=1 \\ i \neq j}}^{N} \left| \mathcal{A}_i^{(\tau)} \cap \mathcal{A}_j^{(\tau)} \right| , \tag{4.2.1}$$

Stellenbosch University https://scholar.sun.ac.za

by trying to trade off maximizing the first term

$$\sum_{i=1}^{N} \left| \mathcal{A}_i^{(\tau)} \right|$$

with minimizing the second term

$$\sum_{i=1}^{N} \sum_{\substack{j=1 \\ i \neq j}}^{N} \left| \mathcal{A}_i^{(\tau)} \cap \mathcal{A}_j^{(\tau)} \right| . \tag{4.2.2}$$

Robots applying this approach are likely to achieve at least a lower search performance bound while keeping individual search performance high.

4.2.1 Intuition for the soft obstacle strategy

To contribute effectively to a group search, each robot should avoid searching areas others have already covered or are likely to cover in future. To achieve this, we make use of the following two insights:

1. **(Bounded exploration region)**. Since the upper bound on search time is known in advance, a robot can use this information to evaluate the maximum area it and other robots can cover.

2. **(Soft obstacles)**. If a robot knows the regions assigned to other robots, it can plan to avoid exploring those regions. Note that a robot becomes aware of other robots' assigned exploration regions only through rendezvous.

These insights are used in our approach when constructing the bounded exploration regions for the cellular decomposition strategy [17], described in Subsection 4.3.1. However, just as with the approach of sectors (unbounded regions which are used in the two benchmark strategies discussed in Chapter 3) applied in [109], there remain other major concerns when applying cellular decomposition in an unknown environment.

Let $\mathcal{X}_i$ denote the current exploration region of R_i. (An exploration region is assumed to be rectangular and is described in Algorithm F.4).

- The region assigned to a robot could be partly outside the environment, if the coordinating robots are not yet aware of boundary locations (e.g. region $\mathcal{X}_3$ in Figure 4.1).

Stellenbosch University https://scholar.sun.ac.za

- The exploration region of a robot is likely to be occupied by obstacles (e.g. the circular obstacle in $\mathcal{X}_4$ in Figure 4.1), since the environment is not all free space.

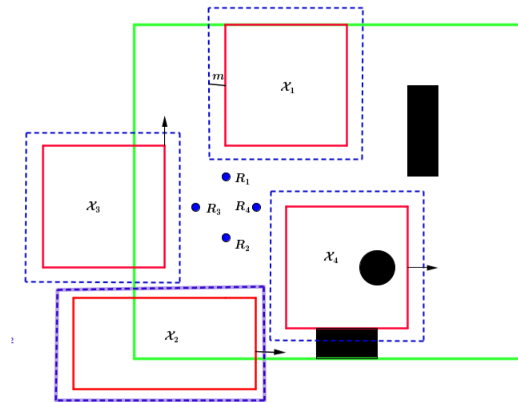


Figure 4.1: *Illustration of the proposed soft obstacle coordination with four robots. Solid black regions denote obstacles. The green lines denote boundaries of the search space, the red squares are assigned exploration regions $\mathcal{X}_i$ and the dashed lines are margins (which we will explain in Subsection 4.3.2).*

The two points described above can lead to robots having unbalanced assignments, making interference more likely. Let S, W and O (with $S \supseteq W \supset O$) denote an open space (i.e. an outdoor area), the search environment, and the set of obstacles respectively. The set S is an arbitrary superset of W to serve as a universal set (any set containing W). Given the grid representation used here (as discussed in Subsection 2.3.2), W is represented by a grid. The set of obstacles O is the set of cells that are occupied—robots learn about O as they explore. All boundaries of W are assumed to be in O.

Initially, a solitary robot is neither aware of other robots involved in the search space nor the locations of obstacles. To address this concern, a robot first creates a virtual world, meaning that some parts of its exploration region might not be in the real environment or may already be explored by other robots. When its exploration region is partly accessible, the robot explores the accessible part first (details on exploring accessible regions will be provided in Appendix A). Subsequently, the robot selects another exploration region if necessary, taking into account the exploration regions assigned to other robots that it is aware of which are treated as soft obstacles.

Stellenbosch University https://scholar.sun.ac.za

No matter what might prevent one robot from exploring its assigned exploration region (such as unbalanced assignment due to unexpected boundaries or inaccessible regions), other robots that are aware of this assigned exploration region are prohibited from searching in that exploration region.

In Figure 4.1, for example, most of the exploration region $\mathcal{X}_3$ assigned to robot R_3 is outside the environment (i.e. it is inaccessible). The robot R_3, after exploring the accessible area of its exploration region, must select another exploration region. If R_3 has encoded the exploration regions of the other three robots as soft obstacles, it should select an exploration region in the complement of the soft obstacle. In Figure 4.1, for example, R_3 can follow the direction indicated by the arrow above its exploration region $\mathcal{X}_3$.

So far, we have presented some intuition for the proposed coordination strategy for solitary robots. In what follows, we introduce a relevant property of coordination for effective coordination of solitary robots.

4.2.2 Unbounded world property

This section discusses a relevant property of successful coordination of solitary robots: *unbounded world*.

When the number of robots involved in search is unpredictable and the size of the search environment is unknown, robots can represent the search environment by an unbounded world. Each robot initialises its own virtual world without prior knowledge of the environment, but can grow it as necessary as the robot encounters other robots or obstacles. A robot starts with a small virtual world for which the size is based on the available search time (bound) τ. A robot resizes its virtual world as it encounters obstacles in its region or gains additional information: for example, when it encounters another robot which informs it that some part of its region has been explored already or allocated to other robots. Resizing a region involves expanding its virtual world to allow it to compensate for an inaccessible region or the area explored by or assigned to other robots. An unbounded world is required because the number of robots involved in search is unknown from the perspective of a single robot.

Let $\mathcal{V}_i$ denote the current virtual world of robot R_i. The robot uses this virtual world $\mathcal{V}_i$ (it is assumed to be rectangular) as a universal set encompassing portions of the search space W. Initially, the exploration region of

Stellenbosch University https://scholar.sun.ac.za

R_i is $\mathcal{X}_i$, and it uses the virtual world $\mathcal{V}_i = \mathcal{X}_i$.

We next describe our proposed coordination method for solitary robots: the soft obstacle strategy.

4.3 Soft obstacle coordination strategy

The proposed coordination strategy is based on the concept of soft obstacles. Regions that comprise the soft obstacles are first discussed, after which the coordination strategy is described.

The proposed strategy assumes the following:

- The environment is a static, large and initially unknown 2D world—the robots have insufficient time to cover it entirely.
- The boundaries of the environment are unknown but can be detected when encountered. Here boundaries of the search environment are treated as obstacles, i.e. robots can not distinguish boundaries of the search environment from obstacles.
- Robots are solitary and uniquely identifiable.
- Solitary robots are assumed to have the same coordinate system. Situations where each robot has its own internal coordinate system would require a suitable map merging method [54].
- Perception range is equal to communication range. This is for clarity of exposition, but it should be straightforward to extend the approach to relax the assumption.

The key concern with allocating sectors (i.e. unbounded regions) which this work aims to tackle is the behaviour of robots in cases of unbalanced assignment. We propose a way to allow a robot to explore outside of its allocated region while taking into account regions allocated to other robots, thus reducing interference. The main difference between our soft obstacle strategy and the strategy of sectors [109] is thus that the soft obstacle strategy assigns bounded regions to robots, while sectors are not bounded (as discussed in Chapter 3).

This idea of soft obstacles is similar to the solution to the problem of deployment of mobile sensors [43], where autonomous sensors use a reactive

Stellenbosch University https://scholar.sun.ac.za

strategy (which means a sensor takes actions by considering actions of other sensors) to spread out towards uncovered regions in the search space. The reactive strategy is based on potential fields. The fields are constructed such that each sensor is repelled by other sensors, and therefore sensors tend to move toward uncovered regions. The key difference between the method based on potential fields and our method is as follows. In the method based on potential fields, robots do not need a coordinator: nodes find exploration areas and repel from each other independently. This works well for robots with unlimited perception and communication range, i.e. a robot can detect any other robot in the search environment at any time. Our method requires a coordinator to assign exploration areas to participating robots. Our method is for situations where robots have limited perception and communication range.

In what follows, variable notations will consider the following: quantities typically changing at each time step incorporate a time index, while quantities only changing periodically do not incorporate a time index.

4.3.1 Constructing soft obstacles

Two regions are considered soft obstacles by a robot: the region R_i knows has already been explored at time t, $\mathcal{C}_i^{(t)}$, and the interference region $\mathcal{Z}_i$, i.e., the exploration regions assigned to the robots R_i is aware of, including itself. The soft obstacle $\mathcal{B}_i^{(t)}$ of robot R_i at time t is then

$$\mathcal{B}_i^{(t)} = \mathcal{C}_i^{(t)} \cup \mathcal{Z}_i \,. \tag{4.3.1}$$

The soft obstacles can take the shape of any bounded object. In this thesis, we use an occupancy grid-based approach [100], but an alternative representation, such as a geometric approach [111], could also be used.

During a rendezvous, robots exchange their interaction histories. Thus, two robots do not need to interact with each other directly to obtain information about each other. At the end of any coordination during the search, the explored regions of the involved robots are updated to include the other robots' explored and exploration regions. Let $\mathcal{I}$ denote a set of interacting robots. Consider an interaction involving robots with identifiers in the set $\mathcal{I}$, including R_i. The new explored region $\mathcal{C}_i^{(t)}$ of R_i is the union of explored

Stellenbosch University https://scholar.sun.ac.za

regions of the interacting robots, given by

$$\mathcal{C}_i^{(t)} \leftarrow \bigcup_{j \in \mathcal{I}} \mathcal{C}_j^{(t)} . \tag{4.3.2}$$

The updated interference region $Z_i^{(t)}$ of R_i is the union of interference regions of the interacting robots, given by

$$\mathcal{Z}_i \leftarrow \bigcup_{j \in \mathcal{I}} \mathcal{Z}_j . \tag{4.3.3}$$

Before interaction, the interference region of a robot is the union of its own previous exploration regions. The robot R_i includes its own previous exploration region in the interference region so that after this current interaction of robots, each robot will continue to explore its previous exploration region if there is still unexplored area in it before moving to a new exploration region.

In applying cellular decomposition for the proposed coordination strategy, the coordinator intentionally leaves unexplored spaces (or margins) around the robots' exploration regions. These margins allow sustained exploration when a robot leaves its exploration region to travel to a new exploration region for some reason—such as discovering that its region has been explored already or that it is partially obstructed. Such use of margins is part of the novelty of the proposed approach.

4.3.2 Use of margins

Let d be the perception range of robots. In Figure 4.1, margins are sections surrounding exploration regions, with m denoting the width of the margins. To prevent robots from moving through explored regions, the value of m should be at least $2d$. We set $m = 2d$ in this work. When searching, a robot may not be able to cover its exploration region entirely. Encountered obstacles or boundaries of the search environment, or interaction with other robots, are possible reasons a robot may not cover its current exploration region entirely. When this happens, the robot can use such margins to move from its current exploration region to another one. While margins are determined by the coordinator in interaction of robots, they are not included in the soft obstacles of the participating robots. Thus other robots can use

Stellenbosch University https://scholar.sun.ac.za

margins around another robot's exploration region when needed. But exploration regions allocated respect margins. Let $\mathcal{M}_i$ denote the margin area around the new exploration region assigned to R_i. Let $\mathcal{Q}_i$ denote the new exploration region assigned to robot R_i. The augmented exploration region of R_i is $\mathcal{M}_i \cup \mathcal{Q}_i$. In Chapter 5 we will see the benefit obtained by the use of these margins.

As mentioned above, in our proposed coordination method margins are selected during interaction only, i.e. a single robot does not select margins while it is in the process of selecting its personal exploration region. The reason that a single robot does not select margins in our proposed coordination method is because margins are used to allow robots that are from a recent interaction to avoid moving through exploration regions of other colleagues, especially when it is found that they can not search their assigned exploration regions effectively. The use of margins is found to be relevant for coordination because assigned exploration regions generally fall in the same area, so the chance that these assigned exploration regions to be near to each other is high. One needs to consider how robots should avoid moving through their colleagues' exploration regions when travelling to their new exploration regions. However, we note that the use of margins for a single robot might have some positive as well as negative impacts to our proposed method.

We next consider the implementation of the proposed soft obstacle strategy in coordination of robots.

4.3.3 Implementation of the soft obstacle strategy

Suppose P solitary robots with identifiers in $\mathcal{I}$ are involved in an interaction. During the interaction, the exploration regions of the participating robots typically change. A coordination strategy is required to select P exploration regions for $i \in \mathcal{I}$ to spread the P robots to non-overlapping areas. The participating robots apply the cellular decomposition strategy [17] which is the decomposition of search space into non-overlapping regions.

Before selecting the exploration regions, robots send their explored region (the $\mathcal{C}_i^{(t)}$'s), interaction histories and the interference regions (the $\mathcal{Z}_i$'s) to an elected leader using a procedure DATAFUSION in Algorithm 5.3. Let $t_\circ$ and t° denote times at the start and the end of interaction. If R_l is the leader,

Stellenbosch University https://scholar.sun.ac.za

the fused explored region is then $\mathcal{C}_l^{(t^\circ)}$. The elected leader can decide to enlarge its virtual world during the application of the cellular decomposition as discussed below. The leader is responsible for assigning non-overlapping exploration regions to the interacting robots. These exploration regions are found from unexplored areas of its virtual world. If the leader is unable to find exploration regions from its current virtual world, it must enlarge its virtual world create further unexplored area. In other words, enlargement of the fused virtual world takes place on demand if there is insufficient space or the sampling approach used has failed to find a suitable exploration region after a number of iterations.

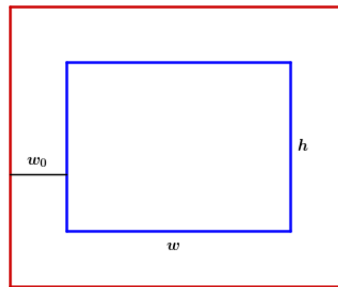


Figure 4.2: *Virtual world enlargement. The blue rectangle denotes virtual world before enlargement and the red rectangle denotes virtual world after enlargement.*

Enlarging the virtual world. The elected leader enlarges its virtual world as follows. Let w and h be the width and height of the virtual world before enlargement and $V(w_0)$ denote the new area to add to the virtual world for its enlargement where w_0 denotes the enlargement parameter as indicated in Figure 4.2 (w_0 is a user-defined parameter). The size of the set $V(w_0)$ is a

$$|V(w_0)| = 2w_0(h + 2w_0 + w)\,.$$

The virtual world of the elected leader will then be

$$\mathcal{V}_i \leftarrow \mathcal{V}_i \cup V(w_0)\,.$$

To enlarge its exploration region, a robot uses the procedure VIRTUALWORLDENLARGEMENT in Algorithm 4.1. The corresponding Python implementation for the virtual world enlargement is given in Appendix G.2.1—our implementation of virtual world enlargement assumes that a robot has sufficient

Stellenbosch University https://scholar.sun.ac.za

Algorithm 4.1 *Choice of exploration regions. The procedures defined in this environment are run by R_i. We use* `Python` *built-in functions for various basic set operations.*

```
1: Global variables
2:  B_i^(t)   soft obstacles of robot R_i at time t
3:  Γ_i       a hash table of assigned exploration regions
4:  d         perception range of R_i
5:  J_i       set of identifiers of interacting robots; V_i virtual world
6:  L_i       set of locations of interacting robots
7:  L         number of attempts of selecting a region before virtual world enlargement
8:  m         margins width
9:  N         sampling size
10: U_i^(t)   unexplored region
11: w_0       width for virtual world enlargement
12: ----------------------------------------------------------------
13: procedure GETGROUPEXPLORATIONREGIONS()
14:     VIRTUALWORLDENLARGEMENT()
15:     (k, U, P, l) ← (1, B_i^(t).clone(), |I_i|, 1)
16:     c ← GETCENTRALPOINT()                              ▷ using Algorithm 5.3
17:     A ← 2m + sqrt(Â(τ − t))
18:     while k ≤ P do
19:         next ← true
20:         l ← 1
21:         while next do
22:             S ← V_i \ U
23:             w ← (A/2)(1 + uniform(0, 1))
24:             h ← A^2 / w
25:             for j = 1 to N do
26:                 generate a sampled point (x_U, y_U) randomly and uniformly distributed over S
27:                 R ← RECTANGLE(x_U, y_U, w, h)                ▷ "using Algorithm F.4"
28:                 E ← RECTANGLETOSET(R)                        ▷ using Algorithm F.4
29:                 if E ∩ U = ∅ and E ⊆ S then
30:                     Q_k ← RECTANGLE(R.x + m, R.y + m, R.w − 2m, R.h − 2m)
31:                     F ← RECTANGLETOSET(Q_k)
32:                     B_k ← Q_k.GETCLOSESTCORNER(c)            ▷ using Algorithm F.4
33:                     U ← U ∪ E
34:                     next ← false
35:                     break
36:                 end if
37:             end for
38:             if l = L then
39:                 VIRTUALWORLDENLARGEMENT()
40:                 l ← 1
41:             end if
42:             l ← l + 1
43:         end while
44:     end while
45:     B ← (B_1, B_2, ··· , B_P)
46:     robotsIDS, cost ← GETCOSTMATRIX(B)
47:     ids, indices ← HUNGARIAN(cost)     ▷ from SciPy [15] assuming that P robots are
        interacting. This subroutine returns identifiers corresponding to closest exploration regions for
        the P robots based on their locations
48:     for (j, k) ∈ zip(ids, indices) do
49:         r ← robotIDS(j)
50:         Γ_i(r) ← Q_k
51:     end for
52: end procedure
53:
54: function GETCOSTMATRIX(B)
55:     cost ← []
56:     ids ← []
57:     for j ∈ L_i.size() do
58:         ids.add(j)
59:         row ← []
60:         for b ∈ B do
61:             a_i ← L_i[j]
62:             a ← δ(a_i, b)
63:             row.add(a)
64:         end for
```

Stellenbosch University https://scholar.sun.ac.za


```
65:         cost.add(row)
66:     end for
67:     return (ids, cost)
68: end function
69:
70: procedure VIRTUALWORLDENLARGEMENT()
71:     V ← 𝒱_i
72:     𝒱_i ← RECTANGLE(V.x − w_0, V.y − w_0, V.w + 2w_0, V.h + 2w_0)
73: end procedure
```

memory to represent the virtual world. From the fused information $\mathcal{C}_l^{(t^\circ)}$ and $\mathcal{Z}_l$, the soft obstacle $\mathcal{B}_l^{(t^\circ)}$ is constructed using the following equation

$$\mathcal{B}_l^{(t^\circ)} = \mathcal{C}_l^{(t^\circ)} \cup \mathcal{Z}_l \,.$$

At the end of coordination, each robot R_i sets $\mathcal{C}_i^{(t^\circ)} \leftarrow \mathcal{C}_l^{(t_\circ)}$, $\mathcal{Z}_i \leftarrow \mathcal{Z}_l$, $\mathcal{B}_i^{(t^\circ)} \leftarrow \mathcal{B}_l^{(t_\circ)}$ and $\mathcal{V}_i \leftarrow \mathcal{V}_l$ (all the interacting robots have the same virtual world $\mathcal{V}_l$) using a procedure TASKASSIGNMENT in Algorithm 5.3.

Once the robots' soft obstacles are known, a cellular decomposition respecting these soft obstacles is obtained using the following approach.

Choice of exploration regions. We intend to choose P exploration regions that will direct the P robots to disparate places. The P exploration regions are chosen from the unexplored region $\mathcal{U}_l^{(t)} = \mathcal{V}_l \setminus \mathcal{B}_l^{(t)}$ in the fused region known by the leader R_l. Since the required area of each exploration region can be determined from the search time bound, we first set the width and the height of one exploration region large enough for exploration region and margins. Then we arbitrarily choose bottom left corner of a rectangle with that width and height. We next need to check whether the chosen rectangle lies entirely within the unexplored region $\mathcal{U}_l^{(t)}$. To do that, the procedure consists of uniformly generating random points in the rectangle (the number of points is user-defined). We do not brute force check if a rectangle lies in the unexplored area because it is inefficient: every single point of the rectangle will need to be checked. The rectangle is considered valid if all the points generated are found in the unexplored region. If the elected leader fails to find P exploration regions in this way, it further increases the size of its virtual world. The coordinator can decide to increase its virtual world as much as required. This procedure is repeated until all the interacting robots have been assigned exploration regions. Recall that $\mathcal{Q}_i$ denotes the exploration region assigned to R_i from its current interaction.

Stellenbosch University https://scholar.sun.ac.za

The P regions are chosen sequentially: The exploration regions are chosen such that

$$\bigcup_{i\in\mathcal{I}}(\mathcal{Q}_i \cup \mathcal{M}_i) \subseteq \mathcal{U}_l^{(t)} \text{ and } \bigcap_{i\in\mathcal{I}}(\mathcal{Q}_i \cup \mathcal{M}_i) = \varnothing\,. \tag{4.3.4}$$

The procedure for determination of an exploration region by a leader is given in the procedure GETGROUPEXPLORATIONREGIONS in Algorithm 4.1. A Python implementation of the soft obstacle strategy can be found in Appendix G.2.2.

Exploration of previous exploration regions. When robots meet, their current exploration regions are also part of the soft obstacles. Although robots are given new exploration regions in an interaction, a robot which has enough unexplored space remaining in its previous exploration region will cover it first before moving to the new exploration region. Let $\mathcal{Y}_i$ denote a queue of available exploration regions for R_i and, recall that $\mathcal{X}_i$ and $B_i^{(t_\circ)}$ denote the exploration region of R_i and its soft obstacle before its current interaction. The robot R_i queues its previous exploration region $\mathcal{X}_i$ to $\mathcal{Y}_i$ if it contains enough unexplored space (at least one sweep through region is required):

$$|\mathcal{X}_i \setminus \mathcal{B}_i^{(t_\circ)}| > 2d\sqrt{wh}\,, \tag{4.3.5}$$

where w and h denote the width and height of $\mathcal{X}_i$ respectively.

Assignment of exploration regions. Once P exploration regions have been identified, the assignment of these exploration regions to the participating robots is performed using the Hungarian algorithm [58] implemented in SciPy [15] to minimize straight-line robot-to-region travelling time. The leader considers the closest corner of an exploration region when calculating the distance between a robot and the exploration region. Once the allocation is completed by the elected leader, the exploration regions are communicated to robots by the leader.

Python code that a robot uses to travel to its exploration region can be found in Appendix G.2.3.

While robots are leaving for their respective exploration regions after interaction, they will typically be within communication range of each other for some time. To prevent triggering further interactions between robots

Stellenbosch University https://scholar.sun.ac.za

with no significant additional information to share, robots leaving a meeting are prohibited from interacting with other robots from the same meeting until some distance $T_i^{(t)}$ has been covered. Let $\delta(x_i, a_i)$ be the distance between the current location x_i of robot R_i and the closest corner a_i of its newly assigned exploration region $\mathcal{Q}_i$. Then the distance $T_i^{(t)}$ is given by

$$T_i^{(t)} = \max_{j \in \mathcal{I}} \delta(x_j, a_j)\,, \tag{4.3.6}$$

(the longest distance to a new exploration region for a robot in the interaction). Obstacles or boundaries of the search environment can make a robot's exploration region inaccessible via the corner closest to its current location. If that happens, a robot considers the next closest corner of its exploration region as its new target to reach its exploration region. If encountered obstacles or boundaries prevent the robot from reaching all the corners of its exploration region, the robot selects another exploration region.

Another reason a robot may select another exploration region is if it has covered an unexpectedly long distance since its most recent interaction travelling to its exploration region without reaching it. We set the window distance, which a robot considers as a long distance since travelling, to be $\eta\delta(x_i, a_i)$ (the value η is user-defined).

Optimality of the selected exploration regions. Two criteria for effective coordination were discussed in Section 3.2. Our proposed approach has fully implemented the first criterion and has partly implemented the second criterion for optimal selection of exploration regions. As discussed in the description of SOS above, selected exploration regions are non-overlapping regions that maximise information utility (i.e. the first criterion).

The second criterion, which is navigation utility, is partly taken into account implicitly by the fact that exploration regions are selected from the virtual world. A virtual world could be relatively small, and we only enlarge it on demand when there is insufficient space to allocate exploration regions. Our argument here is that the approach has a tendency to lead to good selections because the virtual world limits examples with bad navigation utility. In what follows, we illustrate some aspects of our proposed coordination strategy.

Example. Consider three solitary robots R_1, R_2 and R_3 and assume they have not met up to time t. During a solitary search, each robot R_i selects

Stellenbosch University https://scholar.sun.ac.za

an exploration region $\mathcal{X}_i$ and updates its explored region $\mathcal{C}_i^{(t)}$ as it explores the environment. When a robot is performing a solitary search, the only challenge it will face is management of obstacles: a robot must avoid obstacles in a way that does not hinder its progress.

Let us now suppose that R_1 and R_2 meet at some time t_{12}. Upon the meeting of these two robots, seven things will happen.

(1) First, R_1 and R_2 will select one of them as a coordinator. Based on the approach we will use for leader election (discussed in Section 5.3), R_1 will be chosen as the leader.

(2) Second, R_2 will send its explored region $\mathcal{C}_2^{(t_{12})}$ to R_1.

(3) Third, R_1 (the leader) selects two disjoint exploration regions ($\mathcal{Q}_1$ and $\mathcal{Q}_2$) and margins ($\mathcal{M}_1$ and $\mathcal{M}_2$) such that

$$\left(\mathcal{Q}_1 \cup \mathcal{M}_1\right) \cap \left(\mathcal{C}_1^{(t_{12})} \cup \mathcal{C}_2^{(t_{12})}\right) = \emptyset,$$

and

$$\left(\mathcal{Q}_2 \cup \mathcal{M}_2\right) \cap \left(\mathcal{C}_1^{(t_{12})} \cup \mathcal{C}_2^{(t_{12})}\right) = \emptyset.$$

A challenge here is that, since the search environment is unknown to R_1 and R_2, the exploration regions $\mathcal{Q}_i$ could be completely outside of the search environment W or (a part of) $\mathcal{Q}_i$ could be occupied by obstacles without the robots knowing that in advance.

(4) Fourth, R_1 will build their soft obstacles $\mathcal{B}_1^{(t_{12})}$ and $\mathcal{B}_2^{(t_{12})}$ by

$$\mathcal{B}_1^{(t_{12})} \leftarrow \mathcal{B}_2^{(t_{12})} \leftarrow \mathcal{C}_1^{(t_{12})} \cup \mathcal{C}_2^{(t_{12})} \cup \mathcal{Z}_1 \cup \mathcal{Z}_2,$$

where

$$\mathcal{Z}_1 \leftarrow \mathcal{Z}_2 \leftarrow \mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{Q}_1 \cup \mathcal{Q}_2.$$

(5) Fifth, R_1 will send $\mathcal{B}_2^{(t_{12})}$ and $\mathcal{Q}_2$ to R_2.

(6) Sixth, R_1 and R_2 will update their exploration regions $\mathcal{X}_1$ and $\mathcal{X}_2$ by

$$\mathcal{X}_1 \leftarrow \mathcal{Q}_2 \text{ and } \mathcal{X}_2 \leftarrow \mathcal{Q}_2.$$

Stellenbosch University https://scholar.sun.ac.za

(7) Finally, R_1 and R_2 will move to their respective exploration regions, $\mathcal{X}_1$ and $\mathcal{X}_2$, for further solitary search to update their individual sets $\mathcal{C}_1^{(t)}$ and $\mathcal{C}_2^{(t)}$ (assuming their previous exploration regions are explored completely, otherwise they will need to complete them first before they move to their new exploration regions). If a robot R_i finds it difficult to explore its exploration region $\mathcal{X}_i$ for any reason (e.g. obstacle occupancy) at time t°, it can choose another exploration region $\mathcal{Q}_i$ such that

$$\mathcal{Q}_i \cap \left(\mathcal{X}_i \cup \mathcal{C}_i^{(t^{\circ})} \cup \mathcal{B}_i^{(t^{\circ})} \right) = \varnothing \,.$$

Let us now suppose that, at some later time $t_{13} > t_{12}$, R_1 meets with R_3. The same procedure described above which was applied between R_1 and R_2 also applies for the interaction between R_1 and R_3. But here there is something additional to highlight. Apart from the sharing of $\mathcal{C}_1^{(t_{13})}$ and $\mathcal{C}_3^{(t_{13})}$ between R_1 and R_3, the robot R_1 will also share its set $\mathcal{B}_1^{(t_{12})}$ with the robot R_3. After their interaction, the set of soft obstacles $\mathcal{B}_3^{(t_{13})}$ of R_3 becomes

$$\mathcal{B}_3^{(t_{13})} \leftarrow \mathcal{C}_1^{(t_{13})} \cup \mathcal{C}_3^{(t_{13})} \cup \mathcal{B}_1^{(t_{12})} \,.$$

The coordination protocol described above is pursued by the three robots until the search time is elapsed.

4.4 Conclusion

This chapter proposed a new accidental rendezvous strategy for the coordination of solitary robots. The existing accidental rendezvous strategy [109] suffers from interference due to unbalanced assignments. The proposed algorithm refines the accidental rendezvous strategy, as each robot, after an accidental meeting, can better predict actions of other robots involved.

The periodic rendezvous strategy is another strategy proposed to mitigate the interference the accidental rendezvous strategy suffers from. The periodic rendezvous strategy has the advantage that members of a local interaction share their additional information on a regular basis, by scheduling meetings. However, these scheduled meetings cause interruptibility: when their meeting time approaches, the rendezvous is prioritized over exploration and thus robots stop searching to ensure they meet on time.

Stellenbosch University https://scholar.sun.ac.za

The soft obstacle strategy mitigates the typical interference in the accidental rendezvous strategy without incurring high interruptibility as in the periodic rendezvous strategy. With the soft obstacle strategy, to mitigate interference, robots involved in an interaction apply cellular decomposition and each robot considers regions assigned to others as soft obstacles. However, the soft obstacle strategy requires a search time bound and large enough search environment. We also introduced margins which robots use to move to new exploration regions when it can not entirely explore its current exploration region.

This chapter described our proposed method for coordination of solitary robots. The idea was to show how robots coordinate when they interact with each other. The next step should be an experimental assessment of our method. However, we cannot run experiments without providing further details about the functioning and interaction of solitary robots. In the following chapter, we discuss these details.

Stellenbosch University https://scholar.sun.ac.za

Chapter 5

Distributed system of solitary robots

5.1 Introduction

We described the core of our coordination strategy in Chapter 4, where we considered allocations and management of exploration regions. Now, we aim to consider other details about the behaviour and interaction of robots to make a single distributed system, namely: search strategy, obstacle avoidance, data fusion, task assignment and leader election. These aspects are combined in a single system for the assessment of our proposed coordination method in Chapter 6. It should be noted that this chapter does not aim to be state of the art, but it provides a context in which we can evaluate the contributions of Chapter 4.

5.2 Distributed methods for coordination of solitary robots

This section summarises the various distributed strategies considered in this chapter.

1. **Distributed view construction**: the first step in the interaction of robots consists of generating a view of the ad-hoc network of their interaction. The question is, how do robots distributively build such a view under restricted communication? Each of the robots initially has a lim-

Stellenbosch University https://scholar.sun.ac.za

ited view of the network, but by iteratively passing messages between neighbours, they can end up with a broader view.

Every robot independently starts the process of constructing its view of the communication graph, as follows. When some robots interact with each other, they stop and wait for a predefined period of time, T_w seconds, before they start sending messages to each other, to wait for other robots that may be nearby to join the interaction. Otherwise, a limited number of robots involved in an interaction could quickly finish and miss other robots in their neighbourhood moving towards them. This is valuable because larger interactions might yield more effective explorations. The parameter T_w is common to all robots. When one of the robot in the interaction detects another robot in its neighbourhood, it communicates the time t_w already spent in waiting to the new robot ($t_w = 0$ for new robots in an interaction). Communication is assumed to be asynchronous. When a robot sends the time t_w, it also sends its state (Figure 5.1 presents different robot states). The signal sent by the robot is: $\langle \texttt{WaitingSignal(}t_w\texttt{, robotState)} \rangle$. This distributed procedure run on robot R_i is described in Algorithm 5.1.

After T_w seconds has elapsed, robots in the interaction start the process of coordination. For the construction of the communication graph formed in interaction of robots, we apply the pruning method for construction of a view of a communication graph that will be proposed in Chapter 8 (Algorithm 8.1).

2. **Leader election**: the second step in the interaction of robots is the choice of a leader that guides the actions of the interacting robots. A leader combines individual information, applies the coordination strategy and relays decisions to others. Only situations where the choice of a leader is achieved by passing messages between robots are treated. The method applied to choose a leader is presented in Algorithm 5.2 and discussed in Section 5.3.

   Recall that, using Algorithm 8.2, view construction ends at different times for different robots. A way to handle this situation is that, a robot starts the process of choosing a leader only when its view construction is completed. While still busy constructing its view, a robot will queue leader election messages received to treat them later.

Stellenbosch University https://scholar.sun.ac.za

3. **Data fusion**: the elected leader facilitates fusion of individual information of nodes in the network. Here, each robot sends its information (which include maps, exploration regions and interaction histories) to the leader via intermediate robots. A robot receiving information fuses it with its own before the robot forwards the combined information towards the leader as in packet-switched store-and-forward networks [62]. The method DATAFUSION applied to fuse data is presented in Algorithm 5.3 and discussed in Section 5.4.

4. **Coordination strategy**: this is where robots' knowledge is updated and effectively used for further search. This point was the main subject of Chapter 4.

5. **Task assignment**: finally, after applying the coordination strategy, the leader shares the coordination decisions with others (the method TASKASSIGNMENT in Algorithm 5.3 discussed in Section 5.4).

Algorithm 5.1 *Waiting for other interactions in its neighbourhood by a robot. The procedure defined in this algorithmic environment is run by R_i, and uses time since waiting t_w and identifiers $\mathcal{I}_i$ of interacting robots which R_i knows about as global variables.*

1: **procedure** WAITING()
2:     **if** R_i meets a searching or waiting robot R_j **then**
3:         $\mathcal{I}_i \leftarrow \mathcal{I}_i \cup \{j\}$
4:         R_i sends $\langle$`WaitingSignal(`t_w`, robotState)`$\rangle$ to R_j
5:         **when** R_i receives $\langle$`WaitingSignal(`t'_w`, robotState)`$\rangle$ from R_j **do**
6:             $t_w \leftarrow \max(t_w, t'_w)$
7:         **end when**
8:     **end if**
9: **end procedure**

These five tasks facilitate sharing of knowledge and coordination between robots and are distributed strategies.

During the search, each participating robots can be in one of the following states: *searching*, *waiting*, *stopped*, *interacting*, *planning*, or *completed*. These states control the behaviour of robots while exploring an environment. A state transition diagram is given in Figure 5.1. A robot starts in the searching state but can detect another robot once it enters its perception range. When a robot detects another robot in its vicinity, it stops and initiates view construction after waiting for at least T_w seconds.

Stellenbosch University https://scholar.sun.ac.za

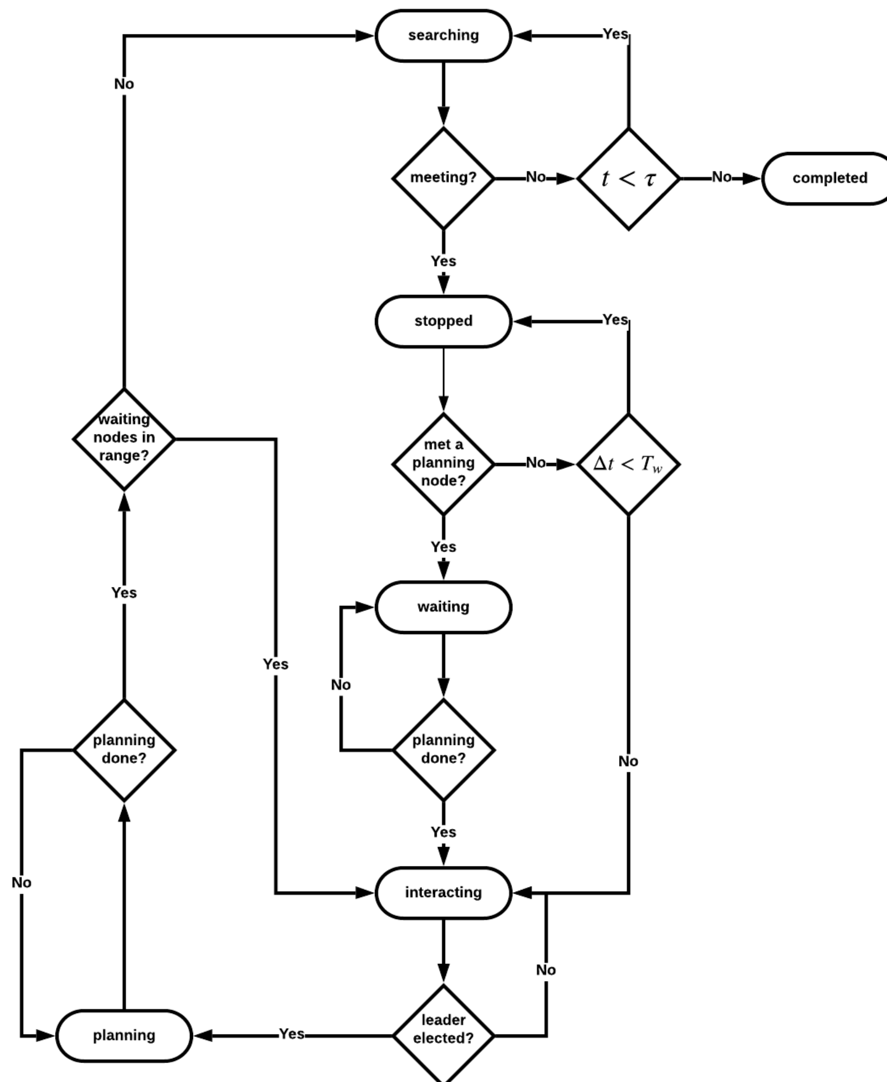


Figure 5.1: *State transition diagram of a single robot during search. The variable t denotes how long a robot has been searching, τ the search time limit, Δt the time elapsed since a robot first encountered other robots for the current meeting and* T_w *the waiting time after which the robot starts interacting.*

A robot initiates an interaction by sending its neighbouring information to all its immediate neighbours. Once a robot detects other robots in its vicinity and the waiting time T_w has elapsed, the robot starts interaction, which means the robot starts exchanging its neighbour information with other robots in its neighbourhood in order to learn a view of topology of the network (see Chapter 8). Once robots have exchanged messages a fixed number D of times, they start identifying a leader. An interacting robot

Stellenbosch University https://scholar.sun.ac.za

becomes a planning robot after the choice of a leader. When a new robot encounters a planning robot, it waits until the planning procedure is terminated. A planning robot is aware of new robots (i.e. robots which arrive in its perception range after it has become a planning robot) within its perception range. After planning is complete, a planning robot goes on to the searching state if no other robots are waiting for it. Otherwise, the planning robot interacts with the waiting robot before going onto the searching state. A robot moves when it is in searching mode and is stationary otherwise. A robot completes searching when the search time has elapsed.

A list of robot states (for implementation purpose) is given in Algorithm F.1.

So far we have established a state transition diagram to show how robots change states during search. It was mentioned in Chapter 4 that solitary robots are assigned bounded exploration regions for effective search by multiple robots. A search mechanism used for exploring these exploration regions is described in Appendix A.

In addition to the problem of search, an obstacle avoidance system must be incorporated. The techniques used to avoid obstacles in our system are discussed in Appendix B.

Another important aspect of coordination is the election of a leader among interacting solitary robots that facilitates their coordination. We discuss leader election next.

5.3 Leader election based on closeness centrality

In Chapter 8, we introduced a distributed method for communication graph view construction. After the construction of these views of the communication graph, each node is responsible for computing its own centrality.

5.3.1 Distributed determination of closeness centrality

We now show how a node determines its closeness centrality after building its view of the communication graph.

To compute v_i's closeness centrality, the node must know whether it is an element of the set $\mathcal{F}_{i,t}$, which denotes the set of pruned nodes known by v_i after iteration t using Algorithm 8.1. At the end of Algorithm 8.1, if a

Stellenbosch University https://scholar.sun.ac.za

node satisfies $v_i \in \mathcal{F}_{i,t}$, then its closeness centrality is set to 0. Otherwise, the node evaluates its centrality given in Equation 2.4.1. The procedure is PRUNINGCLOSENESSCENTRALITY given in Algorithm 5.2.

5.3.2 Construction of a spanning tree for communication

After selecting a leader, nodes typically build a spanning tree for effective communication by combining their individual information. We will consider the selected leader as the root of the spanning tree. Fusion on a spanning tree has been studied in the literature—for instance, fusion using a spanning tree is used to analyse networks in decentralised systems [24, 38, 106]. For effective communication in a distributed system as considered in this thesis, a robot needs only to know its spanning tree parent and children, because for the leader to receive information of other nodes in the data fusion process, spanning tree children send their information to the leader via their spanning tree parent. Before a node forwards its own data to the leader via its spanning tree parent, it must receive data from its tree children. When a node receives data from its tree children, it fuses them with its own data before forwarding the fused data to the leader via its tree parent. This corresponds to a packet-switched store-and-forward network [62] as was discussed in Chapter 3, and is known as fusion by propagation [24].

5.3.3 Leader election

The authors of the original algorithm for graph view construction [118] which we improved in terms of number of messages in Chapter 8 did not give further detail on how individual centralities should be exchanged between the nodes so that every node knows which is the most central node. If every node has built the same view of the communication graph, which occurs when every node learns the actual communication graph, each node can evaluate the centralities of other nodes, so nodes do not need to share their centrality information. But exchange of individual centralities between the nodes is very important in situations where nodes can fail to communicate with each other or nodes have only built limited views of the communication graph, i.e. nodes have applied an approximate method where each node builds its own view of the communication graph. In what follows, we

Stellenbosch University https://scholar.sun.ac.za

leverage the idea presented by You et al. [118] where nodes exchange their identifiers to determine the node with the lowest identifier in a distributed manner. For our case, nodes exchange their centralities between themselves to determine the most central node. Let c_i denote the closeness centrality of the node v_i with respect to its view $\mathcal{S}_{T,i}$ of the communication graph. If there exists a node v_l such that $c_l > c_j$ for all v_j then v_l should be the leader. In the case of a tie, the node with the smallest identifier among the nodes with maximal centrality is selected as the leader.

As mentioned before, one way that nodes can apply to identify the most central node after each node has evaluated its own centrality is to adapt the approach proposed by Naz [77]. He proposed an interesting distributed algorithm (called CHEUNG-BFS-ST-CB), which improves some previous distributed methods, for leader election and spanning tree construction on an unknown connected graph: the node with the lowest identifier is selected as the leader and a spanning tree is constructed by considering the leader as the root. Spanning trees are important in this thesis (as will be discussed in Section 5.4): interacting robots exchange their information using a spanning tree topology where the leader is the root. We consider our communication graph to be unknown to nodes because nodes have built different views of the communication graph (see Chapter 8). In what follows, we first describe the algorithm where a node with lower identifier is identified as a leader, after which we will present our approach where the leader is elected based on closeness centrality.

The CHEUNG-BFS-ST-CB algorithm works as follows. To elect a leader and construct a spanning tree, two important pieces of information are shared among nodes, namely: nodes' identifiers and their distances to their preliminary leaders (i.e. the shortest path length from a node to its preliminary leader). The information about distance allows a node to identify its parent and its children among its immediate neighbours, so that a spanning tree can be constructed. A node updates the value of its distance to a preliminary leader whenever it receives a lower identifier from its neighbours than it has previously encountered. This means that, during the process of leader election, each node can update information about its children and its parent.

Initially, each node considers itself as a leader, sets its distance to zero, and shares its identifier and the value of its distance to its immediate neighbours. When a node receives an identifier lower than it has previously en-

Stellenbosch University https://scholar.sun.ac.za

countered, it updates its information and sends a message to its immediate neighbours.

There are two types of messages exchanged between nodes in the algorithm of [77]: `BFSGOSignal(id, distance)` and `BFSBACKSignal(id, distance, isChild)`. A node uses the first signal message to inform its neighbours about a new potential leader's `id` it has received and its updated distance `distance`. It uses the second type of signal message for two reasons. First, after receiving leader information and possibly updating information about its tree parent, it uses this message to inform its previous tree parent whether it is still its child (i.e. `isChild=False`) or not (i.e. `isChild=True`). Also, after it has discovered its new tree parent, it uses this second message to inform its new tree parent about their tree relationship.

The distributed method proposed by Naz [77] chooses the leader based on nodes' identifiers only. Here, we aim to choose a leader based on closeness centrality. Essentially what changes from the method proposed by Naz [77] is that we replace the comparator between nodes for determination of a leader, and change the parameters so the required values are available for calculating the comparator.

The types of signals sent in the algorithm proposed by Naz [77] are then changed as follows. The first signal is a leader message, $\langle$`BFSGOSignal(`$v_i, v_l,$ c_l`, distance)`$\rangle$, from which a node v_i informs its neighbours about the closeness centrality c_l of a potential leader v_l and its distance to the leader. The second signal is an acknowledgement message, $\langle$`BFSBACKSignal(`$v_i, v_l,$ c_l`, distance, isChild)`$\rangle$, from which a node v_i notifies its neighbour v_j whether v_i is its child or not. A receiving node v_j stores the two types of messages received into two queues $\mathcal{G}_j$ and $\mathcal{B}_j$ respectively. Initially, v_i considers itself as a leader and sends a message `BFSGOSignal(`v_i`, `c_i`, 0)`$\rangle$ containing its identifier, centrality value and its distance of zero to its immediate neighbours. The resulting decentralised algorithm to choose a leader is given in Algorithm 5.2. At the end of this algorithm, each node v_i knows the identifier v_l of the leader, its spanning tree parent `treeParent`, and its spanning tree children `treeChildren`. Algorithm 5.2 makes use of procedures TREECHILDRENUPDATE and TREEPARENTUPDATE. TREECHILDRENUPDATE is used by a node to update its tree children and to confirm their tree relationship with its tree parent. TREEPARENTUPDATE is used by a node to notify its previous tree parent when the state of their tree relationship changes

Stellenbosch University https://scholar.sun.ac.za

and its new tree parent about their new relationship.

The Python code corresponding to Algorithm 5.2 is presented in Appendix G.2.6.

We want to see the impact of reducing number of messages during graph view construction for leader election. The method for graph view construction [118] which we improved requires $\mathcal{O}(Nd_{\mathtt{max}}D)$ messages for a single node where N denotes the number of nodes in the communication graph, $d_{\mathtt{max}}$ the maximum node degree and D the maximum number of iterations. The leader election algorithm in Algorithm 5.2 requires $\mathcal{O}(NM)$ messages for a single node [77] where M denotes the number of edges in the communication graph. For situations where $\mathcal{O}(NM)$ does not dominate $\mathcal{O}(Nd_{\mathtt{max}}D)$ (which result depends on D), it is noteworthy to reduce $\mathcal{O}(Nd_{\mathtt{max}}D)$.

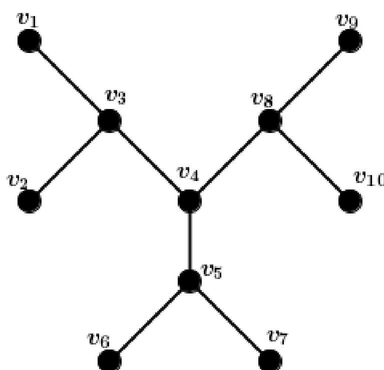


Figure 5.2: *Illustration of leader election using Algorithm 5.2 with D* $= 2$.

Nodes	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
c_l	0	0	0.47	0.6	0.47	0	0	0.47	0	0
`treeParent`	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
`Leader`	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
`treeChildren`	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
`distance`	0	0	0	0	0	0	0	0	0	0

Table 5.1: *Nodes' initialisation of leader election using Algorithm 5.2 for the communication graph in Figure 5.2.* $\perp$ *is used to indicate that a node does not have a tree parent.*

In what follows, we illustrate the functioning of Algorithm 5.2.

Stellenbosch University https://scholar.sun.ac.za

Algorithm 5.2 *Leader election based on the work by Naz [77]. The algorithm is run by a node v_i to choose a leader. The main methods are* TREEPARENTUPDATE *and* TREECHILDRENUPDATE *which are run by v_i once each step. An example of code to execute the class is given in the procedure* RUNLEADERELECTION.

```
 1: procedure RUNLEADERELECTION()
 2:   Pruningobject←PRUNING(N_i, D)                      ▷ using Algorithm 8.1
 3:   Pruningobject.INITIALONEHOP()
 4:   Pruningobject.INITIALUDATE()
 5:   Pruningobject.FIRSTPRUNINGDETECTION()
 6:   while not Pruningobject.ISENDED() do
 7:     Pruningobject.NEXTONEHOP()
 8:     Pruningobject.NEXTUPDATE()
 9:   end while
10:
11:   Leaderobject←LeaderElection(Pruningobject)
12:   Leaderobject.FIRSTHOP()
13:   while not Leaderobject.ISENDED() do
14:     Leaderobject.TREEPARENTUPDATE()
15:     Leaderobject.TREECHILDRENUPDATE()
16:   end while
17: end procedure
18:
19: class LEADERELECTION
20:   Class variables
21:   B_i               a queue of acknowledgement messages received
22:   c_i               closeness centrality of v_i
23:   distance          number of edges between node v_i and a provisional leader
24:   finished          a variable used to indicate the end of the algorithm
25:   G_i               a queue of leader acknowledgement messages received
26:   leaderCloseness   a variable to keep the closeness centrality of a (provisional) leader
27:   leaderID          a variable to save the identifier of a (provisional) leader
28:   treeParent        a (provisional) tree parent
29:   treeChildren      a set of (provisional) tree children
30:   wait              a variable to store queue of neighbours to receive feedback from
31: ------------------------------------------------------------------------------------
32:   constructor (pruningObject)                           ▷ Algorithm 8.1
33:     S_{i,T} ← pruningObject.S_{i,T}
34:     F_{i,T} ← pruningObject.F_{i,T}
35:     N_i ← pruningObject.N_i
36:     leaderCloseness ← PRUNINGCLOSENESSCENTRALITY(S_{i,T}, F_{i,T})
37:     c_i ← leaderCloseness
38:     leaderID ← i
39:     distance ← 0
40:     treeParent ←⊥
41:     treeChildren ← ∅
42:     wait ← ∅
43:     finished← False
44:   end constructor
45:
46:   procedure TREEPARENTUPDATE()
47:     while G_i.size() ≥ 1 do
48:       (v_j, v_l, c_l, dist) ← G_i.dequeue()
49:       if ISBIGGER(l, v_l, c_l) then
50:         leaderID ← l
51:         leaderCloseness ← c_l
52:         distance ← +∞
53:         treeParent ←⊥
54:       end if
55:       if l = leaderID and dist < distance − 1 then
56:         if treeParent ≠⊥ then
57:           send ⟨BFSBACKSignal(v_i, v_l, c_l, distance-1, False)⟩ to treeParent
58:         end if
59:         treeParent ← v_j
60:         treeChildren ← ∅
61:         distance ← dist + 1
62:         wait ← ∅
63:         for v_k ∈ N_i \ {treeParent} do
64:           send ⟨BFSGOSignal(v_l, c_l, distance)⟩ to v_k
```

Stellenbosch University https://scholar.sun.ac.za


```
65:                 wait ← wait ∪ {v_k}
66:             end for
67:             if wait = ∅ then
68:                 send ⟨BFSBACKSignal(v_i, v_l, c_l, distance-1, False)⟩ to treeParent
69:                 finished ← true
70:             end if
71:         else if l = leaderID then
72:             send ⟨BFSBACKSignal(v_i, v_l, c_l, dist, False)⟩ to v_j
73:         end if
74:     end while
75: end procedure
76:
77: procedure TREECHILDRENUPDATE()
78:     while B_i.size() ≥ 1 do
79:         (v_j, v_l, c_l, dist, isChild) ← B_i.dequeue()
80:         if j = leaderID and distance = dist and not finished then
81:             wait ← wait \ {v_j}
82:             if isChild then
83:                 treeChildren ← treeChildren ∪ {v_j}
84:             else
85:                 treeChildren ← treeChildren \ {v_j}
86:             end if
87:             if wait = ∅ then
88:                 if treeParent = ⊥ then
89:                     finished ← True
90:                 else
91:                     l ← leaderID
92:                     send ⟨BFSBACKSignal(v_i, v_l, c_l, distance-1, True)⟩ to its tree parent
93:                 end if
94:             end if
95:         end if
96:     end while
97: end procedure
98:
99: procedure FIRSTHOP()
100:     for v_j ∈ N_i do
101:         send ⟨BFSGOSignal(v_i, v_i, c_i, 0)⟩ to v_j
102:         wait ← wait ∪ {v_j}
103:     end for
104:     finished ← wait = ∅
105: end procedure
106:
107: function ISBIGGER(l, v_l, c_l)
108:     k ← leaderID
109:     c_k ← leaderCloseness
110:     return (c_k < c_l) or (c_k = c_l and l < k)
111: end function
112:
113: function ISENDED()
114:     return finished
115: end function
116:
117: function PRUNINGCLOSENESSCENTRALITY(F_{i,T}, S_{i,T})
118:     if v_i ∈ F_{i,T} then
119:         return 0
120:     else
121:         N ← S_{i,T}
122:         t ← 1
123:         c_i ← 0
124:         repeat
125:             c_i ← c_i + |S_i^(t)|
126:             t ← t + 1
127:         until t = T
128:         return (N−1)/c_i
129:     end if
130: end function
131: end class
```

Stellenbosch University https://scholar.sun.ac.za

Nodes	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
c_l	0.47	0.47	0.6	0.6	0.6	0.47	0.47	0.6	0.47	0.47
`treeParent`	v_3	v_3	v_4	$\perp$	v_4	v_5	v_5	v_4	v_8	v_8
`Leader`	v_3	v_3	v_4	$\perp$	v_4	v_5	v_5	v_4	v_8	v_8
`treeChildren`	$\emptyset$	$\emptyset$	v_1, v_2	v_3, v_5, v_8	v_6, v_7	$\emptyset$	$\emptyset$	v_9, v_{10}	$\emptyset$	$\emptyset$
`distance`	1	1	1	0	1	1	1	1	1	1

Table 5.2: *First iteration of leader election using Algorithm 5.2 for the communication graph in Figure 5.2.*

Nodes	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
c_l	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
`treeParent`	v_3	v_3	v_4	$\perp$	v_4	v_5	v_5	v_4	v_8	v_8
`Leader`	v_4	v_4	v_4	$\perp$	v_4	v_4	v_4	v_4	v_4	v_4
`treeChildren`	$\emptyset$	$\emptyset$	v_1, v_2	v_3, v_5, v_8	v_6, v_7	$\emptyset$	$\emptyset$	v_9, v_{10}	$\emptyset$	$\emptyset$
`distance`	2	2	1	0	1	2	2	1	2	2

Table 5.3: *Second iteration of leader election using Algorithm 5.2 for the communication graph in Figure 5.2.*

Example. As mentioned before, each node initially considers itself as the leader, sets its distance to the leader to zero, and sends its information including its closeness centrality to all its immediate neighbours. This stage is illustrated in Table 5.1, which shows that none of the nodes have a tree parent or any tree children yet.

After each node has received leader information from its immediate neighbours, they update their information, i.e. nodes with smaller closeness centralities update their information about the leader and information about the spanning tree. The results of subsequent iterations are shown in Tables 5.2 and 5.3.

Chapter 2 noted that a leader election method should satisfy three conditions [20]: uniqueness of the elected leader, termination of the leader election process and agreement on the choice of leader, assuming communication is eventually reliable. Algorithm 5.2 satisfies these three properties, assuming fully reliable failure-free communication. Termination of the algorithm is guaranteed because the communication graph is finite, communication is failure-free and nodes are uniquely identifiable. Uniqueness and agreement in the choice of a leader are also guaranteed because nodes are uniquely identifiable and choose a leader based on closeness centrality and node identifier which is unique for each node.

After choosing a leader, with each node knowing its spanning tree parent

Stellenbosch University https://scholar.sun.ac.za

and children, the next step consists of fusing the data from individual robots and the application of the coordination strategy.

5.4 Data fusion and task assignment

Section 5.2 identified various problems involved in interaction of robots, and noted that one effective way to handle interaction of robots is to choose a leader to coordinate the process. In this context, a leader has three roles: it receives data from others, it coordinates plans for their next actions based on the available data, and it relays data and decisions (missions) to others. Since this thesis considers search by solitary robots, the robot data will contain map information. Robots must combine information from individual maps to coordinate more effectively. Robots use the identified spanning tree from leader election (Section 5.3) to propagate information from the leader to other robots and vice-versa. Thus, robots know their spanning tree parent and children during data fusion and task assignment. We next discuss data fusion.

5.4.1 Data fusion

This step of interaction of solitary robots involves sending individual robot data to the leader in a distributed manner.

In this context, data fusion consists of merging the data of individual robots [24]. Fusion is gradually established in a distributed manner [24]. A map can be made of 2D point clouds [61]. Since we use a grid-based map representation as mentioned in Subsection 2.3.2, we discretise point clouds of an environment in a grid representation. The method proposed by Konolige et al. [54] is suitable for such maps.

When robots interact in our setting, they send six types of information to the leader. They send the area covered by each, their interference region (i.e. each of the robots informs the leader about all prior interactions it is aware of so that they all have the same view of the situation), their identifiers, the history of identifiers of robots from previous meetings, their current locations and their tree descendants. Specifically, a robot R_i sends the following data to the leader via its tree parent R_j:

$$(\mathcal{C}_i^{(t)}, \mathcal{Z}_i, \mathcal{I}_i, \mathcal{J}_i, \mathcal{L}_i, \mathcal{D}_i)\,.$$

Stellenbosch University https://scholar.sun.ac.za

Recall from Subsection 4.3.1 that

- $\mathcal{C}_i^{(t)}$: denotes the explored region known by R_i,
- $\mathcal{Z}_i$: denotes the interference region known by R_i,
- $\mathcal{I}_i$: denotes the identifiers of currently interacting robots which R_i is aware of,
- $\mathcal{J}_i$: denotes the identifiers of robots which R_i is aware of,
- $\mathcal{L}_i$: denotes the locations of robots which R_i is aware of and $\mathcal{D}_i$ denotes the set of tree descendants of robot R_i.

Robots share their current locations so that the leader can use them to assign robots to their closest exploration regions after coordination. The data fusion algorithm applied by a single robot R_i is illustrated in the procedure called DATAFUSION in Algorithm 5.3.

When the leader has received individual information from other robots, it begins the coordination strategy as described in Section 4.3. At the end of coordination, robots are informed of their assigned tasks (i.e. exploration regions).

5.4.2 Task assignment

After combining individual data from its neighbours and making plans, the leader relays its decisions to others using the topology of the spanning tree. Unlike data fusion where information was relayed from tree children to tree parents, information for task assignment is relayed from tree parents to tree children. The distributed algorithm that a robot R_i applies to spread the coordination decisions to its spanning tree children is given in the procedure TASKASSIGNMENT in Algorithm 5.3 (where Γ_{ij} denotes the set of exploration regions of tree descendants $\mathcal{D}_{ij}$ of R_i via its tree child R_j).

From what precedes, there are four states when robots are involved in interaction: network view construction, leader election, data fusion and task assignment. For implementation purpose, a list of states of robots involved in an interaction is given in Algorithm F.2.

In the following, the individual strategies we have presented in Chapters 4 and 5 are combined to build a multi-robot search and rescue algorithm.

Stellenbosch University https://scholar.sun.ac.za

5.5 General algorithm

We need to present the overall algorithm (Algorithm 5.3) for coordination of solitary robots. Algorithm 5.3 is applied by each robot R_i in a distributed manner. The algorithm receives as arguments the search bound time τ, the perception and communication range d of robots, and enlargement width parameter w_0 (see Section 4.3). Before presenting the general algorithm, we discuss a few important details.

5.5.1 Choice of exploration region size

The maximum possible area that a robot with circular scanning region[1] of radius d and maximum velocity γ can scan in t time units is (Figure 5.3)

$$\hat{A}(t) = \pi d^2 + 2\gamma t d\,, \tag{5.5.1}$$

which corresponds to the area of the scanned region around a robot moving

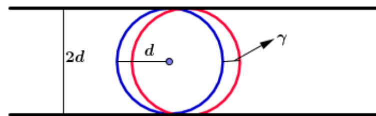


Figure 5.3: *Illustration of the area scanned per motion step by a robot.*

along a straight line for t time units. Thus, the initial size of the exploration region $\mathcal{X}_i^{(0)}$ of a robot R_i is obtained by requiring

$$|\mathcal{X}_i^{(0)}| = \hat{A}(\tau)\,.$$

Likewise, after interacting with other robots at time t, the size of a robot's exploration region is set to

$$|\mathcal{X}_i| = \hat{A}(\tau - t)\,.$$

5.5.2 Travel to exploration region

When a robot is assigned an exploration region, the robot first must travel to its exploration region. To travel to its exploration region, it prefers to move

[1]Alternatively shaped scanning regions should have only a minor impact on this discussion.

Stellenbosch University https://scholar.sun.ac.za

in a straight line, but must avoid obstacles and soft obstacles if encountered. Robots use the Distance Bug algorithm [80] to avoid obstacles and soft obstacles as discussed in Appendix B. It should be noted that the method for obstacle avoidance discussed in Appendix B is a modified version of the Distance Bug algorithm—we modify it to encourage but not enforce avoidance of soft obstacles while a robot is travelling to its new exploration region. The method for the robot's motion to its exploration region is given in Algorithm B.1.

It follows that a robot can search either within its exploration region or while moving to its exploration region. For implementation purpose, this can give rise to two search states: being inside and outside of its exploration region. A list of states of robots involved in an interaction is given in Algorithm F.3.

5.5.3 Selection of a new exploration region by a single robot

There are two main reasons that a robot may elect to select a new exploration region. The first reason is that a robot discovers that some part of its exploration region is inaccessible. The second reason is when the placement of obstacles in its exploration region does not allow the robot to search effectively. The algorithm that a robot applies to select a new exploration region is given in the procedure GETINDIVIDUALEXPLORATIONREGION in Algorithm 5.3. It should be noted that robots do not allocate themselves margins when they choose individual exploration regions.

5.5.3.1 Inaccessible exploration region

An exploration region of a robot can contain inaccessible area. Each robot applies the zigzag search algorithm in the accessible area of their exploration region part as discussed in Appendix A. As robots have limited perception and communication range, a region assigned to a robot in an interaction may not fall entirely in the search space or some other unknown robots may be exploring it. After interaction, members of the interaction do not search in other interacting robot's assigned exploration regions (i.e. the robots apply the soft obstacle strategy, which was described in Section 4.3) even when their own exploration regions are not fully accessible. In

Stellenbosch University https://scholar.sun.ac.za

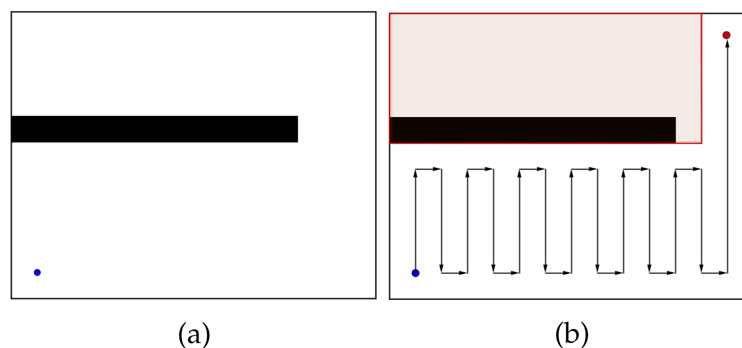
such cases, the robots involved must select new exploration regions independently based on their knowledge of the environment, but also taking into account the soft obstacles, thus avoiding searching in the planned exploration regions of other robots. This is also done using the procedure GETINDIVIDUALEXPLORATIONREGION in Algorithm 5.3.

5.5.3.2 Obstacles in exploration regions

The placement of obstacles in an exploration region can complicate zigzag search of a region as originally planned. A robot decides that it needs to change its exploration region when it does not cover a third of what it is supposed to cover after any sequence of three moves (this is an arbitrary parameter). When a robot encounters obstacles in its exploration region, it must calculate the size of unexplored area in its exploration region (unexplored area can contain obstacles). To estimate the size of unexplored area in its exploration region, the robot generates sampled points randomly and uniformly distributed over its exploration region. The size of unexplored area is approximately proportional to to the fraction of sampled points on the unexplored region. For example if we sample 1000 points from which 730 points lie in the unexplored region, then one says that approximately 73% of the exploration region is unexplored. The robot stays in its current exploration region only if the unexplored area remaining in its exploration region is big enough for further exploration and it can easily get in the unexplored region for exploration. Otherwise, it looks for another region outside of its current region for further search again using the procedure GETINDIVIDUALEXPLORATIONREGION in Algorithm 5.3. We propose a simple strategy for a robot to search the unexplored area in its region after encountering an obstacle. Consider a robot applying a motion pattern (such as the zigzag motion described in Appendix A) to search its assigned region. If it encounters an obstacle, it attempts to maintain the pattern to the extent possible while avoiding the obstacle. This means that when it completes the current motion pattern, some parts of the region might still be relatively unexplored. When it finishes applying its current motion pattern, it therefore must determine whether its current exploration region still contains enough unexplored areas to focus on, or whether it is better to select a fresh search region based on the remaining search time. For example in Figure 5.4, a robot is first given an exploration region (Figure 5.4a) in an oblivious

Stellenbosch University https://scholar.sun.ac.za

manner (i.e. not initially aware of obstacles). Its initial location is indicated by a blue circle and the robot intends to apply (N, E, S, E) as motion pattern. When it first detects an obstacle, it cuts its trip short (Figure 5.4b). When it arrives at the red circle in Figure 5.4b, it realises that it can not continue to use its actual motion pattern but there is still unexplored area in its actual exploration region (see shaded area within red rectangle in Figure 5.4b). If there is more than one option for a new exploration region in the current region, the robot will explore all of them, one after another. In that case, the robot will select options based distances between its current location and the closest exploration region's corners. An algorithm that a robot applies to determine the unexplored area in its exploration region is provided in the procedure FINDEXPLORATIONAREAREGION in Algorithm 5.3. A Python implementation of how a robot determines an unexplored region in its exploration region can be found in Appendix G.2.7.



(a) (b)

Figure 5.4: *Illustration of how a robot can get a new exploration region inside its assigned exploration region. In black is the obstacle.* ***(5.4a)****: Initial exploration region of the robot. The blue circle indicates the initial location of the robot.* ***(5.4b)****: New exploration region within an exploration region. The red circle indicates the initial location of the robot where it realises that it needs to choose a new exploration region inside its assigned exploration region (the new exploration region is shaded).*

Figures A.2b and A.2c further illustrate obstacle avoidance and further exploration in the region using this approach.

5.5.4 New interactions

The last important aspect we need to mention before we provide the overall algorithm for coordination of solitary robots is how a robot detects other

Stellenbosch University https://scholar.sun.ac.za

robots in its perception range. There are two conditions for a robot to consider another robot in its neighbourhood as a candidate for coordination. First, a robot considers another robot as a candidate for coordination if it is their first time encountering each other. Second, a robot considers another robot as candidate for coordination if they have covered some distance $T_i^{(t_m)}$ (as defined in Equation 4.3.6) where t_m denotes an interaction time. Let $\mathcal{I}_i$ denotes identifiers of robots which are currently under the vicinity of R_i. Recall that $\mathcal{J}_i$ denotes the set of identifiers of robots that the robot R_i is aware of; and ΔT_{ij} the amount of distance that has been covered since the last meeting of R_i and R_j up to time t. The algorithm that a robot uses to determine whether to begin a new interaction is given in the procedure NEWINTERACTION in Algorithm 5.3.

5.5.5 Overall algorithm

The overall distributed coordination algorithm run on the robot R_i is given in Algorithm 5.3 which uses earlier algorithms. A portion of the Python implementation of Algorithm 5.3 can be found in Appendix G.2.8. At the end of the search, robots stay at their respective final locations.

5.6 Conclusion

In this chapter we described the various components of a distributed system for effective assessment of our proposed coordination method in Chapter 4. We considered four relevant aspects. We first considered the aspect of search: a solitary robot requires a search strategy to explore an environment effectively. We also considered the aspects of leader election, data fusion and task assignment which take place during robot interaction. Finally, we presented the high-level algorithm for coordination of robots.

Having described our coordination method, and discussed other important aspects of coordination of solitary robots, we now present and discuss our experimental results on coordination of solitary robots.

Stellenbosch University https://scholar.sun.ac.za

Algorithm 5.3 *Our proposed soft obstacle strategy which is run by each robot R_i. The main method is* UPDATE *which is run by R_i once each step. An example of code to execute the class is given in the procedure* RUNSOS.

```
1:  procedure RUNSOS()
2:      SOSrobot←SOS(τ, d, γ, Δt, s, T_w, D, w_0, η, m, N, L)
3:      while not SOSrobot.ISENDED() do
4:          SOSrobot.UPDATE()
5:      end while
6:  end procedure
7:
8:  class SOS
9:      Class variables
10:     Bug               object for obstacle avoidance
11:     B_i^(t)           its soft obstacles
12:     C_i^(t)           its explored region
13:     γ                 motion velocity
14:     Γ_ij              hash table of exploration regions of tree descendants via its tree child R_j
15:     η                 threshold for it to determine whether it needs a new exploration region
16:     Δt                time step
17:     d                 robot perception range
18:     D                 maximum number of iterations for graph construction
19:     D_i               set of tree descendants of R_i
20:     D_ij              set of tree descendants of R_i via its tree child R_j
21:     ε                 a small threshold for stopping condition
22:     electionObject    object for leader election
23:     GO                a variable to control the steps of leader election
24:     G_i               set of previous meetings of R_i and their times
25:     I_i               identifiers of robots currently in interaction known by R_i
26:     J_i               identifiers of interacting robots it knows about
27:     L                 number of attempts required before virtual world enlargement
28:     L_i               a hash table with robot identifiers as keys and their locations as values
29:     m                 width of margins
30:     M_i               margins
31:     N                 sampling size
32:     N_i               immediate neighbours
33:     O                 set of obstacles
34:     planningState     a variable to indicate robot state during planning
35:     pruningObject     object for network view construction
36:     Q_i               its new exploration region
37:     R_i               queue messages received by R_i during task assignment
38:     s                 number of steps required to check quantities of explored area
39:     robotState        a variable to indicate robot state
40:     searchState       a variable to indicate a search state
41:     S_i               queue messages received by R_i during data fusion
42:     stepB             a variable to control the steps of network view construction
43:     τ                 maximum search time
44:     treeChildren      its tree children
45:     treeParent        its tree parent
46:     t_w               time elapsed since robot has been waiting
47:     T_w               maximum time robots can wait for other robots before they start planning
48:     V_i               its current virtual world
49:     w_0               width for virtual world enlargement
50:     x_i               its current location
51:     X_i               its current exploration region
52:     Y_i               sequence of available exploration regions
53:     Z_i               its interference region
54:     zigzag            object for zigzag motion
55:
56:     constructor (τ, d, γ, Δt, s, T_w, D, w_0, η, m, N, L)
57:         (τ, d, γ, Δt) ← (τ, d, γ, Δt)
58:         (s, T_w, D, w_0, t, C_i^(0), B_i^(0), J_i, G_i) ← (s, T_w, D, w_0, 0, B_d(x_i), B_d(x_i), ∅, ∅)
59:         (A_s, w, η, t_w) ← (2sγd/3, sqrt(Â(τ)), η, 0)          ▷ Â(τ) is defined in Equation 5.5.1
60:         INITIALEXPLORATIONREGION()
61:         (M_i, V_i, Z_i, Y_i) ← (∅, X_i, X_i, [])
62:         (A, m, N, L, A_B) ← (2d sqrt(Â(τ)), m, N, L, ∅)        ▷ m denotes a margin width
63:         (treeChildren, treeParent)←(∅, ⊥)
```

Stellenbosch University https://scholar.sun.ac.za


```
64:         Bug← BUG(d, γ)                                        ▷ using Algorithm B.1
65:         pruningObject ←⊥
66:         electionObject ←⊥
67:         zigzag ← ZIGZAG(d, γ)                                 ▷ using Algorithm A.1
68:         robotState ← ROBOTSTATES.SEARCHING                    ▷ using Algorithm F.1
69:         planningState ← PLANNINGSTATES.NONE                   ▷ using Algorithm F.2
70:         searchState ← SEARCHSTATES.INREGION                   ▷ using Algorithm F.3
71:     end constructor
72:
73:     procedure UPDATE()
74:         if not ISENDED() then
75:             t_0 ← currentTime()
76:             if robotState=ROBOTSTATES.SEARCHING or robotState=ROBOTSTATES.WAITING then
77:                 (I_i, W_i) ← NEWINTERACTION()
78:                 if |I_i| > 0 then
79:                     (𝓘_i, 𝓦_i) ← (I_i, W_i)
80:                     robotState ← ROBOTSTATES.STOP
81:                     t_w ← t
82:                 else if |W_i| > 0 then
83:                     robotState←ROBOTSTATES.WAITING
84:                 else
85:                     SEARCHING()
86:                 end if
87:             else if robotState=ROBOTSTATES.STOP then
88:                 if t − t_w ≤ T_w then
89:                     WAITING()                                 ▷ using Algorithm 5.1
90:                 else
91:                     robotState←ROBOTSTATES.PLANNING
92:                 end if
93:             else if robotState=ROBOTSTATES.PLANNING then
94:                 if planningState=PLANNINGSTATES.NONE then
95:                     pruningObject ← PRUNING(𝓘_i, D)
96:                     pruningObject.INITIALONEHOP()
97:                     planningState ← PLANNINGSTATES.CONSTRUCTION
98:                     stepB ← "update"
99:                     GO ← false
100:                else if planningState=PLANNINGSTATES.CONSTRUCTION then
101:                    TREECONSTRUCTION()
102:                else if planningState=PLANNINGSTATES.ELECTION then
103:                    LEADERELECTION()
104:                else if planningState=PLANNINGSTATES.FUSION then
105:                    DATAFUSION()
106:                else if planningState=PLANNINGSTATES.TASK then
107:                    TASKASSIGNMENT()
108:                end if
109:            end if
110:            t_1 ← currentTime()
111:            Δt_01 ← max(0, T_w − t_1 + t_0)        ▷ R_i shall wait for some time T_w before it can increment its counter.
112:            DELAY(Δt_01)
113:            t ← t + Δt
114:            𝓒_i^(t) ← 𝓒_i^(t−Δt) ∪ B_d(x_i)
115:            𝓑_i^(t) ← 𝓑_i^(t−Δt) ∪ B_d(x_i)
116:            ΔT_i ← ΔT_i + Δt
117:        end if
118:    end procedure
119:
120:    procedure REINITINTERACTIONINFO()
121:        (treeChildren, treeParent)←(∅, ⊥)
122:        (𝓘_i, 𝓦_i, 𝓛_i, Γ_i, 𝓓_i) ← (∅, ∅, ∅, ∅, ∅)
123:        t_w ← 0
124:        ΔT_i ← 0
125:        A_B ← ∅
126:        planningState ← PLANNINGSTATES.NONE
127:        robotState ← ROBOTSTATES.SEARCHING
128:        searchState ← SEARCHSTATES.OUTREGION
129:        if 𝓨_i.size() > 0 then
130:            X ← 𝓧_i.clone()
131:            X ←RECTANGLETOSET(X)                              ▷ using Algorithm F.4
132:            Y_i ← ∅
```

Stellenbosch University https://scholar.sun.ac.za

133: **for** $y_i \in \mathcal{Y}_i$ **do**
134: $Y_i \leftarrow Y_i \cup$ RECTANGLETOSET(y_i) ▷ using Algorithm F.4
135: **end for**
136: $X \leftarrow X \setminus \mathcal{B}_i^{(t)}$
137: $X \leftarrow X \setminus Y_i$
138: $A \leftarrow 2d\sqrt{\hat{A}(\tau - t)}$
139: **if** $|X| > 0$ **then**
140: $Q \leftarrow$ RECTANGLEEXTRACTION(N, X) ▷ using Algorithm F.4
141: **if** $Q.w \times Q.h > \frac{A}{2}$ **then**
142: $\mathcal{X}_i \leftarrow Q$
143: **else**
144: $\mathcal{X}_i \leftarrow \mathcal{Y}_i$.pop()
145: **end if**
146: **else**
147: $\mathcal{X}_i \leftarrow \mathcal{Y}_i$.pop()
148: **end if**
149: REINITZIGZAGPARAMS()
150: $y_0 \leftarrow x_i$
151: $\Delta y \leftarrow 0$
152: $A_B \leftarrow \emptyset$
153: **end if**
154: **end procedure**
155:
156: **procedure** INITIALEXPLORATIONREGION()
157: $X \leftarrow [$RECTANGLE$(x_{ix} - d, x_{iy} - d, w, w)]$ ▷ with its current location $x_i = (x_{ix}, x_{iy})$
158: X.add(RECTANGLE$(x_{ix} - d, x_{iy} - w + d, w, w)$)
159: X.add(RECTANGLE$(x_{ix} - w + d, x_{iy} - d, w, w)$)
160: X.add(RECTANGLE$(x_{ix} - w + d, x_{iy} - w + d, w, w)$)
161: $i \leftarrow$ uniform$(0, 4)$
162: $\mathcal{X}_i \leftarrow X[i]$.clone()
163: $\mathcal{V}_i = \mathcal{X}_i$.clone()
164: searchState ← SEARCHSTATES.INREGION
165: REINITZIGZAGPARAMS()
166: **end procedure**
167:
168: **procedure** REINITZIGZAGPARAMS()
169: $c_i \leftarrow \mathcal{X}_i$ .GETCLOSESTCORNER(x_i)
170: zigzag.PARAMREINIT($\mathcal{X}_i, x_i, c_i$) ▷ using Algorithm A.1
171: $a_i \leftarrow$ zigzag.GETSTARTINGPOINT() ▷ using Algorithm A.1
172: **end procedure**
173:
174: **function** ISENDED()
175: **if** $t >= \tau$ **then**
176: robotState ← ROBOTSTATES.COMPLETE
177: **return true**
178: **end if**
179: **return false**
180: **end function**
181:
182: **procedure** SEARCH()
183: **if** searchState = SEARCHSTATES.INREGION **then**
184: **if** $t < s$ **or** $|\mathcal{C}_i^{(t)} \setminus \mathcal{C}_i^{(t-s)}| \geq A_s$ **then** ▷ we use short-circuit evaluation to prevent accessing negative array indices
185: zigzag.ZIGZAGMOVE($\mathcal{B}_i^{(t)}, \mathcal{O}$)
186: $x_i \leftarrow$ zigzag.x_i
187: $\mathcal{B}_i^{(t)} \leftarrow \mathcal{B}_i^{(t)} \cup$ zigzag.B
188: $(y_0, \Delta y, A_B) \leftarrow (x_i, 0, \emptyset)$
189: **else**
190: FINEXPLORATIONAREAINREGION()
191: searchState ← SEARCHSTATES.OUTREGION
192: REINITZIGZAGPARAMS()
193: $(y_0, \Delta y, A_B) \leftarrow (x_i, 0, \emptyset)$
194: **end if**
195: **else if** searchState = SEARCHSTATES.OUTREGION **then**
196: $x_i \leftarrow$Bug.DISTANCEBUG($A_B, x_i, a_i, \mathcal{C}_i^{(t)}, \mathcal{O}$)
197: $\Delta y \leftarrow \Delta y + \gamma$
198: REINITZIGZAGPARAMS()
199: **if** $\delta(x_i, a_i) < \varepsilon$ **then**
200: searchState ← SEARCHSTATES.INREGION

Stellenbosch University https://scholar.sun.ac.za


```
201:            REINITZIGZAGPARAMS()
202:            (y_0, Δy, A_B) ← (x_i, 0, ∅)
203:        else if Δy > ηδ(a_i, y_0) then
204:            B_i^(t) ← B_i^(t) ∪ X_i
205:            if Y_i.size() > 0 then
206:                X_i ← Y_i.pop()
207:            else
208:                X_i ←GETINDIVIDUALEXPLORATIONREGION()
209:            end if
210:            searchState← SEARCHSTATES.OUTREGION
211:            REINITZIGZAGPARAMS()
212:            (y_0, Δy, A_B) ← (x_i, 0, ∅)
213:        end if
214:    end if
215: end procedure
216:
217: procedure TREECONSTRUCTION()
218:    if not pruningObject.ISENDED() then
219:        if stepB= "update" then
220:            pruningObject.INITIALUPDATE()
221:            pruningObject.FIRSTPRUNINGDETECTION()
222:            stepB← "nextHop"
223:        else if stepB= "nextHop" then
224:            pruningObject.NEXTONEHOP()
225:            pruningObject.FIRSTPRUNINGDETECTION()
226:            stepB← "nextUpdate"
227:        else
228:            pruningObject.NEXTUPDATE()
229:            stepB← "nextHop"
230:        end if
231:    else
232:        planningState ← PLANNINGSTATES.ELECTION
233:        leaderObject ← LEADERELECTION(PRUNINGOBJECT)
234:        leaderObject.FIRSTHOP()
235:        GO ← true
236:    end if
237: end procedure
238:
239: procedure LEADERELECTION()
240:    if GO then
241:        leaderObject.TREEPARENTUPDATE()
242:        GO← false
243:    else
244:        leaderObject.TREECHILDRENUPDATE()
245:        GO← true
246:        if leaderObject.ISENDED() then:
247:            treeChildren← leaderObject.treeChildren
248:            treeParent ← leaderObject.treeParent
249:            planningState←PLANNINGSTATES.FUSION
250:        end if
251:    end if
252: end procedure
253:
254: function GETCENTRALPOINT()
255:    P ← L_i.size()
256:    c ← (0, 0)
257:    for j ∈ L_i.keys() do
258:        c ← c + L_i[j]
259:    end for
260:    return c/P
261: end function
262:
263: function TIMEOFPREVIOUSMEETING(R_j)
264:    t_m ← −1
265:    if R_j ∈ G_i.keys() then
266:        t_m ← G_i[R_j]
267:    end if
268:    return t_m
269: end function
```

Stellenbosch University https://scholar.sun.ac.za

270: **function** NEWINTERACTION()
271: $(I_i, W_i) \leftarrow (\emptyset, \emptyset)$
272: **for** $(R_j,$ `jstate`$)$ such that $\delta(x_i, x_j) \leq d$ **do**
273: $t_m \leftarrow$TIMEOFPREVIOUSMEETING(R_j)
274: **if** $t_m = -1$ **or** $\Delta T_{ij} > T_i^{(t_m)}$ **then** ▷ we use short-circuit evaluation to prevent accessing negative array indices
275: $I_i \leftarrow I_i \cup \{j\}$
276: **else if** `jstate=RobotStates`.PLANNING **and** $j \notin \mathcal{I}_i$ **then**
277: $W_i \leftarrow W_i \cup \{j\}$
278: **end if**
279: **end for**
280: **return** I_i, W_i
281: **end function**
282:
283: **procedure** RELAYINFORMATION()
284: $F \leftarrow$ `treeChildren.clone()`
285: **while** $F \neq \emptyset$ **do**
286: R_i chooses an $R_j \in F$
287: $\Gamma_{ij} \leftarrow \emptyset$
288: **for** $R_k \in \mathcal{D}_{ij}$ **do**
289: $\Gamma_{ij}[R_k] \leftarrow \mathcal{Q}_k$
290: **end for**
291: R_i sends $\mathcal{B}_i^{(t)}$, $\mathcal{Z}_i$, $\mathcal{V}_i$, $\mathcal{J}_i$, $\mathcal{G}_i$, and Γ_{ij} to R_j
292: $F \leftarrow F \setminus \{R_j\}$
293: **end while**
294: **end procedure**
295:
296: **procedure** TASKASSIGNMENT()
297: **if** `treeParent` $=\perp$ **then**
298: GETGROUPEXPLORATIONREGIONS()
299: **for** each tree child R_j of R_i **do**
300: **for** $R_k \in \Gamma_{ij}$`.keys()` **do**
301: $\mathcal{Q}_k \leftarrow \Gamma_{ij}[R_k]$
302: $\mathcal{B}_i^{(t)} \leftarrow \mathcal{B}_i^{(t)} \cup \mathcal{Q}_k$
303: $\mathcal{Z}_i \leftarrow \mathcal{Z}_i \cup \mathcal{Q}_k$
304: $\mathcal{G}_i[R_k] \leftarrow t$ ▷ $\mathcal{G}_i$ is a hash table
305: **end for**
306: **end for**
307: RELAYINFORMATION()
308: **else**
309: **while** $\mathcal{R}_i$`.size()` ≥ 1 **do**
310: $(j, \mathcal{B}_j^{(t)}, \mathcal{Z}_j, \mathcal{V}_j, \mathcal{J}_j, \Gamma_{ji}, \mathcal{G}_j) \leftarrow \mathcal{R}_i$`.dequeue()`
311: $(\mathcal{B}_i^{(t)}, \mathcal{Z}_i, \mathcal{V}_i, \mathcal{J}_i) \leftarrow (\mathcal{B}_j^{(t)}$`.clone()`$, \mathcal{Z}_j$`.clone()`$, \mathcal{V}_j$`.clone()`$, \mathcal{J}_j$`.clone()`$)$
312: $\mathcal{G}_i \leftarrow \mathcal{G}_i \cup \mathcal{G}_j$
313: RELAYINFORMATION()
314: **end while**
315: **end if**
316: REINITINTERACTIONINFO()
317: **end procedure**
318:
319: **procedure** DATAFUSION()
320: $\mathcal{L}_i[R_i] \leftarrow x_i$ ▷ $\mathcal{L}_i$ is a hash table
321: $\mathcal{D}_i \leftarrow \emptyset$
322: **if** `treeChildren` $= \emptyset$ **then**
323: R_i sends $\mathcal{C}_i^{(t)}$, $\mathcal{Z}_i$, $\mathcal{I}_i$, $\mathcal{L}_i$ and $\mathcal{D}_i$ to `treeParent`
324: `planningState` $\leftarrow$ PLANNINGSTATES.TASK
325: **else**
326: **for** tree child R_j of R_i **do**
327: $\mathcal{D}_{ij} \leftarrow \emptyset$
328: **end for**
329: **while** $\mathcal{S}_i$`.size()` ≥ 1 **do**
330: $(j, \mathcal{C}_j^{(t)} j, \mathcal{Z}_j, \mathcal{I}_j, \mathcal{L}_j, \mathcal{D}_j) \leftarrow \mathcal{S}_i$`.dequeue()`

Stellenbosch University https://scholar.sun.ac.za

331: $\mathcal{N}_i^{(t)} \leftarrow \mathcal{N}_i^{(t)} \cup \mathcal{N}_j^{(t-1)}$
332: $(\mathcal{C}_i^{(t)}, \mathcal{Z}_i, \mathcal{I}_i) \leftarrow (\mathcal{C}_i^{(t)} \cup \mathcal{C}_j^{(t)}, \mathcal{Z}_i \cup \mathcal{Z}_j, \mathcal{I}_i \cup \mathcal{I}_j)$
333: $(\mathcal{J}, \mathcal{D}_{ij}) \leftarrow (\mathcal{J}_i \cup \mathcal{J}_j \cup \mathcal{I}_i, \mathcal{D}_{ij} \cup \mathcal{D}_j)$
334: $\mathcal{L}_i$.add$(\mathcal{L}_j)$ ▷ adding a hash table into another hash table
335: $\mathcal{D}_{ij} \leftarrow \mathcal{D}_{ij} \cup \mathcal{D}_j$
336: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \mathcal{D}_j$
337: **end while**
338: **if** treeParent $\neq \perp$ **then**
339: R_i sends (its combined information) $\mathcal{C}_i^{(t)}$, $\mathcal{Z}_i$, $\mathcal{I}_i$, $\mathcal{L}_i$ and $\mathcal{D}_i$ to treeParent
340: planningState $\leftarrow$ PLANNINGSTATES.TASK
341: **else**
342: planningState $\leftarrow$ PLANNINGSTATES.TASK
343: **end if**
344: **end if**
345: **end procedure**
346:
347: **function** GETINDIVIDUALEXPLORATIONREGION()
348: $l \leftarrow 1$
349: $A \leftarrow \sqrt{\hat{A}(\tau - t)}$
350: **while true do**
351: $\mathcal{U}_i^{(t)} \leftarrow \mathcal{V}_i \setminus \mathcal{B}_i^{(t)}$
352: $w \leftarrow \frac{A}{2}(1 + \texttt{uniform}(0,1))$
353: $h \leftarrow \frac{A^2}{w}$
354: **for** $j = 1$ **to** N **do**
355: generate a sampled point (x_U, y_U) randomly and uniformly distributed over $\mathcal{U}_i^{(t)}$
356: $R \leftarrow$ RECTANGLE(x_U, y_U, w, h) ▷ using Algorithm F.4
357: $S \leftarrow$ RECTANGLETOSET(R) ▷ using Algorithm F.4
358: **if** $S \subseteq \mathcal{U}_i^{(t)}$ **then**
359: $\mathcal{Z}_I \leftarrow \mathcal{Z}_i \cup S$
360: **return** R
361: **end if**
362: **end for**
363: **if** $l = L$ **then**
364: VIRTUALWORLDENLARGEMENT(w_0) ▷ using Algorithm 4.1
365: $l \leftarrow 1$
366: **end if**
367: $l \leftarrow l + 1$
368: **end while**
369: **end function**
370:
371: **procedure** FINDEXPLORATIONAREAINREGION()
372: $A \leftarrow \sqrt{\hat{A}(\tau - t)}$
373: $E \leftarrow$ RECTANGLEEXTRACTION$(N, \mathcal{V}_i \setminus \mathcal{C}_i^{(t)})$ ▷ using Algorithm F.4
374: **if** $|E| > A$ **then**
375: $\mathcal{X}_i \leftarrow E$
376: **else if** $\mathcal{Y}_i$.size$() > 0$ **then**
377: $\mathcal{X}_i \leftarrow \mathcal{Y}_i$.pop()
378: **else**
379: $\mathcal{Q}_i \leftarrow$GETINDIVIDUALEXPLORATIONREGION()
380: $\mathcal{Z}_i \leftarrow \mathcal{Z}_i \cup \mathcal{Q}_i$
381: $\mathcal{X}_i \leftarrow \mathcal{Q}_i$
382: **end if**
383: **end procedure**
384: **end class**

Stellenbosch University https://scholar.sun.ac.za

Chapter 6

Metrics, results and discussion

In this chapter, we present and discuss the results of our proposed coordination algorithm to govern solitary robots. This proposed coordination strategy applies in situation where the search time is known in advance and is too short to allow robots to explore the entire environment, robots are solitary, and they are exploring an unknown and static environment. To assess our soft-obstacle coordination strategy (SOS) experimentally, we compare it to the accidental rendezvous strategy (ARS) and the periodic rendezvous strategy (PRS). We also implement an hybrid approach which combines ARS with PRS for solitary robots.

Summary on the coordination strategies

- In ARS (discussed in Section 3.2), robots do not prearrange rendezvous, but only meet by chance and are assigned unbounded regions for further exploration.
- In PRS (discussed in Section 3.2), robots are aware of each other from the outset, prearrange their rendezvous and are assigned unbounded regions for further exploration.
- In the hybrid approach, robots do not prearrange rendezvous, but initially only meet by chance. At rendezvous, robots involved share their individual information, are assigned sectors for further search and schedule future rendezvous. There are a number of challenges when combining ARS and PRS for solitary robots, in particular management of future rendezvous: how many rendezvous should a robot

Stellenbosch University https://scholar.sun.ac.za

schedule, because a robot can encounter different robots at different times.

For robots to be incorporated into a team, a means would be required to schedule them into future rendezvous. Moreover, a robot encountering more than one team could have the problem of being a candidate for multiple possible conflicting other rendezvous. For instance, what happens if a robot which has a rendezvous scheduled encounters another robot with a different rendezvous scheduled? If both decide to interact and share information, they might miss their respective meetings. If they do not interact, each misses the additional information from the other. As far as we are aware, this issue is not addressed in the literature.

Here, we implement a naive version which adheres to the following principles:

  – only one rendezvous per robot; and
  – a robot travelling to attend a rendezvous trumps interactions.

- In SOS, robots meet by chance as in ARS and are assigned bounded regions for further exploration.

6.1 Metrics

For the coordination strategy, various metrics including exploration time, map quality and exploration cost can be considered to quantify performance. Yan et al. [116] conducted a survey of performance metrics for robot exploration. Performance metrics are used for different purposes subject to various assumptions. For example it does not make sense to use map quality performance here simply because we assume that maps built by robots are accurate. Similarly, we can not sensibly use exploration time because exploration time is assumed to be common to all robots.

For our purposes, we identify two main measures to quantify the performance of one or more robots, namely: the covered area (known as map completeness [116]) and the number of located targets. Finding targets is the goal of a solitary robot. Under the assumption that targets are independently and uniformly distributed in the search space, the expected number

Stellenbosch University https://scholar.sun.ac.za

of targets found by a robot is proportional to the amount of area covered by the robot. In this work, we thus equate the absolute performance of a robot with coverage of the robot. In addition, solitary robots applying a good coordination model should have sustainable performance. Solitary robots which know about each other after an interaction should have small overlapping area in their explored region.

We intend to compare our proposed method to benchmark techniques by determining the ideal coverage of each robot, and comparing this to the empirical coverage for each method. The comparison determines the performance of the techniques relative to a theoretical ideal.

In practice we expect sub-ideal empirical performance for all approaches since the robot can encounter obstacles, and may explore areas already explored by other robots which it is unaware of.

Let $S_i^{(t)}$ denote the region which the robot R_i was the first robot to explore before time t, and $|S_i^{(t)}|$ its size. We define the performance of the robot R_i after spending t units of time as

$$\varphi_i(t) = \frac{|S_i^{(t)}|}{\hat{A}(t)}, \tag{6.1.1}$$

where $\hat{A}(t)$ is given in Equation 5.5.1. The mean $\varphi(t)$ of the individual performances is expressed by

$$\varphi(t) = \frac{\sum_{i=1}^{N} \varphi_i(t)}{N}. \tag{6.1.2}$$

We use the standard deviation of performance to measure sustainability of robot's performance—this should be small in the case of sustainable performance. The sustainability $\sigma_i(t)$ of robots' performances until time t is given by

$$\sigma_i(t) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\varphi_i(t) - \varphi(t))^2}. \tag{6.1.3}$$

Another aspect of interest is to quantify the differences in the results of our coordination methods. Researchers commonly use the p-value from hypothesis tests as an indication of noteworthy differences between populations. Gail and Richard [33] mentioned that the p-value is not enough to describe the statistical difference between two groups. They reviewed metrics to evaluate another important feature—effect size. The p-value and

Stellenbosch University https://scholar.sun.ac.za

the effect size should be combined for more refined statistical interpretation. We will use Welch's t-test [108] to verify whether the means of robots' coverage using ARS, Hybrid and SOS are significantly different. We use Welch's t-test because we wish to compare two unpaired and independent populations of unequal variances. We will also use a paired-sample test, the Wilcoxon signed-rank test [110], to verify whether the means of robots' coverage using ARS, PRS and SOS are significantly different. We use the Wilcoxon signed-rank test because distributions are not necessarily normal, and robots are run from the same initial conditions which makes paired-sample tests more appropriate. We will also use effect size e to complement the p-value we obtain. The significance level we use throughout is 0.01.

Let μ_i and σ_i be the average and the standard deviation of the coverage of the i-th group respectively—a group refers to robots using a specific coordination technique. The effect size e (Cohen's formula) is given by

$$e = \frac{\mu_1 - \mu_2}{\sqrt{\frac{(N-1)(\sigma_1^2+\sigma_2^2)}{2N}}} \text{ [19]}. \tag{6.1.4}$$

The effect size e describes the amount of overlap between the distributions of the two groups. The values of e are categorised into three groups: small for $e \in [0, 0.5)$; medium for $e \in [0.5, 0.8)$; and large for $e \geq 0.8$. For example if $e = 0$, the means are equal. When e is at least medium, i.e. $e \geq 0.5$, the results of the two groups differ markedly.

6.2 Simulation description and experimental setup

This section describes the experimental simulations we intend to present and discuss the experimental results of coordination techniques in order to assess the performance of our proposed method for coordination of solitary robots. It also describes the programming setup we used for our simulations.

6.2.1 Simulation description

To have a fair comparison between the four coordination methods we consider, the methods should run under the same experimental conditions as

Stellenbosch University https://scholar.sun.ac.za

far as possible. However, robots in PRS have *a priori* knowledge about each other (they start within perception range of each other) [42, 109], while SOS and ARS generally consider solitary robots: robots which have no *a priori* knowledge about each other. This will lead to two scenarios. We will first compare ARS, PRS and SOS by considering the performance of robots starting within perception range of each other—Hybrid is reduced to PRS when robots start within the same perception range. Thereafter SOS will be compared to the ARS and Hybrid methods for the case of solitary robots where robots start from independent locations.

We will run several experiments in this regards. We will consider up to 3000 solitary robots for different cases. We will consider two situations: (i) solitary robots start all from nearby locations and (ii) solitary robots where most of them start from separated locations.

Four main aspects are investigated.

(1) We aim for our proposed coordination method to achieve sustainable performance. To assess that, we will run simulations in some search environments to measure the exploration performance of robots using ARS, PRS, Hybrid and SOS. (For the case of robots starting within the same vicinity, Hybrid is the same as PRS).

(2) We claimed that the use of margins is a part of the novelty of SOS. To assess their impact, we will perform an experiment in various search environments using ARS, PRS, Hybrid and SOS with and without margins, and see how the performance differs.

(3) We claimed in Chapter 5 that a search method to explore an exploration region should be selected carefully, since some search algorithms can yield poor performance, and proposed the use of the zigzag algorithm for SOS. Since the technique of frontiers [114] is used in the benchmark methods, here an SOS robot will also use the technique of frontiers to explore its assigned exploration region. We expect SOS with zigzag search to outperform SOS with the technique of frontiers in terms of coverage because the method of frontiers can suffer from backtracking behaviour which hinders robot progress. To verify this, we will run simulations in various search environments using ARS, PRS, Hybrid and SOS with zigzag search and with the method of frontiers.

Stellenbosch University https://scholar.sun.ac.za

(4) We proposed a method for graph view construction in interaction of robots as part of the leader election process and we will show (in Part II) that our proposed method reduces the number of messages exchanged between robots. To verify the impact of the graph view construction method on coordination, we will perform experiments with large networks and see the benefits of our method in coverage. For these experiments, we will run SOS using both the benchmark method (Algorithm 7.1) and our proposed method (Algorithm 8.1) for network construction. In this thesis, search time is quantified in terms of the robot's number of moves (time steps). We (somewhat arbitrarily) set the duration of sending of one message to 0.0002 moves, i.e. if during the process of leader election robots have exchanged 10000 messages, then that interaction of robots will last for 2 moves. So if the search time τ was set to 100 moves and this is their only interaction, these robots will only spend 98 searching moves.

In terms of results, for robots starting near to each other, we expect ARS and SOS to outperform PRS and Hybrid in terms of coverage, because PRS and Hybrid suffer from interruptibility. On the other hand, PRS and Hybrid might be expected to outperform ARS and SOS in sustainability. We expect SOS to outperform ARS in both coverage and sustainability. We will expect SOS using our pruning method (Algorithm 8.1) to outperform SOS using the benchmark method (Algorithm 7.1) for network view construction.

6.2.2 Experimental setup

A simulator was built in Python and Qt (a C++ cross-platform application framework for graphical user interfaces), which includes simulation of location-based sensing. We implemented a distributed Python class for the various distributed methods involved in coordination.

The perception range was typically set to $d = 20$, but $d = 10$ was used for cases with 50 robots (see Table 6.1) and $d = 8$ for more than 50 robots. The velocity was set to $\gamma = 1$.

In PRS and Hybrid, the rendezvous time t_r was set as done by Hourani et al. [42]: $t_r = 2a + b$, where a is how long a robot can explore after the previous rendezvous before it starts travelling back to the next rendezvous point, and b is a threshold which provides time for robots to explore as they

Stellenbosch University https://scholar.sun.ac.za

move to the rendezvous point. Different values of a can yield different behaviours of PRS robots and the value of a is user-defined. In our setting, we use 3 scenarios, each with a different value of a. For the first rendezvous in the three scenarios, a is set to $30, 50$ or 60 moves respectively. For Hybrid robots in the 3 scenarios, the value of a is typically set to 10, 20 or 30 moves respectively. We used smaller initial values of a for the hybrid method because we consider a large number of Hybrid robots (i.e. 50 robots); these smaller initial values of a are chosen to limit the possibility of interference which is likely to happen because of the number of solitary robots. These initial values of a are multiplied by 1.5 for each subsequent rendezvous, b is set to $\frac{a}{10}$ throughout. Let a_j and b_j denote the values for a and b at the j-th rendezvous. We thus have

$$a_j = \frac{3}{2} a_{j-1} \text{ for } j \geq 2$$

and

$$b_j = \frac{a_j}{10}.$$

Although we used three different values for the parameter a in our investigation, we only report the best results for PRS and Hybrid, and use these for the comparison. Also for rendezvous in PRS and Hybrid, some robots can arrive at a rendezvous point before others. In our experiments, when some robots arrive before others, robots which arrive first wait for other robots. Each rendezvous point is chosen by the leader from the combined explored area until that time.

Let $A = |W|$ denote the area of the search environment. The known upper bounds on the search time considered for up to 50 robots are chosen as follows for various values of k (see Table 6.1):

$$\tau = k \times \left(\frac{A}{2\gamma d N} - \frac{\pi d}{2\gamma} \right). \tag{6.2.1}$$

This choice is motivated by the form of Equation 5.5.1; The factor k is used to control the fraction of the environment that can be explored under ideal conditions. For larger number of solitary robots, τ will be set manually.

6.2.3 Search environments

Here we present search environments considered in our experiments. The environment used to run simulations was a 480×600 grid.

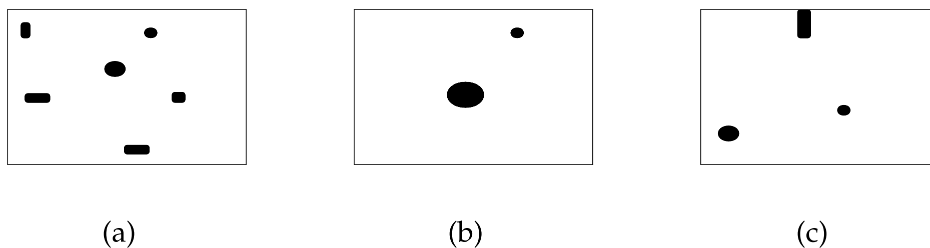
Stellenbosch University https://scholar.sun.ac.za

N	2	3	4	7	8	10	50
τ	2141	1421	1061	598	521	413	217
k	0.6	0.6	0.6	0.6	0.6	0.6	0.8
d	20	20	20	20	20	20	10
$\hat{A}(\tau)$	86896	58096	43696	25176	22096	17776	4371

Table 6.1: *Experimental setup: for each number of robots (N), the time for search (τ), the factor (k), the perception range (d), and the maximum possible coverage per robot $\hat{A}(\tau)$ for this time will be used to evaluate the robots' performance.*

6.2.3.1 Solitary robots starting within the same vicinity

We consider three environments to assess our methods for robots starting within the same vicinity. The three environments are presented in Figure 6.1.



(a) (b) (c)

Figure 6.1: *The simulation environments used for the first experiment.*

6.2.3.2 Randomly generated search environments for solitary robots starting from separated locations

For the case of solitary robots, we generated 50 environments (the number of obstacles and their positions, and the initial pose of robots) randomly from uniform distributions. Initial poses of robots are restricted to free regions. The occupancy rate of obstacles can reach up to 62.5% of the environment. A sample of these environments is shown in Figure 6.2. We considered up to 3000 solitary robots. For these environments, we compare our proposed method (SOS) to ARS and Hybrid—PRS is not appropriate for solitary robots since robots do not all start nearby each other.

We set the search time to $\tau = 200$. In this case, the maximum area each robot can cover is approximately 4400 using Equation 5.5.1. So the 50 robots

Stellenbosch University https://scholar.sun.ac.za

will be able to maximally cover an area of size $50 \times 4400 = 220000$, which is about 76% of the search area of $480 \times 600 = 288000$.

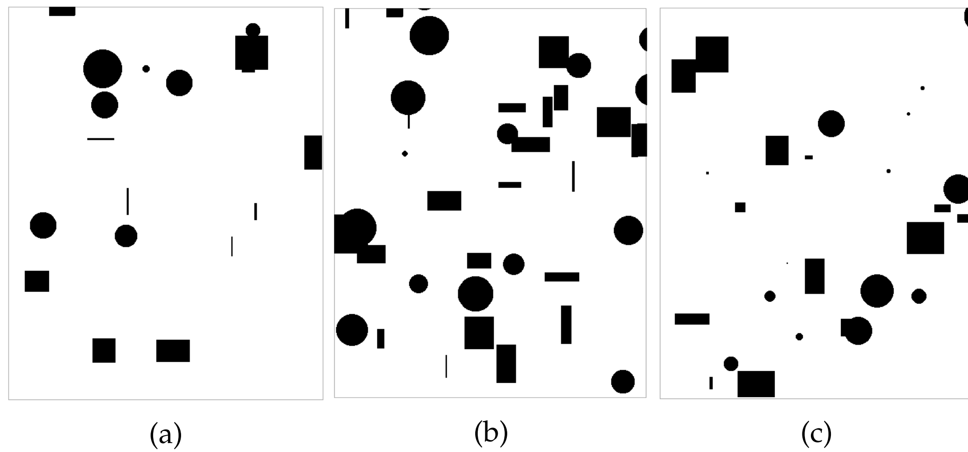


Figure 6.2: *Illustration of some environments considered in this work. Obstacles are in black.*

6.2.3.3 Tests with Victoria Park grid map

An important aspect of simulation is the consideration of actual search environments to test our methods. Real-world environments are different from simulated ones, so it is important to incorporate real-world environment to capture realistic tasks, behaviours and outcomes of robots. We will investigate the exploration performance of 50 solitary robots using ARS, Hybrid and SOS using the Victoria Park grid map [39], a commonly used data set for robotics applications such as navigation and exploration [100]. The Victoria Park image (Figure 6.3) represents a natural and unstructured terrain, and contains 2D laser scanning information of the terrain, collected using a vehicle equipped with a laser range finder. An important characteristic of the Victoria Park grid map is that each tree in the park is seen as a separate feature by the vehicle. The vehicle is also equipped with GPS to capture a map of the terrain. While this data set is often used to provide ground truth for evaluation of the accuracy of localisation and mapping algorithms [102], we treat this data set as an unknown environment for search in this thesis.

For our Victoria Park environment, we generated initial locations of robots randomly from uniform distributions. To create the grid map of the Victoria Park terrain shown in Figure 6.3b, we created a binary image, where 0

Stellenbosch University https://scholar.sun.ac.za

(a) (b)

Figure 6.3: *The Victoria Park grid map.* ***(6.3a)***: *A plan view of the Victoria Park terrain [39].* ***(6.3b)***: *An occupancy grid map of the Victoria Park terrain.*

denotes that a location is occupied and 1 unoccupied—which might not be so accurate—as follows: From the Victoria Park terrain RGB image (see Figure 6.3a), we consider each pixel λ as a tuple $\lambda = (\alpha, \beta, \gamma)$ where $\alpha, \beta, \gamma \in [0, 255]$. To convert the image to a binary matrix, we set $\hat{\lambda}$ the binary representation of a pixel λ to $\hat{\lambda} = 1$ if $\alpha \in [140, 160], \beta \in [180, 205]$ and $\gamma \in [180, 200]$, and $\hat{\lambda} = 0$ otherwise. These thresholds were obtained by exploring and analysing the pixels of the RGB image. In this data set, the obstacles are trees and boundaries of the terrain. The Victoria Park terrain has an occupancy of approximately 30%. We considered 50 solitary robots, which start from locations sampled uniformly from the obstacle-free region.

6.3 Results and discussion

6.3.1 Sustainability of exploration performance

Single simulation

Here, we run simulations from the same starting point in the same environment with each strategy and plot individual robot coverage over time until the end of the search, as well as the robots' individual trajectories (following the representation of Hourani et al. [42]). We also perform 150 simulated runs—50 on each of the three environments in Figure 6.1—and show means and standard deviations of coverage per strategy.

Figures 6.4–6.6 show the results for individual coverage progress (Fig-

Stellenbosch University https://scholar.sun.ac.za

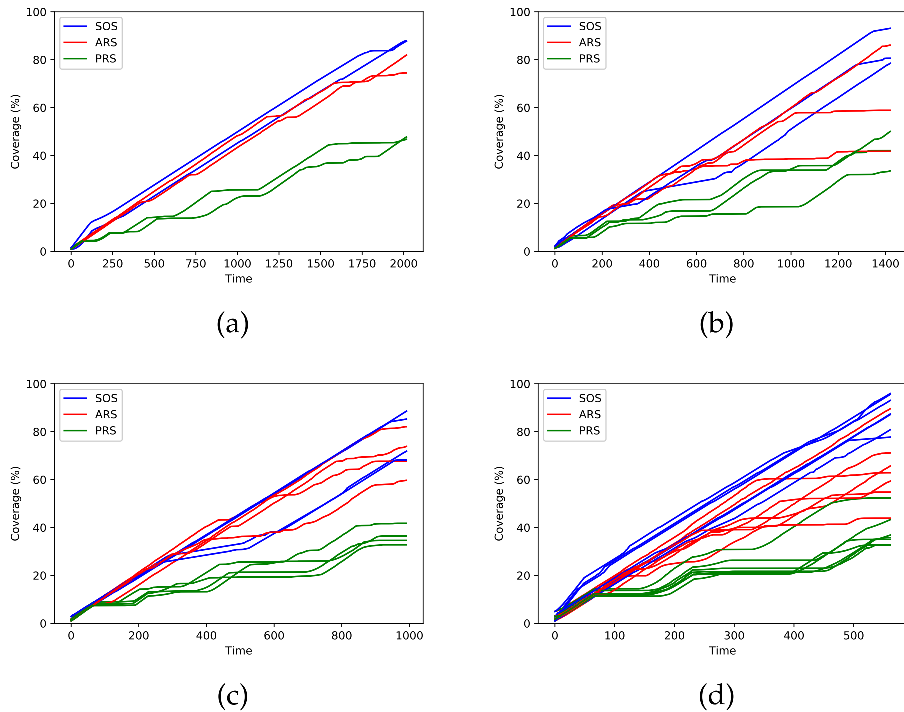


Figure 6.4: *Individual coverage progress (using Equation 6.1.1) for different numbers of robots on the environment in Figure 6.1a.* ***(6.4a)****: Two robots.* ***(6.4b)****: Three robots.* ***(6.4c)****: Four robots.* ***(6.4d)****: Seven robots.*

ure 6.4), differences of final coverages between the benchmark and our proposed methods (Figure 6.5), and the robot trajectories (Figure 6.6) for different teams of robots starting their search from locations within sensing range (i.e. all robots are initially in the same communication network) for the environments in Figure 6.1.

In the case of two robots, all three strategies perform well in terms of sustainability using the environments in Figure 6.1a. This can be observed in Figure 6.4a as robots have similar performance within each approach. Robots had a balanced assignment. ARS and SOS had good performance in terms of coverage, but PRS has less coverage which can be attributed to interruptibility, which took roughly 49% of the search time: as can be seen in Figure 6.4, the PRS robots typically covered half the area covered by other robots.

The experiments with groups of three and four robots investigate the case of unbalanced assignments because robots start close to a boundary which they are unaware of. The unbalanced assignments can be viewed in Figures

Stellenbosch University https://scholar.sun.ac.za

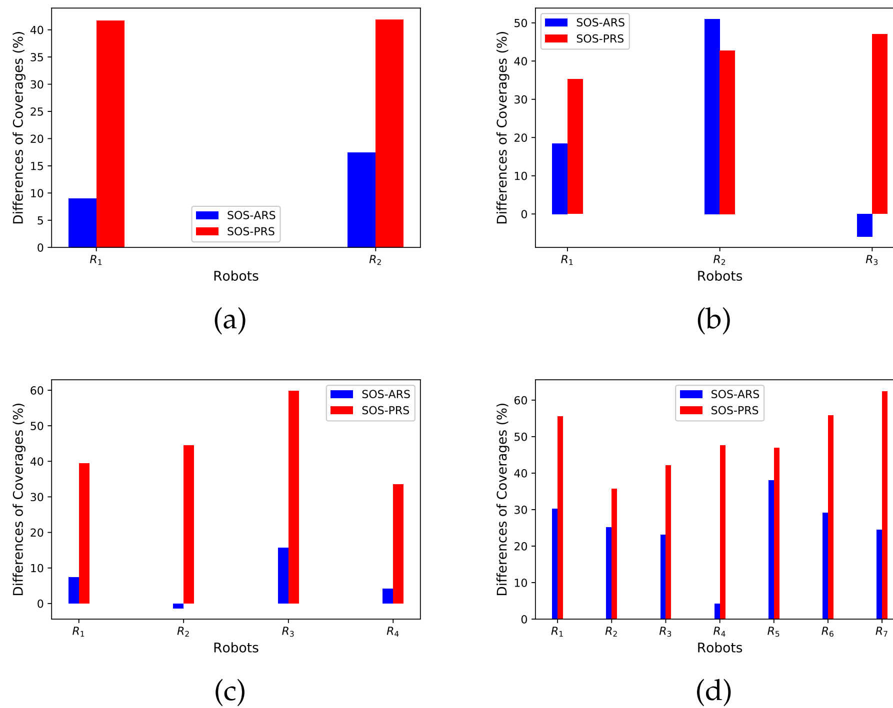


Figure 6.5: *Individual coverage performance for different teams of robots following the representation of Hourani et al. [42]. Positive values indicate that our proposed method outperforms the benchmark methods.* ***(6.5a)****: A team of two robots.* ***(6.5b)****: A team of three robots.* ***(6.5c)****: A team of four robots.* ***(6.5d)****: A team of seven robots.*

6.6d and 6.6g. When treating an unbalanced assignment, our method gives better results than ARS and PRS in coverage (Figures 6.5b–6.5c). With ARS in the team of three robots for instance, two robots (R_1 and R_3) suffered from high interference. Trajectories of robots (as illustrated in Figure 6.6) confirm that robots using SOS generally explore non-overlapping regions, whereas in the ARS case there is overlapping of explored regions.

Figure 6.5d shows coverage results for the case of seven robots. In Figure 6.5d with ARS, R_1, for example, has low coverage as it was exploring an already explored region (see the green robot in Figure 6.6k).

As expected, from results run on environments in Figure 6.1 PRS and SOS provide more sustainable performance than ARS. PRS has a very small standard deviation of coverage per robot—partly also attributable to lower coverage values of PRS.

The issue of unbalanced assignments does not affect the sustainability of PRS significantly because robots meet on a regular basis and share indi-

Stellenbosch University https://scholar.sun.ac.za

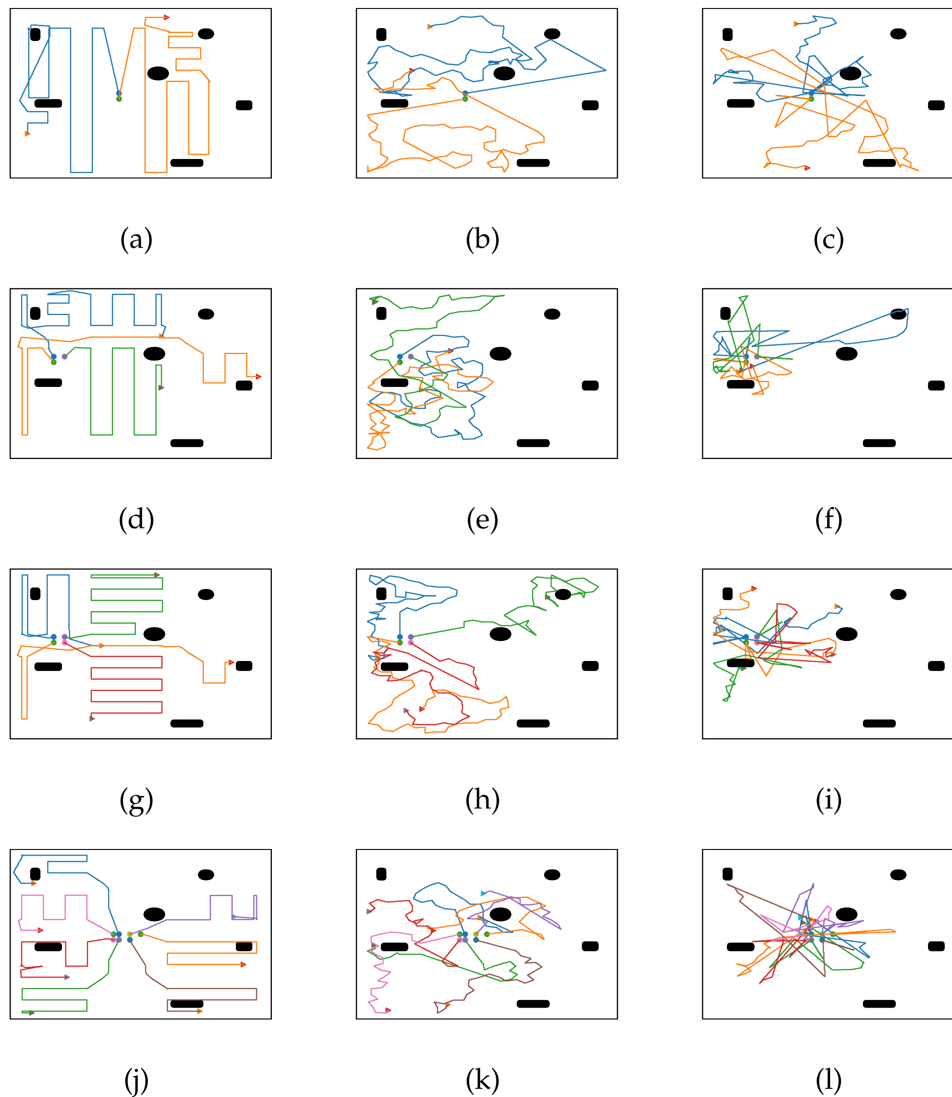


Figure 6.6: *Individual robot trajectories for different teams of robots (SOS, ARS and PRS from left to right). Starting points are depicted by dots.* ***(6.6a–6.6c)****: A team of two robots.* ***(6.6d–6.6f)****: A team of three robots.* ***(6.6g–6.6i)****: A team of four robots.* ***(6.6j–6.6l)****: A team of seven robots.*

vidual information so that each robot involved in the meeting is aware of activity of other robots.

For ARS in particular, ARS robots provides good coverage performance with approximately 25% of interference which is small. We evaluated the level of interference by determining overlapping area in the explored regions of robots as a post hoc.

Stellenbosch University https://scholar.sun.ac.za

Multiple simulations

Figure 6.7 summarises the coverage results from 50 simulations for each environment in Figure 6.1 for the cases of two, three, four, seven, eight and ten robots. Robots start from random and obstacle-free locations in close proximity to each other (with the same starting positions for each technique). For each coordination strategy and number of robots, we give a box-and-whiskers plot of the performance.

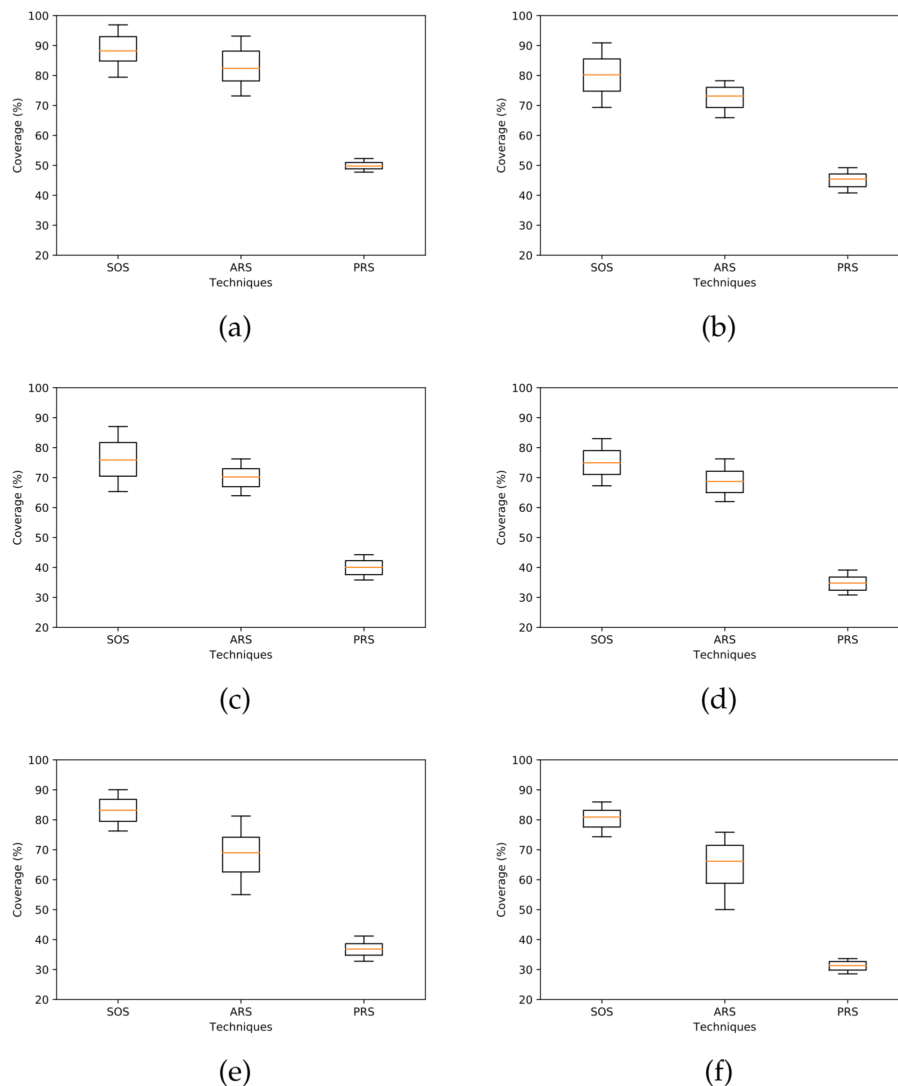


Figure 6.7: *Box-and-whiskers plots of the performance of each coordination strategy.* ***(6.7a)****: Two robots.* ***(6.7b)****: Three robots.* ***(6.7c)****: Four robots.* ***(6.7d)****: Seven robots.* ***(6.7e)****: Eight robots.* ***(6.7f)****: Ten robots.*

Stellenbosch University https://scholar.sun.ac.za

For the coverage results presented in Figure 6.7, the results confirm again that PRS provides the most sustainable performance as had been expected: its standard deviation is very small. The smaller deviation is also due in part to smaller average coverage. Still, it achieves less coverage due to interruptibility.

Hypothesis test and effect size

We also used a Wilcoxon signed-rank test and the effect size (see Table 6.2) to verify whether the mean coverages for ARS, PRS and SOS are significantly different.

(p, e)	ARS	PRS
SOS	$(2.29 \times 10^{-26}, 1.62)$	$(2.29 \times 10^{-26}, 1.98)$
ARS	–	$(2.29 \times 10^{-26}, 1.89)$

Table 6.2: *The p-value and the effect size e for the coordination strategies. This table indicates the p-value using Wilcoxon signed-rank test and the effect size e by comparing the results summarised in Figure 6.7.*

We observed a p-value of 2.29×10^{-26} between every pair of the methods (see Table 6.2), which are less than the threshold 0.01. So the means of the coverage results using the two strategies (ARS and PRS) against SOS are significantly different.

In terms of effect size, according to the classification in Gail and Richard [33], the effect sizes between SOS and ARS and between SOS and PRS are large ($e \geq 0.8$). This indicates that the coverage results between all pairs are markedly different.

In the case of balanced assignments for robots starting from nearby locations, we did not observe difference between SOS and ARS in terms of coverage. Rather, the robot performance obtained with SOS is more sustainable than that with ARS in cases of unbalanced assignments. However, since robots can not determine the type of their assignment (i.e. whether it is balanced or unbalanced) *a priori*, it is beneficial for robots to apply SOS rather than ARS as it is robust to unbalanced assignments. Our results confirm that SOS outperforms other strategies in coverage by utilizing information on the available search time.

Stellenbosch University https://scholar.sun.ac.za

6.3.1.1 Solitary robots starting from separate locations

Sustainability

We now consider sustainability of exploration performance for solitary robots using ARS, Hybrid and SOS.

Results of 50 scenarios using ARS, Hybrid and SOS																									
Scenarios	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
ARS	46	**65**	40	**43**	60	**57**	52	37	46	**63**	37	**54**	45	36	56	**65**	51	36	41	57	55	49	**59**	45	55
Hybrid	**55**	36	**57**	32	39	51	57	33	45	45	38	33	44	44	38	50	53	35	**59**	37	30	54	48	34	43
SOS	48	63	55	41	**67**	45	**69**	**66**	**60**	44	**52**	**54**	**56**	**57**	**69**	50	**68**	**41**	57	**66**	**63**	**55**	46	**54**	**56**
Scenarios	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
ARS	41	41	38	**60**	50	**58**	45	38	**61**	46	51	39	46	41	**49**	40	**64**	51	43	37	55	47	**64**	53	54
Hybrid	31	**51**	**57**	49	40	45	**59**	48	44	46	58	**52**	57	41	32	38	54	**55**	**56**	33	49	56	49	30	50
SOS	**44**	47	52	50	**53**	49	42	**58**	48	**68**	**61**	51	**64**	**67**	48	**60**	60	54	52	**48**	**70**	**69**	62	**65**	**66**

Table 6.3: *Results of* 50 *scenarios using SOS, ARS and Hybrid. The results are in terms of the average percentage of ideal coverage achieved by individual robots in each experiment. We evaluate the average coverage by individual robots in each strategy. Bold values indicate high coverage.*

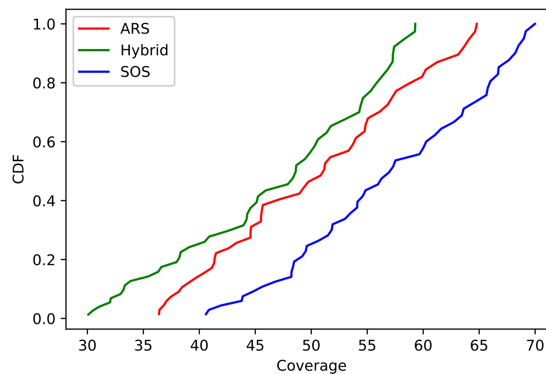


Figure 6.8: *Cumulative distribution function (CDF) of* 50 *scenarios using SOS, ARS and Hybrid. The average percentage of ideal coverage achieved by individual robots for these* 50 *scenarios are presented in Table 6.3.*

Table 6.3 and Figure 6.8 show the results of each of the 50 scenarios, each using a different environment. In Table 6.3, the averages and standard deviations (in percentages) of coverage using ARS, Hybrid and SOS are 49.22 ± 8.61, 45.49 ± 9 and 56.08 ± 8.48 respectively. So SOS outperforms the other two approaches, in terms of average and standard deviation of the coverage on these simulations. Figure 6.8 shows that coverage progresses

Stellenbosch University https://scholar.sun.ac.za

obtained with the three techniques are linear with different slopes; SOS outperformed the other two techniques noticeably.

Table 6.4 shows the individual robot coverage results using SOS, ARS and Hybrid, and Table 6.5 shows the number of interactions that each robot participated in. In SOS, robots have achieved 49.7% of average coverage with standard deviation of 18.0%. In ARS they have achieved 46.1% average coverage with standard deviation of 18.8%. In Hybrid they have achieved 45.1% average coverage with standard deviation of 15.8%.

Individual robot coverage per technique																									
Robots	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}	R_{17}	R_{18}	R_{19}	R_{20}	R_{21}	R_{22}	R_{23}	R_{24}	R_{25}
ARS	25	**52**	**65**	**55**	38	**51**	12	25	34	**88**	42	24	22	**68**	15	**64**	**65**	**54**	**87**	34	64	**77**	**59**	57	**52**
Hybrid	32	34	47	46	39	18	44	**58**	55	63	26	22	23	65	36	41	36	44	72	**42**	**76**	45	48	56	36
SOS	**77**	49	34	39	**53**	43	**46**	35	**56**	78	**50**	**65**	**38**	61	**54**	43	55	45	46	29	31	62	36	**72**	31
Robots	R_{26}	R_{27}	R_{28}	R_{29}	R_{30}	R_{31}	R_{32}	R_{33}	R_{34}	R_{35}	R_{36}	R_{37}	R_{38}	R_{39}	R_{40}	R_{41}	R_{42}	R_{43}	R_{44}	R_{45}	R_{46}	R_{47}	R_{48}	R_{49}	R_{50}
ARS	28	**39**	44	38	38	**52**	32	**73**	14	29	49	59	58	17	46	33	46	29	**49**	**82**	60	58	**27**	46	29
Hybrid	37	41	38	27	43	46	39	63	37	30	**60**	**72**	**61**	**65**	**62**	**54**	25	33	36	67	**69**	**66**	14	46	17
SOS	**68**	19	**52**	**44**	**62**	45	**62**	29	**79**	**56**	44	38	38	36	52	45	**47**	**45**	21	50	42	35	26	**49**	**74**

Table 6.4: *Individual robot coverage results for search and rescue using SOS, ARS and Hybrid. Bold values indicate highest coverage. Robots using SOS outperform robots using ARS with averages of* $49.7 \pm 18.0\%$ *and* $46.1 \pm 18.8\%$ *respectively.*

Number of meetings per robot																									
Robots	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}	R_{17}	R_{18}	R_{19}	R_{20}	R_{21}	R_{22}	R_{23}	R_{24}	R_{25}
ARS	**2**	**1**	**4**	**6**	**3**	1	0	0	**1**	0	2	1	0	0	3	0	**3**	0	0	0	**3**	**1**	2	0	**2**
Hybrid	0	0	0	4	1	0	**2**	0	0	0	2	**2**	**3**	**1**	**9**	0	1	**2**	0	0	0	0	1	0	0
SOS	0	0	0	2	0	**3**	0	0	**1**	**3**	**3**	1	**3**	0	0	**1**	1	1	**3**	**1**	0	**1**	**4**	**2**	1
Robots	R_{26}	R_{27}	R_{28}	R_{29}	R_{30}	R_{31}	R_{32}	R_{33}	R_{34}	R_{35}	R_{36}	R_{37}	R_{38}	R_{39}	R_{40}	R_{41}	R_{42}	R_{43}	R_{44}	R_{45}	R_{46}	R_{47}	R_{48}	R_{49}	R_{50}
ARS	1	**4**	**2**	0	0	**4**	**1**	**7**	1	0	**2**	0	0	0	0	3	**0**	**4**	**7**	0	**2**	**1**	**0**	0	**2**
Hybrid	**2**	3	**2**	0	0	1	0	2	0	0	0	**2**	0	**1**	**8**	1	**0**	3	0	0	0	0	**0**	0	0
SOS	0	1	**2**	**1**	**1**	0	**1**	0	0	**1**	**2**	**2**	**2**	**1**	1	**5**	**0**	1	2	**6**	0	**1**	**0**	**1**	0

Table 6.5: *Number of interactions that each robot participated in for search and rescue using SOS, ARS and Hybrid which coverage results presented in Table 6.4. Bold values indicate highest number of meetings.*

Table 6.6 and Figure 6.9 show the results of 30 simulations using our Victoria Park grid map each using 50 solitary robots. In Table 6.6, the averages and standards deviations (in percentages) of coverage using ARS, Hybrid and SOS are 57.7 ± 3.80, 56.46 ± 4.65 and 60.7 ± 4.46 respectively. So SOS outperforms the other two in average coverage, while ARS slightly outperforms the other approaches in standard deviation. That means ARS was more sustainable than SOS in this setting. Figure 6.9 shows that coverage progresses obtained with the three techniques are linear with SOS slightly better than the other two techniques.

Table 6.3 has confirmed that SOS outperforms ARS and Hybrid in coordinating solitary robots in these specific environments. This difference is

Stellenbosch University https://scholar.sun.ac.za

Results of 30 scenarios using ARS, Hybrid and SOS on the Victoria Park grid map																														
Scenarios	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
ARS	59	53	**63**	58	50	55	60	**59**	61	52	**64**	58	57	**64**	60	53	**60**	**58**	59	54	58	59	52	**61**	62	62	**60**	56	52	54
Hybrid	52	**61**	62	**63**	51	52	**64**	**59**	52	56	52	60	53	60	56	**61**	50	50	62	52	59	**63**	**58**	59	50	54	52	52	56	63
SOS	**60**	58	55	**65**	**55**	**67**	54	57	**67**	**68**	57	**62**	**60**	59	**61**	59	56	54	**65**	**61**	**65**	56	**58**	60	**68**	**68**	59	**65**	**57**	**65**

Table 6.6: *Results of 30 scenarios using SOS, ARS and Hybrid on the Victoria Park grid map. The results are in terms of the average percentage of ideal coverage achieved by individual robots in each experiment. We evaluate the average coverage by individual robots in each strategy. Bold values indicate highest coverage.*

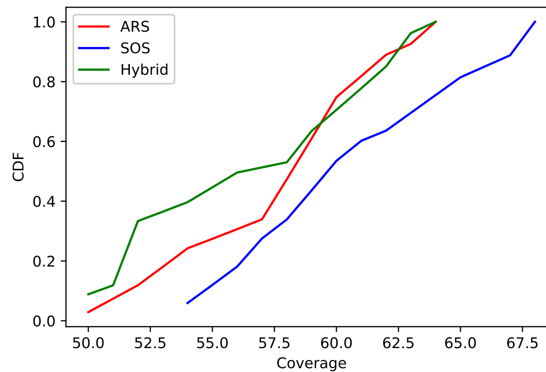


Figure 6.9: *CDF of 30 scenarios using SOS, ARS and Hybrid on the Victoria Park grid map. The average percentage of ideal coverage achieved by individual robots for these 30 scenarios are shown in Table 6.6.*

well observed when treating coverage of solitary robots given interactions (Figure 6.5). Although the results show that SOS outperforms ARS and Hybrid in most cases, as in Table 6.3, this does not mean SOS always performs better than ARS or Hybrid on average when robots have different configurations in the three strategies (i.e. robots have different behaviour in the three methods but they all start from the same locations). The results show that in some cases ARS still outperforms SOS, in situations where ARS robots have a better initial scattering distribution.

In the case of unbalanced assignments with a large number of interacting robots, SOS might not outperform the other two strategies—in Table 6.3, ARS outperforms SOS in some experiments for example. This is probably because when considering a large number of robots, robots may have different experiences using ARS and SOS.

We also used Welch's t-test and the effect size to verify whether the means of results of ARS, Hybrid and SOS are significantly different. We test the null hypotheses that the means of the coverage for each strategy are equal.

Stellenbosch University https://scholar.sun.ac.za

(p,e)	ARS	Hybrid
SOS	$(1.35 \times 10^{-4}, 0.7449)$	$(3.38 \times 10^{-8}, 1.036)$
ARS	–	$(3.8627 \times 10^{-2}, 0.4143)$

Table 6.7: *The p-value and the effect size e for the coordination strategies. This table indicates the p-value using Welch's t-test and the effect size e by comparing the results using our coordination strategies with* 50 *solitary robots in the simulated search environment.*

We observed a p-value of 1.35×10^{-4} between SOS and ARS, and a p-value of 3.38×10^{-8} between SOS and the hybrid method (see Table 6.7), which are less than the threshold 0.01. So the means of the coverage results using the two strategies (ARS and Hybrid) against SOS are significantly different for the environments. Table 6.7 shows that there is not enough evidence in coverage to confidently distinguish the performances of ARS and Hybrid.

In terms of effect size, according to the classification in Gail and Richard [33], the effect size between SOS and ARS is medium ($e \in [0.5, 0.8)$) and the effect size between SOS and the hybrid method is large ($e \geq 0.8$). This indicates that the differences in coverages between SOS and both ARS and Hybrid are marked: the average discrepancy in performance is about 0.75-1.05 standard deviations.

In the three coordination strategies it seems that achieving a higher number of interactions does not necessarily mean the strategy will achieve better performance. In Table 6.5, R_5 had three interactions using ARS and one interaction using Hybrid, but none using SOS. However R_5 using SOS outperformed R_5 using ARS and Hybrid, and R_5 using both ARS and Hybrid yield almost the same result (Table 6.4). This result may mean that R_5 using SOS had a better search behaviour than R_5 using ARS and Hybrid.

Similarly, although R_{45} had six interactions in SOS and no interaction in ARS or Hybrid, R_{45} using ARS and Hybrid both outperformed R_{45} using SOS. This result supports our earlier claim that performance of robots not only depends on their initial configurations (location and type of environment) but also on the balance of their assignment. Another important point to highlight is that the high performance of R_{45} in ARS could cause poor performance of other robots which could be in its region. That is because more robots in a region leads to higher interference before interactions. Furthermore, the robots R_{15} and R_{40} have many interactions using Hybrid. These

Stellenbosch University https://scholar.sun.ac.za

two robots have chances to meet with other robots after their initial interactions. In coverage, R_{40} achieves high performance while R_{15} has a low performance. Many considerations could explain the poor performance of R_{15} despite its high number of interactions; the most likely factor is interruptibility. For R_{40}, it probably learns more information about the environment from its later interactions than from its initial team. This could allow R_{40} to search in regions which lead to low interference.

While robots using SOS outperformed robots using ARS and Hybrid overall, we observe more interactions in ARS and Hybrid than in SOS (Table 6.5). This indicates that ARS robots or Hybrid robots may end up encountering one another again after earlier interactions, but SOS splits robots up more effectively.

It was said earlier that ARS, Hybrid, and SOS can perform differently even when applied to the same setting. The same robot R_j may have different opportunities to interact, although the robot uses the same initial configuration in these strategies. However, running several experiments on different environments may give some indications of the superiority of a method: If one method outperforms another method in most situations with different configurations, we may conclude that the first method seems to be better than the second. This is the view we consider below when comparing the various methods.

In light of this, we conclude that SOS seems to be generally better than ARS. It was observed that SOS outperforms ARS in 31 out of the 50 cases: both methods give almost the same results in 11 cases, and ARS outperforms SOS in the 8 remaining cases. Based on the p-value (see Table 6.7), there is not enough evidence in coverage to confidently distinguish the performances of ARS and Hybrid, but Hybrid outperforms ARS in 3 cases only (Table 6.3).

Table 6.5 again indicates that there are more interactions in ARS than in SOS, while SOS outperforms ARS in overall coverage. Interactions are the only reliable means by which robots may avoid overlapping search. This indicates that SOS has some superiority over ARS because, though fewer meetings are performed in SOS, SOS still achieves higher overall coverage.

Stellenbosch University https://scholar.sun.ac.za

The Victoria Park grid map

Table 6.6 indicates that SOS typically, but not always, outperforms ARS and Hybrid in coordinating solitary robots on the Victoria Park grid map. We also used the Welch's t-test to verify whether the mean coverages of ARS, Hybrid and SOS are pair-wise significantly different for this map. We observed a p-value of 0.00925 between SOS and ARS, and a p-value of 0.0008 between SOS and Hybrid, which are both less than the threshold 0.01. So the means of the coverages using the two benchmark strategies (ARS and Hybrid) are significantly different to that of SOS. The coverages of ARS and Hybrid do not differ significantly ($p = 0.2482$). We also observed an effect size of 0.668 between SOS and ARS, and an effect size of 0.8439 between SOS and the Hybrid approach. From [33], the effect size between SOS and ARS is medium ($e \in [0.5, 0.8)$), and between SOS and the Hybrid method is large ($e \geq 0.8$), so the means of the coverage results using the two benchmark strategies (ARS and Hybrid) differ markedly from that of SOS.

Large networks of solitary robots

We also consider large networks of solitary robots using the Victoria Park grid map and compare our coordination strategies. Results are shown in Table 6.8.

(p, e)	ARS	Hybrid
$N = 100, \tau = 150$	$(0.0008, 0.4863)$	$(4.5601 \times 10^{-16}, 1.2659)$
$N = 1000, \tau = 90$	$(4.5011 \times 10^{-19}, 0.4036)$	$(5.3330 \times 10^{-122}, 1.2659)$
$N = 2000, \tau = 50$	$(6.3885 \times 10^{-8}, -0.1714)$	$(6.8996 \times 10^{-275}, 1.2149)$
$N = 3000, \tau = 30$	$(1.7937 \times 10^{-72}, 0.4714)$	$(0, 1.1876)$

Table 6.8: *p-values and effect sizes e for the two benchmark coordination strategies (ARS and Hybrid) against SOS. This table indicates the p-value using Welch's t-test and the effect size e by comparing the coverage of individual robots on a single run using our coordination strategies with up to* 3000 *solitary robots in the Victoria Park grid map.*

With regards to the p-values and effect sizes observed in Table 6.8, Hybrid achieves considerably poorer results than SOS: results of these two coordination strategies are significantly different. The interruptibility of Hybrid has a definite negative impact on the performance of Hybrid robots. Compared to ARS, SOS achieves better results in 3 cases out of 4. ARS

Stellenbosch University https://scholar.sun.ac.za

achieves better than SOS for the case with 2000 solitary robots. This is why the corresponding effect size is negative.

For solitary robots, our results indicate that SOS is better than ARS and Hybrid as it outperforms ARS and Hybrid in most cases, both for simulated search environments and for the Victoria Park grid map. We conclude that it is beneficial for solitary robots to apply SOS when an upper bound on search time is available. We note that the simulated search environments are more cluttered than the Victoria Park grid map, but SOS performed better relative to ARS and Hybrid in these simulated environments compared to the Victoria Park grid map. This may mean that the benefits offered by SOS are more likely to manifest in very cluttered search environments.

6.3.2 The benefits of margins in SOS

6.3.2.1 Solitary robots starting from nearby locations

For this simulation, the team of robots start their searches from locations within sensing range, and use the search environment in Figure 6.1a. Figures 6.10 and 6.11 show the individual trajectories (Figures 6.10a and 6.10b for the case of unbalanced assignment, and Figures 6.11a and 6.11b for the case of balanced assignment), coverage results (Figures 6.10c and 6.11c for the cases of unbalanced and balanced assignments respectively), and progress results (Figures 6.10d and 6.11d for the cases of unbalanced and balanced assignments respectively) for a team of 8 robots using SOS with and without margins. Some margins are illustrated as gray boxes in Figures 6.10a and 6.11a. These results give a preliminary indication that it is beneficial to SOS robots to use margins when robots have unbalanced assignment, which is the type of situation for which we motivated the design of SOS.

The advantage of using margins with SOS is that robots can use some parts of them while travelling to their new exploration regions without overlapping with other robots which they know about. This situation happens mostly when robots have an unbalanced assignment as is the case in Figure 6.10, which we consider below. Among the 8 robots, the exploration regions for the four left-hand side robots fall entirely in the search environment, and the four exploration regions for the four right-hand side robots are partly outside of the search environment. The four robots with balanced assignment achieve almost the same performance, with or without the use

Stellenbosch University https://scholar.sun.ac.za

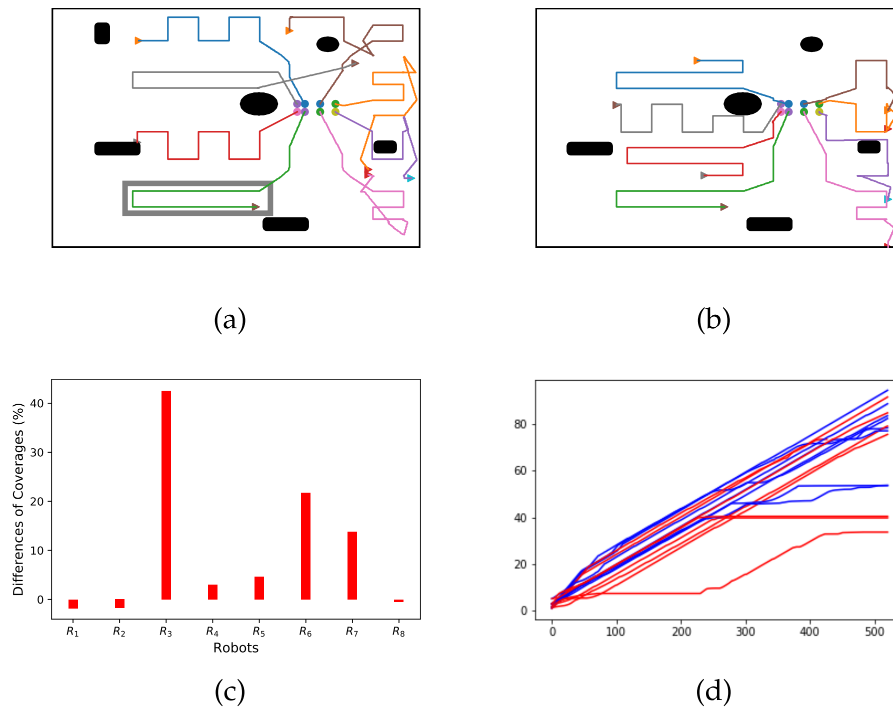


Figure 6.10: *SOS with and without the use of margins for the case of unbalanced assignment.* ***(6.10a)****: Trajectories of the* 8 *SOS robots using margins. Gray box is an illustration of margins between exploration regions.* ***(6.10b)****: Trajectories of the* 8 *SOS robots without the use of margins.* ***(6.10c)****: Differences of coverages for SOS with and without margins.* ***(6.10d)****: Coverage progress of SOS robots with and without margins.*

of margins—although we could see benefits from margins in very cluttered environments as well. This is not surprising because their exploration regions have enough space where robots can search. But the other four robots need to select new exploration regions after covering their assigned exploration regions early. In this situation, robots are most likely to overlap. So by the use of margins, robots can minimise this chance of overlapping between themselves.

In Figure 6.11, where the 8 robots have balanced assignment using SOS with or without margins yields almost the same results, both in terms of robot coverage (Figure 6.11c) and coverage progress (Figure 6.11d).

6.3.2.2 Solitary robots starting from separated locations

We also compare SOS without the use of margins to ARS and Hybrid for region exploration, for the 50 solitary robots in simulated environments. We

Stellenbosch University https://scholar.sun.ac.za

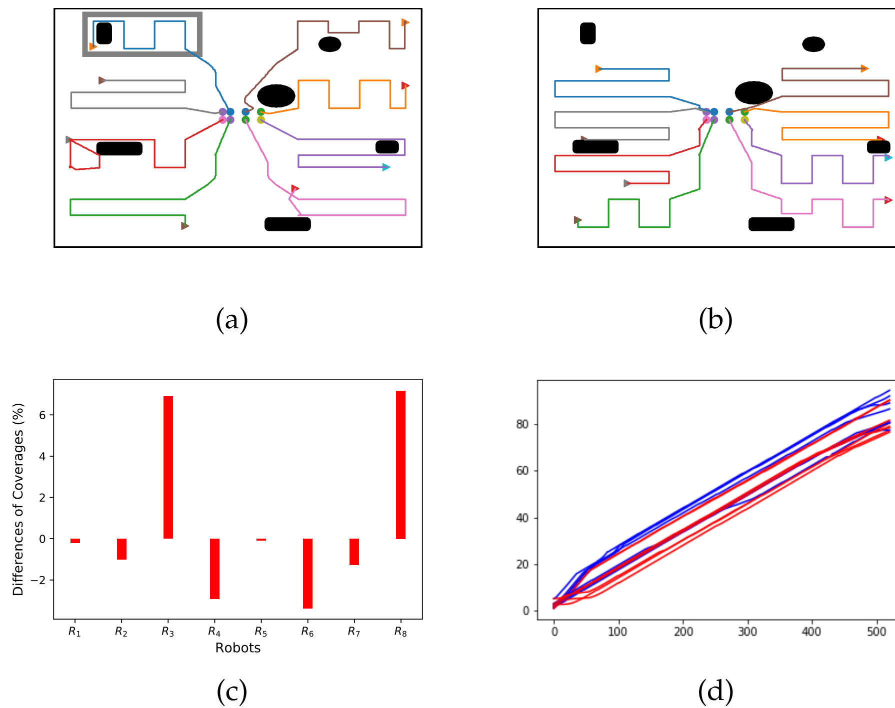


Figure 6.11: *SOS with and without the use of margins for the case of balanced assignment. Some trajectories through obstacles are caused by the image rendering effects.* ***(6.11a)****: Trajectories of the* 8 *SOS robots using margins.* ***(6.11b)****: Trajectories of the* 8 *SOS robots without the use of margins.* ***(6.11c)****: Differences of coverages for SOS with and without margins.* ***(6.11d)****: Coverage progress of SOS robots with and without margins.*

compare these variants of SOS using Welch's t-test and consider the effect size, and the statistical results are as follows (see Table 6.9).

The p-values obtained indicate that the coverage using SOS is significantly better than ARS and Hybrid with a medium effect size. However, compared to the results obtained between SOS with margins, and ARS and Hybrid (that were discussed in the previous section), the use of margins is beneficial to SOS, when solitary robots explore an unknown environment. The same conclusions can also be drawn from the results of the effect size.

(p,e)	ARS	Hybrid
SOS without margins	$(0.00025, 0.525)$	$(1.18 \times 10^{-5}, 0.68)$

Table 6.9: *The p-value and the effect size e for the coordination strategies. This table indicates the p-value using Welch's t-test and the effect size e by comparing the results using our coordination strategies with* 50 *solitary robots in the simulated search environment.*

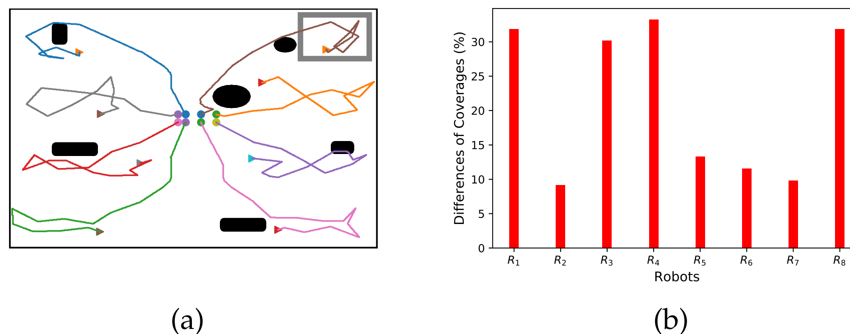
Stellenbosch University https://scholar.sun.ac.za

These results indicate that it is beneficial for SOS robots to use margins.

6.3.3 Selection of a search mechanism for SOS robots

6.3.3.1 Solitary robots starting from nearby location

Figure 6.11a and Figure 6.12a show the results for individual trajectories for a team of 8 robots using SOS with its default zigzag search and SOS using the technique of frontiers for exploration of exploration regions respectively. The relative coverages of each team's robots are plotted in Figure 6.12b. The teams of robots start their searches from locations within sensing range, and use the search environment in Figure 6.1a. These results indicate that it is slightly beneficial to SOS robots to use zigzag for exploration of their regions rather than the technique of frontiers.



(a) (b)

Figure 6.12: *SOS using the technique of frontiers as the search algorithm.* ***(6.12a)****: Trajectory of the* 8 *SOS robots using the technique of frontiers.* ***(6.12b)****: Differences of coverages for SOS using the technique of frontiers and zigzag search.*

The advantage of using zigzag search with SOS is that robots do not backtrack, as in the case of the technique of frontiers. A robot selects frontiers which maximise some utility function. In our case, since robots explore unknown environments, we define utility as the area of unexplored region. So it is very likely that robots choose frontiers which will lead them to some backtracking behaviour. Such backtracking behaviour is indicated by crossing trajectories in Figure 6.12a. This situation often happens with frontier-based search because frontiers are selected from local information and the search environment is unknown to robots. This backtracking behaviour hinders robot progress. In Figure 6.12b, we see that SOS

Stellenbosch University https://scholar.sun.ac.za

robots using zigzag search yielded better performance than SOS robots using frontiers. Another reason why SOS robots using the technique of frontiers achieve worse results than SOS robots with zigzag search is because robots explore a bounded region. Exploration of a bounded region with the technique of frontiers adds too many constraints to the motion behaviour of robots, so some backtracking behaviour is likely to occur even when the region is obstacle-free. The technique of frontiers is good for unbounded search, i.e sectors: when the region assigned to a robot to search is large, environment boundaries should not be encountered as often, so less backtracking would be required—in such a setting, the technique of frontiers could be better than zigzag search.

6.3.3.2 Solitary robots starting from separate locations

For the comparison of SOS frontier techniques and the other two methods (ARS and Hybrid) for region exploration by 50 solitary robots, the statistical results are as follows (see Table 6.10). For SOS with the technique of frontier and ARS, we obtained a p-value of 0.021 (which is greater than the threshold 0.001) and on effect size e of 0.124 (i.e. effect size e is small). For SOS with the technique of frontier and Hybrid, we obtained a p-value less than the threshold and a small effect size. This means that the results of SOS and ARS are almost equal. This indicates that a careful choice of a search technique for SOS is important.

(p, e)	ARS	Hybrid
SOS with frontiers	$(0.021, 0.124)$	$(0.000912, 0.41)$

Table 6.10: *The p-value and the effect size e for the coordination strategies. This table indicates the p-value using Welch's t-test and the effect size e by comparing the results using our strategies with* 50 *solitary robots in the simulated search environment.*

6.3.4 The benefits of our pruning method in coordination of solitary robots

We also consider SOS with two different methods (Algorithms 8.3 and 8.2) that interacting solitary robots use to select a coordinator. Results are shown in Table 6.11. The maximum and distribution of interaction network sizes

Stellenbosch University https://scholar.sun.ac.za

(i.e. the number of robots involved in interactions) for different numbers of solitary robots are also given in Table 6.11 and Figure 6.13 respectively.

N	P	(p, e)
2	0	$(0.6856, 0.9367)$
8	2	$(0.8837, 0.0851)$
10	3	$(0.2873, -0.5449)$
50	11	$(0.0095, 0.5410)$
100	20	$(0.7813, 0.0397)$
1000	42	$(2.2232 \times 10^{-20}, 0.5068)$
2000	65	$(4.7325 \times 10^{-60}, 0.5259)$
3000	140	$(1.3359 \times 10^{-173}, 0.7500)$

Table 6.11: *The p-value and the effect size e for our coordination strategy (SOS) using two different leader election methods. This table indicates the p-value using Welch's t-test and the effect size e by comparing the results using our coordination strategy with at up to* $N = 3000$ *solitary robots in the simulated search environment. P denotes the maximum interaction network size for the corresponding total number of solitary robots N.*

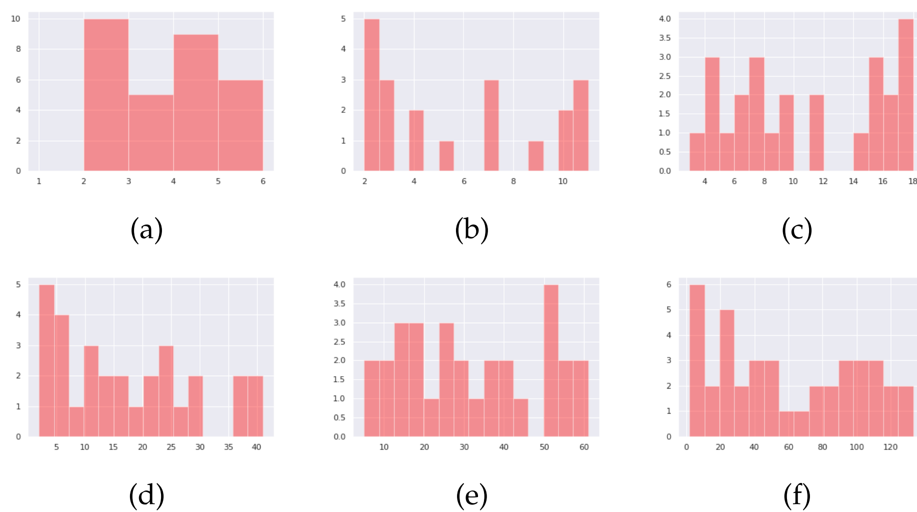


Figure 6.13: *Histograms of interaction network sizes observed in coordination of:* ***(6.13a)****:* 10, ***(6.13b)****:* 50, ***(6.13c)****:* 100, ***(6.13d)****:* 1000, ***(6.13e)****:* 2000 *and* ***(6.13f)****:* 3000, *solitary robots.*

Table 6.11 shows that there is no need for using our pruning method for leader election when small networks of solitary robots are considered. We see that for up to 100 solitary robots, the largest interaction network was about 20 robots. With such small interaction networks, one can simply use

Stellenbosch University https://scholar.sun.ac.za

the benchmark technique [118] for graph construction because our pruning method does not offer a significant advantage. But for larger networks of solitary robots (i.e. from 1000 to 3000 solitary robots), the interaction networks become larger (140 robots approximately). For these cases, the use of our pruning method for leader election has a substantial impact on the coverage performance of the robots as shown Table 6.11: because the larger the interaction network, the higher time required for view construction gets. Most large interaction networks are formed in initialisation of search: a large number of robots are found to start from nearby locations.

For these experiments, we set the duration of sending one message in interaction of robots to 0.0002, which is small. But for situations where the cost of sending a message is high, our pruning method will have a more substantial impact even on small networks. In other words, the higher the cost of sending a message or the larger the interaction network, the more substantial impact our pruning method has relative to the benchmark method for graph view construction.

6.4 Conclusion and avenues for future work

Our aim was to propose a coordination method with low interference and low interruptibility. Details of our contributions and their novelty can be found in Section 4.1.1.

This part contrasted three main coordination strategies, namely: ARS , PRS and SOS in terms of their applicability and the use of solitary robots.

PRS is used to manage a team of robots which can schedule meetings among themselves [42, 109]. In the case of solitary robots, a robot can encounter different robots at different times. We implemented a naive way of combining ARS and PRS for solitary robots (the Hybrid approach).

However, for situations such as map building where robots are bound by a team goal, ARS and SOS should not be recommended. When bound by a team goal, robots must maintain collaboration. Scheduled meetings are a sound strategy that solitary robots can apply to maintain collaboration among themselves. In ARS and SOS, collaboration of solitary robots is always accidental.

Stellenbosch University https://scholar.sun.ac.za

6.4.1 Findings summary

In our experiments we found that SOS provides more sustainable coverage performance than other methods in general. This indicates that bounding an exploration region based on the available known upper bound search time is important for coordination of solitary robots in an unknown environment.

We also found that the use of margins is beneficial to SOS. Since the nature of assignments (balanced or unbalanced) is not known in advance, using SOS with margins is recommended to provide robustness to unbalanced assignments.

Both the choice of search technique and coordination strategy have an impact on the performance of robots. We noted that, when using SOS, the search method should be selected carefully. The use of SOS with some search methods may not yield good performance.

Finally, we showed the impact of the cost of sending message in coordination of robots. We showed that a careful choice of a graph view construction technique for larger networks can be important in reducing the number of messages in large networks.

6.4.2 Avenues for future work

We identify the following promising avenues for future work:

1. Treatment of uncertainty in map fusion which is involved in coordination of solitary robots would be important to consider.

2. The proposed coordination strategy is a one-shot process (which means that robots only search in the environment once), applied in static environments. The case of dynamic environments where robots search iteratively could possibly be investigated in future.

3. For the assessment of our coordination method, we implemented a naive hybrid method which combines ARS and PRS for solitary robots. More sophisticated hybrid versions which consider different approaches to combining ARS and PRS could be investigated in future.

Stellenbosch University https://scholar.sun.ac.za

In this part we have covered our proposal for coordination of solitary robots. In the following part, we cover our proposal for the second problem considered in this thesis: view construction.

Stellenbosch University https://scholar.sun.ac.za

Part II

Network view construction

Stellenbosch University https://scholar.sun.ac.za

Chapter 7

Background and literature

In interaction of robots, a communication graph is a graph that indicates which robots can communicate with which other robots. Communication is considered here as the exchange (sending and receiving) of messages or information between adjacent nodes in a network. We are restricted to peer-to-peer communication because nodes have limited communication capacity. This chapter reviews background and literature for distributed graph view construction methods. Our contribution will be to enhance a distributed method for graph view construction by reducing the number of messages exchanged.

It should be noted that the ultimate goal of graph view construction is decentralised computation of node centrality quantities. In Chapter 1, for effective coordination during interaction of robots, we motivated the selection of a leader based on closeness centrality.

7.1 General background

In a distributed system, the process of building a view of a communication graph can be centralised or decentralised. It is centralised when the construction of the view of a communication graph is performed by a single node, known as an initiator. Once the initiator has the view of a communication graph, it may send it to the other nodes [77]. If the initiator is not known in advance, the first stage of constructing a view of a communication graph involves selecting the initiator.

Robustness to failure in the implementation of distributed systems is es-

Stellenbosch University https://scholar.sun.ac.za

sential in critical applications. We consider a decentralised solution because centralized solutions do not offer robustness to failure of a central node, and are seldom adequate due to computational and communication limitations [67]. A decentralised method to construct a view of a communication graph, on the other hand, is one in which each node builds its own view of their interaction network, with no single initiator [118]. We propose a decentralised method to construct a view of a communication graph. Since we consider an approximate and decentralised method to construct a view of a communication graph, inevitably nodes will construct different topologies describing their interaction network. We will use the term *view* as shorthand for view of a communication graph.

7.2 Literature discussion

Decentralised methods to construct views can be exact or approximate. Naz [77] reviews recent literature on methods to construct views. Exhaustive methods build the complete topology of the communication graph—they involve an all-pairs shortest path computation. As a consequence, exact approaches suffer from problems of scalability [77]. This can be a particular issue for large networks of nodes with constrained computational power and restricted memory resources.

Approximate methods have been proposed to overcome this problem. Unlike exhaustive methods, approximate methods do not provide complete knowledge of the communication graph. These methods provide only an approximate topology.

Many distributed methods for view construction are centralised, i.e. they require a single initiator in the process of network construction. The disadvantage of a unique initiator for approximate methods is that the structure of a view depends on the choice of the initiator. For example, Kim and Wu [47] propose an interesting distributed approach for view construction, where an initiator constructs a tree as the view of a communication graph for which it is the root.

To the best of our knowledge and according to Naz [77], decentralised approximate methods for view construction are very scarce and the algorithm proposed by You et al. [118] is the state of the art for decentralised approximate algorithms for view construction. This method simply runs

Stellenbosch University https://scholar.sun.ac.za

breadth-first search [92] on each node. Here we focus on reducing the number of messages received by nodes. A careful treatment of network communication for robotics search and rescue by low power robots under weak signal conditions is crucial [84, 103]. Weak signal conditions could be caused by the nature of the search environment or sensor conditions. A network of low power robots working under weak signal conditions can face high propagation time of communication.

For the comparison of our proposed method with You et al.'s method [118] in terms of the number of messages, we consider the total and maximum number of messages received per node. We wish to see the impact of our proposed mechanism on message complexity in the entire network. We wish to reduce the maximum number of messages per node because if the communication time per message is the bottleneck, reducing only the total number of messages may not be helpful.

We next show the decentralised construction of a view using the algorithm in You et al. [118].

7.2.1 Decentralised view construction

The method by You et al. was proposed for decentralised approximate computation of centrality measures (closeness, degree and betweenness centralities) of nodes in arbitrary networks. As treated in You et al.'s method [118], these computations require a limited view of the communication graph. At the end of the interaction between nodes, each node will evaluate its centrality based on its own view of the network (which may be different from other nodes' views). Here we consider connected, undirected and unweighted (i.e. edges of the communication graph are of unit weight) graphs of uniquely identifiable nodes.

In You et al.'s method [118], nodes construct a view of the communication graph only from local interactions. A node's view of a communication graph is gradually constructed by message passing. Each node sends its neighbour information (i.e. information about nodes it is aware of but not shared to its immediate neighbours yet) to all of its immediate neighbours which relay it onward through the network. Nodes apply asynchronous communication to build their views of the communication graph, i.e. there is no common clock signal between the sender and receiver.

Stellenbosch University https://scholar.sun.ac.za

Algorithm 7.1 *Our presentation of the benchmark method [118]. The algorithm gives the code run on node v_i. The main methods are* ONEHOP *and* UPDATE *which are run by v_i once each iteration. An example of code to execute the class is given in the procedure* RUNDISTRIBUTEDBFS.

```
1:  procedure RUNDISTRIBUTEDBFS()
2:      BFSobject←DISTRIBUTEDBFS(N_i, D)
3:      while not BFSobject.ISENDED() do
4:          BFSobject.ONEHOP()
5:          BFSobject.UPDATE()
6:      end while
7:  end procedure
8:
9:  class DISTRIBUTEDBFS
10:     Class variables
11:     D            the maximum number of iterations
12:     M_i          a queue used for messages received by the node v_i
13:     N_i          a list of immediate neighbours of the node v_i
14:     N_i^(t)      the set of (t+1)-hop neighbours of v_i
15:     N_{i,t}      the set nodes known by v_i until the end of iteration t
16:     t            the current iteration number
17: ------------------------------------------------------------
18:     constructor (N_i, D)
19:         t ← 0
20:         N_i^(0) ← {v_j : v_j ∈ N_i}
21:         D ← D
22:         N_{t,i} ← N_i^(0).clone()
23:     end constructor
24:
25:     procedure ONEHOP()
26:         if not isEnded() then
27:             for v_j ∈ N_i do
28:                 v_i sends ⟨NeighbouringMessage(i, N_i^(t−1))⟩ to v_j
29:             end for
30:         end if
31:     end procedure
32:
33:     procedure UPDATE()
34:         if not isEnded() then
35:             t ← t + 1
36:             N_i^(t) ← ∅
37:             while M_i.size() ≥ 1 do
38:                 (j, N_j^(t−1)) ← M_i.dequeue()
39:                 N_i^(t) ← N_i^(t) ∪ N_j^(t−1)        ▷ v_i fuses the messages received
40:             end while
41:             N_i^(t) ← N_i^(t) \ N_{t−1,i}
42:             N_{t,i} ← N_{t−1,i} ∪ N_i^(t)
43:         end if
44:     end procedure
45:
46:     function ISENDED()
47:         return N_i^(t) = ∅ or t = D
48:     end function
49: end class
```

Stellenbosch University https://scholar.sun.ac.za

Let $\mathcal{N}_i$ denote the set of neighbours of v_i and $\mathcal{N}_i^{(t)}$ the set of $(t+1)$-hop neighbours of v_i. The initial set of neighbours $\mathcal{N}_i^{(0)}$ is assumed to be known and has the following form:

$$\mathcal{N}_i^{(0)} = \{v_j : \; v_i \text{ is a one-hop neighbour of } v_j\} . \tag{7.2.1}$$

During each iteration, each node sends its neighbouring information to all its immediate neighbours. Each node waits for communication from all of its immediate neighbours after which it updates an internal round counter which indicates the distance between the node and the farthest nodes it knows about. The set $\mathcal{N}_j^{(t-1)}$ is what is being communicated from v_j to v_i after round t. The neighbouring message sent by a node v_j to its neighbours is $\langle \texttt{NeighbouringMessage(}j, \mathcal{N}_j^{(t-1)}\texttt{)}\rangle$. A node v_i stores messages received in a queue, represented by $\mathcal{M}_i$. The node queues messages based on their arrival times. After round t, the topology of v_i's view of the communication graph is updated as follows

$$\mathcal{N}_i^{(t)} = \bigcup_{v_j \in \mathcal{N}_i} \mathcal{N}_j^{(t-1)} \setminus \mathcal{N}_{i,t} , \tag{7.2.2}$$

where

$$\mathcal{N}_{i,t} = \bigcup_{k=0}^{t-1} \mathcal{N}_i^{(k)} . \tag{7.2.3}$$

The algorithm in You et al. [118] that each node v_i uses to construct its view of the communication graph is given in Algorithm 7.1.

The algorithm terminates after at most D iterations, where D is the input of the algorithm. It can also be set or determined in a distributed manner (i.e. the value of D can be determined by nodes during interaction) [35]. But some nodes can also reach their equilibrium stage before the D-th iteration (e.g. nodes which are more central than others). Such nodes should terminate when equilibrium is reached (see Line 47 in Algorithm 7.1).

At the end, every node has a view of network, and so the required centralities can be calculated locally. This view construction method is approximate when the total number of iterations D is less than the diameter of the communication graph, otherwise it is exhaustive, i.e. all views correspond to the exact correct information, assuming a failure-free scenario. With a decentralised approximate method as presented in Algorithm 7.1, nodes may have different views of the network at the end. Each node will evaluate its centrality based on the view of the network it has.

Stellenbosch University https://scholar.sun.ac.za

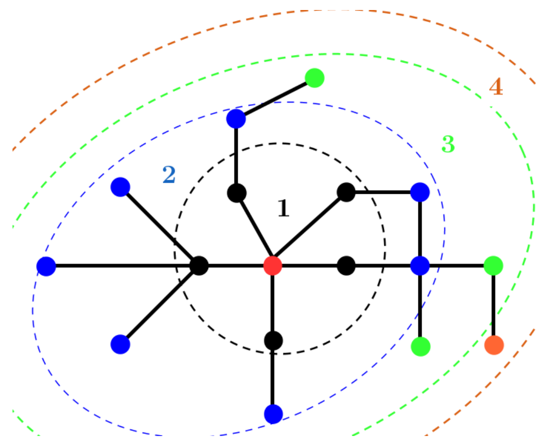


Figure 7.1: *An example of the construction of a view of a communication graph of diameter $d = 6$ using the method of You et al. [118]. The method is run on the red node: black nodes are its 1-hop neighbours, blue nodes are its 2-hop neighbours, green nodes are its 3-hop neighbours and pink node its 4-hop neighbour.*

This approach is similar to breadth-first search [92]: after round t, the view of a node v_i is the subset of the set of nodes of the communication graph containing nodes up to distance $t+1$ from itself. As with many other decentralised approaches, such as decentralised data fusion, we use propagation of information in neighbourhoods to make global estimates from local communication and computation [23].

In what follows, we illustrate Algorithm 7.1 with an example.

Example. Figures 7.1 and 7.2 illustrate the development of a view of the communication graph in Algorithm 7.1. Each node learns about its t-hop neighbours after iteration $t-1$. In Figure 7.1 the red node is initially aware of its one-hop neighbours (nodes in black). After the first iteration, it is aware of its two-hop neighbours (nodes in blue) and so on.

Figure 7.2 illustrates Algorithm 7.1 in a large communication graph with 250 nodes, 581 edges and a diameter of 56 (Figure 7.2a). The various subfigures show how the green node learns the communication graph.

In what follows, we illustrate Algorithm 7.1 with another example.

Example. Consider the communication graph in Figure 7.3, with $D = 3$. The results for the execution of Algorithm 7.1 at each node are shown in Table 7.1. At each iteration t, each node v_i learns its $(t+1)$-hop neighbours. At the end of the algorithm (i.e. after iteration $t = 3$), only the node v_3 has learnt the complete view of the communication graph.

Stellenbosch University https://scholar.sun.ac.za

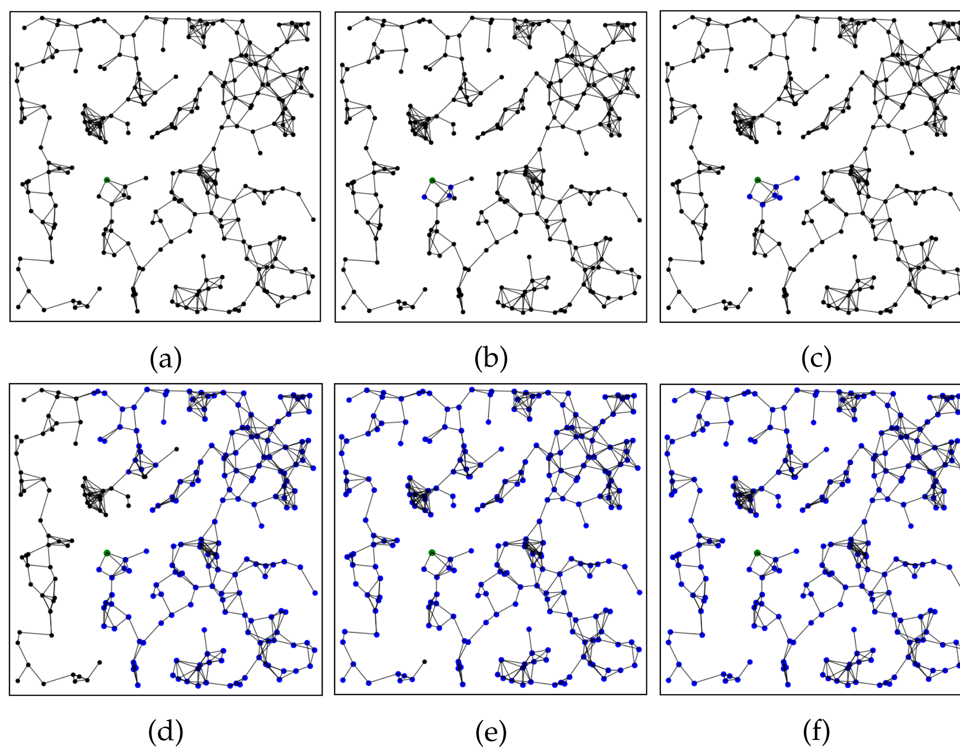


Figure 7.2: *An example of the benchmark method [118] showing the graph discovered by the green node (node on which the algorithm execution is being visualized) with a network of* 250 *nodes,* 581 *edges, and a diameter of* 56*. Blue nodes are nodes discovered by the green node.* ***(7.2a)****: The communication graph considered.* ***(7.2b)****: This illustrates the discovered subgraph after the first iteration of the algorithm.* ***(7.2c–7.2f)****: Later rounds up to round* 35*.*

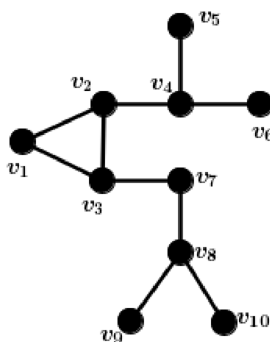


Figure 7.3: *Illustration of a communication graph.*

Stellenbosch University https://scholar.sun.ac.za

Discovery	$t=0$	$t=1$	$t=2$	$t=3$
$\mathcal{N}_1^{(t)}$	v_2, v_3	v_4, v_7	v_5, v_6, v_8	v_9, v_{10}
$\mathcal{N}_2^{(t)}$	v_1, v_3, v_4	v_5, v_6, v_7	v_8	v_9, v_{10}
$\mathcal{N}_3^{(t)}$	v_1, v_2, v_7	v_4, v_8	v_5, v_6, v_9, v_{10}	$\emptyset$
$\mathcal{N}_4^{(t)}$	v_2, v_5, v_6	v_1, v_3	v_7	v_8
$\mathcal{N}_5^{(t)}$	v_4	v_2, v_6	v_1, v_3	v_7
$\mathcal{N}_6^{(t)}$	v_4	v_2, v_5	v_1, v_3	v_7
$\mathcal{N}_7^{(t)}$	v_3, v_8	v_1, v_2, v_9, v_{10}	v_4	v_5, v_6
$\mathcal{N}_8^{(t)}$	v_7, v_9, v_{10}	v_3	v_1, v_2	v_4
$\mathcal{N}_9^{(t)}$	v_8	v_7, v_{10}	v_3	v_1, v_2
$\mathcal{N}_{10}^{(t)}$	v_8	v_7, v_9	v_3	v_1, v_2

Table 7.1: *Illustration of the use of Algorithm 7.1 applied on the graph in Figure 7.3.*

Let d_i denote the degree of node v_i. In Algorithm 7.1, the number of messages received by v_i is $\mathcal{O}(d_i D)$, and the average number of messages per node is $\mathcal{O}(\bar{d}D)$, where $\bar{d}$ is the average node degree of the communication graph.

We next review some existing methods for failure detection and handling of recovery from communication failures.

7.3 Background of methods for failure detection

Network communication is subject to failures: some nodes or edges could malfunction at any time. Communication is successful when messages sent are received uncorrupted, and ineffective communication is considered a failure. There are two main techniques involved in communication failures to enable reliable delivery of messages: error detection and error correction. The former is the detection of malfunctions of some entities (nodes or edges) in the network. The latter not only detects errors, but also reconstructs the original message to produce an error-free message. In this thesis, we consider error detection only, i.e. we can only detect failure of a functioning node/edge or recovery of a failed node/edge. We consider a technique to detect and recover from communication failures when applying our pruning method.

There are several solutions for communication failures [56, 81]. Based on architectural organization, solutions to communication failures are grouped into two categories: centralized and decentralized approaches [81]. Central-

Stellenbosch University https://scholar.sun.ac.za

ized solutions have a single node which detects and manages failures in the system. For example, Stoller [96] proposed a simple centralized solution for failure detection: it proposes implementing a failure detector (FD) to detect and report failures of nodes, and also to report the restoration of nodes when they recover. When the FD detects the failure of a node, it reports a down signal to other nodes. Note that Stoller [96] considers the FD as a separate module which monitors all the nodes. In this thesis, we are interested in decentralized solutions.

Kshirsagar and Jirapure [56] group decentralized solutions for management of communication failures into three categories:

- node self-detection approaches, where each node can identify its own malfunction by performing self-diagnosis;
- neighbour coordination approaches, where nodes coordinate with their immediate neighbours to detect malfunctioning nodes and report their recoveries; and
- clustering approaches, where the entire network is split into clusters and each cluster has a local coordinator responsible for performing tasks for failure management task.

From the three categories of decentralized approaches for failure detection, we find the neighbour coordination approach proposed by Sheth et al. [89] most suitable. The disadvantage of the node self-detection approach is that a node is not aware of failure of other nodes in the system. This approach may be suitable for monitoring system where one needs to study, for example, the impact of failure of a node in the entire system without propagating such information to other components of the system. The clustering approach is quite similar to the neighbour coordination approach, but the former is inappropriate for our use since it requires that the network be split in advance and that the local coordinators be known to other cluster members. In the neighbour coordination approach, each node can monitor the behaviour of its immediate neighbours, which suits our decentralised approach well.

Two situations can cause communication failure in our case. First, a node can fail to send a message. Second, a communication channel (i.e. an edge) can fail to (fully) transmit a message. A node or an edge is said to be down

Stellenbosch University https://scholar.sun.ac.za

when it fails to transmit messages/information. For our purposes, a node will only need to know the states of nodes and edges in its neighbourhood.

7.4 Summary

This chapter reviewed background work and literature for decentralised distributed communication graph construction. According to discussions by Naz [77], there has not been much work done in this direction. Most methods for graph construction are centralised, i.e. there exists, or the participating nodes choose, one node to be the initiator. Based on the nature of solitary robots for search and rescue which we consider in this thesis, and the need for robustness to failures, we were interested in developing a decentralised method for graph construction. To this end, we described an existing method, proposed in You et al. [118], which, based on our knowledge, seems to be the leading decentralised distributed method for view construction. In this method, each node effectively runs breadth-first search [92] to construct its graph view.

While research has been conducted in the area of communication graph construction, there is still room for improvement for this area. This thesis makes contributions by refining an existing approach [118], providing a favourable tradeoff for the problem that we consider.

The next chapter presents our proposed pruning method.

Stellenbosch University https://scholar.sun.ac.za

Chapter 8

Decentralised view construction

8.1 Introduction

In distributed networks, the topology of a network is very important, especially when one needs to know a central node in a network. Chapter 7 has reviewed background and literature for distributed views construction of a communication graph. This chapter introduces our proposed distributed methods for view construction with a reduced number of messages.

Each node builds a view of the communication graph to evaluate its closeness centrality. We consider applications where we care about nodes with high closeness centrality (e.g. leader election based on a closeness centrality measure as discussed in Subsection 5.3).

8.1.1 Contributions

Not all nodes ultimately need to build a view of the network formed in their interaction since some nodes may realize quickly that they are not suitably central, i.e. they have small closeness centralities. The question is how to identify such insignificant nodes in a decentralised algorithm? This work tackles this problem by introducing a pruning strategy that also reduces the number of messages exchanged between nodes compared to the algorithm from You et al. [118]. When a leader is chosen based on closeness centrality, we observe that in the algorithm of You et al. [118], even nodes which can not play a central role, for example leaves, overload communication by receiving messages. (Except in special cases, a leaf node should not play an important role).

Stellenbosch University https://scholar.sun.ac.za

At each iteration of view construction, each node prunes some nodes in its neighbourhood once and thereafter interacts only with the unpruned nodes to construct its view of the communication graph. We refer to this approach as pruning. The more of these prunable nodes a network contains the better our algorithm performs relative to the algorithm proposed by You et al. [118] in terms of number of messages. We further modify the algorithm to take communication failure into account (and analyse the impact of such failures on performance of the algorithm).

8.1.2 Limitations and potential applications of our proposed method

Our proposed method is recommended for situations where only one node is meant to be selected as a central node. It is not recommended for situations where one wishes to know the approximated centralities of all nodes of the network.

Our proposed method is not limited to interaction networks for solitary robots, but can be applied to arbitrary distributed communication networks, and is most likely to be valuable when nodes form very large networks: reducing the number of messages will be a more pressing concern in large-scale networks. Such applications include driverless or instrumented cars, monitoring systems, swarm robotics, mobile sensor networks or general mobile ad-hoc networks.

8.2 View construction

The idea behind our pruning technique is that, during view construction, some nodes can be deactivated (i.e. some nodes will no longer communicate with other nodes). Nodes put on hold are not involved in subsequent steps of the algorithm and their closeness centralities [8] are treated as zero.

In You et al.'s method [118], at iteration t, a node learns of nodes at distance $t + 1$ from itself, and it initially knows its immediate neighbours. Pruning is applied after each iteration t of communication, i.e., pruning is applied after every iteration a node is aware of its $(t + 1)$-hop neighbour information. At that stage, each node checks whether it or any nodes in its one-hop neighbourhood should be put on hold. It is desirable that nodes

Stellenbosch University https://scholar.sun.ac.za

can determine which other nodes in their neighbourhood are deactivated so that they do not need to wait for or send messages to them. This reduces the number of messages exchanged between nodes.

Given two immediate neighbours v_i and v_j, their sets of nodes known to be pruned are not necessarily the same, and nodes do not exchange such information between themselves.

In what follows we describe our pruning method to construct a view.

8.2.1 Pruning

Before describing our proposed pruning method, we note some changes we make to the benchmark method [118].

8.2.1.1 Replacement of the set of nodes $\mathcal{N}_i^{(t)}$ by a set of edges $\mathcal{S}_i^{(t)}$

In the method proposed by You et al. [118], each node v_i keeps record of new nodes $\mathcal{N}_i^{(t)}$ it discovers at each round t. We find this way to be inconsistent, specifically in situations where nodes can fail to communicate, which we will consider in this thesis. (Note that You et al. [118] did not treat communication failure.) In situations where nodes only keep record of new nodes they have discovered, when some nodes fail, other nodes may not construct their views of the communication graph consistently as a node needs to know which nodes are immediate neighbours. Unlike in the method proposed by You et al. [118], here each node keeps record of new edges it has discovered at each round. In other words, each node knows which of the nodes it has discovered are immediate neighbours. So, in case of failures of some nodes during the process, nodes will construct their views of the communication graph consistently.

An edge between nodes v_i and v_j is denoted by

$$e_{ij} = e_{ji} = (\min(i,j), \max(i,j))\,.$$

We will use $\mathcal{S}_i^{(t)}$ to denote the set of new edges discovered by v_i after round t. The initial set of edges $\mathcal{S}_i^{(0)}$ has the following form:

$$\mathcal{S}_i^{(0)} = \{e_{ij} :\ v_i \text{ is a one-hop neighbour to } v_j\}\,. \tag{8.2.1}$$

Stellenbosch University https://scholar.sun.ac.za

After round t, the topology of v_i's view of the communication graph is updated as follows

$$\mathcal{S}_i^{(t)} = \bigcup_{v_j \in \mathcal{N}_i} \mathcal{S}_j^{(t-1)} \setminus \mathcal{S}_{i,t}\,, \tag{8.2.2}$$

where

$$\mathcal{S}_{i,t} = \bigcup_{k=0}^{t-1} \mathcal{S}_i^{(k)}\,. \tag{8.2.3}$$

8.2.1.2 Description

It is worth to recall that we are not aiming to compute exact closeness centrality distribution on a graph. We aim to propose a method for approximation of closeness centrality values. While we are aiming to compute closeness centrality distribution on a graph using pruning, the concept of pruning can directly be related to eccentricity centrality [40]. The eccentricity of a node is the maximum distance between the node and another node. Eccentricity and eccentricity centrality are reciprocal to each other, i.e. nodes with high eccentricity have low eccentricity centrality. We consider the following points to achieve our goal.

- Pruned nodes have relatively high eccentricities or low eccentricity centralities (as will be discussed later in Lemma 8.2.3).

- It was found that eccentricity and closeness centralities are highly and positively correlated for various types of graphs [7, 73]. This is partly due to the fact that they both operate on the concepts of paths.

From what precedes, our proposed pruning method is then recommended for approximations of closeness centralities for categories of graphs where eccentricity and closeness centralities are highly correlated.

We will first show how a node identifies prunable nodes after each iteration. There are two types of objects (leaves, and nodes causing triangles) that a node can prune after the first iteration. When a node is found to be a leaf or to cause a triangle, this node is put on hold by itself and each of its immediate neighbours. Let d_i denote the degree of node v_i. Since nodes have learnt about their 2-hop neighbours by the first iteration, a node v_i knows the neighbours $\mathcal{N}_j$ of each of its immediate neighbours v_j.

Definition 8.2.1 (A leaf)**.** *A node v_i is a leaf if $d_i = 1$.*

Stellenbosch University https://scholar.sun.ac.za

Definition 8.2.2 (A node causing a triangle)**.** *A non-leaf node v_j causes a triangle if $d_j = 2$ and its two immediate neighbours are non-leaves and also immediate neighbours to each other.*

Let $\mathcal{F}_i^{(t)}$ denote the set of pruned nodes known by a node v_i at the end of iteration t (with $t \geq 1$).

We can illustrate elements of $\mathcal{F}_i^{(1)}$ in Figure 8.1. For example in Figure 8.1, from the perspective of v_3, nodes v_4 and v_5 are leaves. Also, nodes v_{11} and v_{14} are nodes causing a triangle from the perspective of v_{10}.

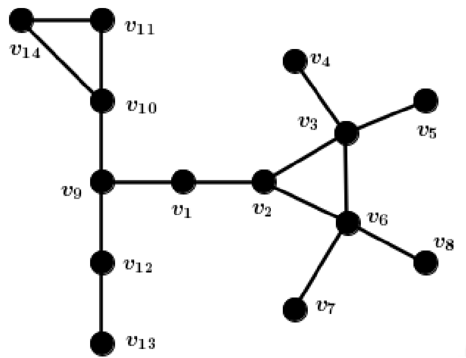


Figure 8.1: *Example of discovery of prunable objects.*

Figure 8.2 illustrates another case of nodes causing triangles. For example, Figure 8.2 has a triangle between the nodes v_1, v_2 and v_3. From the perspective of each of v_1, v_2 and v_3 (using Definition 8.2.2), v_1 causes the triangle. So each of these three nodes will put v_1 on hold.

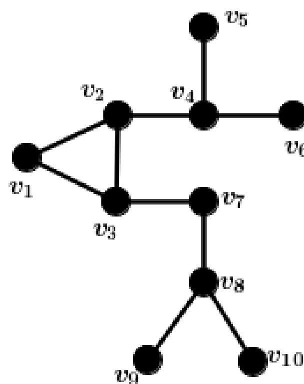


Figure 8.2: *Illustration of triangles (i.e. cycles of size 3) in pruning method. A triangle exists between nodes v_1 v_2 and v_3.*

Stellenbosch University https://scholar.sun.ac.za

Our proposed pruning method is decentralised, so each node is responsible for identifying prunable nodes in its neighbourhood, including itself.

Let $\mathcal{N}_{i,t}^{up}$ denote the set of neighbours of v_i which are still involved in interaction at the beginning of iteration t, i.e. nodes that have not been pruned. Functions that a node v_i applies to detect prunable elements in its neighbourhood after the first iteration are given in Algorithm 8.1.

For graphs of diameter 1 (i.e. a complete graph), pruning is not involved because, after the first iteration each node will realise that it its current view of the communication graph is complete.

So far, we have described how elements of $\mathcal{F}_i^{(1)}$ are identified by node v_i. We now consider the case of further pruning which is straightforward: new nodes should be put on hold when they have no new information to share with their other active neighbours. Thus a node stops relaying neighbouring information to neighbours from which it receives no new information.

At the end of iteration t, a node v_i considers itself as element of $\mathcal{F}_i^{(t)}$ (i.e. prunes itself) if it gets all its new information from only one of its neighbours at that iteration. Also, node v_i prunes v_j if the neighbouring information $\mathcal{S}_j^{(t)}$ sent by v_j to v_i does not contain new information, i.e.

$$\mathcal{S}_j^{(t)} \subseteq \bigcup_{l<t} \mathcal{S}_i^{(l)} . \tag{8.2.4}$$

At the end of our pruning method, we hope that the most central node is among nodes which are unpruned. Equation 8.2.4 indicates for a node v_j to be pruned, there must be another node v_i which is unpruned because a comparison must be done. This is true because there are always unpruned nodes which remain after the first iteration, except a graph of 2 connected nodes in which case pruning is not involved. This means that at the end of our pruning method, there will always remain some unpruned nodes.

Nodes in $\mathcal{F}_i^{(t)}$ for $t \geq 2$ could be viewed as leaves in the subgraph obtained after the removal of all pruned nodes from previous iterations as they can have only one unpruned node remaining. Recall that an unpruned node only interacts with its neighbours which are not deactivated. This means that the number of the unpruned neighbours of a node may get reduced over iterations. So at some iteration an unpruned node can be viewed as leaf if it remains only with one unpruned neighbour.

Stellenbosch University https://scholar.sun.ac.za

Let

$$\mathcal{F}_{i,t} = \bigcup_{l=1}^{t} \mathcal{F}_i^{(l)} .$$

After describing pruning, we now connect it to eccentricity (see Lemma 8.2.3) as mentioned above. Let ecc_i denote the eccentricity of a node v_i and recall that δ_{ij} denotes the shortest path length between the nodes v_i and v_j in an (unweighted) graph G with vertex set $\mathcal{V}$. The eccentricity of a node is given by

$$\mathrm{ecc}_i = \max_{v_j \in \mathcal{V}} \delta_{ij} . \tag{8.2.5}$$

Lemma 8.2.3. *If $v_j \in \mathcal{N}_i$ such that*

$$v_j \in \bigcup_{t \geq 1} \mathcal{F}_i^{(t)} ,$$

then

$$\mathrm{ecc}_j \geq \mathrm{ecc}_i .$$

Proof. Recall that $\mathcal{N}_i^{(t)}$ denotes the set of $(t+1)$-hop neighbours of v_i. According to Definitions 8.2.1 and 8.2.2, and Equation 8.2.4, a node v_j is pruned if it has an immediate neighbour v_i from which it can learn more about the graph (i.e. it can receive new information) while at the same time it can not provide new information to that neighbour. Given two direct neighbours v_j and v_i where v_j has been pruned at iteration t by v_i, we have

$$\mathcal{N}_j^{(t)} \subseteq \bigcup_{l=1}^{t} \mathcal{N}_i^{(l)} . \tag{8.2.6}$$

From the definition of $\mathcal{N}_i^{(t)}$ and Equation 8.2.6, it is straightforward that $\mathrm{ecc}_j \geq \mathrm{ecc}_i$. □

From Lemma 8.2.3 it can be seen that prunable nodes are nodes with relatively high eccentricities. So pruning can not introduce errors when searching for a node of maximum eccentricity centrality. Since eccentricity and closeness centralities are highly correlated [7, 73], it can be concluded that prunable nodes are likely to be nodes with relatively low closeness centrality.

Stellenbosch University https://scholar.sun.ac.za

The procedure for how a node v_i detects elements of $\mathcal{F}_i^{(t)}$ at the end of each iteration $t \geq 2$ is given in Algorithm 8.1 (see function FURTHERPRUNINGDETECTION in Algorithm 8.1). A Python implementation of how prunable nodes are detected in a distributed manner can be found in Appendix G.1.

The algorithm requires one parameter—the maximum number of iterations, D—and returns the view $\mathcal{S}_{i,t}$ of node v_i and the set $\mathcal{F}_{i,t}$ of pruned nodes known by v_i.

Example. Consider the communication graph in Figure 8.2, with $D = 4$. The results of our pruning method on this graph are as follows.

Node sets	ecc$_i$	$t = 1$	$t = 2$	$t = 3$	$t = 4$
$\mathcal{F}_1^{(t)}$	4	v_1	$\perp$	$\perp$	$\perp$
$\mathcal{F}_2^{(t)}$	4	v_1	v_4	v_2	$\perp$
$\mathcal{F}_3^{(t)}$	3	v_1	$\emptyset$	v_2, v_7	$\top$
$\mathcal{F}_4^{(t)}$	5	v_5, v_6	v_4	$\perp$	$\perp$
$\mathcal{F}_5^{(t)}$	6	v_5	$\perp$	$\perp$	$\perp$
$\mathcal{F}_6^{(t)}$	6	v_6	$\perp$	$\perp$	$\perp$
$\mathcal{F}_7^{(t)}$	4	$\emptyset$	v_8	v_7	$\perp$
$\mathcal{F}_8^{(t)}$	5	v_9, v_{10}	v_8	$\perp$	$\perp$
$\mathcal{F}_9^{(t)}$	6	v_9	$\perp$	$\perp$	$\perp$
$\mathcal{F}_{10}^{(t)}$	6	v_{10}	$\perp$	$\perp$	$\perp$

Table 8.1: *Table indicating pruning using the graph in Figure 8.2. $\perp$ indicates that the corresponding node has put itself on hold and $\top$ indicates that the node has reached an equilibrium.*

After the first iteration, each node must identify prunable nodes in its neighbourhood, including itself. Using Algorithm 8.1, $\mathcal{F}_1^{(1)} = \{v_1\}$, so v_1 will put itself on hold. Also, $\mathcal{F}_2^{(1)} = \mathcal{F}_3^{(1)} = \{v_1\}$, so v_2 and v_3 will put v_1 on hold. The same procedure applies for other nodes (see the third column in Table 8.1). At the first iteration, all the leaves (i.e. v_5, v_6, v_9 and v_{10}) are pruned; they all have ecc$_i = 6$. But, a non-leaf node, v_1 (with ecc$_i = 4$), is also pruned on the first iteration. As was mentioned before, the node v_1 is pruned now because it causes a triangle.

At the beginning of iteration $t = 2$, v_1, v_5, v_6, v_9 and v_{10} are no longer involved since they have been identified as prunable nodes at the end of the previous iteration; $\mathcal{F}_2^{(2)} = \mathcal{F}_4^{(2)} = \{v_4\}$ and $\mathcal{F}_7^{(2)} = \mathcal{F}_8^{(2)} = \{v_8\}$; and the

Stellenbosch University https://scholar.sun.ac.za

Algorithm 8.1 *Our proposed pruning method in a failure-free scenario. The main methods are* NEXTONEHOP *and* NEXTUPDATE *which are run by v_i once each iteration. The algorithm gives the code executed for node v_i. An example of code to execute the class is given in the procedure* RUNPRUNING.

```
1:  procedure RUNPRUNING()
2:      Pruningobject ← PRUNING(N_i, D)
3:      Pruningobject.INITIALONEHOP()
4:      Pruningobject.INITIALUDATE()
5:      Pruningobject.FIRSTPRUNINGDETECTION()
6:      while not Pruningobject.ISENDED() do
7:          Pruningobject.NEXTONEHOP()
8:          Pruningobject.NEXTUPDATE()
9:      end while
10: end procedure
11:
12: class PRUNING
13:     Class variables
14:     F_i^(t)        set of pruned nodes known by v_i at the end of iteration t
15:     F_{i,t}        set of pruned nodes known by v_i up to iteration t
16:     M_i            a message queue for neighbouring messages received by the node
17:     N_i            set of immediate neighbours of v_i
18:     N_{i,t}^up     set of immediate neighbours of v_i which are still active up to iteration t
19:     Q_ij           the one-hop neighbours of node v_j sent to v_i at the first iteration
20:     S_i^(t)        set of new edges discovered by v_i at the end of iteration t
21:     S_{i,t}        view of communication graph of v_i up to iteration t
22:     t              current iteration number
23: ------------------------------------------------------------------------
24:     constructor (N_i, D)
25:         S_i^(0) ← {e_ij : ∀v_j ∈ N_i}
26:         S_{i,0} ← S_i^(0).clone()          ▷ v_i detects its immediate neighbours
27:     end constructor
28:
29:     procedure NEXTONEHOP()
30:         if not ISENDED() then
31:             t ← t + 1
32:             N_{i,t}^up ← N_{i,t}^up \ F_{i,t}
33:             S_i^(t) ← ∅
34:             for v_j ∈ N_{i,t}^up do
35:                 v_i sends ⟨NeighbouringMessage(i, S_i^(t-1))⟩ to v_j
36:             end for
37:         end if
38:     end procedure
39:
40:     procedure NEXTUPDATE()
41:         if not ISENDED() then
42:             while M_i.size() ≥ 1 do
43:                 (j, S_j^(t-1)) ← M_i.dequeue()
44:                 S_i^(t) ← S_i^(t) ∪ S_j^(t-1)          ▷ v_i fuses the messages received
45:             end while
46:             S_i^(t) ← S_i^(t) \ S_{t-1,i}
47:             S_{i,t} ← S_{i,t-1} ∪ S_i^(t)
48:             F_i^(t) ← FURTHERPRUNINGDETECTION()
49:             F_{i,t} ← F_{i,t} ∪ F_i^(t)
50:         end if
51:     end procedure
52:
53:     function INITIALONEHOP()
54:         for v_j ∈ N_i do
55:             v_i sends ⟨NeighbouringMessage(i, S_i^(0))⟩ to v_j
56:         end for
57:     end function
```

Stellenbosch University https://scholar.sun.ac.za

58: **procedure** INITIALUPDATE()
59: $\mathcal{S}_i^{(1)} \leftarrow \emptyset$
60: **while** $\mathcal{M}_i$`.size()` ≥ 1 **do**
61: $(j, \mathcal{S}_j^{(0)}) \leftarrow \mathcal{M}_i$`.dequeue()`
62: $\mathcal{S}_i^{(1)} \leftarrow \mathcal{S}_i^{(1)} \cup \mathcal{S}_j^{(0)}$ ▷ v_i fuses the messages received
63: $\mathcal{Q}_{ij} \leftarrow \mathcal{S}_j^{(0)}$
64: **end while**
65: $\mathcal{S}_i^{(1)} \leftarrow \mathcal{S}_i^{(1)} \setminus \mathcal{S}_{i,0}$
66: $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,0} \cup \mathcal{S}_i^{(t)}$
67: **end procedure**
68:
69: **function** LEAVESDETECTION() ▷ v_i detects leaves in its neighbourhood
70: $\mathcal{F}_i^{(1)} \leftarrow \emptyset$
71: **for** $v_j \in \mathcal{N}_i \cup \{v_i\}$ **do**
72: **if** $|\mathcal{Q}_{ij}| = 1$ **then**
73: $\mathcal{F}_i^{(1)} \leftarrow \mathcal{F}_i^{(1)} \cup \{v_j\}$
74: **end if**
75: **end for**
76: **return** $\mathcal{F}_i^{(1)}$
77: **end function**
78:
79: **function** TRIANGLEDETECTION($\mathcal{F}_i^{(1)}$) ▷ v_i detects elements causing triangles in its neighbourhood
80: **for** $v_j \in \mathcal{N}_{i,1}^{up} \cup \{v_i\}$ **do**
81: **if** $|\mathcal{Q}_{ij}| = 2$ **then**
82: Let v_f, v_g be the two immediate neighbours of v_j
83: **if** v_f is a node in $\mathcal{Q}_{ig}$ **then**
84: $\mathcal{F}_i^{(1)} \leftarrow \mathcal{F}_i^{(1)} \cup \{v_j\}$
85: **end if**
86: **end if**
87: **end for**
88: **return** $\mathcal{F}_i^{(1)}$
89: **end function**
90:
91: **procedure** FIRSTPRUNINGDETECTION()
92: $\mathcal{F}_i^{(t)} \leftarrow$ LEAVESDETECTION()
93: $\mathcal{F}_i^{(t)} \leftarrow$ TRIANGLEDETECTION($\mathcal{F}_i^{(t)}$)
94: $\mathcal{F}_{i,t} \leftarrow \mathcal{F}_i^{(t)}$`.clone()`
95: **end procedure**
96:
97: **function** FURTHERPRUNINGDETECTION() ▷ v_i detects elements of $\mathcal{F}_i^{(t)}$ in its neighbourhood
98: $\mathcal{F}_i^{(t)} \leftarrow \emptyset$
99: **for** $v_j \in \mathcal{N}_{i,t}^{up}$ **do**
100: **if** $\mathcal{S}_j^{(t)} \subseteq \mathcal{S}_{i,t-1}$ **then**
101: $\mathcal{F}_i^{(t)} \leftarrow \mathcal{F}_i^{(t)} \cup \{v_j\}$
102: **end if**
103: **end for**
104: **if** $|\mathcal{N}_{i,t}^{up}| = 1$ **and** $\mathcal{S}_i^{(t)} \neq \emptyset$ **then**
105: $\mathcal{F}_i^{(t)} \leftarrow \mathcal{F}_i^{(t)} \cup \{v_i\}$
106: **end if**
107: **return** $\mathcal{F}_i^{(t)}$
108: **end function**
109:
110: **function** ISENDED()
111: **return** $t = D$ **or** $\mathcal{S}_i^{(t)} = \emptyset$ **or** $v_i \in \mathcal{F}_{i,t}$
112: **end function**
113: **end class**

Stellenbosch University https://scholar.sun.ac.za

node v_3 has not identified any prunable node at this iteration. At this iteration, the remaining nodes with high ecc_i (i.e. nodes v_4 and v_8) are pruned. The results for the remaining iterations are shown in Table 8.1.

A detailed toy example of pruning is illustrated in Appendix C.

We are aware that starvation or deadlock [20] are situations which must be addressed in distributed systems. In our proposed distributed system, at the end of each iteration each node is aware of its immediate neighbour's status (whether they are pruned or not). A pruned node neither sends a message nor waits for a message. So when a node is still unpruned, it knows which immediate neighbours to send messages to and which to wait messages from. This prevents the nodes from suffering from starvation or deadlock.

Having described the pruning method, we next evaluate the amount of communication of the pruning method we have described. This complexity analysis considers algorithms without failure detection.

8.3 Communication analysis

In this section we analyse the amount of communication required by our proposed pruning method, against the benchmark technique [118]. Let d_i and $u_i^{(t)}$ denote the number of neighbours of v_i and the number of neighbours of v_i which are pruned at the end of iteration t respectively. Let $Y_i^{(D)}$ and $P_i^{(D)}$ be the number of messages that the node v_i receives according to the benchmark algorithm (Algorithm 8.3) [118] and the number of messages v_i receives through the use of the pruning strategy for D rounds respectively. Then the number of messages $\Delta_i^{(D)}$ the node v_i saves due to the use of a pruning strategy (Algorithm 8.2) is

$$\Delta_i^{(D)} = Y_i^{(D)} - P_i^{(D)} \,. \tag{8.3.1}$$

We expect $\Delta_i^{(D)} \geq 0$ for all $v_i \in \mathcal{V}$. We consider situations with the number of iterations $D \geq 2$ since, our proposed algorithm and the algorithm in [118] give the same number of messages for $D = 1$, for failure-free cases.

The maximum number of iterations to terminate the algorithm is D and a node v_i is supposed to receive d_i messages at the end of each iteration t using the benchmark method. Recall that our proposed pruning and the

Stellenbosch University https://scholar.sun.ac.za

benchmark methods can also terminate when an equilibrium is reached. Let H_i denote the iteration after which a node v_i applying the benchmark and our proposed methods reaches an equilibrium. The value of H_i is the same for both algorithms because at iteration t, a node v_i (which should be an unpruned node using our proposed method) has the same view (i.e. it has learnt up to $(t+1)$-hop neighbours) using both algorithms. Note that for our pruning method, a pruned node does not reach an equilibrium. Let $h_i^{(t)}$ be the number of neighbours of v_i which have reached equilibrium at iteration t. It is clear that if there exists at least one neighbour of v_i which has reached equilibrium at iteration t, then v_i will reach its equilibrium at iteration $t+1$. For the benchmark method,

$$Y_i^{(D)} = \sum_{t=1}^{\min(H_i,D)} d_i - h_i^{(H_i-1)} \;\; \forall v_i \in \mathcal{V}\,. \tag{8.3.2}$$

In the following, we present results showing how our proposed pruning method decreases the number of messages exchanged between nodes while communicating in a failure-free scenario.

Lemma 8.3.1. *Let L_i denote the round at which a node v_i is put on hold ($L_i = +\infty$ for unpruned node v_i). The number of messages received by a node $v_i \in \mathcal{V}$ is*

$$P_i^{(D)} = \sum_{t=1}^{\min(D,H_i)} (d_i - u_i^{(t-1)}) - h_i^{(H_i-1)}\,,$$

for unpruned nodes and

$$P_i^{(D)} = \sum_{t=1}^{T_i} (d_i - u_i^{(t-1)})\,,$$

for pruned nodes where

$$T_i = \min(D, L_i)\,.$$

Proof. After any round of pruning, the number of messages received by v_i in the next round is reduced depending on the number of its neighbours put on hold at that round of pruning. So at the end of each iteration t, the node v_i receives $(d_i - u_i^{(t-1)})$ messages. Note that $u_i^{(0)} = 0$. Also a node can stop interacting with other nodes after it is put on hold. If the node v_i is put on hold at the end of iteration $T_i \leq D$, then it stops receiving messages. Also at iteration H_i, an unpruned node v_i does not receive messages from its direct neighbours which have reached equilibrium at iteration $H_i - 1$. □

Stellenbosch University https://scholar.sun.ac.za

Theorem 8.3.2. *Let L_i denote the round at which a node v_i is put on hold (with $L_i = +\infty$ for an unpruned node v_i). The number of messages saved by a node $v_i \in \mathcal{V}$ in a failure-free scenario is*

$$\Delta_i^{(D)} = \sum_{t=2}^{T_i} u_i^{(t-1)} + \sum_{t=T_i+1}^{\min(H_i,D)} d_i + h_i^{(H_i-1)}.$$

Proof. This is straightforward by Equations 8.3.1 and 8.3.2, and Lemma 8.3.1. □

It is clear that the pruning method requires more computation than the benchmark method in order to detect nodes to prune. Also pruned nodes build very limited views of a communication graph as they stop interacting with others once they are pruned. These nodes would have built broader views using the benchmark method [118]. Thus if considering applications where all nodes are required to build broader views of the communication graph, our pruning method is not recommended.

Having described the pruning method for failure-free cases and evaluated the amount of communication of the pruning method, we next describe the pruning method for failure cases.

8.4 Communication failure

So far we have introduced a pruning method for constructing a view of a communication graph, with an advantageous characteristic: pruning should reduce the number of messages exchanged between nodes during view construction. But network communication is subject to failures (as discussed in Section 7.3): we assumed that pruned nodes can not fail when building network views using our proposed method.

We next describe the approach (Algorithm 8.2) we propose for failure detection. Our proposed method is a neighbour coordination approach [89]: each node monitors the behaviour of its immediate neighbours, which suits our decentralised approach well (as motivated in Section 7.3).

8.4.1 Neighbour coordination approach

In our case, each node will have an FD component to monitor its immediate neighbours or incident edges in order to detect and report their failures. It

Stellenbosch University https://scholar.sun.ac.za

also reports when failed nodes and edges recover. In other words, a node is monitored by all its immediate neighbours and an edge is monitored by both nodes that the edge connects. There are two ways a node can detect failure of its neighbour or an incident edge. A node v_i considers another node v_j in its neighbourhood down if v_i does not receive a message from v_j for T units of time (the variable T is user-defined), unless the node v_j is on hold. It is assumed that any other nodes, say v_k, than the failed node v_j assigns the same T to v_j. We use checksums in our various algorithms to detect failure in communication channels. In our algorithms, the checksum is a boolean function: it returns false if no errors were detected during transmission, otherwise it returns true.

The value of T depends on the maximum number of iterations. For a typical value of D, T can be set to 1 second for D iterations corresponding to D seconds (i.e. 1 second corresponds to 1 iteration), as done in our experiments. We choose 1 second because each node is supposed to receive information from its neighbours at the end of each iteration (i.e. every second).

To inform its immediate neighbours about the failure of a node or the failure of a channel, the node v_i sends a down signal to its immediate neighbours. Nodes relay these signals further through the network. This approach can lead to loops of failure signals. To avoid such loops, each signal is uniquely identified by the FD nodes. Every node stores failure signals it has received, which it removes when it receives a recovery signal corresponding to that failure signal. In this way, a node relays a failure signal only if such signal is absent in its storage memory under the assumption that a node or an edge only fails once per iteration (this is checked using the subroutine ISPERSISTENT in Algorithm 8.2). This means that if a non-neighbour of a failed node receives multiple failure signals of that failed node, it will only relay the first failure signal. Also a node sends or relays a failure signal to a neighbour only after receiving a message from it. In this way, loop avoidance will be reinforced. The format of a failure signal is $\langle \texttt{Signal}(S, Q, i, c) \rangle$ to denote a signal concerning c (c is either an edge or a node) of type S and Q (S and Q are used to indicate whether it is a failure signal or a recovery signal) is sent from the node v_i. For failure of a node v_j, the signal sent by v_i is $\langle \texttt{Signal}(0, 0, i, j) \rangle$. For failure of an edge e_{jk}, the signal sent by v_i is $\langle \texttt{Signal}(0, 1, i, e_{jk}) \rangle$.

Stellenbosch University https://scholar.sun.ac.za

If some node is down, other nodes keep exchanging messages. Whatever its condition (failed or not), an edge is always used by both the nodes incident to it. In other words, if an edge is down, both the nodes incident to it keep exchanging messages through the failed channel until a possible recovery of the failed edge (i.e. until a message is successfully received)—a better approach might be to poll the connection periodically, after increasing intervals. Otherwise, they will never know the recovery of the edge when that happens. If there is still some failed nodes or edges which have not recovered by the end of view construction, functioning nodes shall remove them from their final views (see Line 161 in Algorithm 8.2). A failed node or edge can speed up or delay the communication process between nodes. That is because the removal of a node or edge from a network can change path lengths between remaining nodes (Figure 8.3) and thus the complexity of the process of constructing a view.

When a failed node or edge is restored, the FD nodes send a restoration signal to their immediate neighbours, for relaying through the network. A failed node is considered restored when the FD node receives a message from it. Also a failed edge is considered restored when it is successful. For restoration of a node v_j, the signal sent by node v_i is $\langle \texttt{Signal}(1, 0, i, j) \rangle$. For restoration of an edge e_{jk}, the signal sent by node v_i is $\langle \texttt{Signal}(1, 1, i, e_{jk}) \rangle$. A node that recovers should simply carry on sending neighbouring messages. A node stores failure signals in a queue $\mathcal{P}_i$.

There are four situations for the report of failure. The first situation is when a node, say v_i, has detected a failure of a node in its neighbourhood. The second situation is when the node v_i has detected a failure of an edge in its neighbourhood. The third situation is when the node v_i has detected a recovery in its neighbourhood. The fourth situation is when a node v_i has received a failure or recovery signal from its neighbour, which means the node v_i is not a neighbour of the failed node or incident to the failed edge. In the last case, it can forward the signal received to its other neighbours. As mentioned before, a node relays a failure signal if only such signal is absent in its storage memory (this is checked using the subroutine ISPERSISTENT in Algorithm 8.2). The approaches for these four situations are given in Algorithm 8.2 (see procedures NODEFAILUREDETECTION, EDGEFAILUREDETECTION, RECOVERYDETECTION and FORWARDFAILURESIGNALS).

One might argue that a node which receives a neighbouring message

Stellenbosch University https://scholar.sun.ac.za

from its neighbour which has just recovered from a failure must consider the time of failure of its neighbour while integrating the message received. Otherwise, it may end up with an incorrect view of the communication network. This would have been true in the case where neighbouring information was described by nodes only, as one may need to know which nodes are direct neighbours for an effective integration. In our case, neighbouring information is represented by edges (i.e. from neighbouring information a node knows which nodes are direct neighbours), so there is no need to consider time of failure while a node is integrating the message received.

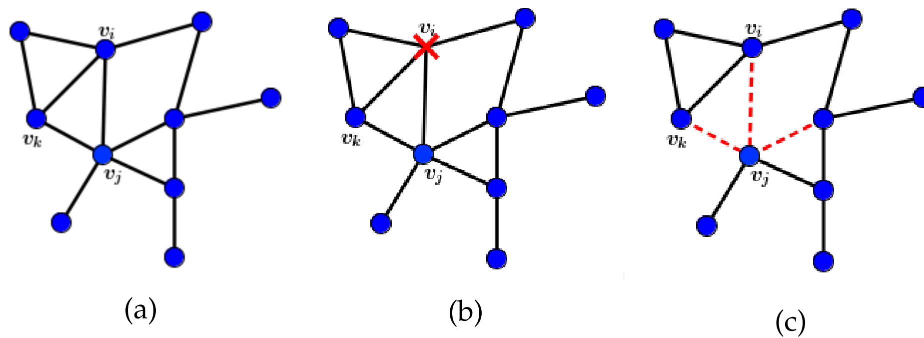


Figure 8.3: *Illustration of failure of a node or edges.* ***(8.3a)****: Initial configuration of a communication graph with a diameter of* 4*.* ***(8.3b)****: Failure of the node v_i to communicate. The new graph has a diameter of* 4*.* ***(8.3c)****: Failure of the three edges in dashed red lines. The new graph has a diameter of* 6*.*

Our last step is to include the FD in the pruning method in Algorithm 8.1. The new pruning algorithm including FD is presented in Algorithm 8.2. The main difference is that Algorithm 8.2 takes into account detection and reporting of failures (and recovery).

The original distributed method [118], presented in Algorithm 7.1, did not discuss any mechanism for communication failures. Since we aim to compare our proposed method to the original method while considering the impact of node failures, it is thus important to also incorporate the mechanism for failure handling into Algorithm 7.1. The modified version of Algorithm 7.1 including FD is presented in Algorithm 8.3.

A Python implementation of Algorithm 8.2 can be found in Appendix G.1.

The downside of this view construction approach with FD (Algorithms 8.2 and 8.3) is that there might be ambiguous situations where a node v_i can

Stellenbosch University https://scholar.sun.ac.za

Algorithm 8.2 *Our proposed pruning method with failure management. The algorithm gives the code executed for node v_i. The main methods are* NEXTONEHOP *and* NEXTUPDATE *which are run by v_i once each iteration. Apart from the variables used in Algorithm 8.1, additional global variables of this algorithm are listed in the beginning of the algorithm. It should be noted that in this algorithm assignments make copies of objects. An example of code to execute the class is given in the procedure* RUNPRUNINGFD*.*

```
1:  procedure RUNPRUNINGFD()
2:      PruningobjectFD ← PRUNINGWITHFD(N_i, D, T)
3:      PruningobjectFD.INITIALONEHOP()
4:      PruningobjectFD.INITIALUDATE()
5:      PruningobjectFD.FIRSTPRUNINGDETECTION()
6:      while not PruningobjectFD.ISENDED() do
7:          PruningobjectFD.NEXTONEHOP()
8:          PruningobjectFD.NEXTUPDATE()
9:      end while
10:     PruningobjectFD.FINALITERATION()
11: end procedure
12:
13: class PRUNINGWITHFD
14:     Class variables
15:     A              set of current neighbours of v_i
16:     Γ_i            the set of signals received by v_i
17:     E_i^down       the set of nodes in N_i incident to at least one edge in S_i^down
18:     N_i^down       set of failed nodes known by v_i
19:     N_{i,t}^up     set of neighbours of v_i which are still involved in interaction at iteration t
20:     P_i            the queue of failure signals received by node v_i
21:     S_i^down       set of failed edges known by v_i
22:     T              maximum number of iterations for a node to detect failures of its neighbours
23:     T_ij           the last time node v_i received a message from node v_j
24: ------------------------------------------------------------------------
25:     constructor (N_i, D, T)
26:         (N_i, D, T) ← (N_i, D, T)
27:         (Γ_i, N_{i,t}^up, S_i^(0)) ← (∅, N_i.clone(), {e_ij : ∀v_j ∈ N_i})
28:         S_{i,0} ← S_i^(0).clone()                    ▷ v_i detects its immediate neighbours
29:         (N_i^down, S_i^down, E_i^down, S_i^(1)) ← (∅, ∅, ∅, ∅)
30:     end constructor
31:
32:     procedure NEXTONEHOP()
33:         if not ISENDED() then
34:             t ← t + 1
35:             N_{i,t}^up ← N_i \ (F_{i,t} ∪ N_i^down ∪ E_i^down)
36:             S_i^(t) ← ∅
37:             for v_j ∈ N_{i,t}^up do
38:                 v_i sends ⟨NeighbouringMessage(i, S_i^(t−1))⟩ to v_j
39:             end for
40:         end if
41:     end procedure
42:
43:     procedure NEXTUPDATE()
44:         if not ISENDED() then
45:             while M_i.size() ≥ 1 do
46:                 for v_k ∈ N_{i,t}^up do
47:                     NODEFAILUREDETECTION(v_k)
48:                 end for
49:                 (j, S_j^(t−1)) ← M_i.dequeue()
50:                 T_ij ← currentTime()
51:                 isDown ← EDGEFAILUREDETECTION(S_j^(0))
52:                 if !isDown then
53:                     S_i^(t) ← S_i^(t) ∪ S_j^(t−1)            ▷ v_i fuses the messages received
54:                 end if
55:                 RECOVERYDETECTION(v_j, isDown)
```

Stellenbosch University https://scholar.sun.ac.za

56: $\mathcal{N}_{i,t}^{\text{up}} \leftarrow \mathcal{N}_{i,t}^{\text{up}} \setminus (\mathcal{N}_i^{\text{down}} \cup \mathcal{E}_i^{\text{down}})$
57: **end while**
58: FORWARDFAILURESIGNALS()
59: $\mathcal{S}_i^{(t)} \leftarrow \mathcal{S}_i^{(t)} \setminus \mathcal{S}_{t-1,i}$
60: $\mathcal{S}_{t,i} \leftarrow \mathcal{S}_{t-1,i} \cup \mathcal{S}_i^{(t)}$
61: $\mathcal{F}_i^{(t)} \leftarrow$ FURTHERPRUNINGDETECTION() ▷ v_i detects elements of $\mathcal{F}_i^{(t)}$ in its neighbourhood, defined in Algorithm 8.1
62: $\mathcal{F}_{i,t} \leftarrow \mathcal{F}_{i,t} \cup \mathcal{F}_i^{(t)}$
63: **end if**
64: **end procedure**
65:
66: **procedure** INITIALONEHOP()
67: $t \leftarrow 1$
68: **for** $v_j \in \mathcal{N}_i$ **do**
69: v_i sends $\langle$`NeighbouringMessage`$(i, \mathcal{S}_i^{(0)})\rangle$ to v_j
70: $T_{ij} \leftarrow$`currentTime()`
71: **end for**
72: **end procedure**
73:
74: **procedure** INITIALUPDATE()
75: **while** $\mathcal{M}_i$.`size()` ≥ 1 **do**
76: **for** $v_k \in \mathcal{N}_{i,t}^{\text{up}}$ **do**
77: NODEFAILUREDETECTION(v_k)
78: **end for**
79: $(j, \mathcal{S}_j^{(0)}) \leftarrow \mathcal{M}_i$.`dequeue()`
80: $T_{ij} \leftarrow$`currentTime()`
81: `isDown`$\leftarrow$EDGEFAILUREDETECTION($\mathcal{S}_j^{(0)}$) ▷ it returns false if no error was detected
82: **if** !`isDown` **then**
83: $\mathcal{S}_i^{(1)} \leftarrow \mathcal{S}_i^{(1)} \cup \mathcal{S}_j^{(0)}$ ▷ v_i fuses the messages received
84: $\mathcal{Q}_{ij} \leftarrow \mathcal{S}_j^{(0)}$.`clone()`
85: **end if**
86: RECOVERYDETECTION(v_j, `isDown`)
87: $\mathcal{N}_{i,t}^{\text{up}} \leftarrow \mathcal{N}_{i,t}^{\text{up}} \setminus (\mathcal{N}_i^{\text{down}} \cup \mathcal{E}_i^{\text{down}})$
88: **end while**
89: FORWARDFAILURESIGNALS()
90: $\mathcal{S}_i^{(1)} \leftarrow \mathcal{S}_i^{(1)} \setminus \mathcal{S}_{i,0}$
91: $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,0} \cup \mathcal{S}_i^{(t)}$
92: **end procedure**
93:
94: **procedure** NODEFAILUREDETECTION(v_j)▷ a procedure to send a failure signal when a node detects failure of a neighbour in its neighbourhood
95: **if** `currentTime()` $- T_{ij} > T$ **then**
96: $\mathcal{N}_{i,t}^{\text{up}} \leftarrow \mathcal{N}_{i,t}^{\text{up}} \setminus \{v_j\}$
97: $\mathcal{N}_i^{\text{down}} \leftarrow \mathcal{N}_i^{\text{down}} \cup \{v_j\}$
98: v_i sends $\langle$`Signal`$(0, 0, i, j)\rangle$ to $\mathcal{N}_{i,t}^{\text{up}}$
99: **end if**
100: **end procedure**
101:
102: **function** EDGEFAILUREDETECTION($v_j, \mathcal{S}_i^{(t)}$) ▷ a procedure to send a failure signal when a node detects failure of an edge in its neighbourhood
103: `isDown` $\leftarrow$ `checksum`$(\mathcal{S}_i^{(t)})$
104: **if** `isDown` **then**
105: $\mathcal{S}_i^{\text{down}} \leftarrow \mathcal{S}_i^{\text{down}} \cup \{e_{ij}\}$
106: $\mathcal{E}_i^{\text{down}} \leftarrow \mathcal{E}_i^{\text{down}} \cup \{v_j\}$
107: v_i sends $\langle$`Signal`$(0, 1, i, e_{ij})\rangle$ to $\mathcal{N}_{i,t}^{\text{up}}$
108: **end if**
109: **return** `isDown`
110: **end function**

Stellenbosch University https://scholar.sun.ac.za

111: **procedure** RECOVERYDETECTION(v_j, `hasFailed`) ▷ a procedure to send a recovery signal when a node detects recovery in its neighbourhood
112: **if** $v_j \in \mathcal{N}_i^{\text{down}}$ **then**
113: **if** !`hasFailed` **then**
114: $\mathcal{N}_i^{\text{up}} \leftarrow \mathcal{N}_i^{\text{up}} \cup \{v_j\}$
115: $\mathcal{N}_i^{\text{down}} \leftarrow \mathcal{N}_i^{\text{down}} \setminus \{v_j\}$
116: v_i sends $\langle \texttt{Signal}(1, 0, i, j) \rangle$ to $\mathcal{N}_{i,t}^{\text{up}}$
117: **end if**
118: **end if**
119: **if** edge $e_{ij} \in \mathcal{S}_i^{\text{down}}$ **then**
120: **if** !`hasFailed` **then**
121: $\mathcal{S}_i^{\text{down}} \leftarrow \mathcal{S}_i^{\text{down}} \setminus \{e_{ij}\}$
122: $\mathcal{E}_i^{\text{down}} \leftarrow \mathcal{E}_i^{\text{down}} \setminus \{v_j\}$
123: v_i sends $\langle \texttt{Signal}(1, 1, i, e_{ij}) \rangle$ to $\mathcal{N}_{i,t}^{\text{up}}$
124: **end if**
125: **end if**
126: **end procedure**
127:
128: **function** ISPERSISTENT($\langle \texttt{Signal}(S, Q, k, c) \rangle, \Gamma$) ▷ a procedure to check whether the node receives the signal for the first time
129: **if** $\langle \texttt{Signal}(S, Q, k, c) \rangle \notin \Gamma$ **then**
130: $\Gamma \leftarrow \Gamma \cup \{\langle \texttt{Signal}(S, Q, k, c) \rangle\}$
131: $\Gamma \leftarrow \Gamma \setminus \{\langle \texttt{Signal}(\neg S, Q, k, c) \rangle\}$
132: **return true**
133: **end if**
134: **return false**
135: **end function**
136:
137: **procedure** FORWARDFAILURESIGNALS() ▷ a procedure to send a failure signal when a node receives a failure signal
138: **while** $\mathcal{P}_i$`.size()` ≥ 1 **do**
139: $(S, Q, k, c) \leftarrow \mathcal{P}_i$`.dequeue()`
140: **if** ISPERSISTENT($\langle \texttt{Signal}(S, Q, k, c) \rangle, \Gamma_i$) **then**
141: **if** $S = 0$ **and** $Q = 0$ **then**
142: $\mathcal{N}_i^{\text{down}} \leftarrow \mathcal{N}_i^{\text{down}} \cup \{v_c\}$
143: **else if** $S = 0$ **and** $Q = 1$ **then**
144: $\mathcal{S}_i^{\text{down}} \leftarrow \mathcal{S}_i^{\text{down}} \cup \{c\}$
145: **else if** $S = 1$ **and** $Q = 0$ **then**
146: $\mathcal{N}_i^{\text{down}} \leftarrow \mathcal{N}_i^{\text{down}} \setminus \{v_c\}$
147: **else if** $S = 1$ **and** $Q = 1$ **then**
148: $\mathcal{S}_i^{\text{down}} \leftarrow \mathcal{S}_i^{\text{down}} \setminus \{c\}$
149: **end if**
150: v_i sends $\langle \texttt{Signal}(S, Q, k, c) \rangle$ to $\mathcal{N}_{i,t}^{\text{up}}$
151: **end if**
152: **end while**
153: **end procedure**
154:
155: **procedure** FINALITERATION()
156: **if** $t < D$ **then**
157: **for** $v_j \in \mathcal{N}_{i,t}^{\text{up}}$ **do**
158: v_i sends $\langle \texttt{NeighbouringMessage}(i, \emptyset) \rangle$ to v_j
159: **end for**
160: **end if**
161: $\mathcal{S}_{i,t} \leftarrow \mathcal{S}_{i,t} \setminus (\mathcal{S}_i^{\text{down}} \cup \{e_{jk} \in \mathcal{S}_{i,t} : v_j \in \mathcal{N}_i^{\text{down}} \vee v_k \in \mathcal{N}_i^{\text{down}}\})$
162: **end procedure**
163: **end class**

Stellenbosch University https://scholar.sun.ac.za

Algorithm 8.3 *Our presentation of the benchmark method [118] with failure management. The algorithm gives the code executed for node v_i. The main methods are* ONEHOP *and* UPDATE *which are run by v_i once each iteration. The same class variables used in Algorithm 8.2 are also treated as class variables here, as appropriate. It should be noted that in this algorithm assignments make copies of objects. An example of code to execute the class is given in the procedure* RUNDISTRIBUTEDBFSFD.

```
1:  procedure RUNDISTRIBUTEDBFSFD()
2:      BFSobjectFD←DISTRIBUTEDBFSWITHFD(N_i, D, T)
3:      while not BFSobjectFD.ISENDED() do
4:          BFSobjectFD.ONEHOP()
5:          BFSobjectFD.UPDATE()
6:      end while
7:  end procedure
8:
9:  class DISTRIBUTEDBFSWITHFD
10:     constructor (N_i, D, T)
11:         (N_i, D, T) ← (N_i, D, T)
12:         t ← 0
13:         S_i^(0) ← {e_ij : v_j ∈ N_i}
14:         S_{t,i} ← S_i^(0).clone()
15:         (Γ_i, N_i^down, S_i^down, E_i^down) ← (∅, ∅, ∅, ∅)
16:         N_{i,t}^up ← N_i.clone()
17:         for v_j ∈ N_{i,t}^up do
18:             T_ij ←currentTime()
19:         end for
20:     end constructor
21:
22:     procedure ONEHOP()
23:         if not ISENDED() then
24:             t ← t + 1
25:             S_i^(t) ← ∅
26:             for v_j ∈ N_{i,t}^up do
27:                 v_i sends ⟨NeighbouringMessage(i, S_i^(t−1))⟩ to v_j
28:             end for
29:         end if
30:     end procedure
31:
32:     procedure UPDATE()
33:         if not ISENDED() then
34:             while M_i.size() ≥ 1 do
35:                 for v_k ∈ N_{i,t}^up do
36:                     NODEFAILUREDETECTION(v_k)            ▷ defined in Algorithm 8.2
37:                 end for
38:                 (j, S_j^(t−1)) ← M_i.dequeue()
39:                 T_ij ←currentTime() isDown←EDGEFAILUREDETECTION(S_j^(0))
40:                 if !isDown then
41:                     S_i^(t) ← S_i^(t) ∪ S_j^(t−1)        ▷ v_i fuses the messages received
42:                 end if
43:                 RECOVERYDETECTION(v_j, isDown)           ▷ defined in Algorithm 8.2
44:                 N_{i,t}^up ← N_{i,t}^up \ (N_i^down ∪ E_i^down)
45:             end while
46:             FORWARDFAILURESIGNALS()                      ▷ defined in Algorithm 8.2
47:             S_i^(t) ← S_i^(t) \ S_{i,t−1}
48:             S_{i,t} ← S_{i,t−1} ∪ S_i^(t)
49:         end if
50:     end procedure
51:
52:     function ISENDED()
53:         return t = D or S_i^(t) = ∅
54:     end function
55: end class
```

Stellenbosch University https://scholar.sun.ac.za

wrongly think that it has reached an equilibrium state at the end of iteration t because its $\mathcal{S}_i^{(t)}$ is found to be empty. This may arise in situations where there is an overload of failures. Such ambiguous situations are acceptable in this work because, as was motivated throughout this part, we do not expect network views across nodes to be the same or exact—each node considers the view it has constructed until the end of the procedure. It should also be noted that this approach invalidates the invariant that nodes know all their t-hop neighbours (in the underlying failure-free communication graph) after $t-1$ iterations.

8.5 Conclusion

This chapter described our proposed pruning method to construct a view of a communication graph in a distributed manner. For the construction of a view of a communication graph, a variant of the decentralised algorithm in You et al. [118] was proposed, which we extended for failures. The benefit of this new variant over the state of the art is in terms of the number of messages involved in the construction of a communication graph view.

The focus of the proposed distributed method was on reducing the number of messages received by nodes during the view construction. However, algorithms that contribute to reducing running time and energy consumption would obviously also be valuable. For instance, the running time that a node requires to construct its view of a communication graph depends on the availability of messages from its immediate neighbours. Our pruning method reduces the number of messages exchanged between nodes. Thus our pruning method may run faster than the method in You et al. [118]. Also pruned nodes are not involved in subsequent communications, thus the proposed algorithm may have lower energy cost. These are by-products, not treated further in this thesis, but interesting in and of themselves. The following chapter presents and discusses our results for view construction.

Stellenbosch University https://scholar.sun.ac.za

Chapter 9

Metrics, results and discussion

This chapter presents and discusses our results for view construction. Our proposed variant (Chapter 8) for constructing a view will be compared to the original algorithm [118]. Our proposed method aims to reduce the number of messages/signals involved in interaction of robots for view construction. Thus, we will compare the methods in terms of number of messages/signals.

Summary on the decentralised methods

1. In the benchmark method, each node exchanges messages with all its immediate neighbours at each iteration (Algorithms 7.1 and 8.3).
2. In the pruning method, each node detects which of its immediate neighbours to put on hold (or prune) at each iteration, including itself. It only interacts with unpruned neighbours in subsequent iterations (Algorithms 8.1 and 8.2).

9.1 Metrics

Recall that $Y_i^{(D)}$ is the number of messages that the node v_i must receive according to the original benchmark algorithm [118] and $P_i^{(D)}$ the number of messages v_i receives through the use of the pruning strategies after D rounds. To compare the methods, we consider two performance measures: the average number of messages received per node and the maximum number of messages received per node. A good method has a low total number

Stellenbosch University https://scholar.sun.ac.za

of messages and a low maximum number of message per node. Recall that N denotes the number of nodes in a given graph. The average number of messages is thus

$$\bar{Y}^{(D)} = \frac{1}{N} \sum_{v_i \in \mathcal{V}} Y_i^{(D)}, \tag{9.1.1}$$

for the benchmark technique and

$$\bar{P}^{(D)} = \frac{1}{N} \sum_{v_i \in \mathcal{V}} P_i^{(D)}, \tag{9.1.2}$$

for our method.

The maximum number of messages is

$$Y_{\max}^{(D)} = \max_{v_i \in \mathcal{V}} \{Y_i^{(D)}\}, \tag{9.1.3}$$

for the benchmark technique and

$$P_{\max}^{(D)} = \max_{v_i \in \mathcal{V}} \{P_i^{(D)}\}, \tag{9.1.4}$$

for our pruning method. Our theoretical results indicate that we should have

$$Y_i^{(D)} \geq P_i^{(D)} \; \forall v_i \in \mathcal{V}, \tag{9.1.5}$$

and so

$$Y_{\max}^{(D)} \geq P_{\max}^{(D)}. \tag{9.1.6}$$

We can have $Y_i^{(D)} = P_i^{(D)}$ or $Y_{\max}^{(D)} = P_{\max}^{(D)}$ when the communication graph does not contain a prunable node. In terms of the structure of the graph, this means that the exact diameter of the graph is 1 or 2 (e.g. star and complete graphs) or all the nodes are central points (e.g. ring graph)—a central point of a graph is a node with minimum average distances between itself and all other nodes of the graph.

Another aspect of interest is to quantify the differences in the results of our view construction methods. As discussed in Section 6.1, we will use the Wilcoxon signed-rank test [110] and the effect size [19] to verify whether the averages of number of messages using pruning and the benchmark method are significantly different. We will also use Wilcoxon signed-rank test and the effect size to verify quality of approximate node centrality values using pruning and the benchmark method.

Stellenbosch University https://scholar.sun.ac.za

9.2 Simulation description and experimental setup

This section describes what we intend to present and discuss to assess the performance of our proposed method for view construction. It also describes the programming setup we consider for our simulations.

9.2.1 Description of experiments

For the experimental investigation, we consider four ways to compare the two strategies:

(1) In the description of our pruning method in Chapter 8, we claimed that our proposed pruning method improves the benchmark method in terms of the number of messages. We would like to verify these results experimentally. As such, we run simulation experiments and plot the number of messages received by nodes for each method on various networks to assess their number of messages. We consider failure-free cases here.

(2) We discussed the impact of failures on our method. We noted that failures of nodes or edges can cause an increase in the number of messages exchanged between nodes. Also failures can cause a decrease in the number of messages exchanged between nodes due to non-communication. But failures also introduce exchanges of signals between nodes. We run some simulation experiments with the benchmark and pruning methods on networks in the case of failures and plot the variation in the number of messages received per node. To do this, we define the probability p_f of 0.1 that a functioning node or an edge can fail at a given iteration. We also define the probability p_r of 0.8 that a failed node or edge recovers independently at each time step. A node or an edge can fail many times. It should be noted that each node/edge fails or recovers independently at the end of each iteration for both algorithms and these are conditional probabilities, i.e. these are the probability that a working node/edge fails, and the probability a failed node/edge recovers.

(3) We motivated our view construction method for distributed evaluation of node closeness centrality. It is thus important to see how good the

Stellenbosch University https://scholar.sun.ac.za

closeness centralities determined from the views constructed by our method are, compared to those constructed by the benchmark method. To do that, we run the benchmark method and our pruning method on some graphs, set different values of D, and choose the leader based on closeness centrality. We then determine the exact most central node using the actual graph. To compare the two methods, we calculate the shortest path distance between the exact most central node and the approximate most central nodes obtained with our method. A good approximation should choose a most central node with a small distance to the exact most central node. In Chapter 8, we showed that pruning is related to node eccentricity which allows us to ensure approximation of closeness centrality using pruning because eccentricity and closeness centralities are positively and strongly correlated [7, 73]. We run simulation experiments to determine the Spearman's ρ [94] and Kendall's τ [45] coefficients between eccentricity and closeness centralities. We consider the Spearman's ρ and Kendall's τ coefficients because they are appropriate correlation coefficients to measure the correspondence between two rankings. They both output values in the interval $[-1, 1]$. Values close to 1 indicate strong agreement between the two rankings (i.e. the two rankings are strongly and positively correlated), values close to -1 indicate disagreement between the two rankings is strong (i.e. there is a strong negative correlation between the two rankings). A value close to zero indicates a weak relationship between the two rankings.

(4) Our research was primarily motivated by reducing the number of messages in interaction of nodes, compared to the benchmark method [118]. However, we also compare these methods in terms of running time and memory use. We expect the pruning method to outperform the benchmark method in terms of running time and storage use. Intuitively, we think that the number of messages should be positively correlated with running time and memory size since messages need to be processed and their content stored before subsequent communication.

Stellenbosch University https://scholar.sun.ac.za

9.2.2 Experimental setup

We implemented a simulation of our pruning method using Python and `NetworkX`, [1] an open library for networks. Our simulation was run on two HPCs (High Performance Computing clusters) hosted by Stellenbosch University.[2]

We use the Python library `Pympler`[3] to determine the memory size of an object, and the Python library `Timer`[4] to determine the running time of the distributed algorithm. Given a graph, in our implementation each node has its own clock and its own memory and we picked the maximum running time and the maximum memory usage which we report per graph.

9.2.3 Input graphs

We ran several simulations with random graphs generated as discussed below, and considered the following two scenarios.

Randomly generated networks. For the first scenario, we used a grid of 200x200 points with integer coordinates and generated 50 random connected undirected graphs as follows. We generated N uniformly distributed grid locations (sampling without replacement) as nodes. The number of nodes, N, was sampled uniformly from $[50, 500]$. Two nodes are considered connected if the Euclidean distance between them is less than the communication range d—we took $d = 8$ using Euclidean distance. Generating a random graph in this way can give rise to a disconnected graph. If a disconnected graph is generated, we repeat the process until a connected graph is obtained. The number of edges and the diameter range were in $[50, 2000]$ and $[20, 60]$ in all cases respectively. An example of a communication graph of solitary nodes generated in this way can be found in Figure 9.2a. A histogram of diameters of these generated graphs is given in Figure 9.1.

Real-world networks. We also consider 34 real-world networks. The 34 real-world graphs are a phenomenology collaboration network [66], a snap-

[1] `https://networkx.github.io/`

[2] `https://www0.sun.ac.za/hpc/index.php?title=Main_Page`

[3] `Pympler` is a Python library which measures, analyses and monitors the memory behaviour of an object at runtime.

[4] `Timer` is a Python module used to handle algorithmic tasks which involve time.

Stellenbosch University https://scholar.sun.ac.za

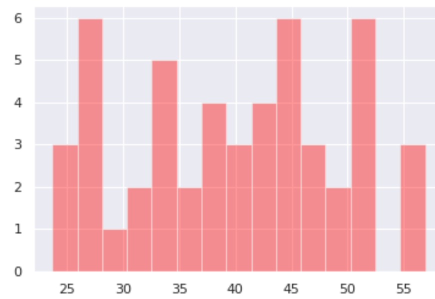


Figure 9.1: *Histogram of diameters of random graphs. We use a binwidth of* 2*.*

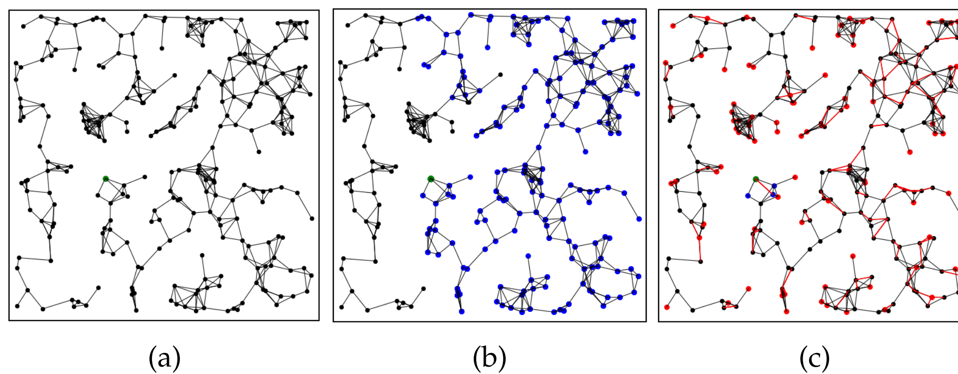


Figure 9.2: *Illustration of a network of solitary nodes.* ***(9.2a)****: A communication graph of* 250 *nodes and a diameter of* 56*.* ***(9.2b)****: Propagation of information using the approach of You et al. [118]. Running the approach on the green node, blue indicates known nodes and black indicates nodes the green node is not aware of, after* 32 *iterations.* ***(9.2c)****: Propagation of information using once-off pruning. Here, red nodes represent pruned nodes and black nodes those that are not pruned.*

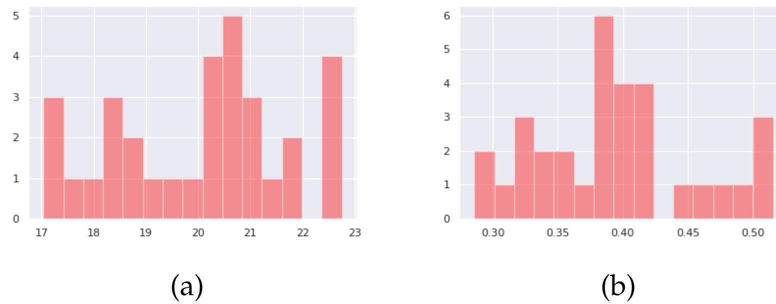
shot of the Gnutella peer-to-peer network [66], and 32 autonomous graphs [65]. The phenomenology collaboration network represents research collaborations between authors of scientific articles submitted to the Journal of High Energy Physics. In the Gnutella peer-to-peer network, nodes represent hosts in the Gnutella network and edges represent connections between the Gnutella hosts. Autonomous graphs are graphs composed of links between Internet routers. These graphs represent communication networks based on Border Gateway Protocol[5] logs. Some characteristics of some of these real-world networks can be found in Table 9.1.

[5]Border Gateway Protocol is a gateway protocol for exchanging routing and reachability information among autonomous systems.

Stellenbosch University https://scholar.sun.ac.za

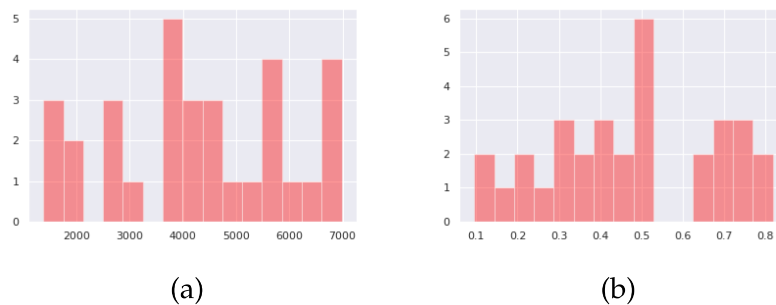
9.3 Results and discussion

9.3.1 Average and maximum number of messages



(a) (b)

Figure 9.3: *Histogram of differences in average number of messages between the benchmark and our pruning methods on 32 autonomous graphs. Positive values indicate that our pruning method is better than the benchmark method in terms of average number of messages.* ***(9.3a)****: Reductions in the average number of messages.* ***(9.3b)****: Percentage reductions in the average number of messages.*



(a) (b)

Figure 9.4: *Histogram of differences in maximum number of messages between the benchmark and our pruning methods on 32 autonomous graphs. Positive values indicate that our pruning method is better than the benchmark method in terms of maximum number of messages.* ***(9.4a)****: Reductions in the maximum number of messages.* ***(9.4b)****: Percentage reductions in the maximum number of messages.*

Our experiments illustrate the improved communication performance over the benchmark method [118] resulting from our proposed enhancement. Figures 9.3 and 9.4 show differences between the averages and differences between the maximum numbers of messages per node between

Stellenbosch University https://scholar.sun.ac.za

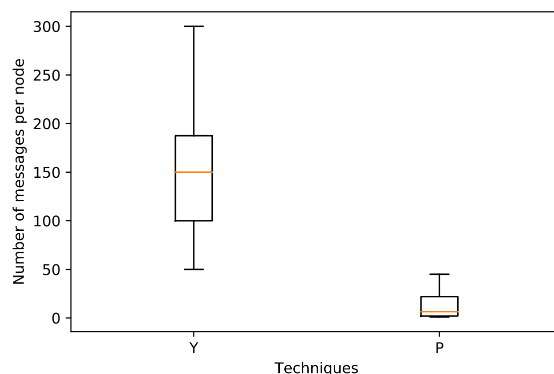


Figure 9.5: *Box-and-whisker plots of the number of messages per node using both methods on a random graph with $D = 10$. On the horizontal axis Y and P denote the benchmark and pruning method respectively.*

the benchmark and our pruning method on 32 autonomous graphs. Positive values indicate that our pruning method is better than the benchmark method.

Nodes	Edges	Diameter	Y	P	Ym	Pm
Three autonomous networks						
G_1						
1486	3422	9	*28.0*	**8.8**	*7005*	**1413**
G_2						
2092	4653	9	*27.2*	**7.9**	*3312*	**552**
G_3						
6232	13460	9	*25.4*	**6.9**	*7295*	**1459**
Phenomenology collaboration network						
10876	39994	9	*52.2*	**14.3**	*721*	**120**
Gnutella peer-to-peer network						
9877	25998	13	*63.0*	**8.3**	*3705*	**787**

Table 9.1: *Properties, average, and maximum numbers of messages for a sample of 5 real-world networks (the phenomenology collaboration network, the Gnutella peer-to-peer network, and 3 autonomous networks) on the benchmark method and our proposed method. Y and P denote average numbers of messages for the benchmark and pruning method respectively. Ym and Pm denote maximum numbers of messages for the benchmark and pruning method respectively. The poorest and best performances are shown in italic and bold respectively.*

The number of messages per node on one random graph using our algorithms are presented in Figure 9.5. Table 9.1 shows the results of the average and the maximum number of messages per node in five real-world graphs respectively.

Stellenbosch University https://scholar.sun.ac.za

For applicable graphs (where the sets of prunable nodes are non-empty), putting some nodes on hold provides an improvement over the benchmark method [118] in terms of messages exchanged. The results confirm that the pruning method is better (in terms of average number of messages) than the benchmark method (Figure 9.3).

Hypothesis test

We also used a Wilcoxon signed-rank test and the effect size to verify whether the mean differences of the number of messages between pruning and the benchmark method are significantly different.

We observed a p-value of 7.96×10^{-90} and an effect size of 21.1140 between the number of messages obtained with our pruning and the benchmark methods on 50 random graphs containing 500 nodes each. The p-value is less than the threshold 0.01, so the means of the number of messages using the benchmark method against pruning are significantly different. In terms of effect size, according to the classification in Gail and Richard [33], the effect size between our pruning and the benchmark methods is large ($e \geq 0.8$). So the means of the number of messages using the benchmark method against pruning are different markedly.

The motivation of pruning was to reduce the communication overhead in the entire network. For this, it was mentioned that the total number of messages is a good indicator. However, this reduction in the total number of messages can speed up the process of view construction only when the maximum number of messages is also reduced. Figure 9.4 shows that the maximum number of messages per node for the same graphs as in Figure 9.3 is also reduced using the pruning method.

Figure 9.5 and Table 9.1 also confirm that pruning method reduces both the total and the maximum numbers of messages. Having both the total number of messages and the maximum number of messages per node reduced indicates that our proposed method may have a positive impact on other performance metrics as will be discussed later in this section.

Furthermore, having the number of messages per node reduced indicates that our proposed method may have a positive impact on the effect of communication failures. Since pruned nodes are not involved in subsequent communication, any impact of their failure is reduced and they do not need to receive failure signals.

Stellenbosch University https://scholar.sun.ac.za

9.3.2 Impact of failure on the number of messages

We also compare the impact of node and edge failures on the number of messages exchanged during the process of view construction using the benchmark and our pruning method. Figure 9.6 shows the number of messages received by nodes when considering both a failure-free situation and the case of failure on the same network for four of the randomly generated networks. Figure 9.7 shows the number of signals received by nodes for the four randomly generated networks. There are a number of nodes or edges which fail in each iteration.

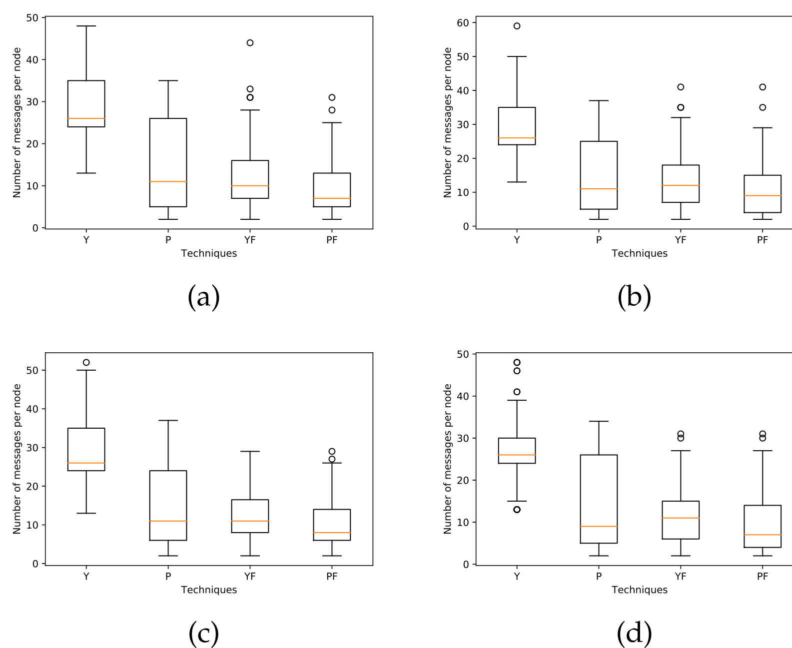


Figure 9.6: *Number of messages per node using both methods for view construction in the case of failures with $D = 13$. On the horizontal axis Y, P, YF and PF denote the benchmark method without failure, the pruning method without failure, the benchmark method with failures and the pruning method with failures respectively. Number of messages on four graphs of diameter, node size and edge size of* ***(9.6a)****:* 115; 965 *and* 2217*.* ***(9.6b)****:* 113; 867 *and* 1921*.* ***(9.6c)****:* 116; 909 *and* 2103*.* ***(9.6d)****:* 71; 999 *and* 2428*.*

Figures 9.6 indicates that when some nodes or edges fail, the number of messages received per node decreases. The number of messages per node decreases because nodes intend to report failure information to the entire network—when a node or an edge fails or recovers, its immediate neighbours report this information to the entire network—as shown in Figure 9.7.

Stellenbosch University https://scholar.sun.ac.za

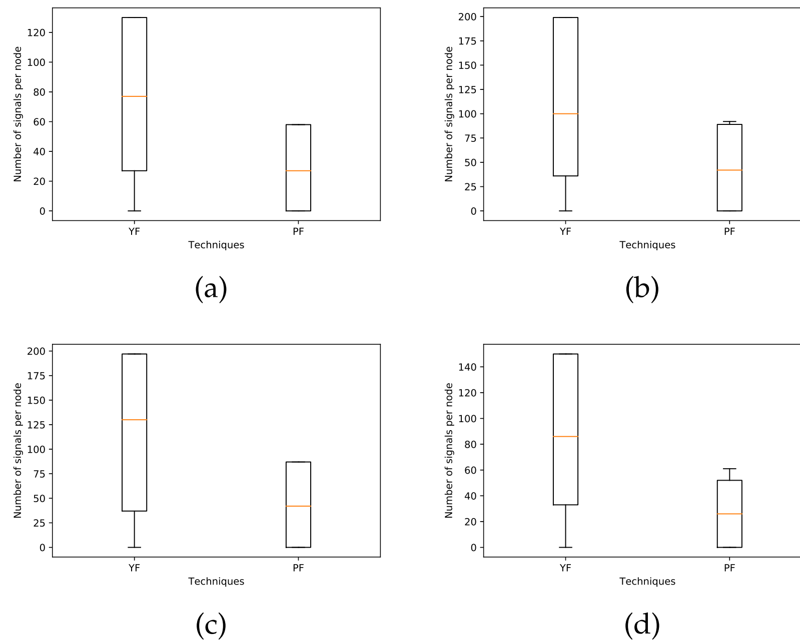


Figure 9.7: *Number of signals per node using both methods for view construction in the case of failures with $D = 10$. On the horizontal axis YF and PF denote the benchmark method and the pruning method respectively. Number of messages on four graphs of diameter, node size and edge size of* ***(9.7a)****:* $115; 965$ *and* 2217. ***(9.7b)****:* $113; 867$ *and* 1921. ***(9.7c)****:* $116; 909$ *and* 2103. ***(9.7d)****:* $71; 999$ *and* 2428.

The larger the network, the higher the risk of node and edge failures. When many nodes or edges fail or recover, it will require a considerable number of signals to be sent to all remaining nodes in order to report the situations of failed nodes or edges.

Figure 9.7 shows that the pruning method is less affected by the impact of node or edge failures. This is simply because pruned nodes are not involved in subsequent communication, so they are not notified of communication failures. This indicates that it is worthwhile to reduce the number of messages during graph view construction in order to reduce the impact of failures on the overall number of messages that must be exchanged between nodes.

9.3.3 Quality of selected most central node

Foremost, for various graphs considered here, the averages and standard deviations of the Spearman's ρ and Kendall's τ coefficients between eccentricity and closeness centralities are 0.9237 ± 0.0508 and 0.7839 ± 0.0728 re-

Stellenbosch University https://scholar.sun.ac.za

spectively, and the correlation coefficients were all positive. This shows that there is predominantly a fairly strong level of correlation between eccentricity and closeness centralities for various graphs considered here. This confirms the results by Batool and Niazi [7], and Meghanathan [73]. Since pruning is related to eccentricity, this indicates that pruning can be used for approximation of closeness centrality.

Tables 9.2 and 9.3 show the shortest path distances between the exact most central node and approximate most central nodes using our pruning method and the benchmark method [118] for 2 random graphs for failure-free cases and failure cases respectively. The first random graph is composed of 70 nodes and has diameter of 35. The second random graph is composed of 100 nodes and has diameter of 49. Also Figures 9.8 and 9.9 show differences of shortest path distances between approximate central nodes obtained between the benchmark and our pruning method with respect to the exact most central node on three random graphs for failure-free cases and failure cases respectively. For each of the graphs, we vary D in a range of values smaller than the diameter of the graph.

D	$\mathbf{Y_1}$	$\mathbf{P_1}$	$\mathbf{Y_2}$	$\mathbf{P_2}$
2	**5**	**5**	**14**	**14**
6	14	**10**	19	**5**
10	13	**1**	20	**0**
14	14	**4**	19	**2**
18	8	**0**	**0**	**0**
22	**0**	**0**	**0**	2
26	**0**	**0**	**0**	**0**

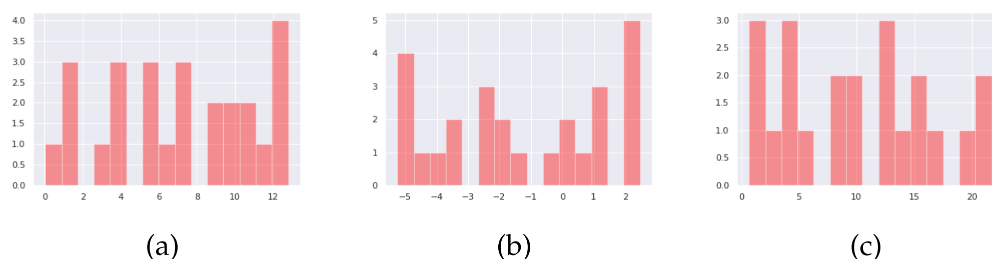
Table 9.2: *Shortest path distances between the exact most central node and approximate most central node using pruning and the benchmark method for failure-free situations. We use two random graphs with* 70 *nodes and diameter of* 35*, and with* 72 *nodes and diameter of* 32 *respectively. The best approximations for each graph and choice of D are highlighted in bold. One method achieves better approximations than another if the distance of the selected node from the true most central node is smaller.* Y_i *and* P_i *indicate shortest path distances for the benchmark and pruning methods on the i-th randomly generated graph respectively.*

The evaluation of node closeness centrality based on a limited view of the communication graph has an impact on the choice of the most central node. When using our pruning method and the benchmark method to choose a leader based on closeness centrality, the methods can yield different results under the same conditions. For failure-free cases, we found

Stellenbosch University https://scholar.sun.ac.za

D	$\mathbf{Y}_1$	$\mathbf{P}_1$	$\mathbf{Y}_2$	$\mathbf{P}_2$
2	**6**	31	40	**15**
6	18	**15**	30	**28**
10	27	**6**	10	**8**
14	15	**6**	**10**	30
18	**6**	38	30	**8**
22	**6**	15	**12**	30
26	**6**	**6**	40	**10**

Table 9.3: *Shortest path distances between the exact most central node and approximate most central node using pruning and the benchmark method for failure situations. We use two random graphs with* 70 *nodes and diameter of* 35*, and with* 72 *nodes and diameter of* 32 *respectively. The best approximations for each graph and choice of D are highlighted in bold. One method achieves better approximations than another if the distance of the selected node from the true most central node is smaller.* Y_i *and* P_i *indicate shortest path distances for the benchmark and pruning methods on the i-th randomly generated graph respectively.*

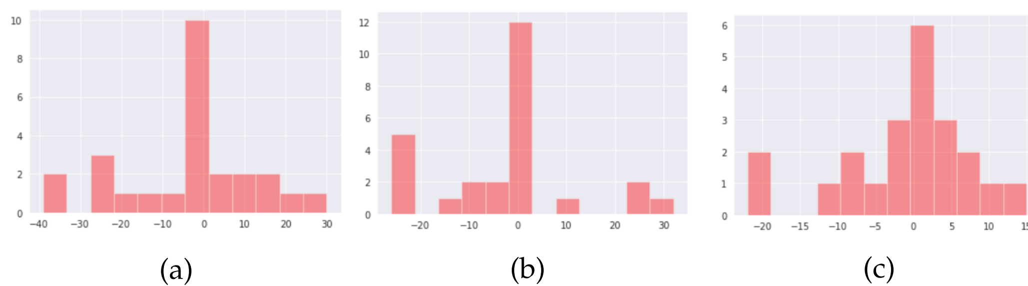


(a) (b) (c)

Figure 9.8: *Histograms of differences of shortest path distances between approximate central nodes obtained using the benchmark and our pruning methods with respect to the exact most central node on three random graphs (for failure-free cases):* ***(9.8a)****: the first (diameter of* 26*,* 125 *nodes and* 180 *edges),* ***(9.8b)****: second (diameter of* 36*,* 289 *nodes and* 597 *edges), and* ***(9.8c)****: third (diameter of* 40*,* 301 *nodes and* 668 *edges) random graphs. Positive values indicate that our method is better than the benchmark method, and negative values indicate the opposite.*

that (Table 9.2) our pruning method gives better approximations to closeness centrality than the benchmark method for failure-free cases when D is smaller than the diameter. In such cases, the benchmark method yields poorer results, as shown in Table 9.2. The results of the benchmark method improve as D increases. This indicates that our pruning method might effectively identify nodes which can not be chosen as leaders as they do not have the highest closeness centrality. Note that, even though the two methods sometimes give the exact most central nodes for some D (for example for $D = 26$ in Table 9.2), these exact most central nodes are not guaranteed. Also the benchmark method can yield better results than pruning as illus-

Stellenbosch University https://scholar.sun.ac.za



(a) (b) (c)

Figure 9.9: *Histograms of differences of shortest path distances between approximate central nodes obtained using the benchmark and our pruning methods with respect to the exact most central node on three random graphs (for failure cases):* ***(9.9a)****: the first (diameter of* 26, 125 *nodes and* 180 *edges),* ***(9.9b)****: second (diameter of* 36, 289 *nodes and* 597 *edges), and* ***(9.9c)****: third (diameter of* 40, 301 *nodes and* 668 *edges) random graphs. Positive values indicate that our method is better than the benchmark method, and negative values indicate the opposite.*

trated in Table 9.2 for $D = 22$. This is also seen in Figure 9.8. As mentioned before, this can be caused by the value of the maximum number of iterations D—this should be expected when this value is smaller than the diameter of the communication graph considered. For failure cases (Table 9.3 and Figure 9.9), the benchmark method outperforms our pruning method in some cases and our pruning method outperforms the benchmark method in other cases—this odd situation can be due to the handling of failure in both methods. This may indicate that both methods can be used for selection of the most central node of a network in failure situations.

The reason why the benchmark method yields poor results when D is smaller than the diameter of the graph for failure-free cases is as follows. When some of the nodes have different views of the communication graph and each evaluates its closeness centrality only based on its view of the communication graph, a node with small exact closeness centrality (which it does not know) may have a high approximate closeness centrality. This can lead to poor conclusions, i.e. nodes may have high approximate centrality, despite having low exact centrality. Our pruning method can also suffer from the same problem. However, the advantage of pruning is that only unpruned nodes compute their approximate closeness centralities, i.e. many such nodes are pruned using our proposed pruning approach. This reduces the chance of yielding poor performance as the central node is selected from a shorter list of candidates, i.e. the unpruned nodes. In the benchmark method, the central node is selected from all nodes.

Stellenbosch University https://scholar.sun.ac.za

Hypothesis test

We also used a Wilcoxon signed-rank test and the effect size to verify whether the mean differences of shortest path distances between approximate central nodes obtained using the benchmark and our pruning methods with respect to the exact most central node on some random graphs are significantly different.

We observed a p-value of 0.1197 and an effect size of 0.3174 between the results obtained with our pruning and the benchmark methods on 50 random graphs of 500 nodes each. The p-value is greater than the threshold 0.01, so there is no significant difference between the means for the two approaches. In terms of effect size, the effect size between our pruning and the benchmark methods is small ($e \leq 0.5$). So the results obtained from our pruning and the benchmark methods are not different. This means that the qualities of the selected most central nodes using both methods are almost the same. This is beneficial to pruning—though they both provide almost the same qualities of selected most central nodes, pruning reduces the number of messages significantly compared to the benchmark method [118].

9.3.4 Running time and memory usage complexities

We also consider maximum running time and maximum memory usage to compare our techniques. Figures 9.10 and 9.11 show differences in the maximum running time (in seconds) results between the benchmark and pruning techniques on the 50 random graphs for failure-free cases and failure cases respectively. Figures 9.12 and 9.13 show difference in the maximum storage memory usage results between the benchmark and pruning methods on the same 50 graphs for failure-free cases and failure cases respectively.

In general, the pruning method builds a view with a reduced number of messages received by nodes, compared to the benchmark method [118]. But the process of pruning can take additional time. However, after pruning a graph, nodes in the subgraph obtained will have a reduced degree. Thus our method may possibly still run faster than the method in [118]. In fact, Figures 9.10 and 9.11 show that the pruning method reduces running time compared to the benchmark method. This means that with a time/communication budget one can get a larger network view (i.e. greater D) with the

Stellenbosch University https://scholar.sun.ac.za

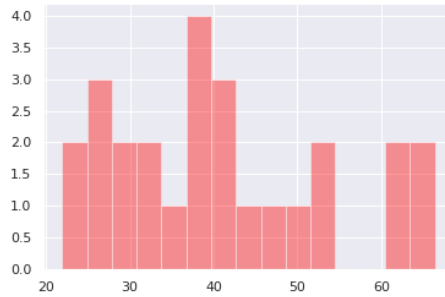


Figure 9.10: *Histogram of percentage reductions of maximum running time (in seconds) on the* 50 *random graphs between the benchmark and our pruning methods for failure-free cases. These values indicate that our pruning method is better than the benchmark method in terms of maximum running time.*

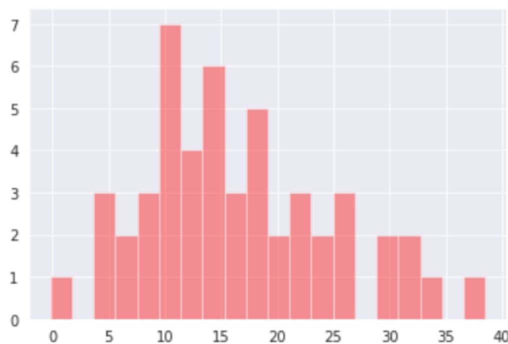


Figure 9.11: *Histogram of percentage reductions of maximum running time (in seconds) on the* 50 *random graphs between the benchmark and our pruning methods for failure cases. These values indicate that our pruning method is better than the benchmark method in terms of maximum running time.*

pruning approach. In the pruning method, prunable nodes finish running earlier than the unprunable nodes. Pruned nodes are also not involved in subsequent computations, thus the proposed algorithm has a lower energy cost. The trade-off is that nodes which are pruned in the pruning method have very limited views of the communication graph, compared to the same nodes using the benchmark technique. These limited views of the communication graph result in less memory being used by the pruning method. This can be observed in Figures 9.12 and 9.13 where maximum memory usage of the pruning method is smaller than that of the benchmark method in all cases for failure-free situations (Figure 9.12) and in most cases for failure situations (Figure 9.13)—the pruning method requires less storage capacity

Stellenbosch University https://scholar.sun.ac.za

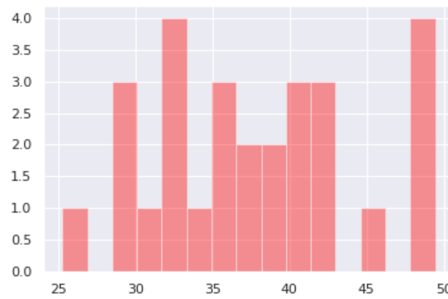


Figure 9.12: *Histogram of percentage reductions of maximum memory usage (in MB) on the* 50 *random graphs between the benchmark and our pruning methods for failure-free cases. These values indicate that our pruning method is better than the benchmark method in terms of maximum memory usage.*

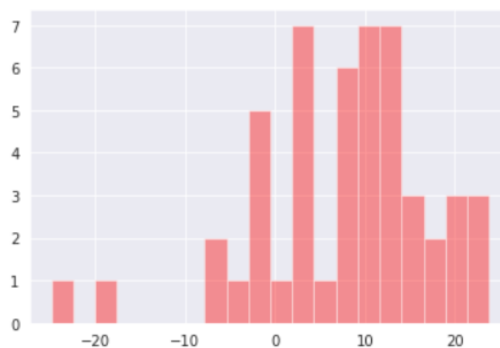


Figure 9.13: *Histogram of percentage reductions of maximum memory usage (in MB) on the* 50 *random graphs between the benchmark and our pruning methods for failure cases. These values indicate that our pruning method is better than the benchmark method in terms of maximum memory usage in most of the random graphs we considered.*

because the memory required for storing pruned nodes by unpruned nodes is less than the the storage capacity that would have been required by the same nodes to store their own views of the communication graph using the benchmark technique.

9.4 Conclusion and avenues for future work

We proposed an enhancement to a benchmark method [118] for view construction in Chapter 8. The main motivation of this enhancement was to reduce communication complexity: we aim to reduce the number of messages exchanged between nodes during interaction. Our main contribution

Stellenbosch University https://scholar.sun.ac.za

was how nodes can identify other nodes, including themselves, less likely to be a central nodes. Such nodes are put on hold during subsequent iterations of view construction. Details of our contributions and novelties for graph view construction can be found in Section 8.1.1.

9.4.1 Findings summary

We confirm our theoretical results showing that our proposed method improves the benchmark method in terms of number of messages. In our experiments we found that our proposed method is less affected by the number of signals exchanged between nodes when handling communication failures. We also found that reduction of the number of messages has a positive impact on other types of performance metrics, namely running time and memory usage.

9.4.2 Future work

When counting messages in our proposed distributed method, we ignore the fact that in large networks, message may need to comprise multiple packets. Analysis of the savings of our proposed approach in terms of the number of edges communicated, or the actual amount of data communicated could be investigated in future. Furthermore, our goal was that our proposed pruning method improves the benchmark method [118], in terms of number of messages (in failure-free cases) and the number of signals (in failure cases), and this was verified empirically in all the experiments run in this chapter. We only considered theoretical analysis of number of messages. We ultimately hope to investigate the number of signals of our proposed pruning method theoretically in future. Finally, it is worth investigating whether we can apply further results from combinatorics or graph theory to study how our proposed method impacts the computation of graph-related quantities.

This concludes our investigation of our proposed method on network view construction. In the previous two parts we have covered proposals for coordination of solitary robots and decentralised view construction. In the following part, we cover our proposal on the last problem considered in this thesis: classifier fusion.

Stellenbosch University https://scholar.sun.ac.za

Part III

Classifier fusion

Stellenbosch University https://scholar.sun.ac.za

Chapter 10

Background and literature

When a robot is exploring an environment, it should be able to recognise targets of interest to achieve its goal. We consider robots provided with multiple sensor nodes (i.e. processing units embedded in a robot) to measure information. Each sensor node of a robot performs classification tasks to identify the types of any objects detected [97]. This is termed object recognition [107] in the machine learning community.

In classifying an observation, the members of an ensemble of (probabilistic) classifiers will typically disagree, with individual classifiers sometimes producing markedly different predictions for the same observation. Leveraging such disagreement can be the very basis of improved overall performance [30]. One of the key motivations for ensemble techniques in machine learning is the no free lunch theorem [112], which states the impossibility of realising a universal best classification methodology. This chapter reviews literature on the problem of classifier fusion when classifiers' outputs are the only information available. This means that we do not know how classifiers work, and do not have information on training or other past performances of classifiers. We consider situations where classifiers have been trained to perform the same task.

10.1 General background

Various approaches have been developed to take advantage of diverse classifier performance in different settings [60, 69, 87, 107] including Bayesian methods [48] and Dempster-Shafer methods [28]. In classifier fusion, two

Stellenbosch University https://scholar.sun.ac.za

cases can be considered: models can be trained collaboratively (e.g. boosting methods [87]) or separately. We consider the case of classifiers being trained separately but from the same underlying distributions. It is not imperative that classifiers be trained on the same data. They can be trained on different data and with possibly different features, but for the same problem. In this case, we consider two categories of classifier fusion methods, depending on the information available:

(1) If classifiers' outputs are the only information explicitly given, then methods based on the principle of indifference with respect to hypotheses could be applied [51, 60].

(2) If other information, such as confusion matrices, is also explicitly given, then Bayesian or Dempster-Shafer methods could be applied [36, 48, 60].

We are interested in the situation (1), i.e. a challenging situation where we have outputs from classifiers, but we are not given additional evidence such as the validation or test error to distinguish hypotheses (i.e. a "zero-knowledge" situation): we do not know what model each classifier uses, how and on what data the classifier was trained, what features it was trained with, its performance on any past data, or any of its predictions on any previous observations. Such information would permit a classifier fusion technique to explicitly take into account the quality of the classifiers when combining different classifiers' outputs. For Dempster-Shafer methods for instance, such information is used to form basic probability assignments (also known as mass functions) of all subsets of the set of hypotheses (or classes in our case) [28]. In the absence of such information, it may seem that classifiers should all be treated equally—this is termed the principle of indifference. The lack of evidence to distinguish hypotheses prompts us to use methods based on the principle of indifference, rather than existing Bayesian or Dempster-Shafer methods.

In this setting, prominent approaches based on the principle of indifference are the majority vote rule, the sum rule and the product rule, with the product rule corresponding to Bayesian classifier fusion assuming a uniform prior over class labels, one of the classifiers is correct, and each classifier is equally likely to be correct a priori.

Stellenbosch University https://scholar.sun.ac.za

10.2 Literature discussion

While there is a vast literature on classifier fusion, the "zero-knowledge" situation we consider in this thesis has received relatively little attention. The lack of evidence to distinguish individual classifiers prompts us to use methods based on the principle of indifference. Kittler et al. [51, 52] indicated that, in terms of performance, the sum rule is the best fusion rule based upon the principle of indifference: after conducting an analysis of error sensitivity and experimental investigations of the sum rule versus the product rule, they showed that the sum rule is much less affected by estimation errors. We next introduce the sum rule, product rule, majority vote rule [51] and Borda count rule [25] after introducing some notation.

The approach taken in this work combines classifiers' outputs on an input vector $\boldsymbol{x}_{\text{in}}$. There are m classifiers, and each one's output values are treated as probability vectors. A classifier f_j outputs d_j, i.e. $d_j = f_j(\boldsymbol{x}_{\text{in}})$. Let d_{jk} denote the probability that the input $\boldsymbol{x}_{\text{in}}$ belongs to the k-th class of objects according to classifier output f_j, and $\varpi = (\varpi_1, \varpi_2, \cdots, \varpi_l)$ be the fusion output (i.e. a probability distribution) we wish to determine and let the classifiers' outputs for input $x_{\mathbf{in}}$ be represented by the following output matrix,

$$\begin{bmatrix} d_1 \\ \vdots \\ d_m \end{bmatrix}, \tag{10.2.1}$$

where $d_j = (d_{j1}, d_{j2}, \cdots, d_{jl})$ and l denotes the number of class labels.

Sum rule. Considering the classifiers' outputs in Equation 10.2.1, the sum rule output is

$$\varpi \equiv \frac{1}{m}\sum_{j=1}^{m} d_j = \frac{1}{m}\left(\sum_{j=1}^{m} d_{j1}, \sum_{j=1}^{m} d_{j2}, \cdots, \sum_{j=1}^{m} d_{jl}\right), \tag{10.2.2}$$

Product rule. Considering the classifiers' outputs in Equation 10.2.1, the product rule output is

$$\varpi \equiv \alpha \prod_{j=1}^{m} d_j = \alpha\left(\prod_{j=1}^{m} d_{j1}, \prod_{j=1}^{m} d_{j2}, \cdots, \prod_{j=1}^{m} d_{jl}\right), \tag{10.2.3}$$

where α denotes a normalisation constant.

Stellenbosch University https://scholar.sun.ac.za

Majority vote rule. This is the typical majority vote rule applied to each probability distribution. Let Δ_{jk} be defined by

$$\Delta_{jk} = \begin{cases} 1 & \text{if } d_{jk} = \max_{s=1}^{l} d_{js} \\ 0 & \text{otherwise.} \end{cases}$$

The fused output according to the majority vote rule is

$$\frac{1}{m}\left(\sum_{j=1}^{m}\Delta_{j1}, \sum_{j=1}^{m}\Delta_{j2}, \cdots, \sum_{j=1}^{m}\Delta_{jl}\right). \tag{10.2.4}$$

Borda count rule. In the Borda count rule, classifiers rank classes in order of preference by giving each class a number of points corresponding to the number of classes ranked lower. Let β_{jk} be defined by

$$\beta_{jk} = l - |\{d_{ji} : d_{ji} > d_{jk} \vee (d_{ji} = d_{jk} \wedge i < k)\}|.$$

The fused output according to the Borda count rule is

$$\frac{2}{ml(l+1)}\left(\sum_{j=1}^{m}\beta_{j1}, \sum_{j=1}^{m}\beta_{j2}, \cdots, \sum_{j=1}^{m}\beta_{jl}\right). \tag{10.2.5}$$

In the classification tasks considered in this work, we employ the terms fused output and fused decision as follows. Given an input, a fused output is a probability distribution over class labels. A fused decision is the class with largest fused output. In the case of ties, which occur when two or more class labels are assigned the maximum probability, we arbitrarily consider the minimum index of corresponding maximum values. For instance, if the fused output is $(0.4, 0.35, 0.25)$ over three class labels, the fused decision could be the first class label as it is assigned the highest probability.

10.3 Summary

This chapter reviewed background work and literature for classifier fusion based on the principle of indifference. Classifier fusion methods based on the principle of indifference are used when information on the performance of individual classifiers is not provided. We propose a classifier fusion method that takes advantage of the reputability of the classifiers when only classifiers' outputs are provided.

Having reviewed and discussed background and literature on our proposed classifier fusion method, we now describe our proposed method.

Stellenbosch University https://scholar.sun.ac.za

Chapter 11

Classifier fusion method

Portions of Part III, and particularly this chapter, appear in Masakuna et al. [71].

11.1 Introduction

This chapter introduces a novel iterative approach to the problem of classifier output fusion that robots use to recognise objects of interest: a once-off classifier fusion method based on the principle of indifference—no external information is provided to enhance the fusion process. We consider the problem of classification where outputs from various classifiers must be combined to make a consensus decision, i.e. a decision all members agree to support in the best interest of the whole group [68].

11.1.1 Contributions

In this work, we assume some regularity between classifiers that is not considered in other methods based upon the principle of indifference. This assumption of regularity is motivated by the fact that classifiers have been optimized for the same problem. Our proposed method is inspired by a negotiation process by which shareholders (corresponding to classifiers) can attempt to reach a consensus decision, with shareholders gradually update their position over time as negotiations progress until consensus is reached. Their initial positions are the outputs obtained from classifiers. Consensus is reached when their transformed positions converge (i.e. they correspond sufficiently) or until some prescribed maximum number of iterations.

Stellenbosch University https://scholar.sun.ac.za

To reach a consensus decision through classifier fusion, a form of support between probability distributions is computed to allow these distributions to contribute unequally to the fused result. A support is a non-negative vector of scores that encodes a notion of similarity between two distributions: when two distributions are more similar, they assign stronger support to each other, as reflected by larger class supports. The key idea of our proposed classifier fusion method is that the similarities of the classifier outputs in regions where they have a fair amount of confidence, and the discrepancies of the classifier outputs in regions where they have contradicting views will be used to update their initial predictions.

11.1.2 Limitations and application of our proposed method

Our proposed method is not suitable in situations where some likelihood-type of loss function is appropriate. When distributions converge, our proposed method assigns high probability (of almost one) to the consensus class, and low probability (of almost zero) to other classes. Thus, our approach may perform poorly for cases where the loss function yields high losses for overconfident misclassification.

Our proposed method has a wider applicability than robotics.

11.2 Yayambo method

We propose a classifier fusion method where distributions are collected from some classifiers and iteratively updated until they converge (or after some pre-specified number of iterations). We call this fusion method the Yayambo[1] algorithm.

11.2.1 Notation

The approach taken in this work combines classifiers' outputs on an input vector x_{in}. Given $x_{\mathbf{in}}$, each classifier f_i outputs a probability distribution $d_i = f_i(x_{\mathbf{in}})$ over class labels which will serve as initial distributions $\pi_i^{(0)}$.

[1]Yayambo means *the first* in Kikongo, one of the languages spoken in the Democratic Republic of Congo.

Stellenbosch University https://scholar.sun.ac.za

11.2.2 Description

During each iteration t, each current distribution $\pi_i^{(t-1)}$ is updated to a new distribution $\pi_i^{(t)}$. These updates are performed based on a vector of non-negative weights computed between each pair of current distributions. Each vector component expresses a level of support for one distribution's prediction for a specific class from another, and is called *class support*. Such a vector of class supports is termed a *support*. These supports are used to update distributions $\pi_i^{(t-1)}$ to $\pi_i^{(t)}$ as follows. Let $\beta_{ji}^{(t)}$ denote the support for $\pi_i^{(t-1)}$ from $\pi_j^{(t-1)}$. The distribution $\pi_i^{(t)}$ and the support $\beta_{ji}^{(t)}$ are non-negative vectors, and each distribution $\pi_{ik}^{(t)}$ is updated in each iteration using the rule

$$\pi_{ik}^{(t)} = \alpha_i^{(t)} \pi_{ik}^{(t-1)} \sum_{j \neq i} \beta_{ji,k}^{(t)}, \tag{11.2.1}$$

where the subscript k denotes a vector component (one per class), and $\alpha_i^{(t)}$ is a normalisation constant. This yields valid probability distributions at each iteration since the supports will be non-negative (as discussed later). When consensus is reached, the algorithm terminates: this occurs when $\pi_i^{(t)} \approx \pi_i^{(t-1)}$ for all i at the end of iteration t, or at a pre-specified maximum iteration T. To verify convergence, our implementation uses the condition

$$\sum_{i=1}^{m} \|\pi_i^{(t)} - \pi_i^{(t-1)}\|_2 < m\varepsilon, \tag{11.2.2}$$

for a predefined small $\varepsilon > 0$.

When the algorithm terminates, say at the end of iteration T, the consensus distribution is then formed using the sum rule:

$$\varpi \equiv \frac{1}{m} \sum_{i=1}^{m} d_i = \frac{1}{m} \left(\sum_{i=1}^{m} \pi_{i1}^{(T)}, \sum_{i=1}^{m} \pi_{i2}^{(T)}, \cdots, \sum_{i=1}^{m} \pi_{il}^{(T)} \right). \tag{11.2.3}$$

On termination, the use of the sum rule in Equation 11.2.3 simply provides a mechanism for resolving potential cases that do not reach consensus by the iteration limit.

The final detail to complete the algorithm is specifying how the required supports $\beta_{ji}^{(t)}$ are calculated based on the current distributions $\pi_j^{(t-1)}$ and $\pi_i^{(t-1)}$. Supports influence how much $\pi_i^{(t)}$ changes. We do not require support of distributions to necessarily be symmetric.

Stellenbosch University https://scholar.sun.ac.za

This notion of support is related to the Matthew Effect [74], where group members with similar beliefs about a topic amplify and collect ever-larger support from each other. In the setting of our work here, if we cluster stakeholders (classifiers) into groups whose predictions are similar, the members of each group will assign stronger support to each other, and assign weaker support to members of other groups.

While various support functions could be defined, the function we consider is inspired by Kullback-Leibler divergence [59].

The class support for $\pi_i^{(t-1)}$ from $\pi_j^{(t-1)}$ with respect to class c_k depends principally on two things: first, the probability $\pi_j^{(t-1)}$ assigns to class c_k (a larger probability corresponds to a higher weight), and second on how closely the probabilities that $\pi_i^{(t-1)}$ and $\pi_j^{(t-1)}$ assign to class c_k align (larger differences in probability lead to a smaller weight). Specifically, we define the support $\beta_{ji,k}^{(t)}$ for $\pi_i^{(t)}$ from $\pi_j^{(t)}$ on $\boldsymbol{x}_{\text{in}}$ for a class label c_k at iteration t as

$$\beta_{ji,k}^{(t)} = \frac{\pi_{jk}^{(t-1)}}{1 + D(\pi_{ik}^{(t-1)}, \pi_{jk}^{(t-1)})}, \tag{11.2.4}$$

where the dissimilarity on the k-th component of distributions $\pi_j^{(t)}$ and $\pi_i^{(t)}$ is

$$D(\pi_{ik}^{(t)}, \pi_{jk}^{(t)}) = \pi_{ik}^{(t)} \left| \log \frac{\pi_{ik}^{(t)} + \varepsilon_0}{\pi_{jk}^{(t)} + \varepsilon_0} \right|, \tag{11.2.5}$$

where ε_0 a smoothing term to avoid zero division. In Equation 11.2.5 we use the absolute value to make the log-ratio symmetric. This is our implementation inspired from the Kullback-Leibler divergence as mentioned above. Other formulations could also be applied.

An additional information about Equation 11.2.5 it is that one should consider the influence ε_0 has on the final decision. An illustration showing the dissimilarities for probabilities is shown in Figure 11.1.

We call this fusion method the Yayambo algorithm, and it is given in Algorithm 11.1. A Python implementation of Yayambo can be found in Appendix G.3.

Example. Let f_1 and f_2 be two classifiers with outputs

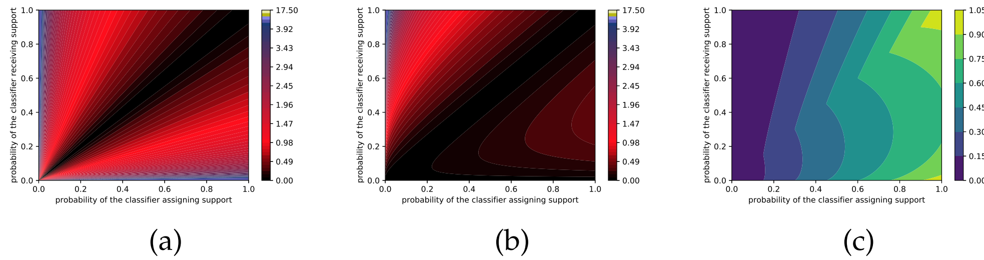
$$\pi_1^{(0)} = (0.3, 0.7) \quad \text{and} \quad \pi_2^{(0)} = (0.8, 0.2),$$

Stellenbosch University https://scholar.sun.ac.za

Algorithm 11.1 *Our proposed classifier fusion method. The main function* (GETCONSENSUS) *receives as input the classifier outputs* $\pi_i^{(0)}$ *and outputs the fusion probability distribution* ϖ *and class prediction.*

```
1:  class YAYAMBO
2:      Class variables
3:      ε            the convergence threshold
4:      ε_0          a small threshold ε_0 to avoid zero division
5:      m            the number of classifiers
6:      T            the maximum number of iterations
7:  ------------------------------------------------------------
8:      constructor (m, T, ε, ε_0)
9:          (m, T, ε, ε_0) ← (m, T, ε, ε_0)          ▷ the number if distributions π_i^(0) to fuse
10:     end constructor
11:
12:     function GETCONSENSUS(π_i^(0) ∀i)
13:         for t = 1 to T do
14:             for i = 1 to m do
15:                 for j ≠ i do
16:                     (β_{ji,1}^(t), β_{ji,2}^(t), ···, β_{ji,l}^(t)) ← SUPPORT(π_i^(t−1), π_j^(t−1))
17:                 end for
18:                 π_i^(t) ←UPDATE(π_i^(t−1), β_{1i}^(t), ···, β_{mi}^(t))
19:             end for
20:             if CONVERGENCE(π_i^(t−1), π_i^(t) ∀i) then
21:                 return SUMRULE(π_i^(t) ∀i)
22:             end if
23:         end for
24:         return SUMRULE(π_i^(t) ∀i)
25:     end function
26:
27:     function SUPPORT(π_i^(t−1), π_j^(t−1))
28:         for k = 1 to l do
29:             β_{ji,k}^(t) ← π_{jk}^(t−1) / (1 + π_{ik}^(t−1) |log((π_{ik}^(t−1) + ε_0)/(π_{jk}^(t−1) + ε_0))|)
30:         end for
31:         return (β_{ji,1}^(t), β_{ji,2}^(t), ···, β_{ji,l}^(t))
32:     end function
33:
34:     function CONVERGENCE(π_i^(t−1), π_i^(t) ∀i)
35:         difference ← 0
36:         for i = 1 to m do
37:             difference ← difference + ‖π_i^(t) − π_i^(t−1)‖_2
38:         end for
39:         return difference < mε
40:     end function
41:
42:     function UPDATE(π_i^(t−1), β_{1i}^(t), ···, β_{mi}^(t))
43:         π_i^(t) ← (0, 0, ···, 0)
44:         α_i^(t) ← 0
45:         for j ≠ i do
46:             for k = 1 to l do
47:                 π_{ik}^(t) ← π_{ik}^(t) + π_{ik}^(t−1) β_{ji,k}^(t)
48:                 α_i^(t) ← α_i^(t) + π_{ik}^(t)
49:             end for
50:         end for
51:         return π_i^(t) / α_i^(t)
52:     end function
53:
54:     function SUMRULE(π_i^(t) ∀i)
55:         c ← 0
```

Stellenbosch University https://scholar.sun.ac.za



(a) (b) (c)

Figure 11.1: *Example of dissimilarities between probabilities (as defined in Equation 11.2.4) showing for which relationship of input probabilities the scaling factor is large, and for which it is small.* ***(11.1a)****: The absolute value of the log-ratio defined in Equation 11.2.5.* ***(11.1b)****: The dissimilarity (i.e. scaled absolute value of the log-ratio) defined in Equation 11.2.5.* ***(11.1c)****: Supports between probabilities.*

56: $\varpi \leftarrow 0$
57: **for** $k = 1$ **to** l **do**
58: $\varpi_k \leftarrow 0$
59: **for** i **do**
60: $\varpi_k \leftarrow \varpi_k + \pi_{ik}^{(t)}$
61: **end for**
62: $\varpi_k \leftarrow \frac{\varpi_k}{m}$
63: **if** $\varpi_k > \varpi$ **then**
64: $\varpi \leftarrow \varpi_k$
65: $c \leftarrow k$
66: **end if**
67: **end for**
68: **return** $((\varpi_1, \varpi_2, \cdots, \varpi_l), c)$
69: **end function**
70: **end class**

---

where we set the convergence threshold ε to 10^{-6}. Let $\pi^{(t)}$ denote the combined distribution.

From the initial probability distributions (i.e. classifier outputs) above illustrated, it can be seen that there is a disagreement between the distributions on the class of the object of interest: the first distribution has high belief for the second class while the second distribution assigns high belief to the first class.

The results when executing Algorithm 11.1 are shown in Table 11.1 (probability vectors or beliefs) and Table 11.2 (supports between beliefs). In this example, consensus is reached at the end of iteration $t = 7$.

It can be observed that towards convergence (from iteration $t = 4$), the first component of supports is getting close to 1 and the second component of supports is getting close to 0. As was mentioned before, with our proposed method, the component of supports on the class which is likely to be the consensus class increases and the components of supports on other

Stellenbosch University https://scholar.sun.ac.za

t	$\pi_1^{(t)}$	$\pi_2^{(t)}$	$\pi^{(t)}$	$\sum_{i=1}^{2} \|\pi_i^{(t)} - \pi_i^{(t-1)}\|$
0	$(0.3, 0.7)$	$(0.8, 0.2)$	$(0.55, 0.45)$	–
1	$(0.7130, 0.2870)$	$(0.5458, 0.4542)$	$(0.6294, 03706)$	0.9435
2	$(0.7394, 0.2606)$	$(0.7589, 0.2411)$	$(0.7491, 0.2509)$	0.3387
3	$(0.8994, 0.1006)$	$(0.8992, 0.1008)$	$(0.8993, 0.1007)$	0.4247
4	$(0.9876, 0.0124)$	$(0.9876, 0.0124)$	$(0.9876, 0.0124)$	0.2498
5	$(9.9984 \times 10^{-1}, 1.5727 \times 10^{-4})$	$(9.9984 \times 10^{-1}, 1.5727 \times 10^{-4})$	$(9.9984 \times 10^{-1}, 1.5727 \times 10^{-4})$	0.0346
6	$(9.9999 \times 10^{-1}, 2.4740 \times 10^{-8})$	$(9.9999 \times 10^{-1}, 2.4740 \times 10^{-8})$	$(9.9999 \times 10^{-1}, 2.4740 \times 10^{-8})$	0.0004
7	$(9.9999 \times 10^{-1}, 6.1208 \times 10^{-16})$	$(9.9999 \times 10^{-1}, 6.1208 \times 10^{-16})$	$(9.9999 \times 10^{-1}, 6.1208 \times 10^{-16})$	6.9976×10^{-16}

Table 11.1: *Evolution of probability distributions over iterations of the Yayambo algorithm.*

t	$\beta_1^{(t)}$	$\beta_2^{(t)}$
0	$(0.6184, 0.1067)$	$(0.1682, 0.5600)$
1	$(0.4585, 0.4105)$	$(0.6224, 0.2376)$
2	$(0.7445, 0.2364)$	$(0.7251, 0.2558)$
3	$(0.8990, 0.1007)$	$(0.8992, 0.1006)$
4	$(0.9876, 0.0124)$	$(0.9876, 0.0124)$
5	$(9.9984 \times 10^{-1}, 1.5727 \times 10^{-4})$	$(9.9984 \times 10^{-1}, 1.5727 \times 10^{-4})$
6	$(9.9999 \times 10^{-1}, 2.4740 \times 10^{-8})$	$(9.9999 \times 10^{-1}, 2.4740 \times 10^{-8})$

Table 11.2: *Evolution of supports of distributions over iterations of the Yayambo algorithm.*

classes decreases. Consequently, the consensus class (the first class) is assigned a probability of almost 1 as shown in $\pi_i^{(7)}$ above.

A detailed example of the execution of the Yayambo algorithm can be found in Appendix D.

11.3 Complexity

We now consider the computational complexity of the Yayambo algorithm. Computing the support $\beta_{ij,k}^{(t)}$ between l class labels, c_k, of two beliefs is $\mathcal{O}(l)$ (Equation 11.2.5). Each iteration requires m^2 computations of $\beta_{ij}^{(t)}$—one for every pair of probability distributions. Assuming that T is the number of iterations until termination of the algorithm, the complexity of Yayambo method is thus $\mathcal{O}(Tm^2 l)$.

The computational complexity of the sum rule and product rule methods is $\mathcal{O}(ml)$, and of the majority vote rule and the Borda count rule is $\mathcal{O}(ml^2)$. In terms of computational cost, the Yayambo method is more costly than the benchmark fusion methods. However, for typical m and T, this complexity is hardly noticeable.

Stellenbosch University https://scholar.sun.ac.za

11.4 Conclusion

Recognition of objects of interest is required in many applications, including search and rescue by solitary robots. This chapter proposed a classifier fusion method. We considered situations where no information about classifiers' performances is provided, only their probabilistic outputs. Individual classifiers may provide different opinions on the type of the same object of interest. Distributions of different classifiers are combined by iteratively assigning support to them until they converge or after the maximum number of iteration is reached. An asymmetric notion of support is used to assign credences to probability distributions, so convergence of distributions means that a consensus decision is reached. While consensus belief does not necessarily mean correct belief, achieving consensus from different initial distributions using an asymmetric support concept may indicate something good about their final judgement on the type of the object of interest.

Unlike existing fusion methods based on the principle of indifference, Yayambo uses and processes the similarities and discrepancies of the classifier outputs before making the final combination. The similarities of the classifier outputs capture regions where classifiers have a fair amount of confidence, and their discrepancies capture regions where they have contradicting views, which will be used to update their initial predictions.

Yayambo assigns a score of almost 1 to the consensus class and of almost 0 to other classes, and is thus an over-confident fusion method and not suitable when calibrated fused probabilities are desired. Our proposed fusion method is better situated to situations where downstream tasks require a commitment to a single object type.

Having described our proposed method for classifier fusion, the following chapter presents and discusses results for classifier fusion.

Stellenbosch University https://scholar.sun.ac.za

Chapter 12

Metrics, results and discussion

In this chapter, we present and discuss results evaluating the proposed classifier fusion method. We consider situations where no information about the performance of classifiers is available, only their outputs. Because of the lack of additional information that can be used to weight classifiers' outputs differently, our proposed classifier fusion method will be compared to the sum rule [51, 52]—a fusion method based on the principle of indifference. We will also consider the product rule, majority vote rule and Borda count rule (as discussed in Section 10.2) in some cases. All the methods we consider are based on the principle of indifference (i.e. classifiers are considered equally), although Yayambo weights classifiers' outputs.

Summary on the classifier fusion methods

- The sum rule averages classifiers' outputs to get a fused decision.
- The product rule performs element-wise multiplications across classifiers' outputs to get a fused decision.
- The majority vote rule performs element-wise votes across classifiers' outputs to get a fused decision.
- The Borda count rule uses a preferential ballot to rank classes in a sequence on the ordinal scale and then performs element-wise votes across these values to get a fused decision.

Stellenbosch University https://scholar.sun.ac.za

- In the Yayambo algorithm, before the fusion of classifiers' outputs occurs, initial probability distributions are collected and iteratively updated based on their similarities until consensus is reached.

12.1 Metrics

There are many ways to measure the quality of a classifier fusion method. One way of evaluating classifier fusion methods is to consider the quality of the fused classifiers, using traditional approaches (i.e. performance metrics) for assessing classifier quality. The most commonly used is accuracy, but other popular metrics include cross-entropy loss and multi-class precision [88].

To compare our classifier fusion method to the benchmark methods, we consider five different performance metrics: accuracy, multi-class precision, multi-class recall, F_1-score and cross-entropy loss. We use these metrics to describe the behaviour of the classifier fusion approaches as one metric may not fully do so. These five metrics, defined later in this section, convey different information which can help us to make a fair decision on the performance of a classifier.

It was mentioned earlier that a classifier aims to assign an object to its true category, but it can fail. Numbers of successful and unsuccessful assignments by a classifier are generally represented by a confusion matrix. Let N be the number of observations in some data set, and l the number of classes. Consider the confusion matrix M given by (for each technique)

$$M = \begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_l \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & \cdots & c_l \end{array} \\ \begin{pmatrix} M_{11} & M_{12} & \cdots & M_{1l} \\ M_{21} & M_{22} & \cdots & M_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ M_{l1} & M_{l2} & \cdots & M_{ll} \end{pmatrix} \end{array}. \tag{12.1.1}$$

In Equation 12.1.1, rows of M represent ground truth classes and columns represent predicted classes. For example, M_{11} indicates the number of test observations in class c_1 correctly classified while M_{12} indicates the number of test observations in class c_1 incorrectly classified as instances of class c_2. Let p_{ik} denote the predicted probability that observation i is of class k, and y_i the correct class of observation i so that $y = (y_1, y_2, \cdots, y_N)$ is the sequence

Stellenbosch University https://scholar.sun.ac.za

of correct classes of the N observations. Let Y be a label binarization or "one-hot encoding" of y [11]. The resulting matrix, composed of entries Y_{ik} for observation i and class label c_k, is given by

$$Y_{ik} = \begin{cases} 1 & \text{if } y_i = c_k \,, \\ 0 & \text{otherwise} \,. \end{cases} \tag{12.1.2}$$

For precision, recall and F_1-score, we average over each class as positive class, since these measures depend on class (i.e. they are sensitive to choice of the positive class). The metrics are defined as follows. The accuracy (ACC) measures the overall proportion of correct assignments of labels to objects and is given by

$$ACC = \frac{1}{N} \sum_{k=1}^{l} M_{kk} \,. \tag{12.1.3}$$

For precision and recall, we use their average over various class labels as the positive class. The averaged precision or positive predictive value (PPV) measures the average of the proportion of objects labelled as each class that are actually members of that class and is given by

$$PPV = \frac{1}{l} \sum_{k=1}^{l} P_k \,, \tag{12.1.4}$$

where the precision on a class c_k is

$$P_k = \frac{M_{kk}}{\sum_{j=1}^{l} M_{jk}} \,. \tag{12.1.5}$$

The averaged recall or true positive rate (TPR) evaluates the average of the proportion of actual class members that are correctly classified, and is given by

$$TPR = \frac{1}{l} \sum_{k=1}^{l} R_k \,, \tag{12.1.6}$$

where the recall on a single class c_k is

$$R_k = \frac{M_{kk}}{\sum_{j=1}^{l} M_{kj}} \,. \tag{12.1.7}$$

The averaged F_1-score (F_1) is the harmonic mean of averaged precision and averaged recall, and is given by

$$F_1 = 2 \times \frac{PPV \times TPR}{PPV + TPR} \,. \tag{12.1.8}$$

Stellenbosch University https://scholar.sun.ac.za

Finally, the cross-entropy loss C measures the performance of a probabilistic classification model (it increases as the predicted probability vector diverges from the actual one-hot encoded label) and is given by

$$C = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{l} Y_{ik}\log p_{ik}\,. \tag{12.1.9}$$

12.2 Simulation description and experimental setup

This section describes what we intend to present and discuss to assess the performance of our proposed classifier fusion method. It also describes our implementation and experimental setup used for our simulations.

12.2.1 Simulation description

Due to their probabilistic formulation, we expect the sum rule and product rule to outperform our proposed method in terms of cross-entropy loss which corresponds to a cross-entropy loss for a probabilistic model. Our hope is that our method will outperform the sum rule, the product rule, the majority vote rule and the Borda count rule in terms of accuracy. Our experiments considered the following cases for each data set, in an attempt to comprehensively test the proposed approach and identify its limitations:

(1) We will train different types of classifiers on exactly the same training data set, apply the fusion techniques and compare the original and fused classifiers on the test data set. This should be an effective way of comparing the fusion techniques if the classifiers are really diverse. Also, Equation 11.2.5 requires a parameter ε_0 during the evaluation of support between distributions and this parameter can influence the final decision. We will vary the values of ε_0 to see how it influences the final decision.

(2) We will train the same type of classifier with different parameters on the same training data set, apply the fusion techniques and compare the original and fused classifiers on the test data set. This investigation is meant to see the impact on the fusion technique when classifiers are trained with different parameters.

Stellenbosch University https://scholar.sun.ac.za

(3) We will also train different types of classifiers on different subsets of the training data set and compare the performances of the fusion methods on the test data set. This is almost like using classifiers with different expertises. This way of training classifiers may produce highly diverse classifiers, which is the type of investigation we are interested in. Data points of subsets are randomly chosen without replacement.

(4) We will also train different classifiers on different subsets of features and see how the fusion methods respond. This corresponds to robotics where sensors can detect different aspects of an object. We are keen to see how the fusion techniques will behave in such cases. Features are randomly chosen without replacement.

(5) We will also consider fusion of an increasing number of classifiers and see how the various classifier fusion methods behave as the number of classifiers increases.

(6) Another aspect will be to verify the active part of our thesis—it was said that solitary robots (the type of robots we consider in this thesis) are active, which means they amend their decisions as additional information is provided. To capture an object in active vision, only a limited number of viewpoints of the object are first captured and gradually more viewpoints are added. The method applied for the collection of one of the data sets we consider (the Columbia data set) corresponds to how active experiments are often done. So we will show the behaviour of classifier fusion techniques as the number of viewpoints of an object increases.

Finally, we will consider some artificial classifier outputs with potentially extreme disagreement, where our assumptions of regularity of trained classifiers (discussed in Chapter 11) may be violated.

12.2.2 Experimental setup

For our experiments, we use Python and the `scikit-learn` library.[1] We used five types of classifiers—classifiers with different behaviours captured in their performances. Any classifier which outputs a probability distribution over classes can be used; the classifiers which we considered in these experiments were:

[1] An open library for machine learning [11].

Stellenbosch University https://scholar.sun.ac.za

- logistic regression (LR) fit with `liblinear` [27] (f_1),
- k-nearest neighbours (k-NN) (f_2),
- a support vector machine (SVM) with the sigmoid function kernel (f_3),
- an SVM with the radial basis function (RBF) kernel (f_4), and
- a multi-layer perceptron (MLP) trained with stochastic gradient descent (f_5).

With the `scikit-learn` library, by the use of softmax function [10], classifiers' outputs are provided in forms of probabilities over classes, except k-nearest neighbours. For k-nearest neighbours, a vote mechanism is used to output a probability distribution. The hyperparameters we selected to train these classifiers are given in Tables 12.1, 12.2 and 12.3. These hyperparameters were deliberately tuned to produce highly similar as well as highly diverse classifiers. For other hyperparameters, we used default values provided by `scikit-learn`.

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
f_1	$C = 2$	$C = 30$	$C = 0.2$	$C = 0.002$	$C = 0.05$	$C = 0.002$	$C = 0.002$
f_2	$k = 50$	$k = 15$	$k = 2000$	$k = 500$	$k = 150$	$k = 500$	$k = 80$
f_3	$C = 200$	$C = 80$	$C = 0.05$	$C = 0.03$	$C = 0.8$	$C = 0.005$	$C = 0.005$
f_4	$C = 300$	$C = 20$	$C = 0.03$	$C = 0.015$	$C = 0.5$	$C = 300$	$C = 300$
f_5	$H = 5$ $\lambda = 0.2$ $T = 200$	$H = 5$ $\lambda = 0.2$ $T = 200$	$H = 5$ $\lambda = 0.2$ $T = 300$	$H = 5$ $\lambda = 0.2$ $T = 300$	$H = 5$ $\lambda = 0.2$ $T = 500$	$H = 5$ $\lambda = 0.2$ $T = 500$	$H = 5$ $\lambda = 0.045$ $T = 500$

Table 12.1: *Hyperparameters used to train classifiers. For f_1, f_3 and f_4, C denotes the penalty parameter of the error term. For the second classifier, which is a k-nearest neighbours classifier, k denotes the number of neighbours. For the last classifier, which is an artificial neural network, H, λ and T denote the number of hidden layers, the learning rate and the number of epochs for training.*

For the first investigation (we mentioned seven investigations above), we considered various values for the parameter ε_0 required in Equation 11.2.5. For the remaining investigations, the convergence threshold ε_0 was set to 10^{-6}. Over all the experiments, the number of iterations for Yayambo to reach convergence was in the range $[5, 23]$.

Stellenbosch University https://scholar.sun.ac.za

Data sets	LR	k-NN	SVM	MLPs
Iris	$C = 100$ solver=newton-cg	$k = 20$, w=uniform solver=ball tree	$C = 100$ solver=rbf	$H = 5, \lambda = 0.1, T = 500$ solver=lbfgs, σ=tanh
Gesture	$C = 0.000001$ solver=lbfgs	$k = 100$, w=distance solver=ball tree	$C = 1$ solver=poly	$H = 1, \lambda = 0.045, T = 300$ solver=sgd, σ=relu
Activity	$C = 0.000001$ solver=liblinear	$k = 800$, w=uniform solver=ball tree	$C = 20$ solver=sigmoid	$H = 3, \lambda = 0.1, T = 500$ solver=adam, σ=logistic
Satellite	$C = 0.01$ solver=sag	$k = 1000$, w=distance solver=kd tree	$C = 0.01$ solver=poly	$H = 5, \lambda = 0.2, T = 500$ solver=lbfgs, σ=tanh
Digits	$C = 0.0000001$ solver=saga	$k = 2000$, w=distance solver=brute	$C = 1000$ solver=sigmoid	$H = 2, \lambda = 0.1, T = 100$ solver=adam, σ=relu
Occupancy	$C = 0.00001$ solver=lbfgs	$k = 1500$, w=uniform solver=kd tree	$C = 1$ solver=rbf	$H = 3, \lambda = 0.02, T = 300$ solver=sgd, σ=logistic
Columbia	$C = 0.25$ solver=liblinear	$k = 200$, w=uniform solver=brute	$C = 0.25$ solver=poly	$H = 5, \lambda = 0.3, T = 400$ solver=lbfgs, σ=logistic

Table 12.2: *Hyperparameters used to train similar classifiers on the subsets of data sets. For support vector machines, C denotes the penalty parameter of the error term. For k-nearest neighbours classifier, k denotes the number of neighbours. For artificial neural network, H, λ and T denote the number of hidden layers, the learning rate and the number of epochs for training.*

Techniques	h_1	h_2	h_3	h_4	h_5
MLP	$H = 10$ solver=sgd $\lambda = 0.045$ $T = 500$ σ=relu	$H = 3$ solver=adam $\lambda = 0.5$ $T = 100$ σ=relu	$H = 3$ solver=lbfgs $\lambda = 0.5$ $T = 100$ σ=tanh	$H = 3$ solver=adam $\lambda = 0.1$ $T = 500$ σ=logistic	$H = 2$ solver=sgd $\lambda = 0.045$ $T = 500$ σ=sigmoid
k-NN	$k = 30$ w=uniform solver=ball tree	$k = 60$ w=distance solver=ball tree	$k = 90$ w=uniform solver=kd tree	$k = 120$ w=uniform solver=brute	$k = 150$ w=distance solver=brute
SVM	$C = 3$ kernel=rbf tol= 0.001 degree= 3	$C = 0.000001$ kernel=linear tol= 0.01 degree= 4	$C = 0.1$ kernel=poly tol= 0.1 degree= 5	$C = 100$ kernel=sigmoid tol= 0.001 degree= 10	$C = 1$ kernel=rbf tol= 0.001 degree= 10

Table 12.3: *Hyperparameters used to train similar classifiers on the Columbia data set. For support vector machines, C denotes the penalty parameter of the error term. For k-nearest neighbours classifier, k denotes the number of neighbours. For artificial neural network, H, λ and T denote the number of hidden layers, the learning rate and the number of epochs for training.*

12.2.3 Data sets

We compared the Yayambo algorithm, the sum rule, the product rule, the majority vote rule and the Borda count rule on six data sets from the UCI Repository [6], and one from from the Columbia Image Library [78]: Iris; Gesture; Activity recognition; Handwritten digits; Satellite; Occupancy and Columbia. Some characteristics of these data sets are given in Table 12.4.

Stellenbosch University https://scholar.sun.ac.za

Data set	Training	Test	Attributes	Classes
Iris	105	45	4	3
Gesture	785	336	19	5
Activity recognition	4725	2025	6	7
Handwritten digits	10000	3000	784	10
Satellite	4435	2000	36	6
Occupancy	8143	2665	5	2
Columbia	765	315	97	15

Table 12.4: *The characteristics of the data set used to train classifiers.*

Further descriptions of the various data sets can be found in Appendix E.

For the Columbia data set, which consists of images of objects, each object was captured from different views (72 views, at 5° intervals around the object, in total). This yields 72 images per object. In our experiment, we only consider 15 types of objects. We convert the RGB images, which we flatten to a sequence of 16384 grayscale values. We use PCA for feature reduction where we retained 97 components. We build the PCA model on all the training data.

For the Columbia data set, it is expected that the more views of the object we consider, the better performance we can achieve. In our experiments, we will consider some limited number of views of the objects and see how the number of views of an object will impact the performance of the classifier fusion methods.

To make a final classification decision using the classifier fusion approaches (sum rule, product rule, majority vote rule, Borda count rule and Yayambo), we select the class with the largest posterior probability.

12.3 Results and discussion

12.3.1 Classifiers trained on the same data set

The results of the five metrics we considered when applying the the majority vote rule, the Borda count rule, the sum rule, the product rule and the Yayambo fusion technique for classifiers trained on the same data set are presented in Tables 12.5–12.7.

While almost all the fusion algorithms exhibited perfect accuracy, precision, recall and F_1-score values on the Iris test data set (the Iris data set is

Stellenbosch University https://scholar.sun.ac.za

especially "easy" because it is very clean), the decisions resulting from the Yayambo method outperformed those from the sum rule, the product rule, the majority vote rule and the Borda count rule on all the other data sets for all of these metrics.[2] Our results also confirm those of Kittler [51] in that the sum rule generally outperformed the product rule and the majority vote rule on these metrics. We see that all five fusion methods behave similarly when all of the individual classifiers are strong. The Yayambo method is most beneficial when the predictions are highly diverse, i.e. the classifiers give different opinions for the same input. This reflects in better fusion performance when combining classifiers with more widely differing performance levels, such as in the Columbia data set in Table 12.5. When classifiers were highly diverse, Yayambo outperformed the sum rule, product rule, majority vote rule and Borda count rule methods more notably in terms of accuracy. These results indicate that it is beneficial to use Yayambo as it provides robustness to the quality of individual classifiers (when accuracy, precision, recall and F_1-score are considered).

We also considered various values for ε_0 as required in Equation 11.2.5 (see Table 12.5). The results show that this parameter can influence the final fusion decision using Yayambo. For various values $\varepsilon_0 \leq 10^{-4}$, Yayambo yields the same performances. But its performance starts degrading when increasing ε_0. In Table 12.5 for instance, Yayambo achieves poorer accuracies for $\varepsilon_0 = 10^{-3}$ than for $\varepsilon_0 > 10^{-3}$ on Activity. Also, Yayambo's accuracies for $\varepsilon_0 = 10^{-2}$ are all poor on most test data. This indicates that the performance of Yayambo is robust to a range of small values for ε_0, but degrades (not badly) once these values become too large, as expected. Also, the choice of values for ε_0 does not impact the number of iterations to convergence.

As expected, the sum rule and product rule outperformed Yayambo in terms of cross-entropy loss (see Table 12.6) in all cases except the Iris data set, where no prediction errors were made. This is because the sum rule and product rule methods each return a sort of average of the probability distributions of individual classifiers, so that these two methods will not return overconfident predictions. Thus, in the case of misclassified observations, the contribution to the loss with the sum rule and product rule are limited. On the other hand, Yayambo's consensus-seeking approach leads to over-

[2] The only exception was a tie in the precision of the product rule and Yayambo on the Activity data set.

Stellenbosch University https://scholar.sun.ac.za

confident classifications, with extremely high corresponding loss values in the case of misclassification: this results from the probability of the correct class being driven down to zero. These high loss values typically easily outweigh the reduction in loss caused by more confident correct classifications. It was also observed that, in most cases, one or more individual classifiers performed better than all the fusion methods in terms of cross-entropy loss. It should be noted that there is no clear explanation of why fusion methods achieve poor results in terms of cross-entropy loss, a clearer explanation requires a more in-depth investigation.

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
f_1	0.9111	0.6845	0.7491	0.5675	0.9063	0.6353	0.8762
f_2	0.9556	0.6667	0.6642	0.7885	0.8753	0.9295	0.6635
f_3	0.9778	0.6488	0.6104	0.6925	0.9047	0.6353	0.5270
f_4	0.9778	0.6101	0.6459	0.6785	0.9117	0.8912	0.6444
f_5	0.9778	0.6815	0.5012	0.6280	0.8217	0.6353	0.7268
Majority vote rule	0.9777	0.6455	0.7356	0.8090	0.8905	0.9033	0.8878
Borda count rule	**1.0000**	0.6728	0.7446	0.8122	0.8886	0.9133	0.8996
Product rule	**1.0000**	0.6665	0.7455	0.8110	0.8905	0.9042	0.8852
Sum rule	**1.0000**	0.6875	0.7644	0.8010	0.9153	0.9163	0.9048
Yayambo ($\varepsilon_0 = 10^{-6}$)	**1.0000**	**0.7054**	**0.7788**	**0.8120**	**0.9167**	**0.9365**	**0.9535**
Yayambo ($\varepsilon_0 = 10^{-5}$)	**1.0000**	**0.7054**	**0.7788**	**0.8120**	**0.9167**	**0.9365**	**0.9535**
Yayambo ($\varepsilon_0 = 10^{-4}$)	**1.0000**	**0.7054**	**0.7788**	**0.8120**	**0.9167**	**0.9365**	**0.9535**
Yayambo ($\varepsilon_0 = 10^{-3}$)	**1.0000**	**0.7054**	0.7786	**0.8120**	**0.9167**	**0.9365**	**0.9535**
Yayambo ($\varepsilon_0 = 10^{-2}$)	**1.0000**	0.6996	0.7600	0.8092	0.9111	**0.9365**	0.9444

Table 12.5: *The majority vote rule, Borda count rule, sum rule, product rule and Yayambo accuracies on test data with various values of ε_0 for Yayambo. Bold values indicate best performance.*

We found that Yayambo is over-confident in its predictions as it assigns a score of almost 1 to the consensus class, so any misclassification will increase the cross-entropy loss disproportionally to the change in accuracy.

The Borda count rule and majority vote rule were outperformed by other fusion methods in various experiments run. A potential weakness of the Borda count rule and majority vote rule is that the specific values of the probabilities are ignored when performing the fusion (although this might make them robust to outliers or overconfident classifiers).

In Tables 12.5–12.7, we considered five fusion methods: majority vote rule, Borda count rule, sum rule, product rule and Yayambo. In what follows, we will focus almost exclusively on Yayambo and the sum rule. We want to focus on Yayambo and the sum rule because the sum rule was re-

Stellenbosch University https://scholar.sun.ac.za

Model	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
f_1	0.4798	0.8798	0.9229	1.3969	0.4225	0.5783	2.7037
f_2	0.4451	1.1447	1.2374	**0.5073**	0.4451	**0.1597**	**0.8634**
f_3	0.0891	0.8817	0.7077	0.5501	0.2899	0.2133	2.7086
f_4	0.1125	0.8791	**0.6057**	0.5532	**0.2720**	0.3716	1.2267
f_5	0.0495	0.8786	0.6889	0.8709	0.7405	0.7304	6.7406
Borda count rule	0.0512	0.8574	1.2443	1.4677	1.0010	0.9000	1.2442
Majority vote rule	0.2012	0.8657	0.8888	1.2282	0.6767	1.1989	1.2111
Product rule	0.1021	0.8541	0.9542	1.2564	0.9985	1.2564	1.2020
Sum rule	0.2135	**0.7485**	0.7586	0.6459	0.3304	0.2873	1.1151
Yayambo ($\varepsilon_0 = 10^{-6}$)	**0.0000**	10.7262	3.5044	3.0495	1.6125	0.6944	1.9882
Yayambo ($\varepsilon_0 = 10^{-5}$)	**0.0000**	10.7262	3.5044	3.0495	1.6125	0.6944	1.9882
Yayambo ($\varepsilon_0 = 10^{-4}$)	**0.0000**	10.7262	3.5044	3.0495	1.6125	0.6944	1.9882
Yayambo ($\varepsilon_0 = 10^{-3}$)	**0.0000**	10.7262	4.5584	5.8452	2.7445	0.6944	1.9882
Yayambo ($\varepsilon_0 = 10^{-2}$)	**0.0000**	14.5241	4.5896	5.2002	2.4214	0.6944	1.9882

Table 12.6: *The majority vote rule, Borda count rule, sum rule, product rule and Yayambo cross-entropy losses on test data. Bold values indicate best performance.*

ported to be the best fusion method [51] for methods based on principle of indifference—it was also found in our experiments that the product rule, the majority vote rule and the Borda count rule typically achieve poorer results than the sum rule and Yayambo.

Also in our further experiments, we will only consider two metrics: accuracy and loss. In our experiments, Yayambo was found to achieve the best on the other three metrics. So to avoid repeating the same conclusions, the results on these other metrics will be omitted.

12.3.2 Similar classifiers trained with different parameters on the same data set

Results of the Borda count rule, majority vote rule, product rule, sum rule and Yayambo on the Columbia data set using the same classifiers with different hyperparameters are presented in Table 12.8. Here h_1 to h_5 are all classifiers of the same type—denoted by the row heading—with hyperparameters as specified in Table 12.3. The two fusion methods yielded the same accuracy with k-NN classifiers, with two of the individual classifiers outperforming both classifier fusion methods. Yayambo outperformed the sum rule with MLP and SVM classifiers, where the two fusion methods outperformed all the individual classifiers. In Table 12.8, individual classifiers have widely differing outputs for the test observations and generally have poor performance. The fusion methods leverage these differences from in-

Stellenbosch University https://scholar.sun.ac.za

Data set	Methods	Precision	Recall	F_1-score
Iris	Majority vote rule	0.9800	0.9800	0.9800
	Borda count rule	**1.0000**	**1.0000**	**1.0000**
	Sum rule	**1.0000**	**1.0000**	**1.0000**
	Product rule	**1.0000**	**1.0000**	**1.0000**
	Yayambo	**1.0000**	**1.0000**	**1.0000**
Gesture	Majority vote rule	0.6700	0.6800	0.6100
	Borda count rule	0.6600	0.6800	0.6000
	Sum rule	0.7000	0.6900	0.6500
	Product rule	0.6900	0.6900	0.6400
	Yayambo	**0.7100**	**0.7100**	**0.6900**
Activity	Majority vote rule	0.7700	0.7500	0.7500
	Borda count rule	**0.7800**	**0.7800**	**0.7800**
	Sum rule	0.7700	0.7600	0.7600
	Product rule	**0.7800**	0.7700	0.7600
	Yayambo	**0.7800**	**0.7800**	**0.7800**
Satellite	Majority vote rule	0.7900	0.7900	0.7800
	Borda count rule	0.7800	0.7900	0.7900
	Sum rule	0.8000	0.8000	0.8000
	Product rule	0.7800	0.7800	0.7900
	Yayambo	**0.8200**	**0.8100**	**0.8100**
Digits	Majority vote rule	0.9100	0.9100	0.9100
	Borda count rule	0.9100	0.9100	0.9100
	Sum rule	0.9100	0.9100	0.9100
	Product rule	0.9100	0.9100	0.9100
	Yayambo	**0.9200**	**0.9200**	**0.9200**
Occupancy	Majority vote rule	0.9000	0.9100	0.9200
	Borda count rule	0.9000	0.9000	0.9200
	Sum rule	0.9300	0.9200	0.9200
	Product rule	0.9000	0.9100	0.9200
	Yayambo	**0.9600**	**0.9500**	**0.9500**
Columbia	Majority vote rule	0.9200	0.9000	0.8900
	Borda count rule	0.9300	0.8900	0.8800
	Sum rule	0.9300	0.9000	0.9000
	Product rule	0.9300	0.8900	0.8800
	Yayambo	**0.9500**	**0.9400**	**0.9400**

Table 12.7: *Precision, recall (the true positive rate), and F_1 score for the fusion techniques on the seven benchmark data set. The Yayambo approach performed best on all metrics for all these data sets. Bold values indicate best performance. As in Tables 12.5 and 12.6, there is no much difference with various values of ε_0. For the performances reported here, ε_0 was set to 10^{-6}.*

dividual classifiers to achieve better prediction.

Since differences observed in classifiers' performances mean that classifiers provide different views on some data points (i.e. classifiers disagree), Yayambo is a good fusion method for fusion of highly diverse classifiers' outputs. This indicates that it is beneficial to use Yayambo as it provides robustness in terms of the quality (i.e. accuracy) of individual classifiers.

In Table 12.8, k-NN outperforms all the fusion methods. For the Yayambo

Stellenbosch University https://scholar.sun.ac.za

Techniques	MLP	k-NN	SVM
h_1	0.6444	**0.8381**	0.6444
h_2	0.4635	0.8127	0.7937
h_3	0.4667	0.5937	0.9534
h_4	0.4857	0.4762	0.6095
h_5	0.1238	0.7400	0.1048
Borda count rule	0.6052	0.6873	0.9016
Majority vote rule	0.7014	0.7714	0.7968
Product rule	0.6557	0.7714	0.9778
Sum rule	0.7486	0.7841	0.9810
Yayambo	**0.8000**	0.7841	**0.9905**

Table 12.8: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using similar classifiers with different hyperparameters trained, on the Columbia data set where h_i denotes the i-th instance of the corresponding classification method. Accuracy of individual classifier and fusion techniques on the Columbia data set. Bold values indicate best performance.*

fusion approach, we might hope that the exchange of supports might help us avoid this, but these results illustrate that it is quite possible for classifiers with low accuracies (i.e. h_3, h_4 and h_5) to support each other's erroneous predictions, outweighing support for the correct predictions by the other classifiers: recall that the classifiers do not have a prior notion of which other classifiers perform better.

12.3.3 Classifiers trained on different subsets of a data set

Results of our proposed fusion method and the benchmark fusion methods when classifiers are trained on different subsets of a data set are presented in Table 12.9. We observe that our fusion technique typically achieved better accuracy than the sum rule. Table 12.10 shows the data set sizes for training of classifiers on different data sets. Data points of subsets were randomly chosen without replacement. Tables 12.11–12.14 show the results of using the same classifier (with hyperparameters as specified in Table 12.2) trained on different subsets of the training data sets.

Table 12.9 shows that, when classifiers are trained on different subsets of a training data set, Yayambo again outperforms the sum rule on most data sets. They both yield the same accuracy on one data set; Yayambo outperforms the sum rule on four data sets and the sum rule outperforms Yayambo in two cases. When classifiers are trained on different data sets, they are more likely to have different views on the prediction of future test data points. If this is the case, the better performance of Yayambo over the

Stellenbosch University https://scholar.sun.ac.za

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
f_1	0.6889	0.3125	0.6089	0.7065	0.6420	0.8640	0.8381
f_2	0.9111	0.5268	0.7728	0.8220	0.8747	0.7809	0.5111
f_3	0.7556	0.3214	0.5807	0.2305	0.9080	0.7595	0.2032
f_4	0.9333	0.5833	0.6336	0.7995	**0.9403**	0.9808	0.5460
f_5	0.9556	0.4791	0.5032	0.7030	0.8933	**0.9852**	0.7302
Borda count rule	0.9122	0.6022	0.7022	0.8001	0.9014	0.9234	0.8015
Majority vote rule	0.9668	0.6211	0.7625	0.8044	0.8325	0.9419	0.9143
Product rule	**0.9778**	0.6328	0.7555	0.8220	0.9044	0.9499	0.8793
Sum rule	**0.9778**	0.6429	0.7644	0.8220	0.9143	0.9535	0.8793
Yayambo	**0.9778**	**0.6607**	**0.7802**	**0.8240**	0.8793	0.9519	**0.9143**

Table 12.9: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using classifiers trained on different subsets of data set. Bold values indicate best performance.*

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
f_1	95	544	2513	3492	9198	4761	619
f_2	57	401	3829	2700	5723	7176	540
f_3	76	473	3563	3024	6252	4271	441
f_4	95	528	2389	3898	8128	5009	620
f_5	56	727	3341	2875	5036	7972	517

Table 12.10: *Data set sizes for training of classifiers on different data sets.*

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
h_1	0.9556	0.2024	0.7807	0.8595	0.3023	0.9782	0.6317
h_2	0.9778	**0.4464**	0.7970	0.8440	0.3223	0.9782	0.6825
h_3	0.9556	0.2024	0.7709	0.8445	0.4987	**0.9786**	0.8349
h_4	0.9556	0.1815	0.7664	0.8420	0.3193	0.9782	0.7968
h_5	0.9556	0.1756	0.8040	0.8555	0.0930	**0.9786**	0.7302
Borda count rule	0.9556	0.3423	**0.8044**	0.8590	0.5530	0.9782	0.9397
Majority vote rule	**0.9778**	**0.4464**	0.8005	0.8565	0.4773	0.9782	0.8921
Product rule	0.9556	0.4196	0.8025	**0.8620**	**0.6890**	0.9782	**0.9587**
Sum rule	**0.9778**	0.4256	0.8005	0.8615	0.6550	0.9782	0.9365
Yayambo	**0.9778**	0.4107	0.8010	0.8605	0.6780	0.9782	0.9524

Table 12.11: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using MLPs trained on different data sets. Bold values indicate best performance.*

sum rule provides support to this view that Yayambo is more suitable for highly diverse classifiers, i.e. classifiers with differing views on some observations (it depends on the extent of the differences in classifier outputs).

Table 12.9 shows results of using various classifiers of the same type but with different hyperparameters when trained on different subsets of the training data sets. Unlike Table 12.9, Tables 12.11–12.14 show that when the same classifier type was trained on different subsets of a data set, Yayambo

Stellenbosch University https://scholar.sun.ac.za

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
h_1	**1.0000**	0.7083	0.7022	0.7440	**0.6170**	0.9006	0.1460
h_2	0.9556	0.5923	**0.7447**	0.7185	0.5550	**0.9212**	0.1175
h_3	0.9778	0.6875	0.7378	0.7275	0.5840	0.8998	0.0794
h_4	0.9778	0.6994	0.7131	**0.7575**	0.6037	0.9036	**0.3714**
h_5	0.9778	**0.7232**	0.7442	0.7245	0.4907	0.9152	0.1492
Borda count rule	0.9778	0.6905	0.7442	0.7275	0.5723	0.9036	0.1587
Majority vote rule	0.9778	0.6935	0.7462	0.7285	0.5723	0.9036	0.1873
Product rule	0.9778	0.6875	0.7536	0.7335	0.5763	0.9186	0.1048
Sum rule	0.9778	0.6935	0.7531	0.7335	0.5790	0.9186	0.0857
Yayambo	0.9778	0.6935	0.7546	0.7310	0.5727	0.9186	0.0984

Table 12.12: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using k-NN trained on different data sets. Bold values indicate best performance.*

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
h_1	0.9778	**0.5982**	0.8281	0.6850	0.9127	**0.9786**	**0.9937**
h_2	0.9778	0.4851	0.8272	**0.7045**	0.9077	**0.9786**	0.9841
h_3	0.9778	0.5387	0.8232	0.6945	0.9143	0.9782	0.9841
h_4	0.9778	0.5565	0.8277	0.6940	0.9113	0.9782	**0.9937**
h_5	0.9556	0.5952	**0.8291**	0.6735	0.9120	**0.9786**	0.9810
Borda count rule	0.9778	0.5685	0.8286	0.6895	0.9220	**0.9786**	0.9873
Majority vote rule	0.9778	0.5565	0.8281	0.6895	0.9213	**0.9786**	**0.9937**
Product rule	**1.0000**	0.5744	0.8242	0.6875	**0.9237**	**0.9786**	**0.9937**
Sum rule	**1.0000**	0.5744	0.8242	0.6870	0.9227	**0.9786**	**0.9937**
Yayambo	**1.0000**	0.5744	0.8252	0.6865	0.9227	**0.9786**	**0.9937**

Table 12.13: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using SVM trained on different data sets. Bold values indicate best performance.*

and the sum rule achieve almost identical performances. This might indicate that the both fusion methods are recommended when classifiers are highly similar. It is also observed that the other three fusion methods (i.e. the Borda count rule, majority vote rule and product rule) outperform the sum rule and Yayambo in some cases when using similar classifiers. In Table 12.11 for instance, the Borda count rule outperforms the other fusion methods on the Activity data set. Also the product rule outperforms the other fusion methods on the Satellite, Digits and Columbia data sets. This might indicate that all the fusion methods are recommended when classifiers are highly similar.

Stellenbosch University https://scholar.sun.ac.za

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
h_1	0.9778	0.6310	**0.7842**	0.7505	**0.1173**	**0.6353**	0.9619
h_2	**1.0000**	0.6310	**0.7842**	0.7385	**0.1173**	**0.6353**	0.9651
h_3	**1.0000**	0.6310	0.7822	0.7420	**0.1173**	**0.6353**	**0.9683**
h_4	**1.0000**	0.6280	0.7793	**0.7540**	**0.1173**	**0.6353**	0.9651
h_5	0.9778	**0.6399**	0.7827	0.7410	**0.1173**	**0.6353**	0.9651
Borda count rule	**1.0000**	0.6250	0.7812	0.7415	**0.1173**	**0.6353**	**0.9683**
Majority vote rule	**1.0000**	0.6250	0.7812	0.7420	**0.1173**	**0.6353**	**0.9683**
Product rule	**1.0000**	0.6250	0.7807	0.7435	**0.1173**	**0.6353**	0.9651
Sum rule	**1.0000**	0.6250	0.7807	0.7445	**0.1173**	**0.6353**	0.9651
Yayambo	**1.0000**	0.6250	0.7817	0.7425	**0.1173**	**0.6353**	**0.9683**

Table 12.14: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using logistic regression trained on different data sets. Bold values indicate best performance.*

12.3.4 Classifiers trained on different subsets of features

The comparison of our proposed fusion method and the benchmark fusion methods when classifiers are trained on different features of a data set is presented in Table 12.15. We observe that, for many of the data sets, most of the individual classifiers outperform the fusion methods, and f_4 (SVM with RBF kernel) outperformed the other classifiers in most of the data sets. This might mean that classifiers have different behaviours for different features, and f_4 works well for the selected features. This might indicate that when classifiers have learnt from different features, it would be desirable to randomly select and report results of a single classifier. (A classifier would need to be randomly chosen in this case because classifiers' performances on the training data sets are not available.) It is also observed that, in the Digits data set, all the fusion methods are almost as bad as the weakest individual classifier (i.e. f_1). This might mean that individual classifiers perform well in different regions of the input space, as illustrated in Table 12.18. We quantify disagreement between classifiers using mean and standard deviation of L^1 distance per point between classifiers' outputs.

When compared to disagreement between classifiers' outputs using the Activity data set (we choose the Activity data set for comparison because the number of classes in the Activity data set is close to the number of classes in the Digits data set) shown Table 12.17, Table 12.18 shows that individual classifiers have high disagreement on the Digits data set, as illustrated by high mean values (although their standard deviations are almost the same). High mean values indicate low agreement between classifiers based on clas-

Stellenbosch University https://scholar.sun.ac.za

sifiers' predictions. It should be noted that there is no clear explanation of why fusion methods achieve poor results, a clearer explanation requires a more in-depth investigation.

The results show that our fusion technique and the sum rule method achieve roughly the same average performance (accuracy in this case) on some data sets. Table 12.16 shows the number of features considered for each data set. Features were randomly selected without replacement.

Techniques	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
f_1	0.2667	0.3095	0.2893	0.2210	0.6423	0.6353	**0.8762**
f_2	0.6285	0.6190	0.7348	0.8535	0.8846	**0.8671**	0.6095
f_3	0.1206	0.5536	0.6128	0.8310	0.8566	0.6352	0.6413
f_4	**0.7238**	**0.9048**	**0.7550**	**0.8775**	**0.9376**	0.5101	0.6413
f_5	0.1333	0.6845	0.6083	0.8475	0.8930	0.5830	0.8190
Borda count rule	0.1223	0.4254	0.2622	0.4880	0.6910	0.6155	0.7587
Majority vote rule	0.1332	0.4624	0.2333	0.4877	0.6770	0.6001	0.7446
Product rule	0.1333	0.4524	0.2555	0.4887	0.6907	0.6111	0.7338
Sum rule	0.1333	0.4524	0.2622	0.4910	0.6910	0.6200	0.7587
Yayambo	0.1333	0.4672	0.2889	0.5300	0.6367	0.6000	0.8031

Table 12.15: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies on test data using different classifiers trained on different features of data set. Bold values indicate best performance.*

Data sets	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
n^-/n	2/4	12/19	4/6	25/36	299/784	2/5	54/97

Table 12.16: *The number of features considered to train classifiers on each data set where n indicates the total number of features of the corresponding data set and n^- the number of features selected. Features were selected without replacement.*

	f_2	f_3	f_4	f_5
f_1	0.6559 ± 0.1550	0.8972 ± 0.3734	0.7446 ± 0.3213	0.9320 ± 0.4045
f_2	–	0.3031 ± 0.2553	0.3793 ± 0.0781	0.5493 ± 0.2937
f_3	–	–	0.3365 ± 0.3920	0.4442 ± 0.5410
f_4	–	–	–	0.4587 ± 0.4273

Table 12.17: *Measurement of disagreement between classifiers, trained using the Activity data set, from a prediction point of view. We quantify disagreement between classifiers using mean and standard deviation of L^1 distance per point between classifiers' outputs.*

Stellenbosch University https://scholar.sun.ac.za

	f_2	f_3	f_4	f_5
f_1	0.6246 ± 0.2449	1.1341 ± 0.3270	0.8435 ± 0.2241	0.8903 ± 0.2006
f_2	–	1.2590 ± 0.3608	0.8276 ± 0.3131	0.8078 ± 0.4430
f_3	–	–	1.0605 ± 0.2305	1.4987 ± 0.6742
f_4	–	–	–	0.9271 ± 0.3704

Table 12.18: *Measurement of disagreement between classifiers, trained using the Digits data set, from a prediction point of view. We quantify disagreement between classifiers using mean and standard deviation of L^1 distance per point between classifiers' outputs.*

12.3.5 Increasing the number of classifiers

Table 12.19 shows the accuracies of the fusion methods when increasing the number m of classifiers. For the selection of classifiers, we sample $2 \leq m \leq 5$ classifiers (without replacement) and average classifier performances. Given m, we consider a single sampling of m classifiers and we apply the same selected classifiers in the different data sets. For $m = 2$ we used f_1 and f_2; for $m = 3$, we used f_2, f_3 and f_5; for $m = 4$, we used f_1, f_3, f_4 and f_5.

In Table 12.19, with two classifiers, the sum rule generally outperformed Yayambo in terms of accuracy. But for $m \geq 3$, Yayambo outperformed the sum rule almost across the board. (On the Iris and Occupancy data sets, both techniques yield almost identical results. This is probably because classifiers trained on these two data sets are fairly similar, i.e. they give almost the same accuracy.)

Obviously, the number of classifiers can influence the quality of the fused output. The quality of the fusion output in this case will also depend on the quality of each classifier.

To conclude this section, Yayambo is found to be more beneficial with more classifiers and it is more robust to one poor classifier.

12.3.6 An active vision case

Recall that coordination of solitary robots for search and rescue is the overarching challenge of this thesis, and one aspect mentioned in the description of solitary robots was active vision [1] (see Section 2.2). In an active vision system, agents are called to take decisions as additional information becomes available. An active vision system is one that can manipulate the viewpoints of an object to get better information from it. This section investigates active vision for classification, and we consider the Columbia data

Stellenbosch University https://scholar.sun.ac.za

Size	Methods	Iris	Gesture	Activity	Satellite	Digits	Occupancy	Columbia
$m = 2$	Borda count rule	**0.9111**	**0.5514**	0.7444	0.8005	0.8651	**0.6352**	0.5212
	Majority vote rule	**0.9111**	**0.5514**	0.7364	0.76604	0.8575	0.6222	**0.5333**
	Product rule	**0.9111**	**0.5514**	0.7444	**0.8225**	0.8762	**0.6352**	**0.5333**
	Sum rule	**0.9111**	**0.5514**	**0.7501**	**0.8225**	**0.8771**	**0.6352**	**0.5333**
	Yayambo	**0.9111**	**0.5514**	0.7466	0.7801	**0.8771**	**0.6352**	**0.5333**
$m = 3$	Borda count rule	0.9777	0.5465	0.7446	0.8032	0.9230	0.6310	0.7596
	Majority vote rule	**0.9778**	0.5449	0.7822	0.8020	0.9111	0.6223	**0.8889**
	Product rule	**0.9778**	0.5444	0.7348	0.8060	0.9207	0.6298	0.7685
	Sum rule	**0.9778**	0.5476	0.7536	0.8130	0.9207	**0.6352**	0.7685
	Yayambo	**0.9778**	**0.5595**	**0.7837**	**0.8150**	**0.9257**	**0.6352**	**0.8889**
$m = 4$	Borda count rule	0.9556	0.6339	0.7544	0.8115	0.9229	0.6345	0.7443
	Majority vote rule	0.9556	0.6499	0.7900	0.8200	0.9334	**0.6407**	0.8777
	Product rule	0.9556	0.6429	0.7646	**0.8265**	0.9322	0.625	0.7522
	Sum rule	0.9556	0.6429	0.7644	**0.8265**	0.9393	0.6353	0.7524
	Yayambo	**0.9778**	**0.6458**	**0.7906**	0.8230	**0.9433**	**0.6407**	**0.8857**
$m = 5$	Borda count rule	0.9777	0.6455	0.7356	0.8090	0.8905	0.9033	0.8878
	Majority vote rule	**1.0000**	0.6728	0.7446	0.8122	0.8886	0.9133	0.8996
	Product rule	**1.0000**	0.6665	0.7455	0.8110	0.8905	0.9042	0.8852
	Sum rule	**1.0000**	0.6875	0.7644	0.8010	0.9153	0.9163	0.9048
	Yayambo	**1.0000**	**0.7054**	**0.7788**	**0.8120**	**0.9167**	**0.9365**	**0.9535**

Table 12.19: *Accuracy values of fusion techniques on benchmark data sets. The value m denotes the number of classifiers considered for each case. Bold values indicate best performance.*

set for this experiment. The idea is to vary the number of viewpoints of objects and observe how the number of object viewpoints influences the performances of the classifier fusion methods.

A fixed camera and a turntable were used to collect the Columbia data. The objects were placed on a motorized turntable against a black background. The turntable was rotated through 360 degrees to vary object pose with respect to the camera. Images of the objects were taken at pose intervals of 5 degrees. Further description of the Columbia data set can be found in Appendix E.

As shown in Table 12.4, we consider 15 types of objects in the Columbia data set where each of the types of objects has 72 views. We used the same five classifiers and considered the following 11 numbers of viewpoints: $1, 6, 11, 16$, 21, 26, 31, 36, 41, 46 and 51. We ran three repetitions with different random initializations for each technique whereby we plot accuracy and loss of classification and fusion techniques in Figure 12.1.

It is observed that, the performance generally improves with the number of viewpoints of the object, as expected. When only one viewpoint of an object is considered, the results show that the fusion methods do not

Stellenbosch University https://scholar.sun.ac.za

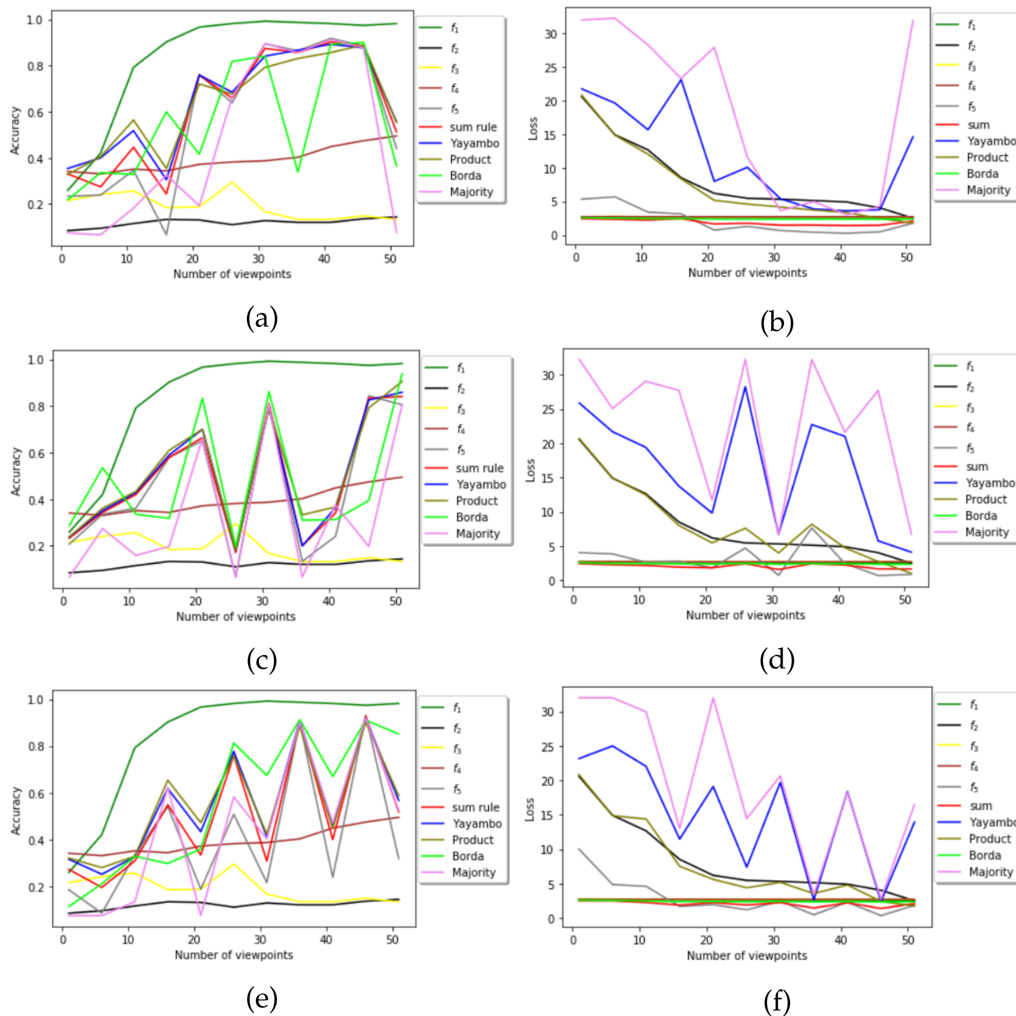


Figure 12.1: *Accuracies (left) and losses (right) of classification and fusion techniques in active vision.* ***(12.1a)****: Accuracies of the techniques in the first simulated run.* ***(12.1b)****: Losses of the techniques in the first simulated run.* ***(12.1c)****: Accuracies of the techniques in the second simulated run.* ***(12.1d)****: Losses of the techniques in the second simulated run.* ***(12.1e)****: Accuracies of the techniques in the third simulated run.* ***(12.1f)****: Losses of the techniques in the third simulated run.*

achieve the best performance, both in accuracy and loss. A possible explanation to the poor results of the fusion methods can be that poor individual classifiers are overconfident, so their outputs dominate the outputs from the strongest individual classifier (f_1). Overconfidence of individual classifiers are indicated by high losses in Figures 12.1b, 12.1d and 12.1f (their losses get improved as the number of views get larger). For Yayambo in particular, this is probably because, when classifying objects made of limited views, most of the classifiers are weak and try to support each other's erroneous

Stellenbosch University https://scholar.sun.ac.za

predictions during the fusion process. Also, given small number of viewpoints, Yayambo outperformed the sum rule in terms of accuracy, while the sum rule achieved lower loss. This matches what we expect in terms of Yayambo's susceptibility to large loss because it is not calibrated for probabilistic outputs. Note that adding an additional viewpoint of an object can decrease or increase the performance of a classification technique (i.e. individual classifiers and classifier fusion methods) which can be illustrated by the zigzagging behaviour in Figure 12.1a. In Figure 12.1a for instance, the Yayambo's performance is worse with 16 viewpoints compared to 11 viewpoints, and with 26 viewpoints compared to 21 viewpoints. This might mean that additional view points can make individual classifiers stronger or weaker, and overconfident as illustrated by high losses in Figure 12.1. This might indicate that selection of features (or view points in this case) for training of classifiers is crucial.

It is also observed (in Figure 12.1) that there is a regularity to the increases and decreases of classifiers' performances when adding new view points. It should be noted that there is no clear explanation of why such regularity is observed—a clearer explanation requires a more in-depth investigation.

12.3.7 Artificial classifiers

Our empirical results so far provide evidence that the Yayambo fusion approach often outperforms other fusion methods, but we also see cases where this does not hold. Here we consider fusing outputs of some hypothetical classifiers which we define by specifying their behaviour, rather than training them. The hope is that the extreme setting we describe here sheds further light on the behaviour of our proposed fusion method.

For the artificial classifier outputs, we have the following situation. We consider fusing five classifiers for a binary classification task, where

- the first classifier, f_1, always classifies correctly, with probability predictions 0.51 for the correct class and 0.49 for the incorrect class;
- the second classifier, f_2, also classifies perfectly, but its probability predictions are 0.90 for the correct class and 0.1 for the incorrect class;
- the third classifier is always wrong: it predicts 0.49 for the correct class, and 0.51 for the incorrect class;

Stellenbosch University https://scholar.sun.ac.za

- with the fourth classifier, for 65% of outputs, the classifier correctly classifies a test point with probability of 0.7 (i.e. probability of 0.7 for correct class and 0.3 for the other class). For the remaining outputs, it outputs a distribution with a first class probability sampled uniformly from $(0,1)$; and
- the fifth classifier always outputs a distribution with a first class probability sampled uniformly from $(0,1)$.

To be able to evaluate accuracies of our artificial classifiers, we provide 5000 outcomes to each class which will be used as ground truth (i.e. each class will contain 5000 data points). Expected performance for each of the 5 individual artificial classifiers is shown in Table 12.20 and expected agreements between classifiers from a classification point of view are shown in Table 12.21. Our results for fusion of artificial classifiers are presented in Tables 12.22 (where we consider an increasing number of classifiers) and 12.23 (where we consider all classifiers).

Note that viewed at the output level, the first and the third classifiers are in close agreement—see the small value at the intersection of f_1 and f_3 in Table 12.24—while they differ substantially from the other three classifiers. On the other hand, from a decision perspective, the first and the second classifiers agree exactly—see the high value at the intersection of f_1 and f_2 in Table 12.21; we used accuracy to evaluate classification disagreement between two classifiers, say f_i and f_j, where f_i's outputs are considered as ground truth and f_j's outputs as predictions. The third classifier disagrees totally with the first two classifiers and, the fourth and fifth classifiers can agree or disagree with others depending on the random outputs.

Techniques	Expected accuracy
f_1	1.0000
f_2	1.0000
f_3	0.0000
f_4	0.8250
f_5	0.5000

Table 12.20: *Expected performances for each of the* 5 *individual artificial classifiers.*

Table 12.22 confirms that the accuracy of fused decisions may degrade as we add weaker (or erratic) classifiers. These results support our earlier argument that Yayambo relies on regularity in the behaviour of classifiers

Stellenbosch University https://scholar.sun.ac.za

	f_2	f_3	f_4	f_5
f_1	1.0000	0.0000	0.8250	0.5000
f_2	–	0.0000	0.8250	0.5000
f_3	–	–	0.1750	0.5000
f_4	–	–	–	0.5000

Table 12.21: *Expected agreement between individual artificial classifiers from a classification point of view. Values range in* $[0, 1]$ *and high values indicate high agreement between classifiers based on their expected accuracy.*

Techniques	Case 1	Case 2	Case 3	Case 4
f_1	**1.0000**	**1.0000**	**1.0000**	**1.0000**
f_2	**1.0000**	**1.0000**	**1.0000**	**1.0000**
f_3	–	0.0000	0.0000	0.0000
f_4	–	–	0.8242	0.8242
f_5	–	–	–	0.5017
Borda count rule	**1.0000**	**1.0000**	0.9123	0.9123
Majority vote rule	**1.0000**	**1.0000**	0.9123	0.9123
Product rule	**1.0000**	**1.0000**	0.9617	0.9011
Sum rule	**1.0000**	**1.0000**	0.9617	0.9365
Yayambo	**1.0000**	**1.0000**	0.9617	0.9333

Table 12.22: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies using artificial classifiers. Here, cases refer to numbers of classifiers considered. Bold values indicate best performance.*

Techniques	Case 1	Case 2	Case 3
f_1	**1.0000**	**1.0000**	**1.0000**
f_2	0.0000	**1.0000**	**1.0000**
f_3	0.0000	0.0000	0.0000
f_4	0.8221	0.8280	0.8164
f_5	0.5013	0.5088	0.5050
Borda count rule	0.9099	0.9137	0.9074
Majority vote rule	0.9087	0.9097	0.9017
Product rule	0.9087	0.9097	0.9017
Sum rule	0.9376	0.9387	0.9283
Yayambo	0.9350	0.9344	0.9253

Table 12.23: *The Borda count rule, majority vote rule, product rule, sum rule and Yayambo accuracies using artificial classifiers. Here, cases refer to different runs of the fourth and fifth classifiers as they are purely random. Bold values indicate best performance.*

resulting from their being trained on the same task, resulting in similar behaviour on future observations.

12.4 Conclusion and avenues for future work

We proposed a once-off classifier fusion method based on the principle of indifference—no external information or prior performance is provided to enhance the fusion process. Our proposed classifier fusion method takes ad-

Stellenbosch University https://scholar.sun.ac.za

	f_2	f_3	f_4	f_5
f_1	55.1443	2.8284	32.66384	41.0806
f_2	–	57.9228	48.1349	69.7933
f_3	–	–	34.1587	41.0897
f_4	–	–	–	51.7427

Table 12.24: *Measurement of disagreement between classifiers from a prediction point of view. We used the Euclidean distance between classifiers' outputs to evaluate prediction disagreement between classifiers. Values range in* $[0, \infty)$ *and high values indicate low agreement between classifiers based on classifiers' predictions.*

vantage of prediction similarities to enhance consensus decision-making in weighting the relative importance of individual classifiers unequally, where the loss function is symmetric. One caveat is that our proposed fusion method is found to be over-confident in its predictions, i.e. the method does not output calibrated probabilities for the various classes. In other words, the approach is not recommended for use in downstream tasks where such probabilities are desirable. Limitations of our proposed classifier fusion method can be found in Subsection 11.1.2.

12.4.1 Findings summary

Our experiments compare our proposed method to four established black-box classifier fusion methods, the sum rule, the product rule, the majority vote rule and the Borda count rule. We found that our proposed method yields the best accuracies in most of the cases we considered, for both highly similar as well as highly diverse classifiers. This indicates that our proposed method is more robust to the quality of individual classifiers than the sum rule, product rule and majority vote rule. However, our proposed method is not a probabilistic method, which results in a high cross-entropy loss. So Yayambo is not suitable in situations where cross-entropy loss function is appropriate. The sum rule and the product rule are recommended for generating calibrated fused outputs. It is possible that other fusion approaches might be more recommended for black-box fusion.

Furthermore, we observed empirically that even though supports assigned to predictions are asymmetric, the updated probability vectors converge. This is interesting as from all the probability vectors we can output the same class of an object of interest for which classifier outputs confer differing views before the fusion process. While a consensus decision is not necessarily correct, we contend that achieving consensus from different ini-

Stellenbosch University https://scholar.sun.ac.za

tial distributions using such an asymmetric notion of support confers some credence on the final decision.

Finally, we observed different behaviours of Yayambo for various values of ε_0 which is required to quantify support between distributions. Values for ε_0 should be chosen carefully, otherwise, Yayambo may yield poor results.

12.4.2 Avenue for future work

We identify the following promising avenues for future work:

1. Our hope was that Yayambo converges, and it converged in all the experiments run in this chapter. We ultimately hope to investigate the convergence of Yayambo theoretically in future.

2. It may be possible to obtain a closed form expression for the consensus class from initial distributions without performing the iteration explicitly, possibly with some modifications to the forms of the equations used for the supports.

3. It would be interesting to consider how to formalize a probabilistic prior interpretation of the regularity assumption we are making, and what posterior it leads to in the Bayesian probability setting. This may lead to support formulae with better theoretical motivations and further improved performance. It would also lay a solid foundation for fusion of multiple sequential observations, rather than the once-off case considered by our algorithm.

Having reviewed and discussed background and literature on our three problems, presented our contributions to each, and discussed the results of investigations, we now conclude this thesis.

Stellenbosch University https://scholar.sun.ac.za

Chapter 13

Conclusions and future work

13.1 Summary

The overarching challenge of this thesis was search and rescue in an unknown static environment by solitary robots. The research question was how solitary robots can coordinate their actions when some of them meet to work more effectively. We introduced various algorithms or algorithm enhancements for three problems arising from this context, namely: coordination, construction of a view of a network, and classifier fusion. Although three problems were treated, coordination is the main problem involved for search and rescue. Network view construction and classifier fusion are related problems inspired by the problem of coordination.

In what follows we provide a brief summary of the methods and results contained in this thesis, excluding the opening chapters (**Chapter 1** and **Chapter 2**) and closing chapter (**Chapter 13**). The thesis was composed of three parts for the three topics considered. Each part considered background work and presented a literature review of the topic, before presenting novelties followed by experimental results and discussion.

Part I discussed a novel coordination strategy for multiple solitary robots: coordination by soft obstacles based on the principle of cellular decomposition given limited available time. The proposed approach targets situations where the available search time is known and limited. We considered search coverage as a proxy to evaluate robots' expected performance, which corresponds to assuming targets located independently and uniformly in space.

Part II discussed the proposed decentralised method each node applies

Stellenbosch University https://scholar.sun.ac.za

to construct its view of their interaction network. Our proposed method focuses on reducing the number of messages received by nodes during the view construction.

Classifiers are used to identify the category of an object. Different classifiers may give different opinions on the type of a target of interest and it might be difficult to select the best classifier to use. One way to improve classification performance is to combine diverse classifiers. In **Part III**, we described a classifier fusion method to combine classifiers' outputs for classification of targets.

13.2 Primary contributions

We now present what we view as the chief contributions of this thesis.

(1) For the problem of coordination of multiple solitary robots, we considered two benchmark methods, ARS and PRS [42, 109]. ARS suffers from high interference while PRS suffers from high interruptibility. Our proposed coordination strategy is based on the cellular decomposition strategy [17] and the use of margins, and produced a low interruptibility and low interference strategy [70]. In this strategy, after applying cellular decomposition by solitary robots, each robot considers regions assigned to others to be soft obstacles which should be avoided as much as possible. Cellular decomposition allows robots to explore non-overlapping regions, thus interference is mitigated; it also allows robots to avoid the need to schedule further rendezvous, thus reducing the problem of interruptibility.

(2) For the problem of managing interaction of robots, our proposed variant on the benchmark method that a robot applies to construct its view of a communication graph [118] reduces the number of messages exchanged between robots in an interaction. Our method is based on the principle of pruning. While constructing a view of the communication graph, we identify certain types of nodes that are not required to be involved in every round of the algorithm; such nodes are deactivated. This consideration significantly reduces the number of messages various nodes must receive while constructing a view of the communication graph. We also propose a version of pruning that is robust to failure.

Stellenbosch University https://scholar.sun.ac.za

(3) For the problem of classifier fusion, we considered situations where we have multiple classifiers and only their outputs on a test point are provided. To reach a consensus in classifier fusion, classifiers assign some support to each other to score their outputs. We believe that classifiers should still be able to find a way to determine the quality of their outputs in the absence of information about their performance. We introduced a novel way [71] to combine such classifiers by quantifying support between predictions, and using these to weight predictions to reach a consensus decision. This way of combining classifiers without additional information about their performance outperformed the sum rule, product rule, majority vote rule and Borda count rule methods which we used as the benchmark methods. The proposed fusion method attempts to take advantage of the expectation of similar behaviour of classifiers trained for the same task, i.e. our proposed fusion method leverages the belief that homogeneity of classifier predictions contains valuable information.

The research reported in this thesis has investigated three problems: management of interaction, coordination and classifier fusion. In our setting, the three problems are considered as an all-in-one problem, i.e. these three problems are subproblems of the overarching search and rescue problem.

13.3 Future work

Avenues for future work for the three methods are discussed in Chapters 9.4, 6.4 and 12.4 respectively.

We ran experiments for our three proposed methods separately. We also ran experiments with the integration of the first two approaches (i.e. coordination and graph view construction). The incorporation of our third proposed method (i.e. classifier fusion method) in the system requires some real applications (applications where real robots are deployed to an unknown environment containing real objects of interest, e.g. mining). We ultimately hope to test our three methods as part of a whole hardware system of mobile robots running real applications.

Stellenbosch University https://scholar.sun.ac.za

List of References

[1] J. Aloimonos, I. Weiss, and A. Bandyopadhyay. Active Vision. *International Journal of Computer Vision*, 1(4):333–356, 1988.

[2] S. Alpern. The Rendezvous Search Problem. *SIAM Journal on Control and Optimization*, 33(3):673–683, 1995.

[3] F. Amigoni, J. Banfi, and N. Basilico. Multirobot Exploration of Communication-Restricted Environments: A Survey. *IEEE Intelligent Systems*, 32(6):48–57, 2017.

[4] R. Aragues, C. Sagues, and Y. Mezouar. Feature-Based Map Merging with Dynamic Consensus on Information Increments. *Autonomous Robots*, 38(3):243–259, 2015.

[5] R. C. Arkin and J. Diaz. Line-of-Sight Constrained Exploration for Reactive Multiagent Robotic Teams. In *7th International Workshop on Advanced Motion Control*, pages 455–461. IEEE, 2002.

[6] A. Asuncion and D. Newman. UCI Machine Learning Repository. Available at `https://archive.ics.uci.edu/ml/datasets.html`, 2007.

[7] K. Batool and M. A. Niazi. Towards a Methodology for Validation of Centrality Measures in Complex Networks. *PloS one*, 9(4):e90283, 2014.

[8] A. Bavelas. Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950.

[9] C. Belta, V. Isler, and G. J. Pappas. Discrete Abstractions for Robot Motion Planning and Control in Polygonal Environments. *IEEE Transactions on Robotics*, 21(5):864–874, 2005.

[10] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[11] E. Bisong. Introduction to Scikit-learn. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pages 215–229. Springer, 2019.

Stellenbosch University https://scholar.sun.ac.za

[12] E. Bonabeau, M. Dorigo, and G. Theraulaz. Inspiration for optimization from social insect behaviour. *Nature*, 406(6791):39, 2000.

[13] P. Bonacich. Power and Centrality: A Family of Measures. *American Journal of Sociology*, 92(5):1170–1182, 1987.

[14] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. " O'Reilly Media, Inc.", 2008.

[15] E. Bressert. *SciPy and NumPy: An Overview for Developers*. " O'Reilly Media, Inc.", 2012.

[16] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider. Coordinated Multi-Robot Exploration. *IEEE Transactions on Robotics*, 21(3):376–386, 2005.

[17] H. Choset. Coverage for Robotics–A Survey of Recent Results. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):113–126, 2001.

[18] L. Cigler and B. Faltings. Decentralized Anti-Coordination Through Multi-Agent Learning. *Journal of Artificial Intelligence Research*, 47:441–473, 2013.

[19] J. Cohen. The Statistical Power of Abnormal-Social Psychological Research: A Review. *The Journal of Abnormal and Social Psychology*, 65(3):145, 1962.

[20] G. F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Pearson Education, 2005.

[21] D. Das, S. Mukhopadhyaya, and D. Nandi. Techniques in Multi-Robot Area Coverage: A Comparative Survey. In *Handbook of Research on Design, Control, and Modeling of Swarm Robotics*, pages 741–765. IGI Global, 2016.

[22] J. de Hoog, S. Cameron, and A. Visser. Selection of Rendezvous Points for Multi-Robot Exploration in Dynamic Environments. In *Workshop on Agents in Realtime and Dynamic Environments, International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2010.

[23] H. Durrant-Whyte. A Beginner's Guide to Decentralised Data Fusion. *Technical Document of Australian Centre for Field Robotics, University of Sydney, Australia*, pages 1–27, 2000.

[24] H. Durrant-Whyte, M. Stevens, and E. Nettleton. Data Fusion in Decentralised Sensing Networks. In *4th International Conference on Information Fusion*, pages 302–307, 2001.

Stellenbosch University https://scholar.sun.ac.za

[25] P. Emerson. The Initial Borda Count and Partial Voting. *Social Choice and Welfare*, 40(2):353–358, 2013.

[26] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. State-of-the-Art Deep Learning: Evolving Machine Intelligence toward Tomorrow's Intelligent Network Traffic Control Systems. *IEEE Communications Surveys & Tutorials*, 19(4):2432–2455, 2017.

[27] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9(Aug):1871–1874, 2008.

[28] L. Fei, J. Xia, Y. Feng, and L. Liu. A Novel Method to Determine Basic Probability Assignment in Dempster–Shafer Theory and its Application in Multisensor Information Fusion. *International Journal of Distributed Sensor Networks*, 15(7):1–16, 2019.

[29] O. Feinerman, A. Korman, Z. Lotker, and J.-S. Sereni. Collaborative Search on the Plane without Communication. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, pages 77–86. ACM, 2012.

[30] A. C. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency Handling in Multiperspective Specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.

[31] G. M. Fricke, J. P. Hecker, A. D. Griego, L. T. Tran, and M. E. Moses. A Distributed Deterministic Spiral Search Algorithm for Swarms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4430–4436. IEEE, 2016.

[32] A. Fujimori, P. N. Nikiforuk, and M. M. Gupta. Adaptive Navigation of Mobile Robots with Obstacle Avoidance. *IEEE Transactions on Robotics and Automation*, 13(4):596–601, 1997.

[33] M. S. Gail and F. Richard. Using Effect Size–or Why the P Value Is Not Enough. *Journal of Graduate Medical Education*, 4(3):279–282, 2012.

[34] R. G. Gallager, P. A. Humblet, and P. M. Spira. A Distributed Algorithm for Minimum-Weight Spanning Trees. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(1):66–77, 1983.

[35] F. Garin, D. Varagnolo, and K. H. Johansson. Distributed Estimation of Diameter, Radius and Eccentricities in Anonymous Networks. *IFAC Proceedings Volumes*, 45(26):13–18, 2012.

Stellenbosch University https://scholar.sun.ac.za

[36] A. Gelman, A. Vehtari, P. Jylänki, T. Sivula, D. Tran, S. Sahai, P. Blomstedt, J. P. Cunningham, D. Schiminovich, and C. Robert. Expectation Propagation as a Way of Life: A Framework for Bayesian Inference on Partitioned Data. *arXiv preprint arXiv:1412.4869*, 2017.

[37] C. Gomez, A. C. Hernandez, and R. Barber. Topological Frontier-Based Exploration and Map-Building Using Semantic Information. *Sensors*, 19(20):4595, 2019.

[38] S. Grime, H. F. Durrant-Whyte, and P. Ho. Communication in Decentralized Data-Fusion Systems. In *1992 American Control Conference*, pages 3299–3303. IEEE, 1992.

[39] J. Guivant, J. Nieto, and E. Nebot. Victoria Park Dataset. Available at `http://www-personal.acfr.usyd.edu.au/nebot/victoria_park.htm`, 2012.

[40] P. Hage and F. Harary. Eccentricity and Centrality in Networks. *Social Networks*, 17(1):57–63, 1995.

[41] Z. B. Hao, N. Sang, and H. Lei. Cooperative Coverage by Multiple Robots with Contact Sensors. In *IEEE Conference on Robotics, Automation and Mechatronics*, pages 543–548. IEEE, 2008.

[42] H. Hourani, E. Hauck, and S. Jeschke. Serendipity Rendezvous as a Mitigation of Exploration's Interruptibility for a Team of Robots. In *IEEE International Conference on Robotics and Automation*, pages 2984–2991. IEEE, 2013.

[43] A. Howard, M. J. Matarić, and G. S. Sukhatme. Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem. In *Distributed Autonomous Robotic Systems 5*, pages 299–308. Springer, 2002.

[44] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbor Rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[45] M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.

[46] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. In *Autonomous Robot Vehicles*, pages 396–404. Springer, 1986.

[47] C. Kim and M. Wu. Leader Election on Tree-Based Centrality in Ad Hoc Networks. *Telecommunication Systems*, 52(2):661–670, 2013.

Stellenbosch University https://scholar.sun.ac.za

[48] H.-C. Kim and Z. Ghahramani. Bayesian Classifier Combination. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, pages 619–627, 2012.

[49] T. W. Kim, E. H. Kim, J. K. Kim, and T. Y. Kim. A Leader Election Algorithm in a Distributed Computing System. In *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 481–485. IEEE, 1995.

[50] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. Robocup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research. In *Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics*, volume 6, pages 739–743. IEEE, 1999.

[51] J. Kittler. Combining Classifiers: A Theoretical Framework. *Pattern Analysis and Applications*, 1(1):18–27, 1998.

[52] J. Kittler, M. Hatef, R. P. Duin, and J. Matas. On Combining Classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.

[53] C. S. Kong, N. A. Peng, and I. Rekleitis. Distributed Coverage with Multi-Robot System. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 2423–2429. IEEE, 2006.

[54] K. Konolige, D. Fox, B. Limketkai, J. Ko, and B. Stewart. Map Merging for Distributed Robot Navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 212–217. IEEE, 2003.

[55] E. Korach, D. Rotem, and N. Santoro. Distributed Algorithms for Finding Centers and Medians in Networks. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(3):380–401, 1984.

[56] R. V. Kshirsagar and B. Jirapure. A Survey on Fault Detection and Fault Tolerance in Wireless Sensor Networks. In *International Conference on Benchmarks in Engineering Science and Technology*, pages 6–9, 2012.

[57] F. Kuhn, N. Lynch, and R. Oshman. Distributed Computation in Dynamic Networks. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, pages 513–522. ACM, 2010.

[58] H. W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

Stellenbosch University https://scholar.sun.ac.za

[59] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[60] L. I. Kuncheva. Switching Between Selection and Fusion in Combining Classifiers: An Experiment. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(2):146–156, 2002.

[61] K. Lai and D. Fox. Object Recognition in 3D Point Clouds using Web Data and Domain Adaptation. *The International Journal of Robotics Research*, 29(8):1019–1037, 2010.

[62] S. Lam and M. Reiser. Congestion Control of Store-and-Forward Networks by Input Buffer Limits–an Analysis. *IEEE Transactions on Communications*, 27(1):127–134, 1979.

[63] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Department, Iowa State University, 10 1998.

[64] A. Le Roux, M. I. Cherry, L. Gygax, and M. B. Manser. Vigilance Behaviour and Fitness Consequences: Comparing a Solitary Foraging and an Obligate Group-Foraging Mammal. *Behavioral Ecology and Sociobiology*, 63(8):1097–1107, 2009.

[65] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 177–187. ACM, 2005.

[66] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[67] X. Li, G. Chen, E. Blasch, J. Patrick, C. Yang, and I. Kadar. Multi-Sensor Management for Data Fusion in Target Tracking. In *Signal Processing, Sensor Fusion, and Target Recognition XVIII*, volume 7336, page 73360Y. International Society for Optics and Photonics, 2009.

[68] K. M. Lynch, I. B. Schwartz, P. Yang, and R. A. Freeman. Decentralized Environmental Modeling by Mobile Sensor Networks. *IEEE Transactions on Robotics*, 24(3):710–724, 2008.

Stellenbosch University https://scholar.sun.ac.za

[69] U. G. Mangai, S. Samanta, S. Das, and P. R. Chowdhury. A Survey of Decision Fusion and Feature Fusion Strategies for Pattern Classification. *IETE Technical review*, 27(4):293–307, 2010.

[70] J. F. Masakuna, S. W. Utete, and S. Kroon. A Coordinated Search Strategy for Solitary Robots. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 433–434. IEEE, 2019.

[71] J. F. Masakuna, S. W. Utete, and S. Kroon. Performance-Agnostic Fusion of Probabilistic Classifier Outputs. In *IEEE 23rd International Conference on Information Fusion*, pages 1–8. IEEE, 2020.

[72] M. McNeill and D. Lyons. An Approach to Fast Multi-Robot Exploration in Buildings with Inaccessible Spaces. In *International Conference on Robotics and Biomimetics (ROBIO19)*, pages 1–8. IEEE, 2019.

[73] N. Meghanathan. Correlation Coefficient Analysis of Centrality Metrics for Complex Network Graphs. In *Computer Science On-line Conference*, pages 11–20. Springer, 2015.

[74] R. K. Merton. The Matthew Effect in Sciencee: The Reward and Communication Systems of Science are Considered. *Science*, 159(3810):56–63, 1968.

[75] A. R. Mosteo, L. Montano, and M. G. Lagoudakis. Multi-Robot Routing under Limited Communication Range. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1531–1536. IEEE, 2008.

[76] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Available at `https://bitcoin.org/bitcoin.pdf`, Accessed in January 2019.

[77] A. Naz. *Distributed Algorithms for Large-Scale Robotic Ensembles: Centrality, Synchronization and Self-Reconfiguration*. PhD thesis, Université Bourgogne Franche-Comté, 2017.

[78] S. A. Nene, S. K. Nayar, and H. Murase. Columbia Object Image Library (coil-100). Technical Report TR CUCS-005-96, Computer Vision Laboratory, Computer Science Department, Columbia University, 2 1996.

[79] A. Nüchter and J. Hertzberg. Towards Semantic Maps for Mobile Robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008.

[80] J. Oroko and G. Nyakoe. Obstacle Avoidance and Path Planning Schemes for Autonomous Navigation of a Mobile Robot: A Review. In *Proceedings of the*

Stellenbosch University https://scholar.sun.ac.za

2012 Mechanical Engineering Conference on Sustainable Research and Innovation, pages 314–318, 2012.

[81] M. Pasin, S. Fontaine, and S. Bouchenak. Failure Detection in Large Scale Systems: A Survey. In *IEEE Network Operations and Management Symposium Workshops*, pages 165–168. IEEE, 2008.

[82] R. J. Ramírez and N. Santoro. Distributed Control of Updates in Multiple-Copy Databases: A Time Optimal Algorithm. In *Proceedings of the 4th Berkeley Conference on Distributed Data Management and Computer Networks (Berkeley, Calif, Aug.)*, pages 191–207, 1979.

[83] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset. Limited Communication, Multi-Robot Team Based Coverage. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, volume 4, pages 3462–3468. IEEE, 2004.

[84] K. A. Remley, G. Koepke, D. G. Camell, C. Grosvenor, G. Hough, and R. T. Johnk. Wireless Communications in Tunnels for Urban Search and Rescue Robots. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, pages 236–243. ACM, 2008.

[85] N. Roy and G. Dudek. Collaborative Robot Exploration and Rendezvous: Algorithms, Performance Bounds and Observations. *Autonomous Robots*, 11(2):117–136, 2001.

[86] N. Santoro. *Design and Analysis of Distributed Algorithms*, volume 56. John Wiley & Sons, 2006.

[87] R. E. Schapire. The Boosting Approach to Machine Learning: An Overview. In *Nonlinear Estimation and Classification*, pages 149–171. Springer, 2003.

[88] N. Seliya, T. M. Khoshgoftaar, and J. Van Hulse. A Study on the Relationships of Classifier Performances Metrics. In *21st International Conference on Tools with Artificial Intelligence*, pages 59–66. IEEE, 2009.

[89] A. Sheth, C. Hartung, and R. Han. A Decentralized Fault Diagnosis System for Wireless Sensor Networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 1–3. IEEE, 2005.

[90] Y. Shiloach and S. Even. An On-Line Edge-Deletion Problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

Stellenbosch University https://scholar.sun.ac.za

[91] B. Siciliano and O. Khatib. *Springer Handbook of Robotics*, chapter 40, pages 921–941. Springer Science & Business Media, 2008.

[92] S. S. Skiena. *The Algorithm Design Manual*, volume 1. Springer Science & Business Media, 1998.

[93] A. Solanas and M. A. Garcia. Coordinated Multi-Robot Exploration Through Unsupervised Clustering of Unknown Space. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* , volume 1, pages 717–721. IEEE, 2004.

[94] C. Spearman. "General Intelligence" Objectively Determined and Measured. *American Journal of Psychology*, 15:663–671, 1961.

[95] A. N. Steinberg, C. L. Bowman, and F. E. White. Revisions to the JDL Data Fusion Model. In *Sensor Fusion: Architectures, Algorithms, and Applications III*, volume 3719, pages 430–442. International Society for Optics and Photonics, 1999.

[96] S. D. Stoller. Leader Election in Distributed Systems with Crash Failures. Technical Report TR 481, Department of Computer Science, Indiana University, 4 1997.

[97] L. Susperregi, B. Sierra, M. Castrillón, J. Lorenzo, J. Martínez-Otzeta, and E. Lazkano. On the Use of a Low-Cost Thermal Sensor to Improve Kinect People Detection in a Mobile Robot. *Sensors*, 13(11):14687–14713, 2013.

[98] A. Tapus and M. J. Mataric. Towards Socially Assistive Robotics. *Journal of the Robotics Society of Japan*, 24(5):576–578, 2006.

[99] S. Thrun. Toward Robotic Cars. *Communications of the ACM*, 53(4):99–106, 2010.

[100] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT press, 2005.

[101] S. Thrun and T. M. Mitchell. Lifelong Robot Learning. *Robotics and Autonomous Systems*, 15(1-2):25–46, 1995.

[102] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot. FastSLAM: An Efficient Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association. *Journal of Machine Learning Research*, 4(3):380–407, 2004.

Stellenbosch University https://scholar.sun.ac.za

[103] T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka. Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue. *IEEE Robotics & Automation Magazine*, 19(3):46–56, 2012.

[104] H. Torres-Contreras, R. Olivares-Donoso, and H. M. Niemeyer. Solitary Foraging in the Ancestral South American Ant, *Pogonomyrmex Vermiculatus*. Is It Due to Constraints in the Production or Perception of Trail Pheromones? *Journal of Chemical Ecology*, 33(2):435–440, 2007.

[105] I. Ulrich and J. Borenstein. VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots. In *IEEE International Conference on Robotics and Automation, 1998*, volume 2, pages 1572–1577. IEEE, 1998.

[106] S. Utete and H. F. Durrant-Whyte. Routing for Reliability in Decentralised Sensing Networks. In *American Control Conference*, volume 2, pages 2268–2272. IEEE, 1994.

[107] S. W. Utete, B. Barshan, and B. Ayrulu. Voting as Validation in Robot Programming. *The International Journal of Robotics Research*, 18(4):401–413, 1999.

[108] B. L. Welch. The Generalization of Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, 1947.

[109] B. L. Wellman, S. Dawson, J. de Hoog, and M. Anderson. Using Rendezvous to Overcome Communication Limitations in Multirobot Exploration. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 2401–2406, 2011.

[110] F. Wilcoxon. Individual Comparisons by Ranking Methods. In *Breakthroughs in Statistics*, pages 196–202. Springer, 1992.

[111] D. Wolf, A. Howard, and G. S. Sukhatme. Towards Geometric 3D Mapping of Outdoor Environments Using Mobile Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1507–1512. IEEE, 2005.

[112] D. H. Wolpert. The Lack of a Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[113] K. M. Wurm, C. Stachniss, and W. Burgard. Coordinated Multi-Robot Exploration using a Segmentation of the Environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1160–1165. IEEE, 2008.

Stellenbosch University https://scholar.sun.ac.za

[114] B. Yamauchi. A Frontier-based Approach for Autonomous Exploration. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151. IEEE, 1997.

[115] B. Yamauchi. Frontier-based Exploration Using Multiple Robots. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 47–53. ACM, 1998.

[116] Z. Yan, L. Fabresse, J. Laval, and N. Bouraqadi. Metrics for Performance Benchmarking of Multi-Robot Exploration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3407–3414. IEEE, 2015.

[117] J. Yick, B. Mukherjee, and D. Ghosal. Wireless Sensor Network Survey. *Computer Networks*, 52(12):2292–2330, 2008.

[118] K. You, R. Tempo, and L. Qiu. Distributed Algorithms for Computation of Centrality Measures in Complex Networks. *IEEE Transactions on Automatic Control*, 62(5):2080–2094, 2017.

Stellenbosch University https://scholar.sun.ac.za

Appendix A

Solitary search

In Chapter 4, we described our coordination method where robots are called to explore assigned exploration regions. In this section, mechanisms for exploration of a region are discussed.

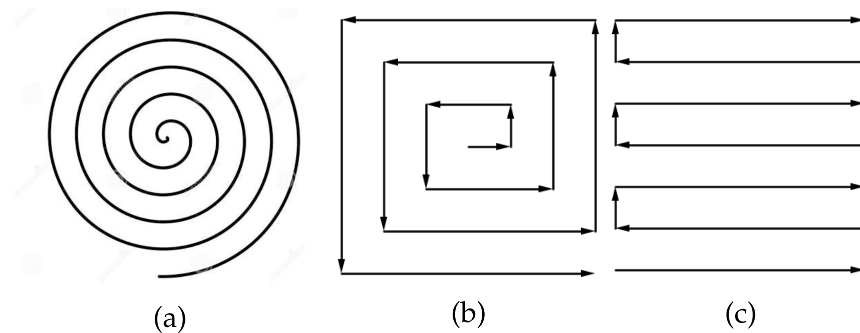


Figure A.1: *Some examples of methods for exploration of a region.* ***A.1a***: *Circular spiral search.* ***A.1b***: *Square spiral search.* ***A.1c***: *Zigzag search.*

Exploration regions discussed in Chapter 4 are bounded. From the perspective of a bounded region, a spiral-based search (Figures A.1a and A.1b) seems to be a good strategy to effectively cover a region. There exist various spiral-based search methods, with two major approaches being: circular and square spiral search methods. A spiral search strategy has the following property [31]: detection of nearest targets. Detection of the nearest targets is particularly important for robots bound by a team goal (e.g. if robots are called to drop discovered targets to some central or rendezvous point), because it minimises the per-target trip time to the rendezvous point.

When compared to the square spiral search method, the circular spiral search method gives complete coverage of exploration region and suf-

Stellenbosch University https://scholar.sun.ac.za

fers from repeated sampling (i.e. repeated exploration or backtracking behaviour). But the level of repeated sampling of the spiral search strategy is low when compared to other search strategies such as random search or frontier-based methods which suffer from high level of repeated sampling. However, since there is repeated sampling in the circular spiral search strategy, a square spiral search (Figure A.1b) could be applied. Repeated sampling in circular spiral is caused by adjustment of circular trajectories. A square spiral search does not have repeated exploration because the trajectory is straight. But a square spiral search leaves uncovered locations inside the covered region (we assume that a robot's sensed area is circular). However, backtracking hinders good search performance (i.e. a square spiral search does not give complete coverage).

To address the above concerns, we use an alternative strategy with completeness, but which does not repeat explored regions (which the circular spiral strategy suffers from) and does not leave uncovered locations inside the covered region (which the square spiral strategy suffers from). To cover an exploration region, the strategy applied is a back-and-forth motion, which is known as zigzag search (Figure A.1c) or the lawnmower pattern [17]. The zigzag search strategy also, when using a circular scanning region in a rectangular region which we consider in this thesis, leaves uncovered locations as the square spiral search but outside the covered region. This means that for zigzag search, completeness without repeated sampling requires exploring uncovered locations from outside the exploration region.

Application of zigzag search

In zigzag search, the robot moves back and forth from one side of its region to the other, along parallel paths. For instance, a robot may move in an east-and-west pattern towards the north (Figure A.2a).

To apply zigzag search, two aspects are important, namely pattern and direction of the robot. Only the four compass directions are considered: North (N), South (S), West (W) and East (E), and the robot's travel is described by moves in these directions. A pattern refers to a sequence of directions for robot motion. Let D and D^- denote a direction and its opposite respectively. Starting from D, going clockwise, one gets D, D^+, D^-, D^+ where D^+ is an orthogonal direction to D. For instance, if D is North then

Stellenbosch University https://scholar.sun.ac.za

N N N
W E W E W E
S S S
(a) (b) (c)

Figure A.2: *Application of zigzag search in different situations.* ***(A.2a)****: An obstacle-free exploration region.* ***(A.2b)****: A region occupied by a regular polygonal obstacle. (This is essentially like a Bug algorithm [80] along obstacles. We discuss Bug algorithms in Appendix B).* ***(A.2c)****: A region occupied by an irregular polygonal obstacle.*

D^- is South, D^+ is West. For D_i and D_j two distinct orthogonal directions, a zigzag pattern is described by,

$$\texttt{pattern} = (D_i, D_j, D_i^-, D_j)\,. \tag{A.0.1}$$

This allows eight possible patterns: (E, N, W, N), (E, S, W, S), (W, N, E, N), (W, S, E, S), (N, E, S, E), (N, W, S, W), (S, E, N, E) and (S, W, N, W). For instance, with the pattern (E, N, W, N), the robot will move east, then north, followed by west and north, repeatedly as in Figure A.2a.

A robot reaches its exploration region from a corner. A robot chooses its pattern when it arrives at a corner of its exploration region—a Python implementation of zigzag patterns can be found in Appendix G.2.4. It randomly chooses a pattern among the two possible patterns for the corner it has reached. For example, if a robot reaches its exploration region from the south-western corner of the region, it chooses between patterns (N, E, N, W) and (E, N, W, N). The procedure SETPATTERN defined in Algorithm A.1 shows how a robot chooses its pattern after reaching the closest corner a_i of its exploration region.

A robot changes directions when applying zigzag behaviour. A trip refers to motion between two turning points. Given a motion scale, $\gamma > 0$, a trip in the northern direction adds $(0, \gamma)$ to the current location x_i of the robot; a southward trip adds $(0, -\gamma)$; a westward trip adds $(-\gamma, 0)$; and an eastward trip adds $(\gamma, 0)$. For instance if $x_i = (0.5, 1.2)$, $\gamma = 0.1$ and robot has travelled north, then x_i becomes $(0.5, 1.3)$. The procedure SETDIRECTION defined in Algorithm A.1 shows how a robot sets its direction once within its exploration region.

Stellenbosch University https://scholar.sun.ac.za

Algorithm A.1 *Zigzag search used by a robot R_i to explore its exploration region $\mathcal{X}_i$. The main function (see* ZIGZAGMOVE*) requires $\mathcal{O}$ a set of obstacles within robot perception range. $\mathcal{X}_i$ is a rectangle implemented using Algorithm F.4 and a corner $a_i = (a_i(0), a_i(1))$.*

```
1:  class ZIGZAG
2:      Class variables
3:      a_i            the closest corner of its current exploration region
4:      γ              its motion velocity
5:      d              its perception range
6:      direction      its current zigzag direction
7:      dist           the distance already travelled in its current zigzag trip
8:      distance       the distance of its current zigzag trip
9:      J              an indicator for its current trip
10:     P              list of zigzag motion patterns
11:     pattern        the current zigzag pattern of R_i
12:     x_i            its current location
13:     X_i            its exploration region
14: ---------------------------------------------------------------
15:     constructor (d, γ)
16:         (d, γ, P) ← (d, γ, [])
17:         P.add((E(), N(), W(), N()))
18:         P.add(((E(), S(), W(), S())))
19:         P.add((W(), N(), E(), N()))
20:         P.add((W(), S(), E(), S()))
21:         P.add((N(), E(), S(), E()))
22:         P.add((N(), W(), S(), W()))
23:         P.add((S(), E(), N(), E()))
24:         P.add((S(), W(), N(), W()))
25:     end constructor
26:
27:     procedure PARAMREINIT(X_i, x_i, a_i)
28:         (X_i, x_i, a_i, dist, J) ← (X_i, x_i, a_i, 0, 1)
29:         SETPATTERN()
30:         SETDIRECTION()
31:         SETDISTANCE()
32:     end procedure
33:
34:     function GETPATTERNS()
35:         return P
36:     end function
37:
38:     function N()
39:         return [0, γ]
40:     end function
41:
42:     function S()
43:         return [0, −γ]
44:     end function
45:
46:     function E()
47:         return [γ, 0]
48:     end function
49:
50:     function W()
51:         return [−γ, 0]
52:     end function
53:
54:     function SETPATTERN()
55:         if X_i.x == a_i(0) and X_i.y == a_i(1) then
56:             pattern← P[random.choice(0, 4)]
57:         else if X_i.x == a_i(0) and X_i.y + X_i.h == a_i(1) then
58:             pattern← P[random.choice(1, 6)]
59:         else if X_i.x + X_i.w == a_i(0) and X_i.y == a_i(1) then
60:             pattern← P[random.choice(2, 5)]
61:         else
62:             pattern← P[random.choice(3, 7)]
63:         end if
64:     end function
```

Stellenbosch University https://scholar.sun.ac.za


```
65:     function SETDIRECTION()
66:         direction ← pattern[(J − 1) (mod 4)]
67:     end function
68:
69:     function GETSTARTINGPOINT()
70:         if (pattern[0] == N() and pattern[1] == E()) or (pattern[0] == E() and
    pattern[1] == N()) then
71:             return a_i + (d, d)
72:         else if (pattern[0] == S() and pattern[1] == E()) or (pattern[0] == E() and
    pattern[1] == S()) then
73:             return a_i + (d, −d)
74:         else if (pattern[0] == N() and pattern[1] == W()) or (pattern[0] == W() and
    pattern[1] == N()) then
75:             return a_i + (−d, d)
76:         else
77:             return a_i + (−d, −d)
78:         end if
79:     end function
80:
81:     function SETDISTANCE()
82:         if J is odd then
83:             if absoluteValue(direction)== (0, γ) then
84:                 distance ← X_i.h − 2d
85:             else
86:                 distance ← X_i.w − 2d
87:             end if
88:         else
89:             if absoluteValue(direction)== (0, γ) then
90:                 distance ← min(2d, X_i.h − dJ)
91:             else
92:                 distance ← min(2d, X_i.w − dJ)
93:             end if
94:         end if
95:     end function
96:
97:     function ZIGZAGMOVE(O)
98:         B ← ∅
99:         x ← x_i + direction
100:        if not OBSTACLEDETECTION(x, O) and dist<=distance−γ and ISINSIDE(x) then
101:            x_i ← x
102:            dist ← dist + γ
103:        else
104:            if OBSTACLEDETECTION(x, O) then
105:                j ← 1
106:                for k =dist to distance do
107:                    y ← x_i + j × distance
108:                    B ← B ∪ B_d(y)    ▷ where B_d(y) denotes a closed ball of radius d centred at y
109:                    j ← j + 1
110:                end for
111:            end if
112:            dist ← γ
113:            J ← (J + 1) (mod 4)
114:            direction ← pattern[J]
115:            SETDISTANCE()
116:            x ← x_i + direction
117:            if not OBSTACLEDETECTION(x, O) and ISINSIDE(x) then
118:                x_i ← x
119:            end if
120:        end if
121:    end function
122:
123:    function OBSTACLEDETECTION(x_i, O)
124:        x' ← x_i + d × direction
125:        return x' ∈ O
126:    end function
127:
128:    function ISINSIDE(x_i)
129:        return x_i ∈ X_i
130:    end function
131: end class
```

Stellenbosch University https://scholar.sun.ac.za

To perform a zigzag trip, a robot follows a direction according to its pattern. Let L be the length of a robot's trip. The robot travels that far in the chosen direction. It then changes its direction and updates the distance, travels again and so on. The full behaviour of zigzag search is shown in Algorithm 5.3.

In zigzag search, all parallel trips have the same length and only two distances are applied (assuming obstacle-free move): a short distance and a long distance. The short distance is always chosen to be $2d$, which avoids gaps or repeated sampling within a covered area arising from smaller or longer distances respectively. The long distance is obtained from the side length of the exploration region of the robot as well as the pattern and the direction of the robot. Let w denote the width of a robot's exploration region. For example if the robot's pattern is (E, N, W, N), the long distance will be $L = w - 2d$. We subtract $2d$ from w because a robot can detect a boundary of its exploration region when the boundary is within its perception range. The value of L is set before the robot starts exploring its exploration region. Once set, the robot uses L throughout the exploration of its current region. But a robot can cut a trip short when it encounters an obstacle in its exploration region, as shown in Figure A.2b.

The zigzag search algorithm is applied when a robot starts searching or when it has reached its assigned exploration region. A robot reaches its exploration region from a corner a_i (see Figure A.3a) and starts applying the zigzag pattern once it is at position s_i (see Figure A.3a). We want the robot to enter its exploration region from a_i so that it can apply the zigzag pattern properly. The procedure of how a robot moves to reach a_i or s_i is described in Section 5.5. The procedure GETSTARTINGPOINT in Algorithm A.1 shows how a robot R_i selects its starting point s_i.

Before we provide further details about a zigzag trip, we must mention a crucial consideration involved in search: obstacle avoidance. A mobile robot needs to be able to detect and avoid obstacles for effective search. The obstacle avoidance mechanism is the subject of Appendix B. Here, in an abstract way, we illustrate how a robot can detect an obstacle. A robot is equipped with sensors to detect an edge of an obstacle once it is within perception range. Sensors which fit this setting would be range sensors or tactile sensors [100].

Let x' denote a location further along robot's current direction such that

Stellenbosch University https://scholar.sun.ac.za

$\delta(x, x') = d$ (Figure A.3b). A robot decides to change its direction when $x' \in O$, where O denotes the set of obstacle locations. The algorithm for obstacle detection given in the procedure OBSTACLEDETECTION in Algorithm A.1.

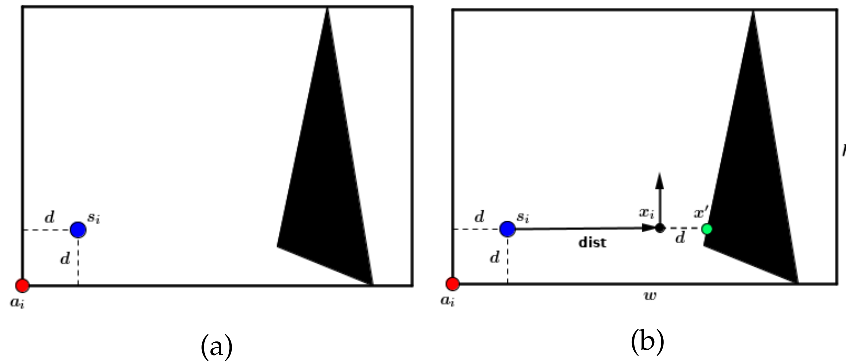


Figure A.3: *Illustration of a zigzag move and obstacle detection.* ***(A.3a)****: An initial configuration of robot's exploration region.* ***(A.3b)****: An illustration of zigzag move and obstacle detection.*

We now present the zigzag move mechanism (Algorithm A.1). This algorithm is presented from the perspective of a solitary search and it is assumed that $\gamma < d$, to avoid obstacles. A robot uses the procedure GETDISTANCE (see Algorithm A.1) to determine the length of its next trip. A Python implementation of the zigzag trip can be found in Appendix G.2.5.

Stellenbosch University https://scholar.sun.ac.za

Appendix B

Obstacle avoidance techniques

A robot needs to avoid obstacles while exploring a region. This section discusses the obstacle avoidance mechanisms a solitary robot applies in our methods to avoid obstacles encountered while searching. Various methods could be applied for this, including the potential field, vector field histogram and local navigation methods [32, 46, 105]. For simplicity, we use a Bug algorithm [80] to avoid obstacles. There exist various variants of Bug algorithms, including Bug1, Bug2, and Distance Bug (also known as Tangent Bug). We use the Distance Bug algorithm [80] as it is an improvement of the other two Bug algorithms. Bug algorithms are simple methods used for path planning to avoid an unexpected obstacle in the robot motion by updating the directional angle of a robot when an obstacle is detected.

We also use Bug algorithms because they assume only local knowledge of the search environment (i.e. obstacle shapes are unknown), the exact direction and distance to its goal, which satisfies the requirements for exploration of an unknown environment by solitary robots. When a robot using a Bug algorithm for obstacle avoidance detects an obstacle, it follows the boundary of the obstacle. This is achieved by using the following two principles:

- the robot attempts to navigate around the nearest detected obstacle;
- if there is no obstacle in perception range, the robot moves straight towards the goal.

In Distance Bug, when a robot encounters an obstacle, it chooses an obstacle-free location to move to, which minimises the distance to the goal while avoiding the obstacle. It should be noted that we modify the implementa-

Stellenbosch University https://scholar.sun.ac.za

tion of Distance Bug in this thesis—apart from obstacle avoidance, robots are also called to avoid moving through soft obstacles. This means that even when no obstacles are encountered by a robot, it might select a location based on the amount of information but which does not necessarily minimise the distance to the goal. This is done with the purpose of maximising coverage. However, we note that this additional condition might have an impact on the performance of Distance Bug when the travelling time to the goal is estimated assuming robots move in a straight line, but actually they are restricted to compass directions. In our setting, the goal can be either the closest corner a_i or the starting point s_i of the exploration region of the robot (when the robot is travelling to its exploration region, see Figure A.3a). Since an exploration region is selected from an unexplored area using our proposed coordination method, we expect robots to take straight lines to their exploration regions most of the time. A robot takes also a straight line to the goal when surrounded by explored area.

When an obstacle is detected, a robot chooses a location which yields high amount of information and short distance to the goal a. Let $u(b)$ denote an utility function associated to locations x_i and b, $I(b)$ the amount of information (i.e. unexplored area) the robot can obtain by moving to a location b and $\delta(b, a)$ the distance from a location b and the goal a. Considering our proposed coordination method (see Algorithm 5.3), this utility function will basically allow the robot to move to its new exploration region through margins (unexplored areas around its current exploration region). Recall that $B_d(b)$ denotes a closed ball of radius d centred at b. Let $\mathcal{B}_i^{(t)}$ denotes the set of (soft) obstacles. The utility function $u(b)$ is given by

$$u(b) = \frac{1 + I(b)}{1 + \delta(b, a)}, \tag{B.0.1}$$

where

$$I(b) = |B_d(b) \setminus \mathcal{B}_i^{(t)}| .$$

A procedure for a robot to select a location to move to is given in the procedure UTILITY in Algorithm B.1.

Situations where a robot is found to be in an enclosed space in the search environment using the Distance Bug method could occur. To avoid such situation, it is assumed that each open areas in the search environment are connected in the following way: each open cell allows movement in at least

Stellenbosch University https://scholar.sun.ac.za

two of the cardinal directions. Also, it is possible that a robot alternate west and east, or north and south while using the Distance Bug algorithm. The use of soft obstacles allows a robot to avoid some backtracking behaviour.

An abstract algorithm for one-step move using Distance Bug is given in Algorithm B.1, while a Python implementation can be found in Appendix G.2.9. An illustration of repeated use of Algorithm B.1 is shown in Figure B.1, i.e. a robot moves from an initial location to a goal using Algorithm B.1 repeatedly.

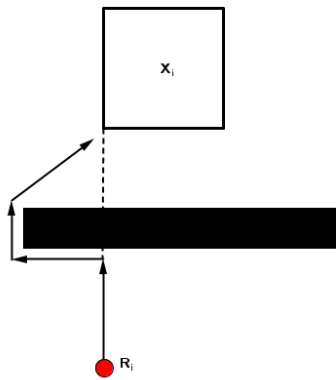


Figure B.1: *Illustration of obstacle avoidance using Distance Bug where the red filled circle denotes a robot R_i, the black rectangle an obstacle, and the white square with black boundary, X_i, the exploration region of R_i. This happens by applying the function* DISTANCEBUG *in Algorithm B.1 repeatedly as applied in Algorithm 5.3.*

Stellenbosch University https://scholar.sun.ac.za

Algorithm B.1 *A one-step move using the Distance Bug method for obstacle avoidance. The main function (see* DISTANCEBUG*) requires A set of visited locations, x_i current location of the robot, a the goal, $\mathcal{B}_i^{(t)}$ the set of soft obstacles and $\mathcal{O}$ the set of obstacles.*

```
1:  class BUG
2:      Class variables
3:      d                perception range
4:      γ                motion velocity
5:  ------------------------------------------------------------
6:      constructor (d, γ)
7:          (d, γ) ← (d, γ)
8:      end constructor
9:
10:     function DISTANCEBUG(A, x_i, a, B_i^(t), O)
11:         x ← x_i + γ (a − x_i)/‖a − x_i‖
12:         if x ∉ O and |B_d(x) \ B_i^(t)| > 0 then  ▷ B_d(x) denotes a closed ball of radius d centred at x
13:             return x
14:         else
15:             B ← {x_i + (0, γ), x_i + (0, −γ), x_i + (γ, 0), x_i + (−γ, 0)}
16:             B ← B \ O
17:             if B == ∅ then
18:                 return x_i
19:             end if
20:             b ← B.pop()                          ▷ returns a random set element
21:             D ← δ(a, b)          ▷ δ(a, b) denotes Euclidean distance between a and b
22:             I ← |B_d(b) \ B_i^(t)|
23:             u ← UTILITY(I, D)
24:             for b_i ∈ B do
25:                 D_i ← δ(a, b_i)
26:                 I_i ← |B_d(b_i) \ B_i^(t)|
27:                 u_i ← UTILITY(I_i, D_i)
28:                 if u < u_i then
29:                     b ← b_i
30:                     u ← u_i
31:                 end if
32:             end for
33:             return b
34:         end if
35:     end function
36:
37:     function UTILITY(I, D)
38:         return (1+I)/(1+D)
39:     end function
40: end class
```

Stellenbosch University https://scholar.sun.ac.za

Appendix C

Toy example of pruning method

To illustrate the pruning method, consider the following communication graph (Figure C.1).

We set D to 3. It is assumed that nodes know their immediate neighbours in advance. Table C.1 shows the initial situation of each node in the communication graph where each node knows its 1-hop neighbours only. At this stage, a node can not identify which nodes in its neighbourhood should be pruned. So they need to exchange their neighbouring information at least once.

For example, during the first round of pruning of this communication graph it can be observed that $v_1, v_{10}, v_{11}, v_{14}$ should be pruned. The updated node information after the first iteration is given in Table C.2. At this stage, all nodes know their 2-hop neighbours, so they can now identify which nodes in their respective neighbourhood should be deactivated in a distributed manner.

The node information after completion of the first pruning round is shown

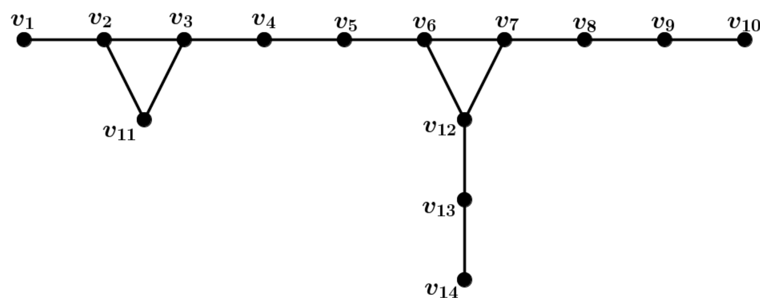


Figure C.1: *Communication graph used to illustrate pruning methods. This communication graph has diameter of* 9*, and contains* 14 *nodes and* 15 *edges.*

Stellenbosch University https://scholar.sun.ac.za

Nodes	Pruned?	$t-$hop	$\mathcal{F}_{i,0}$
v_1	False	1	None
v_2	False	1	None
v_3	False	1	None
v_4	False	1	None
v_5	False	1	None
v_6	False	1	None
v_7	False	1	None
v_8	False	1	None
v_9	False	1	None
v_{10}	False	1	None
v_{11}	False	1	None
v_{12}	False	1	None
v_{13}	False	1	None
v_{14}	False	1	None

Table C.1: *Initial situation before the use of pruning.*

Nodes	Pruned?	$t-$hop	$\mathcal{F}_{i,0}$
v_1	False	2	None
v_2	False	2	None
v_3	False	2	None
v_4	False	2	None
v_5	False	2	None
v_6	False	2	None
v_7	False	2	None
v_8	False	2	None
v_9	False	2	None
v_{10}	False	2	None
v_{11}	False	2	None
v_{12}	False	2	None
v_{13}	False	2	None
v_{14}	False	2	None

Table C.2: *The results of interaction after the first iteration.*

in Table C.3 and Figure C.2. At this stage, each node knows whether it should be pruned or not, as well as which of its immediate neighbours should be put on hold. For example the node v_{10} and its immediate neighbour v_9 know that v_{10} should be pruned. This means that there will be a reduced number of messages to be exchanged between nodes during subsequent communication interactions.

Now nodes which are put on hold do not participate in the subsequent interactions. Table C.4 shows updated information of the remaining nodes after the second iteration.

Other nodes will be pruned during the later rounds of pruning. For instance, in the second iteration, node v_2 will have no new information to

Stellenbosch University https://scholar.sun.ac.za

Nodes	Pruned?	$t-$hop	$\mathcal{F}_{i,1}$
v_1	True	2	v_1
v_2	False	2	v_1, v_{11}
v_3	False	2	v_{11}
v_4	False	2	None
v_5	False	2	None
v_6	False	2	None
v_7	False	2	None
v_8	False	2	None
v_9	False	2	v_{10}
v_{10}	True	2	v_{10}
v_{11}	True	2	v_{11}
v_{12}	False	2	None
v_{13}	False	2	v_{14}
v_{14}	True	2	v_{14}

Table C.3: *The results of the first pruning.*

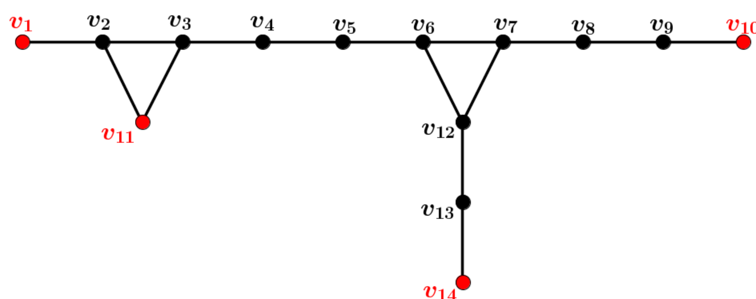


Figure C.2: *Pruned nodes from the first pruning stage. In red are nodes pruned and in black are nodes which are still involved in interaction.*

forward to node v_3, so it should be put on hold after the second iteration.

Nodes	Pruned?	$t-$hop	$\mathcal{F}_{i,1}$
v_2	False	3	v_1, v_{11}
v_3	False	3	v_{11}
v_4	False	3	None
v_5	False	3	None
v_6	False	3	None
v_7	False	3	None
v_8	False	3	None
v_9	False	3	v_{10}
v_{12}	False	3	None
v_{13}	False	3	v_{14}

Table C.4: *The results of interaction after the second iteration.*

After the second round of communication, the remaining nodes once again identify nodes among themselves which need to be put on hold. Table

Stellenbosch University https://scholar.sun.ac.za

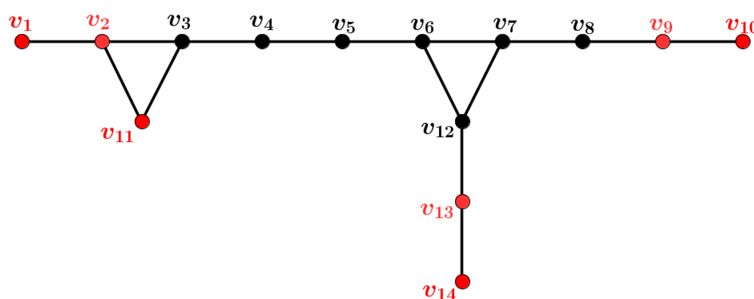


Figure C.3: *Pruned nodes after the second pruning stage. In red are nodes pruned up to this stage, and in black are nodes which are still involved in interaction.*

C.5 and Figure C.3 show new nodes which need to be put on hold. In this case, the nodes v_2, v_9 and v_{13} are now pruned. Only the remaining nodes, i.e. v_3, v_4, v_5, v_6, v_7 and v_8, should further interact with each other.

Nodes	Pruned?	$t-$hop	$\mathcal{F}_{i,2}$
v_2	True	3	v_1, v_2, v_{11}
v_3	False	3	v_2, v_{11}
v_4	False	3	None
v_5	False	3	None
v_6	False	3	None
v_7	False	3	None
v_8	False	3	v_9
v_9	True	3	v_9, v_{10}
v_{12}	False	3	v_{13}
v_{13}	True	3	v_{13}, v_{14}

Table C.5: *The results of the second round of pruning.*

Nodes	Pruned?	$t-$hop	$\mathcal{F}_{i,2}$
v_3	False	4	v_2, v_{11}
v_4	False	4	None
v_5	False	4	None
v_6	False	4	None
v_7	False	4	None
v_8	False	4	v_9
v_{12}	False	4	v_{13}

Table C.6: *Table showing the results of interaction after the third iteration.*

After the third iteration—with results as shown in Table C.6—the remaining nodes would usually again need to identify other nodes which should be put on hold, in this case we stop the algorithm because D was set to 3.

Stellenbosch University https://scholar.sun.ac.za

Appendix D

Toy example of Yayambo

To illustrate the Yayambo method, consider the following matrix giving the outputs of three classifiers for a four-class classification task,

$$\pi^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.1 & 0.5 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.1 \\ 0.195 & 0.005 & 0.795 & 0.005 \end{pmatrix} \end{array}. \tag{D.0.1}$$

The convergence threshold is set to 10^{-6} and the maximum number of iterations is set to 10. We first compute the support that is obtained from the ith distribution to another by applying the divergence (Equation 11.2.4) to the initial belief matrix $\pi^{(0)}$. At time $t = 1$, the supports assigned to distributions 2 and 3 from distribution 1 are

$$\begin{array}{c} \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.2538 & 0.2105 & 0.2644 & 0.0813 \\ 0.1756 & 0.0010 & 0.5449 & 0.0023 \end{pmatrix} \end{array}, \tag{D.0.2}$$

the supports assigned to distributions 1 and 3 from distribution 2 are

$$\begin{array}{c} \\ 1 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.0647 & 0.3985 & 0.1664 & 0.1794 \\ 0.1606 & 0.0016 & 0.5349 & 0.0034 \end{pmatrix} \end{array}, \tag{D.0.3}$$

and the supports assigned to distributions 1 and 2 from distribution 3 are

$$\begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.0823 & 0.4822 & 0.0708 & 0.1943 \\ 0.2632 & 0.2905 & 0.1311 & 0.0977 \end{pmatrix} \end{array}. \tag{D.0.4}$$

Stellenbosch University https://scholar.sun.ac.za

In Yayambo, supports are assigned to distributions, and we use an asymmetric metric to compute support. For example, in Equation D.0.2 a score of 0.1756 is assigned to distribution 3 for class c_1 from distribution 1, while a score of 0.0823 is assigned to distribution 1 for the same class from distribution 3 (Equation D.0.4). Towards convergence, almost the same support will be assigned to all distributions from all others on a particular class—this score could be high or small depending on whether it is meant to be the consensus class or not. That means with more iterations, class supports get closer. This is quite different from the sum rule method or any other fusion method based on indifference where each distribution contributes to a consensus decision equally.

So, using Equation 11.2.1, the belief matrix, after the first iteration is

$$\pi^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.1311 & 0.3232 & 0.4945 & 0.0511 \\ 0.1623 & 0.2883 & 0.5054 & 0.0439 \\ 0.2889 & 0.0165 & 0.6883 & 0.0062 \end{pmatrix} \end{array} .$$

Checking convergence using Equation 11.2.2,

$$\sum_{j=1}^{3} \|\pi_j^{(1)} - \pi_j^{(0)}\|_2 \approx 0.7720 > 3(10^{-6}) . \tag{D.0.5}$$

Since convergence has not yet been reached, Equation 11.2.4 is applied again. At time $t = 2$, the supports assigned to distributions 2 and 3 from distribution 1 are

$$\begin{array}{c} \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.1551 & 0.2717 & 0.4965 & 0.0434 \\ 0.2466 & 0.0064 & 0.5410 & 0.0054 \end{pmatrix} \end{array} , \tag{D.0.6}$$

the supports assigned to distributions 1 and 3 from distribution 2 are

$$\begin{array}{c} \\ 1 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.1240 & 0.3065 & 0.4855 & 0.0505 \\ 0.2500 & 0.0070 & 0.5463 & 0.0055 \end{pmatrix} \end{array} , \tag{D.0.7}$$

and the supports assigned to distributions 1 and 2 from distribution 3 are

$$\begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.0951 & 0.2991 & 0.3587 & 0.0501 \\ 0.1271 & 0.2677 & 0.3733 & 0.0431 \end{pmatrix} \end{array} . \tag{D.0.8}$$

Stellenbosch University https://scholar.sun.ac.za

Thus the distribution matrix becomes (using Equation 11.2.1),

$$\pi^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 0.0800 & 0.1366 & 0.7795 & 0.0038 \\ 0.0899 & 0.1339 & 0.7725 & 0.0036 \\ 0.1110 & 0.0163 & 0.8716 & 0.0010 \end{pmatrix} \end{array} .$$

Again checking convergence using Equation 11.2.2,

$$\sum_{j=1}^{3} \|\pi_j^{(2)} - \pi_j^{(1)}\|_2 \approx 0.9226 > 3(10^{-6}) \,, \tag{D.0.9}$$

we see that another iteration is required. Convergence is finally reached at time $t = 6$, with

$$\sum_{j=1}^{3} \|\pi_j^{(6)} - \pi_j^{(5)}\|_2 \approx 0.2240(10^{-6}) < 3(10^{-6}) \,.$$

Below we show class supports in the last iteration. At time $t = 6$, the supports assigned to distributions 2 and 3 from distribution 1 are

$$\begin{array}{c} \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 4.26 \times 10^{-8} & 1.93 \times 10^{-8} & 9.99 \times 10^{-1} & 2.03 \times 10^{-20} \\ 4.35 \times 10^{-8} & 1.02 \times 10^{-8} & 9.99 \times 10^{-1} & 1.54 \times 10^{-20} \end{pmatrix} \end{array} , \tag{D.0.10}$$

the supports assigned to distributions 1 and 3 from distribution 2 are

$$\begin{array}{c} \\ 1 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 4.16 \times 10^{-8} & 1.93 \times 10^{-8} & 9.99 \times 10^{-1} & 2.03 \times 10^{-20} \\ 4.35 \times 10^{-8} & 1.02 \times 10^{-8} & 9.99 \times 10^{-1} & 2.03 \times 10^{-20} \end{pmatrix} \end{array} , \tag{D.0.11}$$

and the supports assigned to distributions 1 and 2 from distribution 3 are

$$\begin{array}{c} \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 4.16 \times 10^{-8} & 1.93 \times 10^{-8} & 9.99 \times 10^{-1} & 2.03 \times 10^{-20} \\ 4.26 \times 10^{-8} & 1.93 \times 10^{-8} & 9.99 \times 10^{-1} & 2.03 \times 10^{-20} \end{pmatrix} \end{array} . \tag{D.0.12}$$

So the distribution matrix becomes (using Equation 11.2.1),

$$\pi^{(6)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{pmatrix} 4.16 \times 10^{-8} & 1.93 \times 10^{-8} & 9.99 \times 10^{-1} & 2.03 \times 10^{-20} \\ 4.26 \times 10^{-8} & 1.93 \times 10^{-8} & 9.99 \times 10^{-1} & 2.03 \times 10^{-20} \\ 4.35 \times 10^{-8} & 1.02 \times 10^{-8} & 9.99 \times 10^{-1} & 2.03 \times 10^{-20} \end{pmatrix} \end{array} .$$

It is observed that a high score is assigned to the consensus class c_3 from all distributions.

Stellenbosch University https://scholar.sun.ac.za

Appendix E

Dataset characteristics

The Iris dataset contains 150 observations, 50 each of 3 types of iris plant. Each observation is described by four features: sepal length, sepal width, petal length and petal width (all the features are measured in centimetres). The three types of iris plants are: *Iris Setosa*, *Iris Versicolour* and *Iris Virginica*. This is an exceedingly simple domain.

The Gesture dataset is composed of features extracted from 7 videos with people gesticulating holding a Microsoft Kinect sensor, aiming at studying Gesture Phase Segmentation. The dataset contains 19 numeric attributes and five types of gestures which are: rest, preparation, stroke, hold and retraction.

The Activity recognition dataset represents a real-life benchmark in the area of activity recognition applications. The dataset is provided for the task of classifying the activity performed by the user from time-series data generated by a Wireless Sensor Network (WSN). The WSN is composed of 3 sensors and each provides the Received Signal Strength (RSS) of the beacon packets they exchange among themselves in the WSN. The dataset has 6 attributes which are the average and standard deviation of the signal strength provided from each sensor. The dataset has 7 types of activity: walking, standing, sitting, lying, cycling and two types of bending.

The Handwritten digits dataset contains images of digits (0–9) hand written. Each digit is represented by a 28x28 image, which means each digit contains 784 pixels (or attributes). Here, the task is classifying the digits.

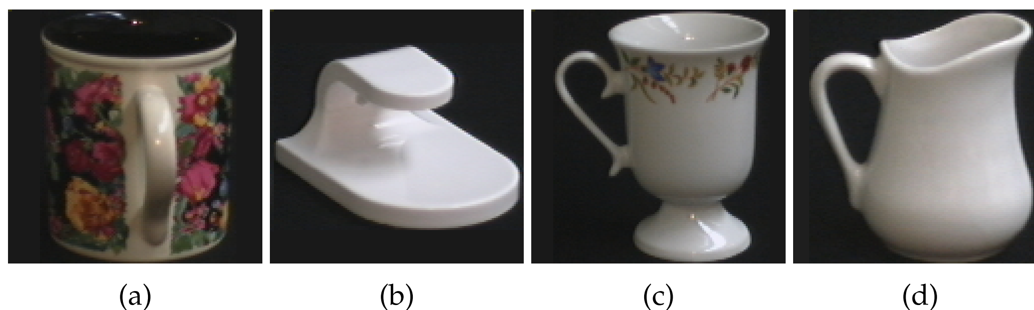
The Satellite database consists of multiple spectral values of pixels in 3x3 neighbourhoods in a satellite image, and the pixel type associated with the central pixel in each neighbourhood. The dataset contains 36 attributes per

Stellenbosch University https://scholar.sun.ac.za

observation (4 spectral bands and 9 pixels in neighbourhood) and 6 types of pixels which are: red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble, and very damp grey soil.

The Occupancy detection dataset consists of observations of occupancy of a room. The attributes of the dataset are temperature, relative humidity, light level, CO2 level and humidity ratio. The task is a binary classification: whether the room is occupied or not.

The Columbia dataset [78] is a database of 128x128 RGB (Red Green Blue) images of 100 types of objects (see Figure E.1 for some examples). A fixed camera and a turntable were used to collect these data. The objects were placed on a motorized turntable against a black background. The turntable was rotated through 360 degrees to vary object pose with respect to the camera. Images of the objects were taken at pose intervals of 5 degrees. This yields 72 images per object. This dataset was used in a multi-sensor system for object recognition. Each sensor could identify an object and display its angular pose. In our experiment, we arbitrarily only consider 15 types of objects. We use the OpenCV[1][14] (Open Source Computer Vision) library to convert the RGB images, which we flatten to a sequence of 16384 grayscale values with Numpy[2] [15]. Finally we use PCA for feature reduction where we retained 97 components. We build the PCA model on the all training data.



(a) (b) (c) (d)

Figure E.1: *Photographs of four of the objects in the Columbia dataset [78].* ***(E.1a)****: Object 1.* ***(E.1b)****: Object 2.* ***(E.1c)****: Object 3.* ***(E.1d)****: Object 4.*

[1]OpenCV is an open source computer vision and machine learning software library. It provides a common infrastructure for computer vision applications and accelerates the use of machine perception.

[2]Numpy is a fundamental Python library for scientific computing of N-dimensional array objects.

Stellenbosch University https://scholar.sun.ac.za

Appendix F

Utilities

This section presents the following algorithmic structures used in our proposed coordination method (which was described in Chapters 3–5): an enumeration of robot states (Algorithm F.1); an enumeration of robot states during planning (Algorithm F.2); an enumeration of robot states during search (Algorithm F.3); a class to model a rectangle to represent an exploration region or a virtual world (Algorithm F.4): we use `Numpy` to find the lowest leftmost point and the highest rightmost point for construction of a bounding rectangle where the x and y components are sorted separately. This leads to efficient approximations.

Algorithm F.1 *Enumeration of robot states.*

```
1: enum ROBOTSTATES
2:     COMPLETE
3:     INTERACTING
4:     PLANNING
5:     SEARCHING
6:     STOP
7:     WAITING
8: end enum
```

Algorithm F.2 *Enumeration of states involved in interaction of robots.*

```
1: enum PLANNINGSTATES
2:     CONSTRUCTION
3:     ELECTION
4:     FUSION
5:     TASK
6: end enum
```

Algorithm F.3 *Enumeration of search states for a robot.*

```
1: enum SEARCHSTATES
2:     INREGION
3:     OUTREGION
4: end enum
```

Stellenbosch University https://scholar.sun.ac.za

Algorithm F.4 *Creation of a rectangle and related subroutines used for coordination of solitary robots. We use* `Math.min` *and* `Math.max` *to find the lowest leftmost point (see* LOWESTLEFTMOSTPOINTOF*) and highest rightmost point (see* HIGHESTRIGHTMOSTPOINTOF*) of a set.*

```
1:  class RECTANGLE
2:      Class variables
3:      x           x-coordinate of the left-lower corner of 𝒳_i
4:      y           y-coordinate of the left-lower corner of 𝒳_i
5:      w           its width
6:      h           its height
7:  ----------------------------------------------------------------
8:      constructor (x, y, w, h)
9:          (x, y, w, h) ← (x, y, w, h)
10:     end constructor
11:
12:     function RECTANGLEEXTRACTION(N, D)
13:         sample a set S of N points randomly and uniformly over D
14:         (x_low, y_low) ← LOWESTLEFTMOSTPOINTOF(S)
15:         (x_high, y_high) ← HIGHESTRIGHTMOSTPOINTOF(S)
16:         R ← ∅
17:         for (l, h) ∈ S do
18:             L ← {(p_x, p_y) ∈ S : (x_low ≤ l ≤ x_high) ∧ (h = p_y)}
19:             H ← {(p_x, p_y) ∈ S : (l = p_x) ∧ (y_low ≤ h ≤ y_high)}
20:             if |L ∪ H| ≥ (y_high − y_low + x_high − x_low)/2 then
21:                 R ← R ∪ L ∪ H
22:             end if
23:         end for
24:         (x_low, y_low) ← LOWESTLEFTMOSTPOINTOF(R)
25:         (x_high, y_high) ← HIGHESTRIGHTMOSTPOINTOF(R)
26:         return RECTANGLE(x_low, y_low, x_high − x_low, y_high − y_low)
27:     end function
28:
29:     function RECTANGLETOSET(R)
30:         S ← ∅
31:         for i = R.x to R.x + R.w do
32:             for j = R.y to R.y + R.h do
33:                 S ← S ∪ {(i, j)}
34:             end for
35:         end for
36:         return S
37:     end function
38:
39:     function SETTORECTANGLE(S)
40:         (x, y) ← LOWESTLEFTMOSTPOINTOF(S)
41:         (q, r) ← HIGHESTRIGHTMOSTPOINTOF(S)
42:         (w, h) ← (q − x, r − y)
43:         R ← RECTANGLE(x, y, w, h)
44:         return R
45:     end function
46:
47:     function GETCORNERS()
48:         return {(x, y), (x, y + h), (x + w, y), (x + w, y + h)}
49:     end function
50:
51:     function GETCLOSESTCORNER(x_i)
52:         corners ← GETCORNERS()
53:         c ← corners.pop()
54:         u ← δ(c, x_i)
55:         for c_i ∈ corners do
56:             u_i ← δ(c_i, x_i)
57:             if u_i < u then
58:                 c ← c_i
59:                 u ← u_i
60:             end if
61:         end for
62:         return c
63:     end function
64: end class
```

Stellenbosch University https://scholar.sun.ac.za

Appendix G

Code of Python functions

This thesis has proposed solutions to three types of problems, namely: decentralised construction of a view of a communication graph, coordination and classifier fusion. We have implemented these solutions to test their relevance in the literature. The source code for our solutions can be found at `https://bitbucket.org/jmf-mas/codes`. This chapter presents some of the main modules involved in our code for the clarification of our thesis, as they are referred to in the main text of this thesis. It should noted that apart from the Python implementations of various classes for our proposed methods found in the repository, we also implemented two additional classes (messaging.py and messageTypes.py) for message management between interacting nodes. In the first class we implement management of sending and reception of messages between interacting nodes (or interacting robots for coordination). We use the concepts of serialization and deserialization, as well as dictionary. The second class is an enumeration for different types of messages nodes can exchange between themselves.

G.1 Network view construction

This is the implementation of how a node checks whether it or its neighbour should be put on hold while constructing a view of the network formed in their interaction. We illustrate the codes for leaves, special neighbours of leaves, nodes and edges causing a triangle.

```
1 from pympler import asizeof
2 import time
3 import networkx as nx
4 import os
```

Stellenbosch University https://scholar.sun.ac.za


```
5  os.chdir("..")
6  from messaging import Messaging
7  from messageTypes import MessageTypes
8
9  class PruningNode:
10
11     def __init__(self, i, Ni, D=10):
12
13         self._id = i
14         self._Ni = Ni.copy()
15         self._NitUp = Ni.copy()
16         self._Sit = [set([self.edge(i, j) for j in self._Ni])]
17         self._Sjt = {j:[] for j in self._Ni}
18         self._Si = set([self.edge(i, j) for j in self._Ni])
19         self._Qi = []
20         self._centrality = -1
21
22         self._D = D
23         self._equilibrium = False
24         self._t = 0
25         self._finished = False
26
27         self._nbMessages = 0
28         self._runningTime = time.time()
29         self._memorySize = 0
30
31         self._msgType = MessageTypes
32         self._messaging = Messaging()
33
34         self._hold = False
35
36     def edge(self, i, j):
37
38         return (min(i, j), max(i, j))
39
40     def initialOneHop(self):
41
42         self._t += 1
43         for j in self._NitUp:
44             M = self._messaging.deserialize(self._msgType.M)
45             M = self._messaging.addTo(M, j, self._id, self._Sit[-1])
46              self._messaging.serialize(M, self._msgType.M)
47              self._nbMessages +=1
48
49     def initialUpdate(self):
50
51         if not self._finished:
52             M = self._messaging.deserialize(self._msgType.M)
53
54              if self._id in M and not self._equilibrium and self._t<self._D:
55                  S = set([])
56                  Q = {self._id:set(self._NitUp)}
57
58                  while(len(M[self._id])>=1):
59                      self._nbMessages +=1
```

Stellenbosch University https://scholar.sun.ac.za


```
60                     j = list(M[self._id].keys())[0]
61                     St = M[self._id][j]
62 
63                     if j in self._Sjt:
64                         self._Sjt[j].append(St)
65                     else:
66                         self._Sjt[j] = [St]
67                     S = S.union(St)
68                     q = set([k for l in St for k in l])
69                     if self._t==1:
70                         q = q.difference(set([j]))
71                     Q[j] = q
72                     M = self._messaging.deleteFrom(M, self._id, j)
73 
74                 self._messaging.serialize(M, self._msgType.M)
75 
76                 self._Qi.append(Q)
77                 S = S.difference(self._Si.copy())
78                 self._Sit.append(S)
79                 self._Si = self._Si.union(S)
80 
81                 self._equilibrium = S==set([])
82 
83     def leavesDetection(self):
84 
85         Fit = set([])
86         for j in set(self._NitUp).union(set([self._id])):
87             if len(self._Qi)>0 and j in self._Qi[0] and len(self._Qi[0][j])==1:
88                 Fit.add(j)
89 
90         return Fit
91 
92     def triangleDetection(self, Fit):
93 
94         for j in set(self._NitUp).union(set([self._id])):
95             if len(self._Qi)>0 and j in self._Qi[0] and len(self._Qi[0][j])==2:
96                 f, g = self._Qi[0][j]
97                 if g in self._Qi[0] and f in self._Qi[0][g]:
98                     Fit.add(j)
99 
100        return Fit
101
102    def furtherPruningDetection(self):
103
104        Fit = set([])
105        St = self._Si.difference(self._Sit[-1])
106
107        for j in self._NitUp:
108            if len(self._Sjt[j])>0 and self._Sjt[j][-1].issubset(St):
109                Fit.add(j)
110
111        if len(self._NitUp)==1 and self._Sit[-1]!=set():
112            Fit.add(self._id)
113            self._equilibrium = True
114            self._hold =True
```

Stellenbosch University https://scholar.sun.ac.za


```
115
116         return Fit
117
118     def firstPruningDetection(self):
119
120         self._hold = False
121         Fit = self.leavesDetection()
122         Fit =self.triangleDetection(Fit)
123         self._Fit = Fit
124         self._Fi = Fit
125         if self._id in self._Fi:
126             self._equilibrium = True
127             self._hold = True
128
129     def nextOneHop(self):
130
131         if not self._finished:
132             self._t += 1
133             self._NitUp = set(self._NitUp).difference(self._Fi)
134             if not self._equilibrium:
135                 for j in self._NitUp:
136                     M = self._messaging.deserialize(self._msgType.M)
137                     M = self._messaging.addTo(M, j, self._id, self._Sit[-1])
138                     self._messaging.serialize(M, self._msgType.M)
139                     self._nbMessages +=1
140
141     def nextUpdate(self):
142
143         if not self._finished:
144             M = self._messaging.deserialize(self._msgType.M)
145
146             if self._id in M and not self._equilibrium and self._t<self._D:
147                 S = set([])
148                 while(len(M[self._id])>=1):
149                     self._nbMessages +=1
150                     j = list(M[self._id].keys())[0]
151                     St = M[self._id][j]
152                     self._Sjt[j].append(St)
153                     S = S.union(St)
154                     M = self._messaging.deleteFrom(M, self._id, j)
155
156                 self._messaging.serialize(M, self._msgType.M)
157
158                 S = S.difference(self._Si.copy())
159                 self._Sit.append(S)
160                 self._Si = self._Si.union(S)
161
162                 self._Fit=self.furtherPruningDetection()
163                 self._Fi = self._Fi.union(self._Fit)
164
165                 self._equilibrium = S==set([])
166
167     def isEnded(self):
168
169         if not self._finished:
```

Stellenbosch University https://scholar.sun.ac.za


```
170             self._finished = self._t==self._D or self._equilibrium or self._hold
171             if self._finished:
172 
173                 self._runningTime = time.time()- self._runningTime
174                 self._memorySize = 0.000001*asizeof.asizeof([self])
175                 if not self._hold and len(self._Si)>0:
176                     G = nx.Graph()
177                     G.add_edges_from(self._Si)
178                     self._centrality = nx.closeness_centrality(G)[self._id]
179                 else:
180                     self._centrality = 0
181 
182         return self._t==self._D or self._equilibrium or self._hold
```

Listing G.1: Implementation of the pruning method for failure-free cases

```
 1 class PruningNodeFD(PruningNode):
 2 
 3     def __init__(self, i, Ni, D=10, T=0.01, pf=0.1, pr=0.8):
 4 
 5         super(PruningNodeFD, self).__init__(i, Ni)
 6 
 7         self._Gi = set([])
 8         self._NiDown = set([])
 9         self._SiDown = set([])
10         self._EiDown = set([])
11         self._Ti = {j: time.time() for j in self._Ni}
12         self._T = T
13         self._pf = pf
14         self._pr = pr
15         self._isDown = False
16 
17     def isPersistent(self, signal):
18 
19         if signal not in self._Gi and np.random.choice([0, 1], p=[self._pf, 1-self
    ._pf])==1:
20             self._Gi  = self._Gi.union(signal)
21             (S, Q, c) = signal
22             if (1-S, Q, c) in self._Gi:
23                 self._Gi.remove((1-S, Q, c))
24             return True
25         return False
26 
27     def nodeFailureDetection(self, j):
28 
29         if time.time()-self._Ti[j]>self._T:
30             self._NitUp.remove(j)
31             self._NiDown.add(j)
32             for k in self._NitUp:
33                 P = self._messaging.deserialize(self._msgType.P)
34                 P = self._messaging.addTo(P, k, self._id, [0, 0, j])
35                 self._messaging.serialize(P, self._msgType.P)
36 
37 
38     def edgeFailureDetection(self, j):
```

Stellenbosch University https://scholar.sun.ac.za


```
39
40          isDown = False
41          if self.edge(self._id, j) not in self._Gi:
42              if np.random.choice([0, 1], p=[self._pf, 1-self._pf])==0:
43                  self._Gi.add(self.edge(self._id, j))
44                  isDown = True
45
46              P = self._messaging.deserialize(self._msgType.P)
47              if isDown:
48                  self._SiDown.add(self.edge(self._id, j))
49                  self._EiDown.add(j)
50                  for k in self._NitUp:
51                      P = self._messaging.deserialize(self._msgType.P)
52                      P = self._messaging.addTo(P, k, self._id, [0, 1, self.edge(
        self._id, j)])
53                      self._messaging.serialize(P, self._msgType.P)
54                  self._nbMessages +=1
55          return isDown
56
57      def recoveryDetection(self, j, isDown):
58
59          if isDown and j in self._NiDown:
60              if np.random.choice([0, 1], p=[self._pr, 1-self._pr])==0:
61                  self._NitUp.add(j)
62                  self._NiDown.remove(j)
63                  self._EiDown.add(j)
64                  P = self._messaging.deserialize(self._msgType.P)
65                  for k in self._NitUp:
66                      P = self._messaging.addTo(P, j, self._id, [0, 1, j])
67                      self._messaging.serialize(P, self._msgType.P)
68                  self._nbMessages +=1
69
70          if isDown and self.edge(self._id, j) in self._SiDown:
71              self._SiDown.remove(self.edge(self._id, j))
72              self._EiDown.remove(j)
73              P = self._messaging.deserialize(self._msgType.P)
74              for k in self._NitUp:
75                  P = self._messaging.addTo(P, j, self._id, [1, 1, self.edge(self.
        _id, j)])
76                  self._messaging.serialize(P, self._msgType.P)
77              self._nbMessages +=1
78
79
80      def forwardFailureSignals(self):
81
82          P = self._messaging.deserialize(self._msgType.P)
83          if self._id in P:
84
85              while(len(P[self._id])>=1):
86                  j = list(P[self._id].keys())[0]
87                  (S, Q, c) = P[self._id][j]
88                  if self.isPersistent((S, Q, c)):
89                      if (S, Q)==(0, 0):
90                          self._NiDown.add(c)
91                      if (S, Q)==(0, 1):
```

Stellenbosch University https://scholar.sun.ac.za


```
92                         self._SiDown.add(c)
93                     if (S, Q)==(1, 0):
94                         self._NiDown = self._NiDown.difference(set([c]))
95                     if (S, Q)==(1, 1):
96                         self._SiDown = self._SiDown.difference(set([c]))
97                     for k in self._NitUp:
98                         P = self._messaging.addTo(P, k, self._id, [S, Q, c])
99                     self._messaging.serialize(P, self._msgType.P)
100
101                self._messaging.deleteFrom(P, self._id, j)
102                self._nbMessages +=1
103
104    def initialOneHop(self):
105
106        if not self._finished:
107            self._t += 1
108
109            if not self._isDown:
110                if np.random.choice([0, 1], p=[self._pf, 1-self._pf])==0:
111                    self._isDown = True
112                else:
113                    for j in self._NitUp:
114                        M = self._messaging.deserialize(self._msgType.M)
115                        M = self._messaging.addTo(M, j, self._id, self._Sit[-1])
116                        self._messaging.serialize(M, self._msgType.M)
117                        self._Ti[j] = time.time()
118
119                        if np.random.choice([0, 1], p=[self._pf, 1-self._pf])==0:
120                            self._Gi.add(self.edge(self._id, j))
121            else:
122                if np.random.choice([0, 1], p=[self._pr, 1-self._pr])==0:
123                    self._isDown = False
124
125    def nextOneHop(self):
126
127        if not self._finished:
128            self._t += 1
129            self._NitUp = set(self._NitUp).difference(self._Fi)
130            if not self._isDown:
131                if np.random.choice([0, 1], p=[self._pf, 1-self._pf])==0:
132                    self._isDown = True
133                else:
134                    if not self._equilibrium:
135                        for j in self._NitUp:
136                            M = self._messaging.deserialize(self._msgType.M)
137                            M = self._messaging.addTo(M, j, self._id, self._Sit
    [-1])
138                            self._messaging.serialize(M, self._msgType.M)
139                            self._Ti[j] = time.time()
140
141                            if np.random.choice([0, 1], p=[self._pf, 1-self._pf])
    ==0:
142                                self._Gi.add(self.edge(self._id, j))
143            else:
144                if np.random.choice([0, 1], p=[self._pr, 1-self._pr])==0:
```

Stellenbosch University https://scholar.sun.ac.za


```
145                     self._isDown = False
146
147     def initialUpdate(self):
148
149         if not self._finished:
150             M = self._messaging.deserialize(self._msgType.M)
151             if self._id in M:
152                 S = set([])
153                 Q = {self._id:set(self._NitUp)}
154
155                 while(len(M[self._id])>=1):
156                     NitUp = self._NitUp.copy()
157                     for k in NitUp:
158                         self.nodeFailureDetection(k)
159                     self._nbMessages +=1
160                     j = list(M[self._id].keys())[0]
161                     St = M[self._id][j]
162                     self._Ti[j] = time.time()
163                     isDown = self.edgeFailureDetection(j)
164                     if not isDown:
165                         self._Sjt[j].append(St)
166                         S = S.union(St)
167                         q = set([k for l in St for k in l])
168                         if self._t==1:
169                             q = q.difference(set([j]))
170                         Q[j] = q
171                     self.recoveryDetection(j, isDown)
172
173                     Nup = self._NitUp.copy()
174                     Nup = Nup.difference(self._EiDown)
175                     Nup = Nup.difference(self._NiDown)
176                     self._NitUp = Nup.copy()
177                     M = self._messaging.deleteFrom(M, self._id, j)
178
179                 self._messaging.serialize(M, self._msgType.M)
180
181                 self._Qi.append(Q)
182                 self.forwardFailureSignals()
183                 S = S.difference(self._Si.copy())
184                 self._Sit.append(S)
185                 self._Si = self._Si.union(S)
186
187                 self._equilibrium = S==set([])
188                 self.finalIteration()
189
190     def nextUpdate(self):
191
192         if not self._finished:
193             M = self._messaging.deserialize(self._msgType.M)
194             if self._id in M and not self._equilibrium:
195                 S = set([])
196                 while(len(M[self._id])>=1):
197                     NitUp = self._NitUp.copy()
198                     for k in NitUp:
199                         self.nodeFailureDetection(k)
```

Stellenbosch University https://scholar.sun.ac.za


```
200                         self._nbMessages +=1
201                         j = list(M[self._id].keys())[0]
202                         St = M[self._id][j]
203                         self._Sjt[j].append(St)
204                         S = S.union(St)
205 
206                         self._Ti[j] = time.time()
207                         isDown = self.edgeFailureDetection(j)
208 
209                         if not isDown:
210                             self._Sjt[j].append(St)
211                             S = S.union(St)
212 
213                         self.recoveryDetection(j, isDown)
214 
215                         Nup = self._NitUp.copy()
216                         Nup = Nup.difference(self._EiDown)
217                         Nup = Nup.difference(self._NiDown)
218                         self._NitUp = Nup.copy()
219                         M = self._messaging.deleteFrom(M, self._id, j)
220 
221                     self._messaging.serialize(M, self._msgType.M)
222 
223 
224                     S = S.difference(self._Si.copy())
225                     self._Sit.append(S)
226                     self._Si = self._Si.union(S)
227                     self.forwardFailureSignals()
228                     self._Fit=self.furtherPruningDetection()
229                     self._Fi = self._Fi.union(self._Fit)
230                     self._equilibrium = S==set([])
231                     self.finalIteration()
232 
233     def finalIteration(self):
234 
235         if self._equilibrium and self._t<self._D:
236             if not self._isDown:
237                 if np.random.choice([0, 1], p=[self._pf, 1-self._pf])==0:
238                     self._isDown = True
239                 else:
240                     if self._equilibrium and self._t<self._D:
241                         Nup = self._NitUp.copy()
242                         Nup = Nup.difference(self._EiDown)
243                         Nup = Nup.difference(self._NiDown)
244                         self._NitUp = Nup.copy()
245                         for j in self._NitUp:
246                             M = self._messaging.deserialize(self._msgType.M)
247                             M = self._messaging.addTo(M, j, self._id, self._Sit
    [-1])
248                             self._messaging.serialize(M, self._msgType.M)
249 
250                     if self._equilibrium:
251                         E = set([])
252                         for j in self._NiDown:
253                             for k in self._NiDown:
```

Stellenbosch University https://scholar.sun.ac.za


```
254                                 if self._id!=k:
255                                     E.add(self.edge(j, k))
256                         self._Si = self._Si.difference(E)
257             else:
258                 if np.random.choice([0, 1], p=[self._pr, 1-self._pr])==0:
259                     self._isDown = False
260 
261     def isEnded(self):
262 
263         if not self._finished:
264             self._finished = self._t==self._D or self._equilibrium
265             if self._finished:
266                 self._runningTime = time.time()- self._runningTime
267                 self._memorySize = 0.000001*asizeof.asizeof([self])
268 
269         return self._t==self._D or self._equilibrium
```

Listing G.2: Implementation of the pruning method for failure cases

G.2 Coordination

We present some code segments for the coordination strategy for solitary robots in this section.

G.2.1 Virtual world enlargement

This code implements how a robot enlarges its virtual world.

```
1 def _virtualWorldEnlargement(self):
2 
3     self._Vi.x -= self._w0
4     self._Vi.y -= self._w0
5     self._Vi.w +=2*self._w0
6     self._Vi.h +=2*self._w0
```

Listing G.3: Implementation of virtual world enlargement

G.2.2 Soft obstacles

This code implements how a leader determines new exploration regions of robots in an interaction—cellular decomposition.

```
1 def _getCoordinationTargets(self, Mi=100):
2 
3         xmin, ymin = min(self._Bit[-1])
4         xmax, ymax = max(self._Bit[-1])
5         self._Vi = Rectangle([xmin, ymin], xmax-xmin, ymax-ymin)
6         dtau = self._tau - self._t
```

Stellenbosch University https://scholar.sun.ac.za


```
7          A = (np.pi*self._d**2+2*self._d*self._gamma*(dtau))
8          A = (np.sqrt(A)+2*self._m)**2
9          xi = self._getCentralPoint()
10         self._virtualWorldEnlargement()
11         P = len(self._Ii)
12         k = 1
13         Xs = []
14         Ms = []
15         self._Bit[-1] = self._Bit[-1].union(self._Cit[-1])
16         U = self._Bit[-1].copy()
17         B = []
18
19         while k<=P:
20             l = 1
21             nextSampling = True
22             while nextSampling:
23
24                 VS = self._rectangleToSet(self._Vi)
25                 Q = VS.difference(U)
26                 Qr = list(Q)
27                 N = min(Mi, len(Qr))
28                 indices = np.random.choice([j for j in range(len(Qr))], replace=
       False, size=N)
29
30                 for i in indices:
31
32                     w = int(np.random.randint(2*(self._d+self._m), np.sqrt(A)+ 2*(
       self._d+self._m)))
33                     h = max(2*(self._d+self._m), int(A/w))
34                     lx, ly = Qr[i]
35                     R = Rectangle([lx, ly], w, h)
36                     RS = self._rectangleToSet(R)
37
38                     if RS.intersection(U)==set([]) and RS.issubset(VS):
39
40                         U.add((lx, ly))
41                         U = U.union(RS)
42                         X = Rectangle([R.x+self._m, R.y+self._m], R.w-2*self._m, R
       .h-2*self._m)
43                         XSet = self._rectangleToSet(X)
44                         M = RS.difference(XSet)
45                         Xs.append(X)
46                         Ms.append(M)
47                         B.append(self._getClosestCorner(X, xi))
48                         k +=1
49                         nextSampling = False
50                         break
51
52                 l +=1
53                 if l==self._L:
54                     l = 1
55                     self._virtualWorldEnlargement()
56
57         robotsIDS, cost = self._getCostMatrix(B)
58         ids, targets = assignment(cost)
```

Stellenbosch University https://scholar.sun.ac.za


```
59        self._Gamma = {robotsIDS[i]:Xs[a] for (i, a) in zip(ids, targets)}
```

Listing G.4: Implementation of soft obstacles

G.2.3 Trip to an exploration region

This function shows how a robot moves to its exploration region, after an interaction, or when it decides to choose a new exploration region under some circumstances.

```
1  def _travellingToXi(self):
2
3          x0 = self._xi + np.array([0, self._gamma])
4          x1 = self._xi + np.array([self._gamma, 0])
5          x2 = self._xi + np.array([-self._gamma, 0])
6          x3 = self._xi + np.array([0, -self._gamma])
7
8          B = set([tuple(x0), tuple(x1), tuple(x2), tuple(x3)])
9          B = B.difference(self._O)
10         B = B.difference(self._AB)
11
12         if B!=set([]):
13             if np.linalg.norm(self._xi-self._ai)>2*self._d:
14                 I = 2*self._gamma*self._d
15                 Us = []
16                 for bi in B:
17                     ball = Ball(bi, self._d + self._gamma)
18                     I_ = len(ball.Ball.difference(self._Bit[-1]).difference(self.
    _O))
19                     b = np.array(bi).copy()
20                     if I_>=I:
21                         Us.append(b)
22                 if len(Us)==0:
23                     b = B.pop()
24                     dist = np.linalg.norm(np.array(b)-self._ai)
25                     for bi in B:
26                         dist_ = np.linalg.norm(np.array(bi)-self._ai)
27                         if dist_<dist:
28                             dist = dist_
29                             b = bi
30                     self._xi = np.array(b)
31                 else:
32                     b = Us.pop()
33                     dist = np.linalg.norm(np.array(b)-self._ai)
34                     for bi in Us:
35                         dist_ = np.linalg.norm(np.array(bi)-self._ai)
36                         if dist_<dist:
37                             b = bi
38                     self._xi = np.array(b)
39             else:
40                 b = B.pop()
41                 dist = np.linalg.norm(np.array(b)-self._ai)
```

Stellenbosch University https://scholar.sun.ac.za


```
42                 for bi in B:
43                     dist_ = np.linalg.norm(np.array(bi)-self._ai)
44                     if dist_<dist:
45                         dist = dist_
46                         b = bi
47                 self._xi = np.array(b)
48         self._AB.add((int(self._xi[0]), int(self._xi[1])))
```

Listing G.5: Trip to an exploration region

G.2.4 Zigzag pattern

Here we define the four zigzag directions and the eight zigzag patterns.

```
1  # direction towards north
2  def N():
3      return [0, gamma]
4  def S():
5      return [0, -gamma]
6  def E():
7      return [gamma, 0]
8  def W():
9      return [-gamma, 0]
10 # list of the eight zigzag patterns
11 def patterns_zigzag():
12     return [[E(), N(),W(), N()], [E(), S(),W(), S()],
13             [W(), N(),E(), N()], [W(), S(),E(), S()],
14             [N(), E(),S(), E()], [N(), W(),S(), W()],
15             [S(), E(),N(), E()], [S(), W(),N(), W()]]
```

Listing G.6: Zigzag pattern

G.2.5 One zigzag trip algorithm

This is the implementation of an algorithm for a zigzag trip which a robot applies to explore its assigned region.

```
1  import numpy as np
2  from coordinationUtils import Ball
3
4  class Zigzag:
5
6      def __init__(self, gamma, d):
7
8          self._d = d
9          self._gamma = gamma
10
11     def paramReinit(self, Xi, xi, ci):
12
13         self._Xi = Xi
14         self._xi = xi
```

Stellenbosch University https://scholar.sun.ac.za


```
15         self._ci = ci
16         self._dist = 0
17         self._J = 1
18         self.setPattern()
19         self.setDirection()
20         self.setDistance()
21 
22     def getPatterns(self):
23 
24         return [[self.East(), self.North(), self.West(), self.North()],
25             [self.East(), self.South(), self.West(), self.South()],
26             [self.West(), self.North(), self.East(), self.North()],
27             [self.West(), self.South(), self.East(), self.South()],
28             [self.North(), self.East(),self.South(), self.East()],
29             [self.North(), self.West(), self.South(), self.West()],
30             [self.South(), self.East(),self.North(), self.East()],
31             [self.South(), self.West(), self.North(), self.West()]]
32 
33     def North(self):
34 
35         return [0, self._gamma]
36 
37     def South(self):
38 
39         return [0, -self._gamma]
40 
41     def East(self):
42 
43         return [self._gamma, 0]
44 
45     def West(self):
46 
47         return [-self._gamma, 0]
48 
49     def setPattern(self):
50 
51         x, y, w, h = int(self._Xi.x), int(self._Xi.y), int(self._Xi.w), int(self.
    _Xi.h)
52         if x == self._ci[0] and y == self._ci[1]:
53             self._pattern =  self.getPatterns()[np.random.choice([0, 4])]
54         elif x == self._ci[0] and y + h == self._ci[1]:
55             self._pattern =  self.getPatterns()[np.random.choice([1, 6])]
56         elif x + w == self._ci[0] and y == self._ci[1]:
57             self._pattern =  self.getPatterns()[np.random.choice([2, 5])]
58         else:
59             self._pattern =  self.getPatterns()[np.random.choice([3, 7])]
60 
61     def setDirection(self):
62 
63         self._direction = self._pattern[int(self._J%4)-1]
64 
65     def getStartingPoint(self):
66 
67         if (self._pattern[0]==self.North() and self._pattern[1]==self.East()) or (
    self._pattern[0]==self.East() and self._pattern[1]==self.North()):
```

Stellenbosch University https://scholar.sun.ac.za


```
68             return np.array(self._ci) + [self._d, self._d]
69         elif (self._pattern[0]==self.South() and self._pattern[1]==self.East()) or
    (self._pattern[0]==self.East() and self._pattern[1]==self.South()):
70             return  np.array(self._ci) + [self._d, -self._d]
71         elif (self._pattern[0]==self.North() and self._pattern[1]==self.West()) or
    (self._pattern[0]==self.West() and self._pattern[1]==self.North()):
72             return  np.array(self._ci) + [-self._d, self._d]
73         else:
74             return  np.array(self._ci) + [-self._d, -self._d]
75
76     def setDistance(self):
77
78         if self._J%2==1:
79             if np.abs(self._direction).tolist()==[0, self._gamma]:
80                 self._distance =  int(self._Xi.h-2*self._d)
81             else:
82                 self._distance = int(self._Xi.w-2*self._d)
83         else:
84             if np.abs(self._direction).tolist()==[0, self._gamma]:
85                 self._distance = int(np.min([2*self._d, self._Xi.h-self._d*self._J
    ]))
86             else:
87                 self._distance = int(np.min([2*self._d, self._Xi.w-self._d*self._J
    ]))
88
89     def zigzagMove(self, B, O):
90
91         self._B = set([])
92         xi = self._xi + self._direction
93         obstacle = self.obstacleDetection(xi, O)
94         if not obstacle and self._dist<=self._distance-self._gamma and self.
    isInside(xi):
95             self._xi = xi
96             self._dist +=self._gamma
97         else:
98
99             if obstacle:
100                j = 1
101                for d in range(self._dist, self._distance):
102                    ball = Ball(self._xi + j*np.array(self._direction), self._d)
103                    j +=1
104                    self._B = B.union(ball.Ball)
105
106            self._J = (self._J+1)%4
107            self._direction = self._pattern[self._J-1]
108            xi = self._xi + self._direction
109            self._dist =self._gamma
110            self.setDistance()
111            obstacle = self.obstacleDetection(xi, O)
112            if not obstacle and self.isInside(xi):
113                self._xi = xi
114
115    def obstacleDetection(self, x, O):
116
117        x0 = int(x[0]) +self._d*self._direction[0]
```

Stellenbosch University https://scholar.sun.ac.za


```
118         y0 = int(x[1]) +self._d*self._direction[1]
119 
120         return (x0, y0) in O
121 
122     def isInside(self, xi):
123 
124         xin = self._d/2 <= xi[0]<=self._Xi.x + self._Xi.w - self._d/2
125         yin = self._Xi.y + self._d/2 <= xi[1]<=self._Xi.y + self._Xi.h - self._d/2
126 
127         return xin and yin
```

Listing G.7: Implementation of a one-zigzag trip

G.2.6 Leader election method

This is the implementation of the leader election algorithm used in interaction of robots.

```
1 class LeaderElection:
2 
3     def __init__(self, Node):
4 
5         self._id = Node._id
6         self._Ni = Node._Ni
7         self._lCentrality = Node._centrality
8 
9         self._treeParent = None
10        self._treeChildren = set([])
11 
12        self._leaderID = self._id
13        self._distance = 0
14        self._wait = set([])
15 
16        self._finished = False
17 
18        self._msgType = MessageTypes
19        self._messaging = Messaging()
20 
21    def firstHop(self):
22 
23        for j in self._Ni:
24 
25            BFS_GO = self._messaging.deserialize(self._msgType.BFS_GO)
26            BFS_GO = self._messaging.addTo(BFS_GO, j, self._id, [self._leaderID,
    self._lCentrality, self._distance])
27            self._messaging.serialize(BFS_GO, self._msgType.BFS_GO)
28            self._wait.add(j)
29 
30        self._finished = self._wait==set([])
31 
32    def isBigger(self, l, cl):
33 
34        k = self._leaderID
```

Stellenbosch University https://scholar.sun.ac.za


```
35            ck = self._lCentrality
36
37            return (ck<cl) or ((ck==cl) and l<k)
38
39        def isEnded(self):
40
41            return self._finished
42
43        def treeParentUpdate(self):
44            BFS_GO = self._messaging.deserialize(self._msgType.BFS_GO)
45            if self._id in BFS_GO:
46                while len(BFS_GO[self._id])>=1:
47
48                    j = list(BFS_GO[self._id].keys())[0]
49                    (mid, cl, dist) = BFS_GO[self._id][j]
50                    BFS_GO = self._messaging.deleteFrom(BFS_GO, self._id, j)
51                    self._messaging.serialize(BFS_GO, self._msgType.BFS_GO)
52
53                    if  self.isBigger(mid, cl):
54                        self._leaderID = mid
55                        self._distance = 99999999999999
56                        self._treeParent = None
57                        self._lCentrality = cl
58
59                    if self._leaderID==mid and dist+1<self._distance:
60                        BFS_BACK = self._messaging.deserialize(self._msgType.BFS_BACK)
61                        if self._treeParent!=None:
62                            BFS_BACK = self._messaging.addTo(BFS_BACK, self.
        _treeParent, self._id, [self._leaderID, self._lCentrality, self._distance-1,
        False])
63
64                        self._treeParent = j
65                        self._treeChildren = set([])
66                        self._distance = dist + 1
67                        self._wait = set([])
68
69                        Ni = set(self._Ni).copy()
70                        Ni =  Ni.difference(set([self._treeParent]))
71                        for k in Ni:
72                            BFS_GO = self._messaging.addTo(BFS_GO, k, self._id, [self.
        _leaderID, self._lCentrality, self._distance])
73                            self._wait.add(k)
74
75                        if self._wait ==set([]):
76                            BFS_BACK = self._messaging.addTo(BFS_BACK, self.
        _treeParent, self._id, [self._leaderID, self._lCentrality, self._distance-1,
        True])
77                            self._finished = True
78                        self._messaging.serialize(BFS_BACK, self._msgType.BFS_BACK)
79                    elif self._leaderID==mid:
80                        BFS_BACK = self._messaging.deserialize(self._msgType.BFS_BACK)
81                        BFS_BACK = self._messaging.addTo(BFS_BACK, j, self._id, [mid,
        cl, dist, False])
82                        self._messaging.serialize(BFS_BACK, self._msgType.BFS_BACK)
83                self._messaging.serialize(BFS_GO, self._msgType.BFS_GO)
```

Stellenbosch University https://scholar.sun.ac.za


```
84
85     def treeChildrenUpdate(self):
86
87         BFS_BACK = self._messaging.deserialize(self._msgType.BFS_BACK)
88
89         if self._id in BFS_BACK:
90             while len(BFS_BACK[self._id])>=1:
91
92                 j = list(BFS_BACK[self._id].keys())[0]
93                 (mid, cl, dist, isChild)= BFS_BACK[self._id][j]
94                 BFS_BACK = self._messaging.deleteFrom(BFS_BACK, self._id, j)
95
96                 if self._leaderID==mid and self._distance==dist and not self.
    _finished:
97                     self._wait = self._wait.difference(set([j]))
98                     if isChild:
99                         self._treeChildren.add(j)
100                    else:
101                        self._treeChildren = self._treeChildren.difference(set([j
    ]))
102                    if self._wait==set():
103                        if self._treeParent==None:
104                            self._finished = True
105                        else:
106                            BFS_BACK = self._messaging.addTo(BFS_BACK, self.
    _treeParent, self._id, [self._leaderID, self._lCentrality, self._distance-1,
    True])
107                            self._finished=True
108            self._messaging.serialize(BFS_BACK, self._msgType.BFS_BACK)
```

Listing G.8: Implementation of the leader election algorithm

G.2.7 Determination of a new region within an exploration region

This code implements how a robot determines an unexplored region within its exploration region when the location of encountered obstacles do not allow it to continue applying its planned zigzag pattern.

```
1     def _finExplorationAreaInRegion(self):
2
3         xmin, xmax = int(self._Xi.x), int(self._Xi.x+self._Xi.w)
4         ymin, ymax = int(self._Xi.y), int(self._Xi.y+self._Xi.h)
5         X = set([(px, py) for px in range(xmin, xmax) for py in range(ymin, ymax)
    ])
6         Q = X.difference(self._Bit[-1])
7         A = np.pi*self._d**2 + 2*self._d*self._gamma*(self._tau-self._t)/2
8
9         xmin, xmax, ymin, ymax = int(self._Vi.x), int(self._Vi.x+self._Vi.w), int(
    self._Vi.y), int(self._Vi.y+self._Vi.h)
```

Stellenbosch University https://scholar.sun.ac.za


```
10        V = set([(px, py) for px in range(xmin, xmax) for py in range(ymin, ymax)
    ])
11        Q = self._rectangleExtraction(self._N, V.difference(self._Cit[-1]))
12
13        if Q.w*Q.h>=A:
14
15            self._Xi = Q
16
17        elif len(self._Yit)>0:
18            self._Xi = self._Yit.pop()
19        else:
20            self._Xi = self._getIndividualExplorationRegion()
21            self._Yit = []
```

Listing G.9: Determination of a new region within an exploration region

G.2.8 A full behaviour of a robot applying the soft obstacle strategy

This code combines some of the main functions during exploration of exploration regions by a single robot.

```
1 if not self.isEnded():
2         if self._state in [RobotStates.S,  RobotStates.W]:
3
4             Ii, Wi = self._newInteraction(meetings)
5             if len(Ii)>=1:
6
7                 self._Ii, self._Wi = Ii.copy(), Wi.copy()
8                 self._state = RobotStates.TW
9                 self._timeSinceWaiting = self._t
10
11            elif len(self._Wi)>=1:
12
13                self._state = RobotStates.W
14            else:
15                self._search()
16
17
18        elif self._state ==RobotStates.TW:
19
20            if (self._t-self._timeSinceWaiting)<=self._tw:
21                self._waiting(meetings)
22            else:
23                self._state = RobotStates.P
24
25        elif self._state == RobotStates.P:
26            if self._planningState ==None:
27                self._pruningObject = PruningNode(self._id, self._Ii)
28                self._pruningObject.oneHop()
29                self._planningState = PlanningStates.B
30                self._stepB = "update"
```

Stellenbosch University https://scholar.sun.ac.za


```
31                     self._GO = False
32
33                 elif self._planningState==PlanningStates.B:
34                     self._treeConstruction()
35
36                 elif self._planningState==PlanningStates.S:
37
38                     self._leaderElection()
39
40                 elif self._planningState==PlanningStates.D:
41                     self._dataFusion()
42
43                 elif self._planningState == PlanningStates.T:
44                     self._taskAssignment()
45
46         ball = Ball(self._xi, self._d)
47         self._Cit.append(self._Cit[-1].union(ball.Ball))
48         self._Bit.append(self._Bit[-1].union(ball.Ball))
49         self._t +=self._deltaT
50         self._DeltaTi +=self._deltaT
```

Listing G.10: A full behaviour of of a robot applying the soft obstacle strategy

G.2.9 Distance bug algorithm

This is the implementation of the distance bug algorithm used to avoid obstacles and soft obstacles when a robot explores.

```
1  from coordinationUtils import Ball
2  import numpy as np
3
4  class Bug:
5
6      def __init__(self, d, gamma):
7          self._d = d
8          self._gamma = gamma
9
10     def distanceBug(self, A, x, a, C, O):
11
12         x0 = x+np.array([0, self._gamma])
13         x1 = x+np.array([self._gamma, 0])
14         x2 = x+np.array([-self._gamma, 0])
15         x3 = x + np.array([0, -self._gamma])
16
17         B = set([tuple(x0), tuple(x1), tuple(x2), tuple(x3)])
18         B = B.difference(O)
19
20         if B==set([]):
21             return x
22
23         b = B.pop()
```

Stellenbosch University https://scholar.sun.ac.za


```
24            dist = np.linalg.norm(np.array(b)-a)
25            ball = Ball(b, self._d)
26            I = len(ball.Ball.difference(C))
27            u = self.utility(dist, I)
28            for bi in B:
29                dist = np.linalg.norm(np.array(bi)-a)
30                ball = Ball(bi, self._d)
31                I = len(ball.Ball.difference(C))
32
33                if u<self.utility(dist, I):
34                    b = np.array(bi).copy()
35                    u = self.utility(dist, I)
36            return np.array(b)
37
38        def utility(self, dist, I):
39
40            return (1 + I)/(1 + dist)
```

Listing G.11: Implementation of the distance bug algorithm

G.3 Classifier fusion

This is the implementation of the proposed classifier fusion method (Yayambo).

```
1  import numpy as np
2  from classifierFusion import sum_rule
3
4  class Yayambo:
5
6      def __init__(self, m, T=50, epsilon = 1.0e-06, epsilon0 = 0.000001):
7
8          self.__T = T
9          self.__epsilon = epsilon
10         self.__epsilon0 = epsilon0
11         self.__m = m
12
13     def support(self, x, y):
14
15         dis = []
16         for xi, yi in zip(x, y):
17             dis.append(np.abs(xi*np.log((self.__epsilon0+xi)/(self.__epsilon0+yi))
    ))
18         return y/(1.+np.array(dis))
19
20     def convergence(self, OD, LD):
21
22         diff = np.linalg.norm(OD-LD, axis=1)
23         diff = np.sum(diff)
24         return diff < self.__m*self.__epsilon
25
26     def update(self, pi, betas):
27
```

Stellenbosch University https://scholar.sun.ac.za


```
28          S = pi*np.sum(betas, axis=0)
29          S /=np.sum(S)
30          return S.tolist()
31
32      def getConsencus(self, D):
33
34          m, l = np.array(D).shape
35          Dcopy = np.copy(D)
36
37          for t in range(self.__T):
38              tempD = []
39
40              for i in range(m):
41                  betas = []
42
43                  for j in range(m):
44
45                      if j!=i:
46                          bji=self.support(Dcopy[i], Dcopy[j])
47                          betas.append(bji)
48
49                  tempD.append(self.update(Dcopy[i], betas))
50
51              if self.convergence(Dcopy, tempD):
52                  distribution, decision = sum_rule(Dcopy)
53                  return distribution, decision
54              Dcopy = np.copy(tempD)
55          distribution, decision = sum_rule(Dcopy)
56
57          return distribution, decision
```

Listing G.12: Implementation of the Yayambo algorithm