

AMERICAN UNIVERSITY OF BEIRUT

ON TAMING LARGE OPTIMIZATION
PROBLEMS: A MACHINE LEARNING
APPROACH FOR AN IMPROVED
PERFORMANCE OF AD HOC TEAMS OF
HETEROGENEOUS AGENTS IN PACKAGE
DELIVERY

by

YARA ANTOINE RIZK

A dissertation

submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
October 2018

AMERICAN UNIVERSITY OF BEIRUT

ON TAMING LARGE OPTIMIZATION
PROBLEMS: A MACHINE LEARNING
APPROACH FOR AN IMPROVED
PERFORMANCE OF AD HOC TEAMS OF
HETEROGENEOUS AGENTS IN PACKAGE
DELIVERY

by
YARA ANTOINE RIZK

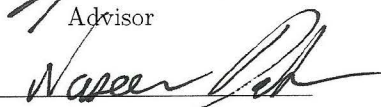
Approved by:

Dr. Karim Kabalan, Professor
Electrical and Computer Engineering, AUB


Committee Chair

Dr. Mariette Awad, Associate Professor
Electrical and Computer Engineering, AUB

Advisor



Dr. Naseem Daher, Assistant Professor
Electrical and Computer Engineering, AUB

Member of Committee

Dr. John Baras, Professor
University of Maryland

Member of Committee

Dr. Jeff Shamma, Professor
King Abdallah University of Science and Technology

Member of Committee

Date of dissertation defense: October 3, 2018

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: RIZK YARA ANTOINE
Last First Middle

Master's Thesis Master's Project Doctoral Dissertation

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after : **One ---- year from the date of submission of my thesis, dissertation, or project.**
Two ---- years from the date of submission of my thesis, dissertation, or project.
Three --X-- years from the date of submission of my thesis, dissertation, or project.

Yara Rizk

10-10-2018

Signature

Date

Acknowledgements

First and foremost, I would like to thank the millions upon millions of biological and artificial neurons that continuously fired throughout my lifetime and especially during my graduate studies, and brought this dissertation to fruition. Without them, I would not have contributed a nanoscopic raindrop in the ocean of artificial intelligence research. Also, I would like to thank my laptops, past and present, who have limped alongside me to the finish line. Finally, I am eternally ungrateful to Thomas Edison, Nicola Tesla, Benjamin Franklin, Alan Turing, and the other founding fathers. Without their contributions, I would have been blissfully ignorant, picking apples, instead of pursuing a PhD.

On a more serious note, I would like to thank my advisor, Prof. Mariette Awad, for her endless support, help, encouragement and belief in my abilities through the ups and downs of my PhD journey. I would also like to thank my committee members, Prof. Karim Kabalan, Prof. Naseem Daher, Prof. John Baras and Prof. Jeff Shamma, for their support and thought-provoking questions. To all the professors at AUB who have given me timely advice, written letters of recommendation for me and taught me during my courses, I would like to thank you. Finally, I would like to thank my parents, sisters, extended family and friends for their support, understanding and “well-wasted” time spent together that made this journey enjoyable.

An Abstract of the Dissertation of

Yara Antoine Rizk for Doctor of Philosophy
Major: Electrical and Computer Engineering

Title: On Taming Large Optimization Problems: A Machine Learning Approach
for an Improved Performance of Ad Hoc Teams of Heterogeneous Agents
in Package Delivery

With the emergence of Internet of Things, cloud computing, and smart cities empowered by artificial intelligence and machine learning, transportation systems have witnessed improved operational performance from safety and sustainability to greener logistics and efficiency. Given that the “last mile” of the delivery process is the most expensive phase, autonomous package delivery systems are gaining traction as they aim for faster and cheaper delivery of goods to city, urban and rural destinations. This interest is further fueled by the emergence of e-commerce, where many applications can benefit from autonomous package delivery solutions. However, the environment stochasticity, variability and task complexity for autonomous operation make it difficult to deploy such systems in real-world applications without the incorporation of advanced machine learning and optimization algorithms. Moving away from designing a “one size fits all” agent to solve the outdoor package delivery problem and considering ad-hoc teams of agents trained within a data-driven framework could provide the answer.

In this work, we argue that heterogeneous multi-agent systems (MAS) can be leveraged to insure some efficient multimodal transport which uses vehicular and non-vehicular agent cooperation for task completion. While the pickup and delivery problem (PDP) is one of the most popular models of package delivery, it does not support MAS. Therefore, we present PDP formulations that allow coalition formation (CF), i.e. a constrained optimization problem is formulated to solve for the delivery schedule while considering teams of agents for task execution. Specifically, 3-index and 2-index mixed integer programming (MIP) approaches

are derived. However, the large number of optimization variables in both formulations causes convergence issues when using branch and bound type optimization solvers, which led to adopting heuristic and data driven approaches. Multiple solvers are presented to find near-optimal schedules including quantum genetic algorithm (QGA), genetic algorithm (GA) and artificial neural networks (ANN). To further improve the performance of the ANN solver, we propose a non-iterative training algorithm for recurrent neural networks (RNN), context-dependent initialization approaches, and a regularization algorithm based on transfer entropy and Kullback-Leibler divergence. Multiple synthetic PDP scenarios with varying problem sizes are generated to evaluate the performance of the proposed solvers and create a labeled training set for ANN.

The algorithmic contributions of this dissertation include: (1) PDP formulations that allow coalition formation (PDP-CF), (2) QGA solver for PDP-CF, (3) GA solver for PDP-CF, (4) ANN solver for PDP-CF, (5) batch and online non-iterative training algorithms for RNN, (6) context dependent initialization algorithms for ANN, and (7) information theoretic regularization algorithms for deep learning.

Contents

| | |
|--|------------|
| Acknowledgements | v |
| Abstract | vi |
| List of Figures | xii |
| List of Tables | xv |
| 1 Introduction | 1 |
| 1.1 Motivation | 1 |
| 1.2 Thesis Question | 3 |
| 1.3 Thesis Approach | 3 |
| 1.4 Thesis Contributions | 4 |
| 1.5 Document Outline | 4 |
| 2 Heterogeneous Multi Agent Systems: A Survey | 6 |
| 2.1 Multi-Agent Systems | 6 |
| 2.1.1 Intelligent Agents | 6 |
| 2.1.2 Multi-Agent Systems | 7 |
| 2.1.3 Tasks | 8 |
| 2.2 Multi-Robot System Workflow | 9 |
| 2.3 Multi-Robot System Applications | 9 |
| 2.3.1 Third Level of Automation | 10 |
| 2.3.2 Second Level of Automation | 11 |
| 2.3.3 First Level of Automation | 12 |
| 2.4 Autonomous Package Delivery | 14 |
| 3 Decision Making Systems: A Survey | 16 |
| 3.1 Introduction | 16 |
| 3.2 Multi-Agent Systems | 19 |
| 3.2.1 Multi-Agent Systems | 19 |
| 3.2.2 Agents | 20 |
| 3.2.3 Interactions | 20 |
| 3.2.4 Decision Making | 21 |

| | | |
|----------|--|-----------|
| 3.3 | Markov Decision Processes | 21 |
| 3.3.1 | Markov Decision Process | 21 |
| 3.3.2 | Partially Observable MDP | 23 |
| 3.3.3 | Some Insights | 24 |
| 3.4 | Game Theory | 24 |
| 3.4.1 | Partially Observable Stochastic Games | 24 |
| 3.4.2 | Bayesian Games | 26 |
| 3.4.3 | Some Insights | 26 |
| 3.5 | Swarm Intelligence | 26 |
| 3.5.1 | Biologically Inspired Algorithms | 26 |
| 3.5.2 | Some Insights | 27 |
| 3.6 | Graph Theory | 27 |
| 3.6.1 | Influence Diagrams | 27 |
| 3.6.2 | Some Insights | 28 |
| 3.7 | Control Theory | 28 |
| 3.7.1 | Distributed Cooperative Control | 29 |
| 3.7.2 | Some Insights | 29 |
| 3.8 | Applications | 30 |
| 3.8.1 | Robotics | 30 |
| 3.8.2 | Repeated Coalition Formation | 30 |
| 3.8.3 | Intelligent Transport Networks | 31 |
| 3.8.4 | Wireless Sensor Networks | 32 |
| 3.8.5 | Intrusion Detection | 32 |
| 3.8.6 | Other Applications | 33 |
| 3.9 | Challenges | 34 |
| 3.9.1 | Scalability | 34 |
| 3.9.2 | Computational Complexity | 34 |
| 3.9.3 | Dynamic Environments | 34 |
| 3.9.4 | System Heterogeneity | 35 |
| 3.9.5 | Big Data | 35 |
| 3.9.6 | Evaluation Standards | 36 |
| 3.9.7 | Other Challenges | 36 |
| 3.10 | Conclusion | 36 |
| 4 | Pickup and Delivery Problem with Coalition Formation (PDP-CF) | |
| | Formulation | 38 |
| 4.1 | Introduction | 38 |
| 4.2 | Literature Review | 39 |
| 4.2.1 | Pickup and Delivery Problem (PDP) Formulations | 39 |
| 4.2.2 | PDP Solvers | 40 |
| 4.3 | Motivating Illustrative Example | 41 |
| 4.4 | PDP Formulation | 43 |
| 4.4.1 | A 3-index PDP-CF Formulation | 44 |

| | | |
|----------|--|-----------|
| 4.4.2 | A 2-index PDP-CF Formulation | 47 |
| 4.4.3 | Cost Function | 49 |
| 4.4.4 | Coalition Overlap | 50 |
| 4.5 | Theoretical Analysis | 51 |
| 4.6 | Conclusion | 52 |
| 5 | PDP Solver: An Evolutionary Approach | 53 |
| 5.1 | Introduction | 53 |
| 5.2 | Literature Review | 54 |
| 5.3 | Methodology | 54 |
| 5.3.1 | Quantum Genetic Algorithm (QGA) Solver | 55 |
| 5.3.2 | Genetic Algorithm (GA) Solver | 58 |
| 5.4 | Empirical Validation | 61 |
| 5.4.1 | Experimental Setup | 61 |
| 5.4.2 | QGA Results | 62 |
| 5.4.3 | GA Results | 62 |
| 5.5 | Conclusion | 63 |
| 6 | PDP Solver: A Deep Learning Approach | 65 |
| 6.1 | Introduction | 65 |
| 6.2 | Literature Review | 66 |
| 6.3 | Methodology | 67 |
| 6.3.1 | Agent Descriptors | 67 |
| 6.3.2 | Package Descriptors | 69 |
| 6.3.3 | Environment Descriptors | 69 |
| 6.3.4 | Artificial Neural Networks (ANN) Formulation | 70 |
| 6.4 | Database Description | 72 |
| 6.4.1 | Descriptor Representation | 72 |
| 6.4.2 | Graphical Representation | 74 |
| 6.5 | Conclusion | 75 |
| 7 | ANN Training Algorithm: A Non-iterative Approach | 76 |
| 7.1 | Introduction | 77 |
| 7.2 | Literature Review | 78 |
| 7.2.1 | Recurrent Neural Networks | 78 |
| 7.2.2 | Non-Iteratively Trained Artificial Neural Networks | 82 |
| 7.3 | Non-Iterative Proximal Randomized Recurrent Neural Networks | 86 |
| 7.3.1 | Randomized Non-iterative RNN Training Algorithm | 87 |
| 7.3.2 | Linear Approximation of Non-iterative RNN Training Algorithm | 87 |
| 7.3.3 | Kaczmarz's Projection Algorithm | 88 |
| 7.4 | Theoretical Analysis | 88 |
| 7.4.1 | Computational Complexity | 89 |

| | | |
|-----------|--|------------|
| 7.4.2 | Model Transformations | 90 |
| 7.5 | Empirical Validation | 91 |
| 7.5.1 | Experimental Setup | 91 |
| 7.5.2 | Non-iteratively-trained RNN Performance Analysis | 93 |
| 7.5.3 | Non-iteratively-trained RNN Statistical Analysis | 107 |
| 7.5.4 | Parameter Sensitivity Analysis | 108 |
| 7.5.5 | Online Learning: RLS versus Kaczmarz's Approximation | 110 |
| 7.6 | Conclusion | 112 |
| 8 | ANN Initialization: A Data Dependent Approach for Regression | 115 |
| 8.1 | Introduction | 115 |
| 8.2 | Literature Review | 116 |
| 8.3 | Proposed Methodology | 118 |
| 8.3.1 | Extreme Learning Machines Overview | 118 |
| 8.3.2 | Context Dependent Extreme Learning Machines (ELM) | 119 |
| 8.4 | Theoretical Computational Complexity | 120 |
| 8.5 | Empirical Validation | 121 |
| 8.5.1 | Experimental Setup | 121 |
| 8.5.2 | Regression Performance Analysis | 122 |
| 8.5.3 | Autonomous Car Steering Wheel Controller | 125 |
| 8.6 | Conclusion | 126 |
| 9 | ANN Initialization: A Least Squares Approach | 127 |
| 9.1 | Introduction | 127 |
| 9.2 | Related Work | 128 |
| 9.3 | Methodology | 129 |
| 9.3.1 | Extreme Learning Machines Overview | 129 |
| 9.3.2 | Backpropagation Overview | 130 |
| 9.3.3 | Least Squares Initialized Backpropagation Training Algorithm | 131 |
| 9.4 | Theoretical Computational Complexity Analysis | 131 |
| 9.5 | Empirical Validation | 133 |
| 9.5.1 | Experimental Setup | 133 |
| 9.5.2 | Performance Evaluation | 133 |
| 9.5.3 | Hyperparameter Sensitivity Analysis | 135 |
| 9.5.4 | Training Error Convergence Analysis | 135 |
| 9.5.5 | Weight Distribution Analysis | 135 |
| 9.6 | Conclusion | 136 |
| 10 | ANN Regularization: An Information Theoretic Approach | 141 |
| 10.1 | Introduction | 141 |
| 10.2 | Literature Review | 142 |
| 10.3 | Methodology | 143 |

| | | |
|-----------|--|------------|
| 10.3.1 | Transfer Entropy Overview | 143 |
| 10.3.2 | Artificial Neural Network Model | 144 |
| 10.3.3 | Two-phase Training Algorithm | 145 |
| 10.3.4 | Information Theoretic Sparsification | 146 |
| 10.4 | Empirical Validation | 147 |
| 10.4.1 | Experimental Setup | 147 |
| 10.4.2 | Performance Analysis | 147 |
| 10.4.3 | Repeatability Analysis | 149 |
| 10.5 | Conclusion | 149 |
| 11 | Conclusion | 151 |
| 11.1 | Summary | 151 |
| 11.2 | Future Research Directions | 152 |
| A | Abbreviations | 154 |
| B | Heterogeneous Multi-Agent System (MAS) Workflow Literature Review | 156 |
| B.1 | Task Decomposition | 156 |
| B.2 | Coalition Formation and Task Allocation | 157 |
| B.2.1 | Coalition Formation | 157 |
| B.2.2 | Task Allocation | 158 |
| B.2.3 | Simultaneous Coalition Formation and Task Allocation | 159 |
| B.3 | MAS Planning and Control | 159 |
| B.4 | Perception | 161 |
| B.5 | Challenges and Insights | 162 |
| B.5.1 | Big Data | 163 |
| B.5.2 | Internet of Things | 163 |
| B.5.3 | Task Complexity | 163 |
| B.5.4 | Autonomous Machine Learning | 164 |
| B.5.5 | Scalability and Heterogeneity Trade off | 164 |
| B.5.6 | Coalition Formation and Task Allocation | 164 |
| B.5.7 | Other Challenges | 164 |
| C | ANN History | 166 |
| C.1 | Concepts from Neuroscience | 166 |
| C.2 | Shallow Beginnings | 167 |
| C.3 | Shallow Networks' Limitations | 168 |
| C.4 | Deep Architectures | 168 |

List of Figures

| | | |
|-----|--|----|
| 1.1 | An autonomous package delivery system in an outdoor environment. | 3 |
| 2.1 | Three-tier heterogeneous Multi-Robot System (MRS) architecture: robot, locally connected MRS, group of MRS connected through the cloud | 7 |
| 2.2 | Proposed workflow | 10 |
| 2.3 | Comparing existing work based on the task complexity and the degree of system automation | 11 |
| 3.1 | Scope of Existing Surveys on MAS Decision Making. | 17 |
| 3.2 | Comparison of Decision Making Model Frameworks. | 19 |
| 4.1 | Illustrative PDP example: (a) Initial map, (b) Solution without transfers or coalitions, (c) Step 1 of solution with transfers and coalitions, (d) Step 2 of solution with transfers and coalitions. . . | 42 |
| 4.2 | PDP's Graphical Representation | 45 |
| 5.1 | QGA Chromosome Encoding based on 3-index Formulation | 56 |
| 5.2 | GA Chromosome Encoding based on 3-index Formulation | 58 |
| 5.3 | GA Chromosome Encoding | 59 |
| 5.4 | GA Initialization | 60 |
| 5.5 | GA Crossover Operation | 61 |
| 5.6 | GA Mutation Operation | 61 |
| 6.1 | The main components of a robotic agent. | 68 |
| 6.2 | The main functions of a package delivery agent. | 69 |
| 6.3 | A feature based ANN formulation. | 70 |
| 6.4 | A graph based ANN formulation. | 71 |
| 6.5 | A GAN formulation. | 71 |
| 6.6 | An image of the graphical representation of a PDP scenario. . . . | 74 |
| 6.7 | An acceptable solution to a PDP scenario. | 75 |
| 7.1 | Different RNN Architectures | 79 |
| 7.2 | Single hidden layer feedforward network trained non-iteratively with randomized input weights | 83 |

| | | |
|------|---|-----|
| 7.3 | Predicted output for AEMO of a fully connected RNN (without DL) trained using ELM-lin | 94 |
| 7.4 | Non-iteratively-trained RNN variants ranking based on Nemenyi post-hoc test | 108 |
| 8.1 | Summary of works in the literature | 118 |
| 8.2 | ELM Architecture | 119 |
| 8.3 | Deep Tesla Data Sample (adapted from [1]) | 125 |
| 9.1 | ELM Architecture | 129 |
| 9.2 | ELM Autoencoder Architecture | 130 |
| 9.3 | Training Error as a Function of Training Epochs for Sports Classification | 136 |
| 9.4 | Weight Distribution of BP-rand and BP-LS after Initialization | 137 |
| 9.5 | Weight Distribution of BP-rand and BP-LS after Training | 137 |
| 10.1 | ANN Architecture | 145 |
| 10.2 | Repeatability Analysis on MNIST for $n = 30$ | 150 |

List of Tables

| | | |
|------|--|-----|
| 2.1 | Existing MRS | 14 |
| 4.1 | Robot Setup | 41 |
| 4.2 | Package Setup | 41 |
| 4.3 | Solution when transfers and coalition formation are not allowed | 43 |
| 4.4 | Solution when transfers and coalitions are allowed | 43 |
| 4.5 | Nomenclature | 44 |
| 4.6 | Graph Complexity | 52 |
| 5.1 | Quantum rotation gate angle adjustment strategy [2] | 56 |
| 5.2 | Instance Description | 62 |
| 5.3 | QGA Performance | 63 |
| 5.4 | GA Performance | 64 |
| 6.1 | Package Descriptor | 70 |
| 7.1 | ANN architecture and training algorithm combinations proposed in this work (denoted by X) | 78 |
| 7.2 | Pseudocode for ELM-rand training algorithm | 87 |
| 7.3 | Pseudocode for ELM-lin training algorithm | 88 |
| 7.4 | Computational complexity of ELM | 89 |
| 7.5 | Number of weights per layer for ANN architectures | 91 |
| 7.6 | Description of the time series benchmarks | 93 |
| 7.7 | Performance on Atmosphere Humidity Data Set | 95 |
| 7.8 | Performance on Atmosphere Temperature Data Set | 96 |
| 7.9 | Performance on Bebida Data Set | 97 |
| 7.10 | Performance on Consumo Data Set | 98 |
| 7.11 | Performance on IPI Data Set | 99 |
| 7.12 | Performance on Inverted Pendulum Data Set | 100 |
| 7.13 | Performance on Quebec Births Data Set | 101 |
| 7.14 | Performance on Santa Fe Data Set | 102 |
| 7.15 | Performance on Deep Tesla Data Set | 103 |
| 7.16 | Performance on Japan Population Data Set | 104 |
| 7.17 | Performance on SP500 Data Set | 105 |

| | | |
|------|--|-----|
| 7.18 | Effect of kernel choice on nRMSE, nMAE and training time . . . | 109 |
| 7.19 | Effect of number of hidden neurons on MSE and training time . . | 111 |
| 7.20 | RLS versus Kaczmarz’s approximation | 113 |
| 7.21 | Statistical comparison of RLS and Kaczmarz’s approximation . . | 114 |
| 8.1 | Nomenclature | 120 |
| 8.2 | Computational Complexity of the considered ELM variants | 121 |
| 8.3 | Database Characteristics | 122 |
| 8.4 | Summary of results | 123 |
| 8.5 | Deep Tesla Results | 126 |
| 9.1 | BP-LS | 132 |
| 9.2 | Computational complexity of a single hidden layer ELM | 132 |
| 9.3 | Database Description | 134 |
| 9.4 | Performance of BP-rand and BP-LS on Regression Datasets . . . | 138 |
| 9.5 | Performance of BP-rand and BP-LS on Classification Datasets . . | 138 |
| 9.6 | Performance of BP-rand and BP-LS on Regression Datasets with Different Activation Functions | 139 |
| 9.7 | Performance of BP-rand and BP-LS on Classification Datasets with Different Activation Functions | 139 |
| 9.8 | Effect of Epochs on Performance | 140 |
| 10.1 | Workflow of the Proposed Algorithm | 146 |
| 10.2 | Pseudo-random Pruning Results | 148 |
| 10.3 | Varying the number of epochs on MNIST | 149 |

Chapter 1

Introduction

1.1 Motivation

With the automation of many everyday tasks resulting from cheaper and better electronics and robotic systems, some tasks such as package delivery still lag behind due to many difficulties. However, the emergence of Internet of Things and smart cities can help make autonomous delivery commonplace, benefiting many industries like retail, healthcare and emergency response.

As e-commerce becomes more popular, evidenced by its \$1,471 billion sales worldwide in 2015 [3], shipping products to consumers is taking up a larger portion of companies' revenues. The "last mile" of the delivery process is the most expensive phase, with Amazon reportedly spending approximately \$11.5 billion in 2015 on shipping costs [4]. Furthermore, the number of packages shipped by Amazon is estimated at about 608 million packages a year and is steadily increasing [5]. With consumers expecting faster and cheaper delivery, companies like Amazon are looking for better package delivery systems.

Automated delivery can also lead to smarter healthcare systems. The first impact involves automating prescription drug delivery. Almost 60% of Americans take at least one prescription drug [6] and about 34% of those are elderly individuals [7] who get more prescriptions than younger individuals [8]. Furthermore, almost 46.2 million people reside in rural areas [9] and 18% of rural area populations are elderly residents [10]. An automated delivery system would allow elderly people to live more independently and improve their quality of life, especially those in rural areas. Furthermore, using drones and all terrain autonomous vehicles would help reduce the need to build new roads and reduce maintenance costs of existing roads due to reduced wear and tear. Since a sharp increase in the 65 and older population is expected in the coming years, especially in rural areas [11], this technology will become even more crucial.

We can envision scenarios in which automated delivery of test samples to laboratories for medical testing can improve the quality of healthcare. This involves

the delivery of tissue or bodily fluids that require special handling conditions to medical laboratories capable of running the required tests. For example, rural area hospitals that would not have sophisticated medical laboratory equipment and account for almost 33% of US hospitals [12], may choose to send blood or other samples to a larger, better equipped hospital to run the required tests, allowing rural area hospitals to improve their diagnosis without spending money on expensive equipment or asking patients to make the trip to bigger hospitals. Furthermore, when encountering a possible virus or rare disease outbreak, the automated package delivery system can efficiently delivery samples to the Center for Disease Control and Prevention (CDC) or other government entities, for further analysis and record keeping, who can take appropriate measures to ensure the safety of the general public. The delivery system can also be used to transport antidotes for rare diseases from the CDC to the hospitals.

Automated package delivery can be applied to emergence response situations to improve response time. Considering wilderness and natural park reservations, transporting anti-venom to people who have been poisoned in the wilderness can be faster than transporting the patients to the nearest hospital. For example, snakebites in rural parts of India and Bangladesh have resulted in thousands of deaths a year [13, 14] which could be prevented with the timely delivery of anti-venom. In addition, park rangers would not be required to leave their base stations to provide supplies to individuals in need of the anti-venom or first aid supplies, reducing time wasted on traveling between various locations in the parks, improving response time and allowing the rangers to aid multiple individuals from their headquarters. This is particularly helpful in undermanned parks or parks with treacherous terrain and in difficult weather conditions. In the event of natural disasters such as earthquakes and hurricanes, infrastructure damage makes many areas inaccessible preventing essential first aid resources from reaching the people that need it most. Therefore, delivering relief packages to these inaccessible areas could be possible with an autonomous delivery system.

A fully automated package delivery system utilizes autonomous ground and aerial vehicles to deliver packages to consumers, as shown in Figure 1.1. This heterogeneous MRS consists of cooperating robots with various capabilities working to accomplish a common goal. The complexity of real world problems, uncertainty, stochasticity and variability of real world environments prevents the design one autonomous robot that can efficiently adapt to all circumstances. This autonomous system could achieve faster package delivery at lower costs. However, many questions still face researchers before heterogeneous MRS can be deployed for autonomous package delivery and in real world applications. How can robots within a MAS make decision that contribute to the completion of the common task, in a decentralized fashion? When should and how can robots form coalitions with other robots to complete specific tasks? How can the repeated Coalition Formation (CF) problem be solved? Can Deep Learning (DL) solve decision making problems more efficiently? Therefore, additional research is necessary in the fields

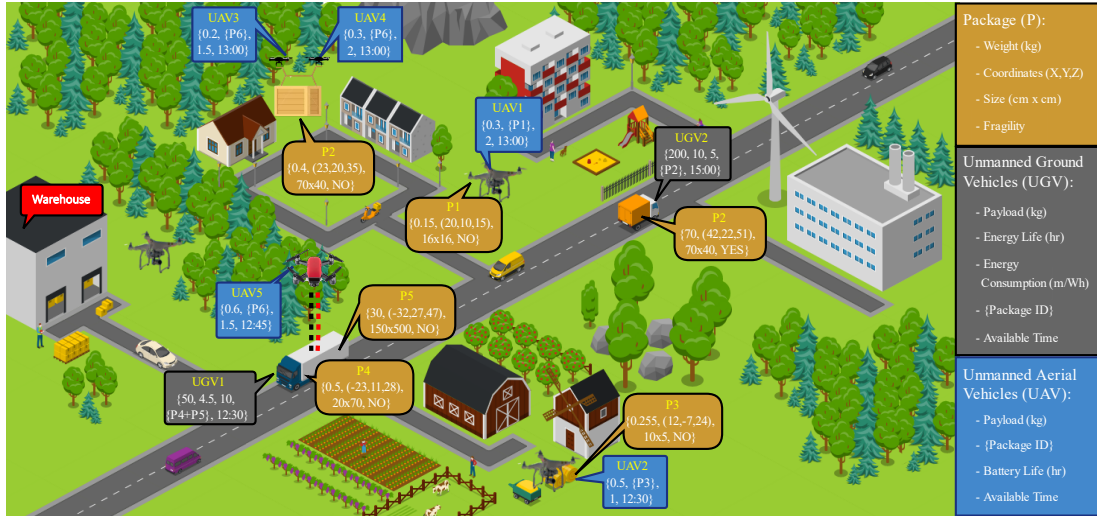


Figure 1.1: An autonomous package delivery system in an outdoor environment.

of artificial intelligence, machine learning, robotics, MAS, distributed optimization and other related fields, to successfully deploy heterogeneous MRS in real world applications.

1.2 Thesis Question

The main research question that will be address in this thesis is the following.

How can agents within a heterogeneous MAS make decisions about coalition formation to cooperatively achieve a complex task?

1.3 Thesis Approach

We build on the work of the scheduling community by formulating a PDP that allows CF based on optimization theory. Specifically, we modify the objective function and constraints of Mixed Integer Programming (MIP) PDP formulations to allow CF. However, this approach resulted in an optimization problem that scaled exponentially with the number of agents. To improve the formulation's scalability, we investigated search-based approaches such as GA and data driven approaches such as ANN. Then, we evaluated the performance of the proposed approaches on multiple PDP scenarios. Furthermore, we sought to improve the performance of ANN solvers by proposing a computationally efficient training algorithm for Recurrent Neural Networks (RNN). Context-dependent initialization algorithms and information theoretic pruning techniques were also investigated and tested on multiple publicly available benchmarks.

1.4 Thesis Contributions

In this work, the main contributions are the following:

- A survey on heterogeneous MAS in various applications, the state of the art in four main building blocks of MAS, and their challenges.
- A survey on decision making models, their applications and challenges.
- A formal definition of PDP that allows CF.
- A 3-index MIP formulation of PDP-CF.
- A 2-index MIP formulation of PDP-CF.
- A GA solver formulation for PDP-CF.
- A QGA solver formulation for PDP-CF.
- A non-iterative training algorithm for RNN.
- An online learning algorithm for non-iteratively trained RNN.
- A context-dependent initialization algorithm for non-iteratively trained ANN.
- A context-dependent initialization algorithm for iteratively trained ANN.
- A regularization approach for DL.

1.5 Document Outline

The thesis is organizing into the following chapters:

- Chapter 2 summarizes relevant work on MRS in autonomous package delivery and other complex tasks after presenting an overview of MAS.
- Chapter 3 summarizes relevant work on decision making models in MAS which are an integral part of any intelligent agent.
- Chapter 4 presents a formulation of PDP-CF based on a 3-index and 2-index PDP formulation. The cost function and constraints are detailed.
- Chapter 5 presents a GA and QGA solver for PDP-CF.
- Chapter 6 presents ANN solver formulations for PDP and discusses the necessary steps before empirical results can be generated.

- Chapter 7 presents a non-iterative training algorithm for RNN. Two variants are formulated for multiple RNN architectures and shown to outperform iteratively trained RNN on time-series regression problems. An online learning formulation is also presented based on Kaczmarz’s approximation of Recursive Least Squares (RLS).
- Chapter 8 introduces a context-dependent initialization algorithm for non-iteratively trained ANN. The algorithm is validated on multiple regression benchmarks and shown to outperform randomly initialized networks.
- Chapter 9 presents an initialization algorithm for iteratively trained ANN. The algorithm, which computes initial weights from a non-iteratively trained autoencoder, outperforms randomly initialized networks on multiple classification and regression benchmarks.
- Chapter 10 presents an information theoretic approach to DL regularization where transfer entropy is adopted as a criterion to neuronal connection pruning. The proposed algorithm is shown to reduce the risk of overfitting by reducing the ANN model complexity.
- Chapter 11 summarizes the contributions of this thesis, comments on its findings and proposes future research directions.

Chapter 2

Heterogeneous Multi Agent Systems: A Survey

In this chapter, we present a brief overview of MAS before discussing related work on MRS applications including autonomous package delivery systems.

2.1 Multi-Agent Systems

2.1.1 Intelligent Agents

An intelligent agent is a physical (robot) or virtual (software program) entity that can autonomously perform actions on an environment while perceiving this environment to accomplish a goal [15]. A rational agent seeks to perform actions that result in the best outcome [15]. A cognitive architecture is the “underlying infrastructure for an intelligent agent” [16]: the agent’s brain. It consists of perception, reasoning, learning, decision making, problem solving, interaction and communication. Its evaluation is based on domain specific performance measures, generality, versatility, rationality, optimality, efficiency, scalability, autonomy and improvability [16].

Many agent categorizations have been proposed in the literature. One categorization distinguishes three types of agents: reactive, deliberative and hybrid agents. Reactive agents simply react to environmental changes. Their workflow contains two primitives: sense (S) and act (A). Deliberative agents initiate actions without any external trigger and rely on planning. This sense-plan-act or sense-model-plan-act paradigm contains three primitives which are performed sequentially: sense (S), plan (P) and act (A). Hybrid agents perform actions based on a planning algorithm or react to current perceptions. The workflows for these three types of agents are represented in Figure 2.1. A finer categorization divides agents into: simple reflex (react to current sensory input), model-based reflex (keep an internal state of the environment), goal-based (perform actions

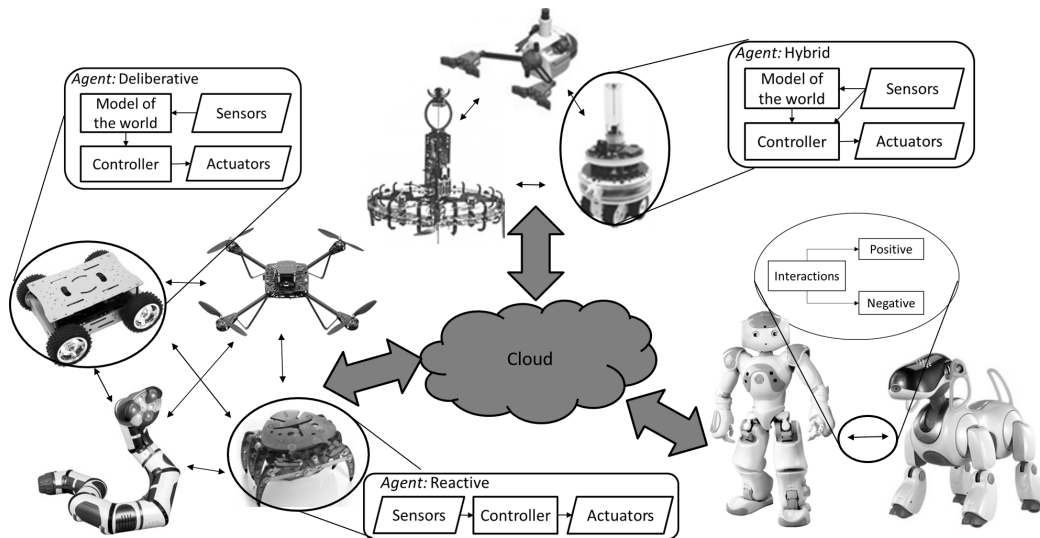


Figure 2.1: Three-tier heterogeneous MRS architecture: robot, locally connected MRS, group of MRS connected through the cloud

to complete a goal), and utility-based (maximize a utility function) agents [15]. These four categories are considered learning agents if they learn an element of the environment or their control algorithms' parameters with the help of a critic.

2.1.2 Multi-Agent Systems

MAS are composed of multiple autonomous, interacting agents that have common or conflicting goals and sensory information [17]. They are characterized by decentralized and incomplete information, asynchronous computations and decentralized control [18]. However, centralized or hybrid systems are also considered MAS. MRS restrict agents to physical robots [19]. MAS has been viewed as an area in distributed artificial intelligence “concerned with coordinated, concurrent action and problem solving” [20], with the second sub-field being distributed problem solving. On the other hand, [21] defined distributed intelligence as a group of entities that perform cognitive functions such as reasoning, solving problems and learning. The term cognitive computing system has also been used to refer to MAS and defined as hardware-software co-optimized architectures composed of diverse intelligent agents that can interact with humans and each other to complete tasks by exploiting each entity’s strengths [22]. Mobile cognition implements distributed cognitive computing architectures on mobile platforms such as robots, cars and smart phones.

MAS evaluation criteria are either domain specific or invariant [23]. Domain specific criteria quantify performance. For search and rescue, performance mea-

sures include the number of rescued persons or extinguished fires [24]. Domain invariant criteria include solution optimality, algorithm time and space complexity, load balancing, fairness, resource utilization and re-allocation quickness, communication overhead, robustness to noise and agent failures, and scalability [23].

MAS have been divided into categories based on multiple criteria. One division, based on agents' diversity and communication capabilities, consists of four classes: homogeneous non-communicative, homogeneous communicative, heterogeneous non-communicative, and heterogeneous communicative [25]. Agent diversity can be either from sensory or actuation capabilities, cognition algorithms or morphology [26]. MAS have also been classified as centralized, hierarchical, decentralized or hybrid architectures [26]. Considering agent interaction complexity lead to three classes: no direct interaction, simple interaction and complex conditional interaction [27].

Focusing on interaction types, MAS can exhibit cooperative, competitive and collaborative interaction based on goals, resources and agent skills [21, 28]. A broader classification is positive versus negative where agents aid or do not interfere with each other versus actively impede other agents. We mainly focus on cooperative interaction where agents are aware of other agents, share the same goals, and their individual actions lead to the accomplishment of the common goal. Examples include search and rescue, exploration, classification of a target, and displacing objects, to name a few [29].

Figure 2.1 represents a heterogeneous MRS architecture with three levels of hierarchy. At the highest level, information about all the robots and the complex task is available in the cloud. Robots can communicate through the cloud and make use of any computational resources and information available in the cloud. The lower level (MRS) contains a subset of robots with an assigned sub-task. Interaction between this subset or coalition is local and can be one of the different types of interactions already discussed. Information is gathered from the various robots' sensors and exchanged among them. Finally, the lowest level is the agent which has access to its sensory input and control of its own actuators. It can communicate with other robots within its coalition and can connect to the cloud. Since the system is heterogeneous, agents are not identical and can have different cognitive architectures (reactive, deliberative or hybrid) or different physical properties (Unmanned Ground Vehicle (UGV), Unmanned Aerial Vehicle (UAV)...). This three tier architecture can lead to automating complex tasks that could not be automated in simpler MRS frameworks.

2.1.3 Tasks

Cooperative MRS are assigned a wide variety of tasks with varying degrees of complexity. Single-robot tasks can be accomplished by one robot [30] such as small scale mapping, pick and place, and navigation problems. Multi-robot tasks require multiple cooperating robots [30]. Multi-robot tasks can be further dis-

tinguished based on the required level of cooperation for successful completion, ranging from loosely to tightly coordinated. Loosely coordinated tasks can be decomposed to sub-tasks that can be independently executed with minimum interaction among robots. Examples include large scale exploration and mapping, hazardous material clean-up, tracking and surveillance. In such scenarios, the environment can be divided into disjoint areas and the robots operate within their specified areas. Tightly coupled tasks are not decomposable and require coordinated execution with significant interaction among robots. Examples include robot soccer, object transport and large scale construction.

2.2 Multi-Robot System Workflow

To systematically design MRS capable of accomplishing complex tasks, we identify four main design blocks: task decomposition, CF, task allocation and task execution or MAS planning and control [31, 32], as shown in Figure 2.2. First, task decomposition is the process of dividing a complex task into a set of sub-tasks that can either be independently or sequentially executed. This division should depend on the set of agents that will complete these sub-tasks and based on the model of the world known to the agents. Given a set of heterogeneous agents, agents with different sensing, actuating and reasoning capabilities, agents should be assigned to sub-tasks while forming coalitions if any one robot can not independently perform a sub-task. The blocks take use information from the overall system or team of MAS level, denoted by the darkest grey blocks. Once the agents have been divided into coalitions and assigned a sub-task, MAS planning and control algorithms can be applied to generate a sequence of actions using knowledge of the state of the world approximated by the perception capabilities of the system. Lighter grey blocks denote that information transfer occurs between agents in a single coalition and the resulting output is at the MAS level. The coalitions will be revised based on the actions performed by the coalitions and environmental constraints obtained from the current model of the world. Finally, perception takes place at the agent level. While some systems may assume shared perceptions, the act of perceiving is performed agents before combining this information. As shown in Figure 2.2, the white blocks represent agent level functions.

2.3 Multi-Robot System Applications

Many MRS have been proposed to solve a broader set of complex tasks. They have all taken a step towards deployment in the real world, but they still make some assumptions or simplifications that limit the system's generalization. We discuss some of these systems next, grouping them based on the complexity of

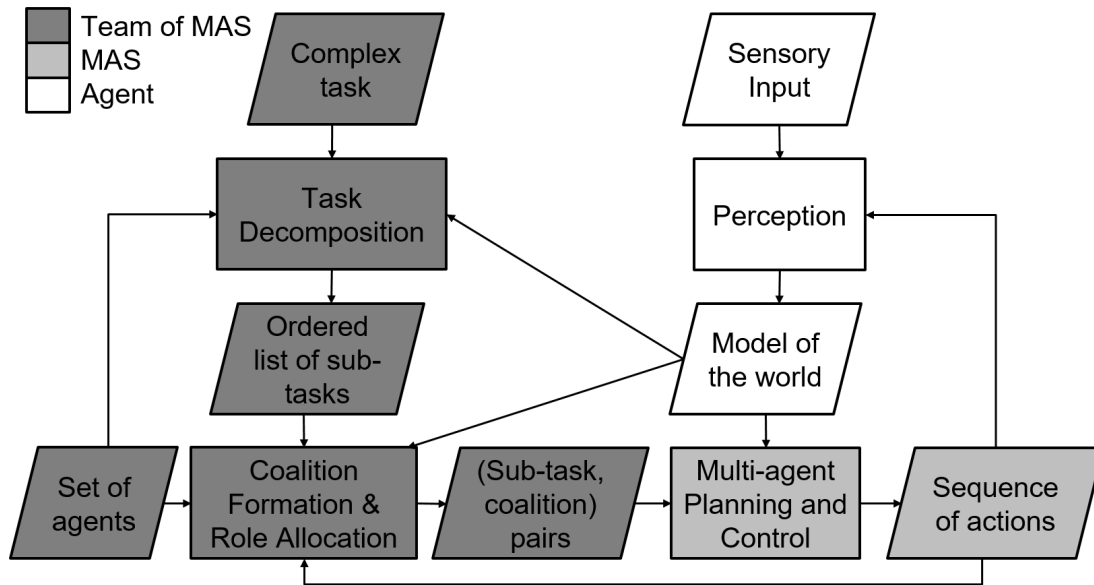


Figure 2.2: Proposed workflow

tasks they execute and their level of automation, from most to least automated. In the first level (least automated), only task execution is automated, while the second level also automates task allocation or CF. The third level does not automate task decomposition. The fourth level automates the entire system. Figure 2.3 portrays the distribution of existing work across levels of automation and task complexity. Table 2.1 summarizes the literature on MRS and their tasks, mentioning only those references with a second or greater automation level. “N/A” stands for not applicable, i.e. this component of the workflow was not considered in the cited work. To the best of our knowledge, no references were found that automated the entire process (fourth level of automation).

2.3.1 Third Level of Automation

A few references achieved the third level of automation where CF, task allocation and task execution were performed by a MRS. In [33], dynamically formed teams of Segways and Pioneer (wheeled) robots hunted for treasure collaboratively in unknown environments using a coordination mechanism based on TraderBots [34]. [35] proposed CF and task allocation algorithms for MRS where UAV were monitoring an environment. [36] simulated multi-robot navigation using a hierarchical clustering algorithm that grouped robots into teams, assigned mobility tasks to these groups and allowed them to navigate without collisions. Finally, [37] attempted to speedup convergence of exploration tasks by developing a supervisory control algorithm. This approach allowed robots to form teams

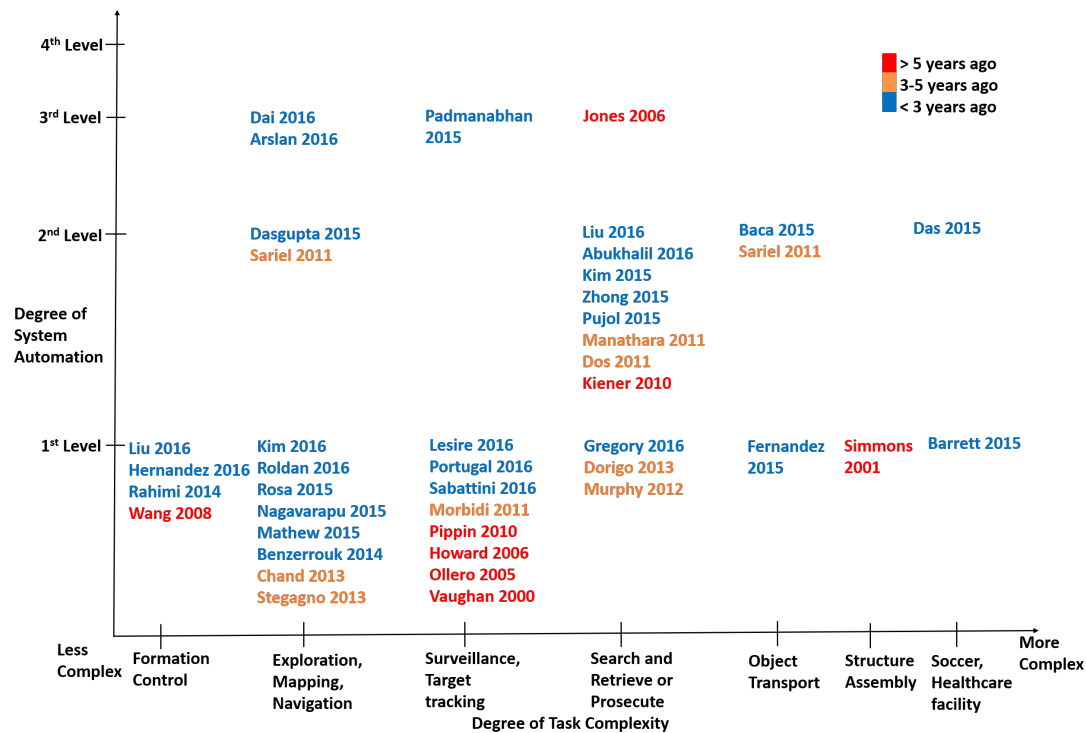


Figure 2.3: Comparing existing work based on the task complexity and the degree of system automation

and assigned a sector zone to explore.

2.3.2 Second Level of Automation

The following references achieved the second level of automation. Task allocation and execution were automated in [38] and validated on MRS performing health care facility tasks. [39] performed multi-robot navigation, cooperation and object construction using a distributed, auctioning-based task (re)-allocation algorithm. [40] presented a cooperative MRS consisting of modular robots, capable of re-configuring their shapes, that was validated on a bar-pushing task. The system, referred to as SMART, relied on inter (between robots) and intra (within the modular robot) communications for cooperative planning to achieve the best modular robot and team configuration for the task at hand. In [32], a humanoid and wheeled robot collaborated to track and kick a ball into a goal. The humanoid kicked the ball after enhancing its tracking accuracy with the help of the wheeled robot. Task decomposition and CF were performed by a human designer. Task allocation was performed using a utility function that assigned a task to the most qualified agent. Then, task execution was performed by the robots' controllers.

Many MRS have been validated on search and rescue or prosecute tasks where task allocation and execution were automated. [41] presented a robot utility based task allocation algorithm that allocated tasks to a swarm of heterogeneous UGV. [42] proposed a binary max sum task allocation algorithm validated in Robocup Rescue, while [43] validated the task allocation algorithm on search and prosecute heterogeneous UAV systems. Other work automated CF and task execution. [44] automated CF in Robocup rescue using swarm intelligence. [45] applied particle swarm optimization based CF for a MRS of UAV performing search and prosecute tasks, while [46, 47] developed a multistage CF algorithm. Finally, [48] developed a framework for land mine detection (COMRADES) that included two types of UGV. This framework performed multi-robot task allocation based on spatial queuing, information gathering and sensor scheduling based on the prediction market algorithm and cooperative exploration based on Voronoi partition coverage.

2.3.3 First Level of Automation

Finally, we present references that only automated the task execution phase in MRS, grouping and ordering them based on task complexity. [49] introduced PLASTIC-Policy to discover the best policy for cooperation among robots in ad hoc teams allowing robots to adapt on the fly and validated on RoboCup soccer. This problem was modeled as a Markov Decision Process (MDP) and solved using a fitted Q-iteration algorithm. In [50], three distinct robots, an overhead crane, a mobile manipulator, and a roving eye were used to precisely place a long heavy beam. Combining ground and aerial robots improved the overall system performance. [51] developed a distributed round robin Q-learning algorithm to transport a hose using a group of UGV.

Swarmanoids is a MRS composed of three types of robot swarms with complementary capabilities: eye-bots (quad-rotors with cameras), hand-bots (grippers), foot-bots (wheeled robots: e-pucks) [52]. The system performed object search and retrieval. A complex task was decomposed into sub-tasks by the human designer. Each sub-task can be performed by one type of robot in the system, i.e. the robots in the system were designed in a way to generate a one to one mapping between sub-tasks and robots, eliminating the need for task allocation. Therefore, this system was able to accomplish a more complex task even though it's workflow is less automated due to the system design which included robots with complementary skills and individual robots that can form predefined coalitions.

[53] performed underwater search and rescue in natural disaster scenarios using Unmanned Underwater Vehicle (UUV) and UAV. Autonomous robots performed a rough preliminary search to identify areas of interest requiring a more thorough search which was later investigated by teleoperated vehicles. [54] developed a navigation and Simultaneous Localization and Mapping (SLAM) algorithm for information gathering in disaster relief. OmniMapper SLAM is based on

graph theory and square-root smoothing and mapping. The NavigationManager integrates the search-based planning and ARA*.

Tracking and surveillance applications adopted MRS to improve coverage by leveraging the heterogeneity of robots and the diverse media (land, air, water). [55] developed a distributed algorithm, HiDDeN, that supervised the completion of UAV and UUV missions in military surveillance and coast securing application. [56] presented a distributed framework to control an air-ground MRS where a UGV cooperated with UAV to complete tasks relayed by a ground station. The system relies heavily on communication between the UGV and UAV. The overall system follows a behavior based autonomy framework where sub-tasks are represented as functional components in a state machine. Field experiments on target detection and surveillance validated the proposed framework on a 3-robot system. [57] focused on UAV, combining helicopters and airships with diverse sensing capabilities, that collaborated to track a target, and inspect and monitor an area [57]. Ground systems included [58] who proposed a behavioral graph based collision avoidance and set point tracking navigation algorithm for MRS. MRS patrolling that adopted a decision making algorithm based on Bayesian decision rules was validated on indoor patrolling with six Pioneer robots [59]. [60] deployed a large number of robots for intruder detection and tracking by first performing exploration and mapping of an unknown indoor environment using a few expensive robots and a large number of cheap sensory robots. [61] developed a framework that allowed UAV to improve their self-localization and localization of objects of interest on the group by cooperating with a GPS-enabled UGV. [62] performed gait monitoring using UAV-UGV MAS.

Navigation, exploration and mapping tasks have also leveraged UAV-UGV cooperation. For example, a large number UGV performing motion tasks were aided by a UAV which communicated location related information [63]. Similarly, [64] used a single UAV to improve UGV SLAM. [65] adopted a MRS as a sensor monitoring solution in greenhouses, using UGV and UAV to record temperature, humidity, specific gas levels and others. While UAV were teleoperated, UGV navigated independently.

[66] developed a decentralized Lyapunov synthesis navigation algorithm with collision avoidance for MRS. [67] proposed a decentralized multi-robot depth first search algorithm that allowed UGV to efficiently explore unknown environments with minimal information exchange and collision avoidance. [68] developed a cooperative navigation algorithm based on graph and networking theory to ensure complete coverage of the environment explored by mobile robots gathering information. [69] developed a hierarchical algorithm that allowed robots with limited capabilities to perform mapping and exploration by receiving tasks from more powerful robots.

Formation control algorithms for MRS have been developed to control UGV and UAV using Lyapunov framework [70], while [71] focused on UGV only but presented a distributed Lyapunov controller. [72] controlled wheeled and legged

Table 2.1: Existing MRS

| Reference, Year | Agents | Task | Task decomposition | CF | Task allocation | Decision making model | Validation |
|-----------------|-----------------------|--|--------------------|-------------------------------|-------------------------------|-----------------------------------|--------------------------|
| [37], 2016 | UGV | Exploration | N/A | Supervisory control algorithm | | | Simulations |
| [36], 2016 | UGV | Navigation | N/A | Hierarchical Clustering | Hierarchical Clustering | Hierarchical Clustering | Simulations |
| [48], 2015 | UGV | Land mine detection | N/A | N/A | Prediction market | Voronoi partition coverage | Simulations, Experiments |
| [35], 2015 | UAV | Surveillance | N/A | Dynamic ANT | Multi-Robot Task allocation | Dynamics controllers | Simulations |
| [33], 2006 | UGV | Treasure hunt | Human Expert | Auctioning | TraderBots | Dynamics controllers | Experiments |
| [46], 2016 | UAV | Search and prosecute | N/A | Multistage sub-optimal CF | N/A | Distributed finite state machines | Simulations |
| [41], 2016 | UGV | Search and rescue | Human Expert | N/A | Robot Utility | Swarm Intelligence | Simulations, Experiments |
| [43], 2015 | UAV | Search and prosecute | N/A | N/A | Welfare based task allocation | Dynamics controllers | Simulations |
| [42], 2015 | UGV | Search and rescue | N/A | Human Expert | Binary Sum Max | Dynamics controllers | Simulations |
| [47], 2015 | UAV | Search and prosecute | N/A | MSOCFA | N/A | Dynamics controllers | Simulations |
| [45], 2011 | UAV | Search and prosecute | N/A | Particle swarm optimization | N/A | Dynamics controller | Simulations |
| [44], 2011 | UGV | RoboCup Rescue | N/A | Swarm intelligence | N/A | Dynamics controllers | Simulations |
| [32], 2010 | Wheeled UGV, humanoid | Kick ball into goal | Human Expert | Human Expert | Utility function | Dynamics controllers | Simulations, Experiments |
| [40], 2015 | Modular robots | Bar pushing | N/A | SMART | N/A | SMART | Experiments |
| [39], 2011 | Khepra II | Object transport, Cooperative navigation | N/A | N/A | Auctioning | Dynamics controllers | Simulations, Experiments |
| [38], 2015 | UGV | Health care facility | N/A | N/A | Auctioning | Dynamics controllers | Simulations |

robots navigating and avoiding obstacles in a given formation. Finally, [73] developed a planar based algorithm for cooperative heterogeneous UAV.

2.4 Autonomous Package Delivery

Automated package delivery systems have been developed and tested for indoor environments, such as in hospitals [74–76], and office building [77] which generally do not require cooperating heterogeneous MRS as a single robot is able to independently deliver packages [74]. Indoor robot package delivery problems are generally modeled as warehouse problems [78]. However, outdoor environments present additional challenges such as weather conditions and large distances that cannot be handled by systems designed for indoor delivery. Other systems that

require similar or overlapping skill sets as automated package delivery have been developed including service robots [79–82], fruit picking robots [83–85] and restaurant waiter robots [86,87]. While they cannot be extended to outdoor delivery without additional considerations, they provide a good starting point for research.

The difficulty of package delivery is further augmented when considering outdoor environments that are less predictable and less constrained. In robotics, the health of the system, which degrades with time from usage, further complicates the problem [88]. Some work in the literature has investigated outdoor automated package delivery such as [89] who proposed a CF algorithm based on real world application assumptions and considered the package delivery domain but did not report any experimental results. [90] studied task allocation for autonomous package delivery in emergency response scenarios, specifically a team of unmanned aerial vehicles delivering relief packages to stranded individuals in the aftermath of an earthquake. Preliminary experiments showed promising results but still needs to be incorporated in end-to-end implementations of heterogeneous MRS. [91] proposed a path planning algorithm for cooperating robots in delivery systems but assumed a pre-formed team of one ground and one aerial vehicle. Furthermore, the algorithm did not make use of Internet of Things to obtain traffic and weather information to improve path planning.

Finally, a survey on cloud robotics has assessed the state of the art of technologies enabling automated package delivery and other applications [92]. They defined cloud robotics as robotic systems that utilize information from the cloud or other non-robotic networks to aid them. They surveyed various areas of cloud robotics including cloud aided learning, crowd sourcing, and accessing big data. They also identified many remaining challenges developing frameworks to move robotics algorithms to the cloud which they termed as “Robotics and Automation as a Service”, developing algorithms to clean and make sense of big data, and developing algorithms robust to varying network latency, among other challenges.

While the state of the art has made significant strides toward deploying autonomous end-to-end package delivery systems, many assumptions and simplifications are made in the presented work, from statically forming cooperating teams to simplifying the environment. A unified framework that allows cooperating heterogeneous robots to utilize smart city, Internet of Things and cloud infrastructure to efficiently delivery packages has not been presented yet.

Chapter 3

Decision Making Systems: A Survey

In this chapter¹, we present a brief overview of decision making models in MAS since it is an integral part of intelligent agents and MAS that will allow such systems to accomplish increasingly complex tasks. Specifically, in this survey, we investigate state of the art work within the past five years on cooperative MAS decision making models, including Markov decision processes, game theory, swarm intelligence and graph theoretic models. We survey algorithms that result in optimal and sub-optimal policies such as reinforcement learning, dynamic programming, evolutionary computing and neural networks. We also discuss the application of these models to robotics, wireless sensor networks, cognitive radio networks, intelligent transport systems and smart electric grids. In addition, we define key terms in the area and discuss remaining challenges that include incorporating big data advancements to decision making, developing autonomous, scalable and computationally efficient algorithms, tackling more complex tasks and developing standardized evaluation metrics. While recent surveys have been published on this topic, we present a broader discussion of related models and applications.

3.1 Introduction

The number of devices connected to the Internet has been increasing over the past few years and projected to exceed 20 billion devices by 2020 [93,94]. These devices can communicate with each other to form MAS that can cooperate to overcome individual limitations and achieve complex tasks. This has led to the emergence of cognitive computing systems which were defined by IBM as systems

¹A version of this chapter has appeared in **Rizk, Y.**, Awad, M., and Tunstel, E., “Decision Making in Multi Agent Systems: A Survey,” *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 3, pp. 514-529, 2018.

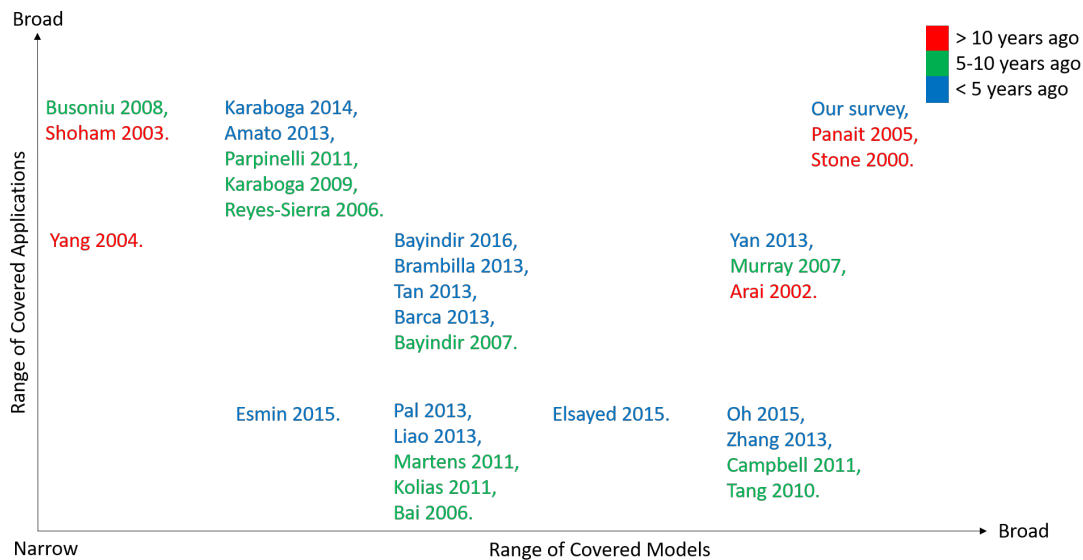


Figure 3.1: Scope of Existing Surveys on MAS Decision Making.

that can interact with each other and humans to exploit their strengths when accomplishing a task [22]. At the heart of cognitive systems is the decision making, or planning and control module which allows agents to generate a sequence of actions that will lead to the accomplishment of their goals.

Multiple surveys on MAS decision making have been published. While some briefly discussed a wide range of models and applications, others focused on a specific model or application. We present a more up-to-date discussion on cooperative MAS decision making, covering a wide range of applications and models. Figure 3.1 depicts the scope of existing surveys in terms of their relative breadth of covered models and applications, while highlighting the targeted scope of our survey. Color coding distinguishes references' publication date: surveys published more than 10 years ago are in red, 5 to 10 years in green and less than 5 years in blue.

The surveys closest to our work, in the upper right corner of Figure 3.1, covered a wide range of MAS decision making methods including game theory, RL, swarm intelligence, and evolutionary computing, and discussed multiple applications including robot soccer, prey-predator pursuit, air traffic control and others [25, 95]. However, these surveys have become outdated and do not cover some key advancements in the field such as the contributions of deep learning. Some surveys covered a wide range of models but focused on multiple problems in robotics. Seven main research areas on MRS were discussed in [19], including robot architectures, mapping and exploration, motion coordination and object transport and manipulation. Multiple unanswered research questions were iden-

tified in [19] including complex task automation using MRS. Cooperative control of multi-vehicle systems and their applications in the military, transportation systems and mobile sensor networks were surveyed in [29] who concluded that additional work in system integration, distributed embedded system verification and decision making at higher level abstractions was necessary before successful deployment of MRS. Yan et al. surveyed multiple aspects of robot coordination including decision making, planning and communication, and observed that more powerful coordination schemes are necessary to automate complex tasks [96].

Surveys discussing one model and its applications included recent work on decentralized Partially Observable Markov Decision Process (POMDP) [97], swarm intelligence in robotics [98–102], and multi-agent RL [103, 104]. Multi-objective particle swarm optimization (PSO) variants [105], algorithms based on bees [106], metaheuristic algorithms [107], and artificial bee colony variants and applications [108] have also been surveyed.

The following surveys discussed multiple models for one application such as formation control and coordination [109, 110], task allocation [111, 112], intrusion detection [113–115] and smart electric grids [116]. Finally, certain surveys focused on a single model applied to one application such as multi-agent RL for robotics [117], swarm intelligence for robot path planning [118], PSO for clustering [119] and swarm intelligence for data mining [120].

In this paper, we survey existing cooperative MAS decision making models including MDP and its variants, game theory, and swarm intelligence. Figure 3.2 depicts a fuzzy comparison between the discussed models based on three criteria: heterogeneity, scalability and communication bandwidth. While other models exist, such as belief-desire-intention models based on the human’s practical reasoning theory [121] and independent choice logic which combined probabilistic information with logic programming to represent knowledge [122], their MAS extensions [123, 124] have not been widely adopted in recent work. Multiple methods that find optimal or sub-optimal action sequences for the various decision making models are surveyed. These include Reinforcement Learning (RL), dynamic programming (DP), RNN and evolutionary computing, to name a few. We present decision making applications in robotics, wireless sensor networks (WSN), traffic signal control and others. Finally, we discuss some of the remaining challenges in cooperative MAS decision making such as leveraging big data advancements, creating more scalable, distributed and computationally efficient algorithms that can tackle more complex tasks, and developing evaluation standards.

In what follows, we first define key terms in the field of MAS in section 3.2. Then, we introduce various decision making models in sections 3.3 through 3.6 and their applications in section 3.8. Remaining challenges and insights on future research directions are discussed in section 3.9 before concluding in section 3.10.

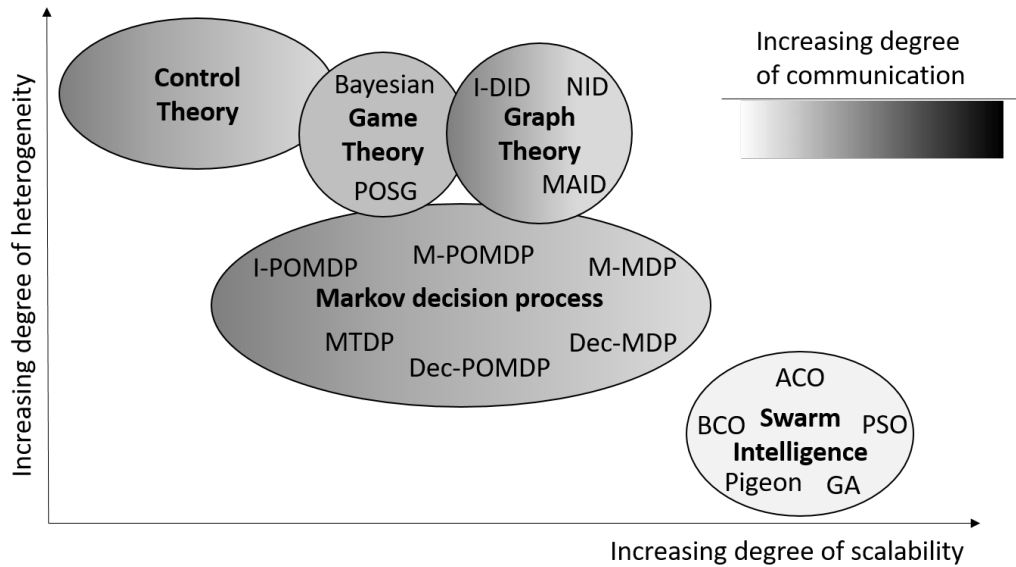


Figure 3.2: Comparison of Decision Making Model Frameworks.

3.2 Multi-Agent Systems

In this section, we define and categorize MAS before discussing its constituting blocks: agents and local interactions. Then, we focus on key terminology for decision making problems, which is an element of intelligent agents.

3.2.1 Multi-Agent Systems

MAS are composed of multiple autonomous, interacting agents that have common or conflicting goals and sensory information [17]. MAS are generally decentralized, asynchronous systems but can sometimes be centralized or hybrid. Their evaluation criteria include domain specific performance metrics and domain invariant criteria such as time and space complexity, load balancing, fairness, resource utilization, communication overhead, robustness and scalability [23]. MAS have been categorized based on multiple criteria such as diversity of agents, communication capabilities and interaction types. Agent heterogeneity stems from diverse sensing and actuating capabilities, computing resources, cognitive algorithms and morphology [26]. Considering agent interaction complexity leads to three classes of MAS: no direct interaction, simple interaction and complex conditional interaction [27].

3.2.2 Agents

An intelligent agent is an autonomous entity capable of performing actions on its environment and perceiving its environment, aiming to accomplish a goal [15]. It can be a physical entity such as robots with sensors and actuators or a virtual entity such as software agents. An intelligent agent exhibits the fundamental properties of perception, reasoning, learning, decision making, problem solving, interaction and communication [16]. It is evaluated based on its solution optimality, generality, robustness, efficiency, autonomy and ability to learn and improve [16]. Agents are categorized based on many different criteria. One categorization depends on the decision making algorithm's instigator and results in three types of agents: reactive, deliberative and hybrid. Reactive agents react to environmental changes. Deliberative agents initiate actions without external triggers. Hybrid agents can react to the environment or initiate actions based on their planning algorithm. Another categorization, proposed in [15], is based on the agent's underlying architecture and contains four classes: simple reflex agents, model-based reflex agents, goal-based agents, and utility-based agents. Simple reflex agents react to current sensory input only while model-based reflex agents keep an internal state of the environment. Goal-based agents perform actions that lead to accomplishing their goals and utility-based agents maximize their utility.

3.2.3 Interactions

In addition to the complexity of interactions, MAS can exhibit different types of interactions based on agent goals, resources and skills [21, 28]. Broadly speaking, interactions can be positive or negative. In the former, agents aid each other in accomplishing their goals, while in the latter, agents actively impede other agents' progress. Positive interaction can be further divided to collective, cooperative, collaborative and coordinative. In collective interaction, agents are unaware of other agents' existence but share a common goal and each agent contributes to its completion, as in robot formation control and foraging. Cooperative interaction is similar to collective interaction except that agents are aware of other agents' existence. Examples include search and rescue, exploration and object displacement. In collaborative interaction, agents do not have common goals but help each other accomplish their individual goals. Finally, in coordinative interaction, agents within an environment work together to minimize interference and complete their individual goals; MRS path planning is one example. Negative interaction can be either conflicting where agents do not have enough resources to complete their goals and fight for external resources or competitive where agents have conflicting goals. In this work, we focus on cooperative MAS since it is an integral part of many smart city systems but still has many open research questions before effective deployment in real-world scenarios is possible.

It is considered by some one of the more challenging interactions due to the need for high correlation and synchronization between agents and time sensitivity of agents' actions, especially in robotics. However, some of the models discussed in this survey can be applied to MAS with positive interactions such as swarm intelligence in collective MAS, game theory in collaborative MAS and graph theory in coordinative MAS.

3.2.4 Decision Making

Decision making, or planning and control, enables an agent to accomplish its goals by determining what action to perform. The decision making problem can either be episodic or sequential [15]. The output of the former is a single action while the latter produces a sequence of actions or policy. The decision making algorithm is evaluated based on policy optimality, search completeness, time complexity and space complexity. A policy is optimal if it has the highest utility. A search algorithm is complete if it guarantees to return an optimal policy in finite time, when it exists. Time complexity quantifies the amount of time needed to search for a solution while space complexity quantifies the amount of computational memory needed. In this work, we focus on sequential decision problems which can be of two types: finite or infinite horizon. Finite horizon implies that decisions need to be made for a finite number of time steps while infinite horizon problems last forever. When discussing decision making in the context of MAS, learning can be either centralized or decentralized. [95] used the terms team learning and concurrent learning. In the former, one learner learns policies for all agents in the system while in the latter, each agent learns its own policies in parallel to other agents. Credit assignment, how to distribute rewards among cooperating agents, is one problem that arises and should be appropriately handled to achieve optimal performance. Communication, whether direct or indirect, is another issue in cooperative decision making that should be considered.

3.3 Markov Decision Processes

In this section, we present the MDP formulation, its extension to MAS and partially observable environments, and conclude with some insights on this method.

3.3.1 Markov Decision Process

An MDP, a discrete time stochastic control process, is characterized by fully observable states and outcomes that are influenced by decision makers. It satisfies the Markov property which states that decisions made at the current time step rely on a finite number of previous time steps. It can also be viewed as a fully observable stochastic game with a single player. In some texts, it is referred to

as a dynamic program, stochastic dynamic program, sequential decision process, and stochastic control problem. An MDP is defined by the tuple (S, A, P, R, γ) . S represents the set of states, s , of the environment. A represents the set of actions, a , an agent can perform. In some states, certain actions are not permissible, i.e. only a subset of the actions can be performed, denoted by A_s . P represents the transition probability. $P_a(s_i, s_j)$ denotes the probability that the environment will transition to state s_j from state s_i when an agent performs action a . R represents the reward. $R_a(s_i, s_j)$ denotes the received reward when performing action a and the environment goes from state s_i to state s_j . γ represents a discount factor, $\gamma \in [0, 1)$, that gives more weight to present reward than future reward. MDP was found to be P-complete [97]. Constrained MDP impose additional constraints on MDP, resulting in more than one cost for every action and the final policy depends on the initial state of the process [125]. Time-dependent MDP [126] extends MDP to continuous time state spaces where value iteration is performed on a piece-wise linear value function.

A solution to an MDP is a policy that should be performed by an agent to maximize its total reward, measured using a value function V . A policy function maps states to actions: $\pi : s \rightarrow a$ or $a = \pi(s)$. Action selection methods determine what action to select next, based on the estimated value functions of the action set, while considering the exploration-exploitation trade-off. A greedy method picks the action with the highest value, ϵ -greedy selects the best action with a probability of $1-\epsilon$. Boltzmann exploration assigns probabilities of selecting actions using an exponential function of the value function.

Many algorithms have been proposed to find optimal and sub-optimal policies for MDP. DP [127], temporal difference learning [128–130], policy search [131], and linear programming [132, 133] require models of the state transition and reward functions. If these models are unknown or too complex, approximate methods are adopted and include model-free RL approaches like Q-learning [134] and SARSA [135], evolutionary computing [136, 137], RNN [138–143], and deep RL [144, 145]. Distributed optimization methods have been adopted to solve MDP problems. First, the alternating direction method of multipliers decomposes the MDP into subproblems. Then, a distributed Newton method [146] or linear programming algorithm [147] find the optimal policy.

MDP extensions have been proposed for MAS. Multi-agent MDP (M-MDP) extends MDP to MAS by assuming a joint action space with a team reward model and fully observable environment. A central learner learns a vector of actions that should be performed by the agents and the reward is common to all agents [17]. The worst-case complexity of finite horizon M-MDP is P-complete [148] which is solvable in polynomial time by a Turing machine, an abstract model of computing devices. As the number of agents increases, the joint state and action spaces' dimensionalities increase exponentially. To ease the computational burden, independence is assumed to make objective functions factorable. Solving the problem iteratively also reduces the computational complexity. Distributed

implementations of the central learner have been developed for factorable objective functions [149]. On the other hand, decentralized MDP (dec-MDP) assumes an independent action space with local reward and jointly fully observable environments [17]. In other words, individual agents view a partially observable environment but the aggregate observations of all agents in the MAS make the environment fully observable. Finite horizon dec-MDP was proven to be worst case NEXP-complete (solvable in exponential time using a non-deterministic Turing machine), when three or more agents are considered [150]. Since actions and rewards are local, this approach falls under the concurrent learning class of MAS learning. Assuming agent observations and transitions are independent, the model is known as TI dec-MDP and its complexity is NP-complete, meaning a solution can be found in polynomial time by a non-deterministic Turing machine. This model can be further simplified by assuming independent rewards to obtain a P-complete complexity in the worst case.

3.3.2 Partially Observable MDP

POMDP is a generalization of MDP to partially observable environments and is defined by $(S, A, P, \Omega, O, R, \gamma)$ where Ω represents the set of observations, O is the observation function and the remaining terms are as defined for MDP. POMDP was found to be PSPACE-complete [97]. Many algorithms have been proposed in the literature to provide exact and approximate solutions for the POMDP and its variants, including value iteration [151, 152], expectation maximization [153, 154], nonlinear optimization [155], quadratically constrained linear programming [156], Monte Carlo methods [157, 158], and DP [159, 160] when state and transition models are known. When they are not known, heuristic search algorithms [161], genetic algorithms [162], RNN [163–166] and model-free RL methods were applied. Authors in [167] learned the number of states to represent in a non-parametric scheme and used RL to find policies for POMDP. Unsupervised learning was adopted to learn an observation space transformation to a latent representation space where policies are learned, in [168]. Forward simulation was used to estimate policy utilities [169].

Decentralized POMDP (Dec-POMDP) generalizes POMDP to MAS where rewards are common and based on joint actions but observations are individualistic [97]. The goal is to maximize the reward of the entire system as agents collaborate to achieve a common task. Communication among agents can be explicit (Dec-POMDP-COM) or implicit (Dec-POMDP). This model is NEXP-complete [150]. Approximate solutions have been proposed based on bounded policy iteration [170], Q-value function methods [171], multi-agent A^* [161], genetic algorithms [162], DP [172–174] and a Bayesian learning, stick-breaking policy algorithm [175]. A set of approximate inferences and heuristics including bootstrapping were used to find approximate solutions to dec-POMDP in [176].

The multi-agent team decision problem (MTDP) [177], equivalent to dec-

POMDP when agents have perfect recall [178], extends economic team theory to robotics. It includes models for implicit and explicit communication and is proven to be NEXP-complete. Multi-agent POMDP (M-POMDP) extends M-MDP to partially observable environments, and is PSAPCE-complete which means the algorithm’s memory requirements are polynomial function of the input size. Like M-MDP, it is a team learning approach that has a central learner, and employed Bayesian RL framework to learn policies [179].

Networked distributed POMDP (ND-POMDP) assumes local interaction among agents to reduce the computational cost of finding policies [180]. ND-POMDP is a factored dec-POMDP model where observations and transitions are independent and rewards are divided among neighboring agents. Its worst case computational complexity is NEXP-complete. Algorithms used to find policies for this model include multi-agent RL [181], DP [182], and distributed constrained optimization [180]. Interactive POMDP (I-POMDP), a concurrent learning approach, generalizes POMDP to MAS by modeling other agents in the system while maintaining a belief of the system state [183]. Finitely nested I-POMDP is PSPACE-complete [178] and approximate solutions have been proposed based on particle filters [184], value iteration [185] and Monte Carlo sampling methods [186].

3.3.3 Some Insights

MDP and its variants have been widely adopted in many complex MAS decision making problems, despite the very restrictive Markovian assumption. Even though these models do not scale well, they are able to handle agent heterogeneity. Recently, deep learning approaches have been adopted to solve various MDP models and provided a roadmap to solve non-Markovian models as well. In addition, deep learning has allowed the extension of MDPs from the discrete space to the continuous space, which is more suitable for robotic MAS.

3.4 Game Theory

Game theory develops models of interaction between rational decision makers under different circumstances [187]. It has been applied in many fields from economics and psychology to artificial intelligence. In this section, we focus on two types of games that have been commonly applied to cooperative MAS in artificial intelligence: stochastic games and Bayesian games.

3.4.1 Partially Observable Stochastic Games

Stochastic or Markov games [188] are sequential probabilistic games. They can also be viewed as a generalization of repeated games where a game from a collection of normal form games can be played at a given step [17]. Payoffs depend on

both actions and the state of the game at the current stage. Players' actions and the game's current state cause the game to transition to other states. Stochastic games are represented using the tuple (Q, N, A, P, r) where Q denotes the set of states that can be played, N denotes the set of players or agents participating in the game, $A = A_1 \times \dots \times A_N$ denotes the actions of the players, P denotes the transition probability function and r denotes the reward or payoff. They belong to the complexity class $NP \cap co - NP$ [189] and can be solved using DP [188], Q-learning [190] and linear programming under certain conditions [17].

To solve a game, a strategy profile or solution concept must be obtained; it is a strategy for each player. A strategy, equivalent to a policy in MDP [15], is a rule used by agents to select an action. An equilibrium strategy is defined as the best response of an agent to another agent's strategy, i.e. the agent cannot improve its expected utility by changing its strategy. It does not always exist in stochastic games but may exist under restricted conditions. For example, stochastic games with a finite number of players, actions and states always have a Nash equilibrium, defined as the strategy profile which maximizes each player's utility knowing the strategy of others in the game [191]. Evolutionary stable strategy is a refinement of the Nash equilibrium which requires a strategy to be stable to any perturbations that may occur to the games as they evolve [192] and was extended to stochastic games [193]. Stochastic games have been shown to have an evolutionary stable strategy under certain conditions [194].

Partially observable stochastic games (POSG) extend stochastic games to partially observable environments where the payoffs are not known to the players. They are represented by the tuple (Q, N, O, A, P, r, b^0) where O denotes the set of observations and b^0 denotes the initial state distribution. POSG have been used to model learning sequential decision making in cooperative MAS [195]. Finding a Nash equilibrium for POSG belongs to the NP-hard computational complexity class [196], meaning they are computationally at least as difficult as NP problems which are solvable by a non-deterministic Turing machine in polynomial time. POSG subclasses include MDP, POMDP and their MAS extensions.

Many exact and approximate solutions have been proposed for POSG. An iterative method to eliminate dominant strategies was proposed in [196]. Authors in [173] combined a generalized version of the DP used for POMDP and eliminated dominated strategies to find a solution to POSG. When agents use the same payoffs, this approach can converge to an optimal solution. The proposed method was tested on multi access broadcast channel control and compared to a policy tree building brute force algorithm. POSG have also been used to model cooperative MAS decision making in partially observable Markovian environments [195, 197]. However, this model's solution is intractable as the number of agents increases. Therefore, an approximate solution was computed based on Bayesian games to achieve decentralized control in robot teams with limited communication. The algorithm was validated on the 2-robot tag problem, 2-agent lady and tiger problem and multiple access broadcast channel problems.

3.4.2 Bayesian Games

Bayesian games are games with incomplete information. Generally, these uncertainties can be modeled as uncertainties in agents' payoffs [17]. They are defined by (N, G, P, I) where N represents the set of agents, G represents the set of games the agents might be playing, P represents the common prior distribution over all the games and $I = (I_1, \dots, I_N)$ represents the partitions of G , for each agent. Examples of Bayesian games include signaling games, bargaining, auctions, and market competitions. Strategic policies can be obtained by converting incomplete games to imperfect information ones. Solving for the Bayesian Nash equilibrium, the Nash equilibrium in Bayesian games, includes best response, RL or other learning rules, linear programming [198] and Monte Carlo methods [199]. Bayesian Nash equilibrium, which consists of a strategy profile and a player's belief about other players' types, always exists.

3.4.3 Some Insights

While game theoretic approaches had been mainly used in competitive MAS, some models have gained popularity in cooperative MAS due to the agents' capabilities of modeling other agents in the game. This property can be useful in robotic systems where robots are unable to communicate with others. However, this restricts the number of agents in the system due to increasing computational costs. Game theory's systematic mathematical approach has been an attractive quality for many applications but combining it with some heuristic approaches such as deep learning might lead to improved performance in robotic applications and others.

3.5 Swarm Intelligence

Swarm intelligence describes the behavior of decentralized cooperative agents, whether natural or artificial, working toward a common global goal [200]. Self-organized and distributed behavior of locally aware and locally interacting agents are pillars of swarm intelligence [18]. Systems modeled in this fashion generally consist of many autonomous but homogeneous agents implementing simple rules with agent interactions restricted to local neighborhoods.

3.5.1 Biologically Inspired Algorithms

Swarm intelligence was inspired by many social insects and animals including ants, bees, wasps, termites, bats, fish, and birds. In some ways, swarm intelligence is similar to RL; both are iterative algorithms that use a reinforcement signal to learn a solution [18]. However, the reinforcement signal modifies the behavior of the agent differently in both algorithms.

Many algorithms have been inspired by bee colony behavior. Bee colony optimization [201] is based on direct communication among agents performing a series of moves for a certain duration based on the strength or fitness of the solution, also known as “waggle dancing”. This recruits other agents to the most fit solution. Navigation is based on path integration where agents continuously update a vector indicating the position of the start location. Ant colony optimization (ACO), inspired by ant colony behavior, is a class of algorithms that rely on indirect communication [202]. Navigation is based on depositing pheromones along the trail. A more fit solution results in stronger pheromones on the trail that lead to recruiting more agents. PSO is inspired by flocks of bird and schools of fish [203]. Agents navigate the environment searching for better solutions using principles from birds’ movements. A pigeon inspired optimization algorithm relied on the magnetic field, sun and landmarks to achieve path planning [204]. Distributed implementations of ACO [205, 206] and PSO [207] have been developed to speedup convergence.

3.5.2 Some Insights

While such systems exhibit desirable properties like robustness, flexibility, scalability, low complexity, inherent parallelism and fault tolerance [52, 99], they have important limitations. Most swarm systems consist of identical agents, leading to their limitations according to [52]. The agents must be homogeneous or can be divided into a small number of homogeneous clusters following simple rules to make decisions. However, there are many applications, such as search and rescue operations, that require heterogeneous, complex agents working toward a common goal.

3.6 Graph Theory

Decision making in MAS have been modeled as graphs with nodes representing agents and edges representing interactions and information flow among agents [208]. In this section, we focus on one popular approach called influence diagrams (IDs), briefly discussing the model and some of its strengths and weaknesses.

3.6.1 Influence Diagrams

IDs are referred to as decision networks in [15] and are a graph theoretic approach that provide a framework for decision making by adding actions and utilities to Bayesian networks [209]. Chance nodes (ellipses) represent random variables. Decision nodes (rectangles) represent choices available to the agent and utility nodes (diamonds) compute the utility of these choices. The action with the highest utility is chosen. IDs can be converted to decision trees by traversing the

diagram from top to bottom, creating a node in the decision tree when a decision node is encountered and adding edges with values equal to probabilities of parent nodes; leaves portray the utility of a path. IDs require the optimization of all parent nodes of a decision variable [122]. Dynamic IDs (DIDs) extend IDs to sequential decision making problems by combining DP with IDs [210] and have been viewed as computationally equivalent to POMDP [211]. They exploit the separability of the value function to generate computationally efficient solutions.

Multi-agent IDs (MAIDs) generalized IDs to MAS by generating decision rules that depend on decision rules made by other agents [212]. This is graphically represented by connecting decision nodes that depend on each other; a directed relevance graph is thus produced. MAIDs represent games with imperfect information graphically and are an alternative to the normal and extensive forms of game representation [211]. They can be converted either to extensive form games or to IDs and then solved.

A network of IDs (NIDs) is built on top of MAIDs to account for uncertainties in other agents' decision making and hierarchy of beliefs [213]. This formalism can represent irrational behavior and distinguishes between different agent models in the systems, i.e. it does not treat all other agents identically. Acyclic NIDs can be solved using a bottom up approach by converting each block to a MAID and solving it. Duplicates are included to account for beliefs about others' strategies. Cyclic NIDs are converted to acyclic NIDs and solved. However, both MAID and NID are applicable to episodic decision making only. Interactive DIDs were proposed in [211] as a MAS extension of DIDs and can be viewed as computational counterparts of I-POMDP. Models of other agents are clustered to reduce computational complexity but lead to approximate solutions.

3.6.2 Some Insights

Graph theory models the interaction of agents, allowing them to exchange information and make decision accordingly. However, the computational complexity of this approach increases exponentially in densely connected graphs with many nodes (agents). The main benefits of graph theory in MAS come from combining it with other approaches such as MDPs and control theory (discussed next) to extend these approaches to MAS. Furthermore, exploiting special structures such as sparsely-connected dense sub-graphs are a common approach to reduce computational cost and improve performance.

3.7 Control Theory

Control theory is an established field that aims to control physical systems by designing controllers using modify the input to achieve the desirable output. Its sub-fields include non-linear, adaptive, optimal, robust and stochastic control, to

name a few, and produce controllers with various properties to overcome limitations imposed by the real-world environment they operate in. However, as automation problems became more complex, researchers extended control theory to MAS by developing distributed controllers. We present a brief overview of this broad field next.

3.7.1 Distributed Cooperative Control

Distributed controllers are designed by combining concepts from control and graph theory. Specifically, interactions among agents are modeled using graph theory and the control problem is decomposed among the agents to obtain a distributed controller. The amount of communication among agents is dependent on the design of the distributed controller and can vary based on the nature and complexity of the task (whether it is easy decomposable), the optimality of the control algorithm and other factors. Since many controllers are based on optimization algorithms, distributed optimization is an integral part of distributed control [214]. Unlike other approaches, distributed cooperative controllers designed using control and graph theory can be mathematically validated to prove optimality, stability, robustness, and convergence, to name a few properties.

Distributed controllers have been applied to various control problems. For example, a Lyapunov based voltage and frequency controller was designed for micro-grid systems that only requires local communication among neighbors [215] and a secondary voltage distributed controller based on input-output feedback linearization that requires sparse communication [216]. A Lyapunov based distributed lead-follower control system was developed that scaled to large MAS when the interaction topology is an undirected graph [217]. Distributed consensus tracking was achieved by designing: distributed adaptive controllers in weakly connected, directed graphs [218], a distributed optimal control algorithm [219], and non-linear distributed impulsive control (control signals are given as impulses instead of continuously) with delayed impulses in undirected graphs [220]. Stochastic sampling in leader-follower consensus problems has been shown to improve scalability of MAS [221]. Distributed impulsive control has also been applied to heterogeneous MAS synchronization problems [222]. Yang et al. proposed distributed output regularization using adaptive control in MAS with a switching topology [223]. Other applications include formation control [224] and navigation [225] in MRS.

3.7.2 Some Insights

While control theory adopts systematic mathematical approaches to develop controllers, some systems are simply too complex and intractable for such methods. For example, most algorithms assume linear systems. Therefore, data-driven

methods such as those in distributed artificial intelligence are necessary to automate certain complex tasks in real-world environments. Nevertheless, distributed cooperative controllers are necessary in some applications where sufficient data is not available or mathematically-proven optimal controllers are crucial like in aviation or military domains.

3.8 Applications

MAS decision making models have been applied to many problems in various fields from robotics to wireless sensor networks. Next, we mention some of the problems that have been solved using the aforementioned decision making models.

3.8.1 Robotics

Cooperative MRS have been applied to many problems that require various degrees of coordination. Loose coordination examples include formation control and foraging, while tight coordination examples include object transport and robot soccer. Environment uncertainty, robot actuating and sensing diversity, system scalability, real-time processing and limited computational resources are a few challenges that should be addressed when designing decision making algorithms. Decisions related to robot actions, information sharing, and coalition formation, are essential to the successful deployment of robots in real world environments. POSG has been applied to multiple problems in robotics [226, 227]. MDPs [228] and POMDP [229–233] have been used for robotics coordination including robot soccer [234]. Graphical models for consensus [235], formation [236, 237], and rendez-vous [236] have also been investigated. Finally, swarm intelligence has been applied to underwater environments [238], 3D space [52] and robot path planning problems [118, 204]. It has been applied to dynamic task allocation [239], distributed localization problems [240], foraging tasks [241–243], collision free navigation [244, 245] and communication free flocking with minimal memory requirements [246]. Swarm-bots, wheeled robots that can physically connect to each other and form larger entities, accomplished coordinated motion, self-assembly, cooperative transport, goal search and path formation [247, 248]. The thermotactic behavior of honeybees inspired the decision making of a swarm of microbots with limited communication capabilities in spatial behavior problems [249]. Swarmanoids, a heterogeneous system composed of three types of complementary swarm robots, performed complex tasks like object retrieval in 3D space [52].

3.8.2 Repeated Coalition Formation

Forming groups of agents that change based on environmental conditions is critical to the successful deployment of MAS in real-world environments. Repeated

coalition formation under uncertainty deals with forming time varying coalitions where agents do not have complete information about other agents’ capabilities. Adopting traditional coalition formation methods such as auctioning and search algorithms cannot handle uncertainty since they assume complete knowledge of agent capabilities. Therefore, this problem has been modeled as a sequential decision making problem by many researchers and solved using some of the decision making models discussed in this work, which can handle information uncertainty. This allows the dynamic formation of robot teams where uncertainty is high and can lead to the automation of complex tasks that was previously unfeasible.

While searching for a solution that strikes a balance between redundancy for fault tolerance and agent’s skill complementarity is challenging enough, attempting to do so with incomplete and noisy information about agents’ skills further complicates matters. Dynamically reforming coalitions also poses its own challenges by requiring the algorithm to determine the lifetime of a coalition. However, repeated coalition formation with uncertainty allows MAS to cope with the stochastic environments and complex tasks.

Matthews et al. assumed the problem was fully observable and adopted MDP to model a football team formation problem [250]. Agent transitions between coalitions were modeled as a MDP, the Shapley value and marginal contributions were used to prune the search space and the best coalition structure was found using Markov probability distributions [251]. A POMDP model was also adopted to allow agents to learn other agents’ capabilities by interacting with each other [252]. IDs solved the problem of coalition formation for complex real-world missions by selecting a subset of coalition formation algorithms suitable for the problem at hand [253]. Swarm intelligence was used to search for the best coalitions to form [254]. Coalition games were generalized to problems with incomplete information through Bayesian games [255, 256].

3.8.3 Intelligent Transport Networks

With the increased awareness on sustainable living, transportation systems are challenged to endorse cutting edge technology and provide better services, while keeping an eye on safety and greener emissions. Intelligent transport networks are formed of autonomous or semi-autonomous communicating vehicles and road infrastructure such as traffic signals and road sensors. Decisions such as when to close a road or change a traffic light color to reduce traffic congestion, give directions to emergency response vehicles to avoid congested roads, improve road safety based on weather conditions, and others, are critical to make transportation smarter. To run efficiently, decisions need to be made in real-time on devices with limited computational resources. POSG modeled directional routing and scheduling of packet delivery in vehicular ad hoc networks [257]. POMDP was used to perform automated driving in urban traffic while dealing with sensor uncertainties [258]. Multi-agent RL has been used for routing algorithms [259–261],

adaptive broadcasting [262], adaptive data collection [263] and traffic signal control [264–271]. Intelligent transportation systems have utilized swarm intelligence to control traffic light scheduling [272], model complex transportation systems [273], develop routing protocols for vehicles [274, 275] and for information dissemination [276–280].

3.8.4 Wireless Sensor Networks

WSN are a collection of autonomous computing and sensing devices with limited computational resources. Their presence is ever increasing with the decreasing cost and size of hardware and emergence of Internet of Things. Integrating decision making in these networks allows us to implement functionality beyond simple information retrieval, making the integration of WSN with other smart city MAS, such as autonomous vehicles, electric grids and transport networks, feasible. The application of MDP was surveyed to model various problems in WSN including intrusion detection, sensor coverage, object detection, data exchange, topology formulation and other problems [281]. POMDP have been used for performance optimization [282], data and memory access control [283], and sleep scheduling [284]. IDs were used for lighting control in WSN and provided robustness to sensor uncertainties [285]. Swarm intelligence has been used for routing in WSN [286–292], clustering [293], cluster head selection [294], for security protocols [293, 295] and node positioning and localization [296, 297].

3.8.5 Intrusion Detection

An essential component of network security is detecting threats before they can compromise the network. Since networks are inherently decentralized, detecting threats can be modeled as a MAS decision-making problem where agents cooperatively determine whether a threat is present. Intrusion detection systems monitor activities in network infrastructures such as WSN and mobile ad hoc networks to identify malicious behavior. It involves detecting malicious packets, tracking their sources and optimizing performance of networks. These systems are considered MAS because each node on the network contributes to keeping the network secure, by making decisions related to the maliciousness of packets. Intrusion detection is a difficult problem because running the decision making algorithms should not use up a significant portion of the network nodes' limited computational resources while identifying threats as early as possible on a wide variety of network technologies.

Many models have been adopted in intrusion detection systems, as surveyed in [298]. MDP identified the network's most vulnerable nodes based on attackers' previous behaviors [299, 300]. Bayesian games modeled attacker/defender games [301]. Multi-agent RL were used to implement distributed intrusion detection systems [302]. Swarm intelligence is a popular approach to intrusion

detection, evidenced by the recently published surveys [113–115]. PSO has been widely applied in combination with support vector machines [303–305] and linear programming [306]. ACO was used in IP traceback problems [307].

3.8.6 Other Applications

Cooperative MAS has been applied to many other fields. Noteworthy applications, briefly discussed next, are cognitive radios, smart electric grids, resource allocation, and distributed optimization. Traditional radio paradigms suffer from spectrum scarcity and usage inefficiency. Cognitive radios have been presented as one possible solution. A cognitive radio is a smart radio that efficiently utilizes the available spectrum. MDP and RL were used to model and solve spectrum sensing and management problem [308–312]. Jamming in networks were modeled using stochastic games [313] and spectrum sharing was modeled using game theory [314]. Decision making models to cognitive radios reduces the wasted, already scarce, spectrum resources and improves their efficiency in switching frequencies.

Smart electric grids will be the primary method of power distribution in smart cities where efficient scheduling, generation and distribution of power are essential. However, the unpredictability of demand and supply as well as plant diversity and plant failures are some of the challenges faced in this field. POSG [315], POMDP [316, 317], multi-agent RL [318, 319] and swarm intelligence [116, 320–322] have been adopted to model and solve power distribution, scheduling, power flow and load forecasting problems.

Resource allocation aims to distribute heterogeneous resources in a fair and efficient manner to maximize resource utilization. Resource allocation has many applications in resource constrained domains where many agents are battling to gain access to these scarce resources such as robotics and cloud computing. Decision making models adopted for this problem solve this problem more efficiently than other approaches such as search or constrained optimization methods. Adopted models include POMDP, used to minimize network bandwidth congestion and fairly allocate resource [323], Bayesian games [324] and stochastic games with multi-agent RL used for job and resource scheduling in grid computing [325].

Distributed optimization, a useful tool in many fields including robotics, electric grids, and large-scale optimization problems, consists of optimizing an objective function in a distributed fashion. MAS decision making leveraged consensus and communication rules to model distributed optimization; each agent optimized part of the objective function before combining their results [326]. For example, game theory has formulated distributed optimization problems as games [327, 328]. In [329], potential games and cooperative control provided a theoretical framework to formulate distributed optimization problems. Also, swarm intelligence including PSO [330, 331] and an algorithm that mimics bacterial foraging [332] have been used to solve multi-objective optimization problems.

Graph theory including time-varying directed graphs [333] and weight-balanced directed graphs [334] have modeled information exchange in a distributed optimization framework.

3.9 Challenges

Although MAS decision making has seen significant improvements in the past decade, it is still plagued with many issues. To reap all the benefits of the Internet of Things boom and improve smart cities and smart living, decision making systems need to address some of the remaining challenges.

3.9.1 Scalability

Decision making algorithms should be scalable, especially in heterogeneous MAS, to accomplish more complex tasks. The scalability of current models greatly relies on agent homogeneity and the level of interaction. Swarm intelligence can scale to large MAS since agents are homogeneous and interaction is minimal and restricted to the agent's neighborhood. MDP variants and game theoretic models do not scale well since the complexity of the algorithm increases exponentially due to the model formulation that results in exponentially large state spaces. Using the graph theoretic formulation for large MAS results in densely connected graphs which are computationally expensive.

3.9.2 Computational Complexity

Decision making algorithms should be computationally efficient due to the need for real-time decision making in some applications or the lack of enough computational resources of agents. Robots generally have limited on board computational resources due to size and weight constraints and might not be able to offload their computations to the cloud due to bandwidth scarcity, poor or unreliable connectivity, and minimum latency requirements. Agent interactions MAS increases the computational cost per agent as the number of agents increases, especially in methods that extend single agent models to MAS if careful consideration of interaction cost is not performed. Tightly coordinated tasks also increase the computational burden due to the large amount of communication and data exchange among agents. Decision making algorithms should be designed with all these constraints in mind to successfully complete complex tasks.

3.9.3 Dynamic Environments

The environment's dynamic and unpredictable nature makes it difficult to foresee, design and test an agent that can handle all these situations. Therefore, decision

making algorithms should generalize well to situations that have not been learned or tested. They should be able to adapt to the dynamic environment and various uncertainties it might encounter and should be robust to noisy and incomplete information generated by sensors, and non-deterministic actions. POMDP, IDs, POSG and Bayesian games are better suited to handle uncertainties than MDP and its variants that assume fully observable environments, because they account for partially observable environments, incomplete and imperfect information in their algorithms. Agent failures are also a source of uncertainty in MAS that hinder the completion of tasks. Unlike other models, swarm intelligence models are better suited to handle agent failures due to the homogeneous nature of agents and minimal interaction necessary. However, this is still an issue that needs to be considered whenever MAS are designed.

3.9.4 System Heterogeneity

Heterogeneous MAS can deal with environment diversity and complex tasks. However, this heterogeneity makes cooperative decision making more complex: agents need to model other agents when capability uncertainty exists, agent capabilities should be compatible, and agents should have a common language to communicate and interact, in addition to other issues. Swarm intelligence simplifies modeling by assuming all agents are homogeneous. Graph theoretic models, POSG and its sub-classes can handle heterogeneous MAS if the state and observation spaces are designed appropriately. I-POMDP and I-DID inherently model other agents, making them better than other graph and game theoretic models in dealing with MAS heterogeneity.

3.9.5 Big Data

Recent advancements in processing big data has led to significant improvements in research areas like object recognition, speech recognition and natural language processing. The next step is to use this information to make better decisions in MAS and handle more complex tasks. Decision making has yet to maximize its benefits from big data. Algorithms that model and generate representations of such data like convolutional neural networks (deep learning) produce computationally expensive models that are not suitable for computationally limited agents or decision making algorithms whose computational cost grows exponentially with the dimensionality of the data. Yet, allowing agents to access these models through the cloud has its own complications with respect to cloud accessibility, bandwidth constraints, representation compatibility, privacy and security.

3.9.6 Evaluation Standards

Evaluation standards are necessary in MAS decision making to compare proposed algorithms and assess the state-of-the-art. General metrics include solution optimality, algorithm completeness, and algorithm time and space complexity. However, additional evaluation metrics of MAS decision making are necessary to enable better comparisons. Some work has developed evaluation metrics and workflows to quantify the performance of MAS. Braubach et al. developed abstract metrics that would be specialized for MAS applications, and include function (e.g. restrictions), usability (e.g. simplicity), operating ability (e.g. performance) and pragmatic metrics (e.g. installation) [335]. Lass et al. distinguished between two metric categories: effectiveness (e.g. success, failure, 90% accuracy) and performance (e.g. resource consumption, time complexity) [336], that could be applied to four MAS levels (agent, framework, platform, host). They presented a framework to select appropriate metrics for a given application and performed a case study on a distributed constrained optimization problem. Di et al. developed a hierarchical metric system where both inter (communication and cooperation) and intra agent metrics measured environment complexity, agent rationality, autonomy, reactivity, and adaptability [337]. This system was tested on a knowledge management problem for the automotive industry with two agents only. Marir et al. proposed an evaluation platform that included metrics like average of communication load and validated the platform on an auctioning problem [338]. Nevertheless, standards to evaluate and compare the performance of MAS on real-world environments are still underdeveloped. Existing metrics have been tested on a hand-full of narrow-scoped scenarios that did not necessarily include robot agents.

3.9.7 Other Challenges

Task complexity poses a challenge for decision making algorithms because they do not have the capability of recognizing what tasks can be decomposed into simpler tasks that they can complete. Adding this capability to decision making algorithms in MAS in addition to dynamically recognizing what tasks require tight coordination and what tasks can be accomplished with minimal interaction among agents will increase the scope of automated tasks. Learning algorithms for decision making and perceiving agents should be autonomous. Reducing the number of manually tunable hyper parameters that require human intervention will allow algorithms to generalize better to unknown environments.

3.10 Conclusion

This survey discusses decision making models and algorithms to find policies for cooperative MAS for different applications. MDP and game theoretic models,

swarm intelligence, and IDs were covered, for which optimal and sub-optimal policies were obtained using RL, DP, direct policy search, Monte Carlo methods, linear, quadratic and mixed integer programming, evolutionary computing, and RNN. MAS applications noted include smart electric grids, WSN, intelligent transportation systems, and robot teams performing search and rescue, object transport and exploration and mapping. While state of the art methods within the past five years are significantly better than their predecessors, research advances in this field are promising but still needs to answer many questions. Decision making algorithms should leverage big data advancements and the Internet of Things to obtain better policies, algorithms should be scalable as more complex tasks require larger MAS, and distributed algorithms should be adopted to ease the computational burden and run on computationally limited devices. Furthermore, evaluation standards or benchmarks need to be developed to enable comparison of algorithms and to facilitate their verification and validation. These improvements would take us a step closer to effective deployment of various MAS in smart cities.

Even though this survey focused on positively interacting MAS, MAS with negative interactions is has many real-world applications. Competitive MAS in robotics, intelligent transportation systems, smart electric grids, among others is an active area of research. Decision making models based on theories in economics, game theory and psychology have been developed. MAS with conflicting interactions is also a prominent area of research especially in robotics and intelligent transportation systems where mobile vehicles use the same infrastructure and must co-exist with minimal interference to complete conflicting goals. Research areas such as conflict management, conflict resolution, and deceptive behavior modeling have emerged to address these issues.

Chapter 4

PDP-CF Formulation

In this chapter, we introduce the PDP-CF formulation that allows the formation of coalitions while determining the delivery schedule of packages. The formulation is based on the PDP optimization formulation, also known as capacitated vehicle routing problem. While PDP is one of the most popular models of package delivery, existing formulations in the literature do not support MAS. Allowing heterogeneous MAS to delivery packages would facilitate the automation of package delivery systems by allowing agents to better handle environmental stochasticity, variable and task complexity.

Next, we first motivate the proposed formulation in section 4.1. We present a brief overview of existing PDP formulations and solvers in the literature in section 4.2. Section 4.3 presents an illustrative example to further motivate our proposed formulation before presenting our PDP-CF formulation in section 4.4. Section 4.5 analyzes the theoretical computational complexity of PDP-CF. Finally, section 4.6 concludes with final remarks before the next chapters present solvers for the PDP-CF formulation.

4.1 Introduction

The autonomous package delivery problem consists of multiple sub-problems from different fields including PDP in scheduling, object transport in robotics, cooperative robot navigation, and others. While vehicle routing problems and PDP have been extensively researched in the field of scheduling [339, 340] and to a certain extent in conjunction with robotics [341], packages are generally assumed to be transported by one robot. The most basic PDP formulation schedules the delivery of packages by modeling the problem as a graph and finds the routes that minimize a cost function. However, allowing MRS to delivery a single package would increase the number of feasible assignments, especially when capacity constraints are considered.

In this work, we propose a PDP formulation that allows multiple robots to

form a coalition to delivery a package that may exceed their individual payloads. A coalition is defined to be a group of robots cooperatively transporting one or more packages simultaneously. Specifically, we develop a 3-index and 2-index MIP formulation that solves for a delivery schedule that minimizes a certain cost function. Multiple cost functions are considered and the possibility of overlapping coalitions is investigated which leads to the possibility of integrating non-robotic agents into coalitions. Finally, the theoretical computational complexity of both formulations are compared. Even though the 2-index formulation has a smaller number of optimization variables to solve for compared to the 3-index formulation, both scale exponentially with the vehicle set cardinality.

4.2 Literature Review

We first present the most common PDP formulations that consider different assumptions and constraints on the system, then discuss algorithms that have been used to find optimal or sub-optimal delivery schedules.

4.2.1 PDP Formulations

A PDP aims to find a schedule that allows a set of vehicles (or possibly robots) to execute transportation requests such as delivering a set of packages from source to destination. To goal is to find a schedule that satisfies various constraints such as capacity, time window and priority constraints, while minimizing an objective function such as distance traveled, energy consumed, and delivery time, among others. A general PDP formulation was proposed in 1995 [342] and many variants with different constraints have been studied. Some of the main assumptions in a PDP formulation are:

- **Number of vehicles** that will deliver items. Single vehicle and multi-vehicle PDPs have been studied.
- **Vehicle start and end locations** specify the start location(s) (e.g. warehouse) and return location(s), if any.
- **Item pickup and delivery locations** specify whether items are picked up from a single location or from different locations and whether they will be delivered to a single location or individual locations.
- **Capacities and demands** specify a vehicle’s maximum capacity and an item’s demand or payload.
- **Time windows** specify the earliest and latest times an item can be picked up and delivered, respectively.

- **Maximum route durations** specify the amount of time a vehicle may spend on any given delivery route.
- **Maximum transport times** specify the amount of time an item can spend in a vehicle before it is delivered.
- **Transfers or transshipments** specify whether items can be exchanged among vehicles at intermediary locations between pickup and delivery.

Static PDPs [343] assume that all delivery requests and vehicles are known before a schedule is formed while dynamic PDPs [344] allow the addition of new requests and vehicles as the delivery schedule is being formed. Dynamic PDP further complicates PDPs since not all the information is available from the start. Thus, greedy algorithms that reach suboptimal solutions tend to be adopted. Online PDP algorithms solve the PDP algorithm in an online fashion as vehicles execute delivery tasks [345].

PDPs have been most commonly formulated as constrained optimization problems with both continuous and integer variables. The objective function may consist of multiple criteria including minimizing distance traveled, total duration, completion time, client inconvenience, and delivery cost, to name a few. Constraints are derived from the initial assumptions of the formulation (e.g. time windows, capacities) but also includes constraints that ensure the validity of a schedule such as delivering an item after it has been picked up.

Compact formulations have been proposed to enable more efficient solutions using general purpose optimization solvers. A 2-index formulation was proposed in [346] based on solving a PDP with time windows as a Hamiltonian tour problem. However, this approach did not perform well on large scale PDPs. Another 2-index formulation was derived based on the Vehicle Routing Problem (VRP) with time windows formulation and explicitly assigning vehicle routes [347].

PDPs have been shown to be NP-hard by reducing them to a traveling salesman problem [348], which implies that find an optimal schedule in polynomial time may not be feasible. Next, we discuss algorithms that attempt to solve PDPs optimally or sub-optimally.

4.2.2 PDP Solvers

Many heuristics and metaheuristics have been proposed to find delivery schedules using search-based algorithms. From optimization, branch-and-cut or branch-and-bound MIP algorithms have been commonly adopted to solve PDP variants including PDP with time windows (PDPTW) and transfers (PDPTW-T) [349–351]. Particle swarm optimization [352] and simulated annealing [353] have also been adopted in large neighborhood PDPTW. A 2-phase heuristic was proposed in [354]. A branch-and-cut algorithm was proposed for the 2-index PDPTW formulation [346].

Table 4.1: Robot Setup

| Robot | Location (x,y) | Payload (units) |
|-------|----------------|-----------------|
| R_1 | (0,1) | 2 |
| R_2 | (0,1) | 2 |
| R_3 | (1,0) | 3 |
| R_4 | (1,0) | 1 |

Table 4.2: Package Setup

| Package | Source Location (x,y) | Destination Location (x,y) | Mass (units) |
|---------|-----------------------|----------------------------|--------------|
| A | (0,1) | (2,1) | 3 |
| B | (0,1) | (1,2) | 1 |
| C | (1,0) | (2,1) | 1 |
| D | (1,0) | (1,2) | 1 |

4.3 Motivating Illustrative Example

We consider an example to illustrate the benefits of allowing transfers and coalitions in the pickup-delivery problem. Given a set of robots and packages, described in Tables 4.1 and 4.2 respectively, we need to find the robot-package assignment that will minimize the total distance traveled by all the robots. In this example, we simplify the PDP formulation to only consider the source, destination and mass of the packages, in addition to the payloads of robots. The environment is a simple 3x3 grid, as shown in Figure 4.1. We define transfers as a package being handed to a different robot at a location other than its final destination. Coalitions are defined as a group of robots collaboratively transporting a package or multiple package, i.e. the robots share the load of the package(s). If transfers and coalitions are not allowed, the assignment in Table 4.3 is optimal and leads to a total distance of 12 units traveled by all robots. The best solution is obtained when both coalitions and transfers are allowed: a total distance of 5 units is traveled by the robots, as shown in Table 4.4.

When transfers and coalitions are not allowed, R_3 must travel a distance of two units (without any load) to pick up package A. Similarly, R_1 must travel a distance of two units to pickup package C before delivering it. Therefore, the total distance traveled by all the robots to deliver all packages is 12 units, as shown in Table 4.3. However, when coalitions can be formed and transferring packages between robots is allowed, the total distance traveled by all the robots

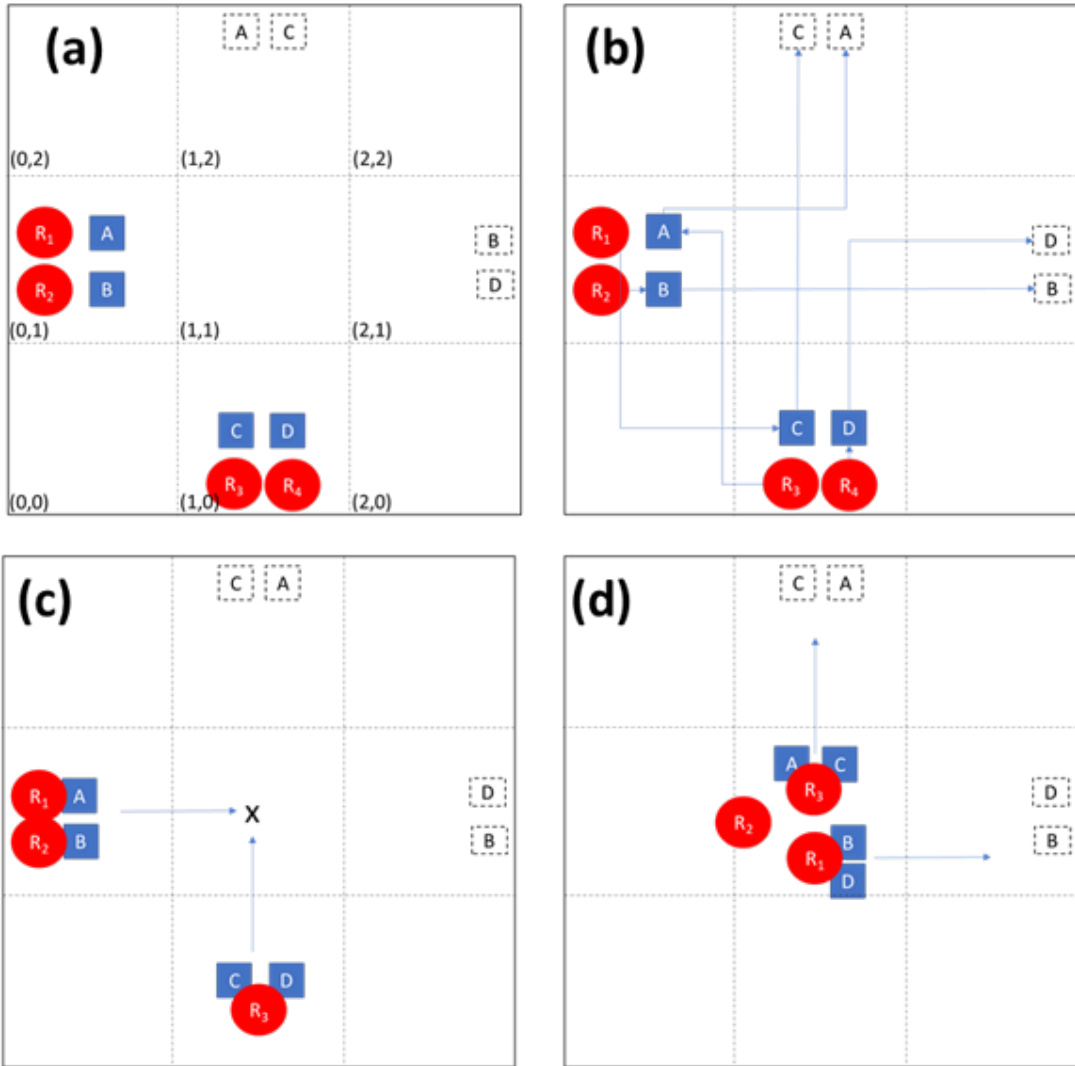


Figure 4.1: Illustrative PDP example: (a) Initial map, (b) Solution without transfers or coalitions, (c) Step 1 of solution with transfers and coalitions, (d) Step 2 of solution with transfers and coalitions.

Table 4.3: Solution when transfers and coalition formation are not allowed

| Time step | Action | Distance traveled |
|------------|---------------------|-------------------|
| 1 | PICKUP(R_1 , C) | 2 |
| 1 | PICKUP(R_2 , B) | 0 |
| 1 | PICKUP(R_3 , A) | 2 |
| 1 | PICKUP(R_4 , D) | 0 |
| 2 | DELIVER(R_1 , C) | 2 |
| 2 | DELIVER(R_2 , B) | 2 |
| 2 | DELIVER(R_3 , A) | 2 |
| 2 | DELIVER(R_4 , D) | 2 |
| Total Cost | | 12 |

Table 4.4: Solution when transfers and coalitions are allowed

| Time step | Action | Distance traveled |
|------------|----------------------------------|-------------------|
| 1 | PICKUP(R_1+R_2 , A) | 0 |
| 1 | PICKUP(R_1+R_2 , B) | 0 |
| 1 | PICKUP(R_3 , C) | 0 |
| 1 | PICKUP(R_3 , D) | 0 |
| 2 | TRANSFER(R_1+R_2 , R_3 , A) | 2 |
| 2 | TRANSFER(R_3 , R_1 , D) | 1 |
| 3 | DELIVER(R_1 , B) | 1 |
| 3 | DELIVER(R_1 , D) | 1 |
| 3 | DELIVER(R_3 , A) | 1 |
| 3 | DELIVER(R_3 , C) | 1 |
| Total Cost | | 5 |

is reduced to 5 units, as shown in Table 4.4. This is due to the fact that robots R_1 and R_2 can form a coalition with a combined total payload of 4 units to transport packages A and B to an intermediary location. At this location, R_3 can transfer package D to one of the other robots and carry package A and C to their final destination while a second robot (R_1 or R_2) deliver package B and D to their final destination. Furthermore, when transfers and coalitions are allowed, only three robots are necessary to deliver the four packages, as opposed to four robots.

4.4 PDP Formulation

Next, we propose a PDP formulation, hereafter referred to as PDD-CF, which allows the formation of coalitions among different agents. This would increase

Table 4.5: Nomenclature

| Symbol | Definition |
|-----------------|---|
| V | Set of vehicles, $v \in V$ |
| m | Number of vehicles |
| P | Set of packages, $p \in P$ |
| n | Number of packages |
| $K_v = 2^V$ | Set of coalitions, $k_v = \{v_i\} \in K_v$ |
| G | Graph represented a PDP instance |
| A | Arc in G , $(i, j) \in A$ |
| N | Nodes in G |
| P | Pickup nodes |
| D | Delivery nodes |
| x_{ijk} | Boolean optimization variable |
| c_{ijk} | Arc cost |
| Q_{jk} | Capacity of coalition k at node j |
| t_{ij} | Travel time associated with (i, j) |
| B_{jk} | Time at which coalition k starts servicing node j |
| M | Constant |
| q_i | Demand at node i |
| $[e_i, \ell_i]$ | Time window |
| k_s | Maximum allowable coalition size |
| d_{ij} | Distance between nodes i and j |
| d_{ijk} | Distance traveled by a coalition k from node i to j |
| $ k $ | Number of vehicles in coalition k |
| $cap(v)$ | Capacity of a vehicle |
| $dem(p)$ | Payload demanded by a package |

the number of feasible assignments, especially when capacity constraints are considered. Table 4.5 summarize the notation adopted throughout this document.

4.4.1 A 3-index PDP-CF Formulation

PDP are most commonly formulated as a 3-index MIP problem [355]; it is represented as a complete graph $G(N, A)$, where $N = \{0, 1, \dots, 2n + 1\}$ denotes the set of nodes in the graph and A the set of arcs or connections between the nodes in N . We assume n total transportation requests or packages; with each request, we associate a pickup node $i \in P$ and delivery node $n + i \in D$. To simplify the notation, we define $P = \{1, \dots, n\} \subset N$ to be the set of pickup nodes and $D = \{n + 1, \dots, 2n\} \subset N$ to be the set of delivery nodes, with nodes $0 \in N$ and $2n + 1 \in N$ representing the origin and final depots. The set of available vehicles is denoted by $V = \{1, 2, \dots, v, \dots, m\}$ with cardinality $|V| = m$ and capacity cap_v .

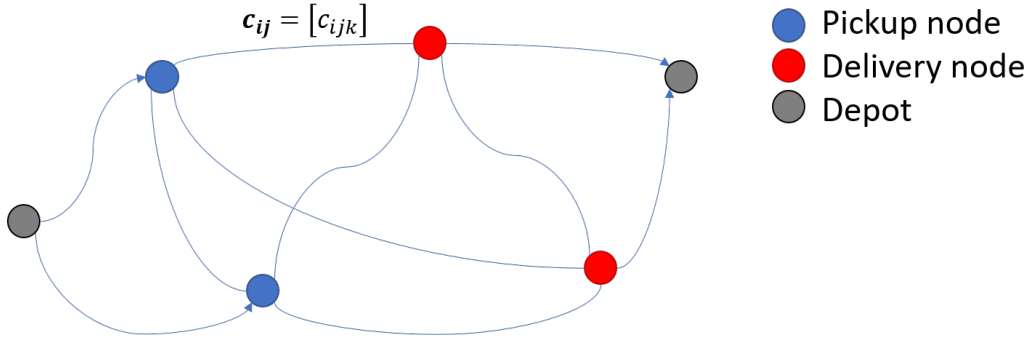


Figure 4.2: PDP's Graphical Representation

Unlike many works in the literature, we do not assume V to be homogeneous, i.e. the capabilities of the vehicles in V are diverse. Figure 4.2 illustrates this graphical representation for $n = 2$.

The objective function in (4.1) minimizes the total routing cost by searching the space of coalitions instead of the space of vehicles. This leads to an exponentially larger search space since the set of coalitions is the power set of V , $K = 2^V$. Furthermore, c_{ijk} is the cost of a coalition k traversing arc $(i, j) \in A$. Existing formulations assumed a routing cost that is independent of the vehicle traversing the route since homogeneous vehicles were considered. However, this does not apply to our formulation since the size of the coalition influences this cost. The routing cost can be one of many functions such as traveled distance, delivery time, consumed energy or a combination of the three. x_{ijk} is a binary variable that is equal to 1 when coalition $k \in K$ traverses arc $(i, j) \in A$.

$$\min_x \sum_{k \in 2^V} \sum_{i \in N} \sum_{j \in N} c_{ijk} x_{ijk} \quad (4.1)$$

We impose multiple constraints on the optimization problem to obtain a valid delivery schedule. First, we ensure that a transportation request is assigned to only one coalition in (4.2) and the corresponding pickup and delivery nodes are visited by this same coalition in (4.3). Then, we ensure that each route starts at the origin depot and terminates at the final depot by adding the constraints in (4.4) and (4.5). The constraint in (4.6) guarantees that each coalition entering a node will leave this node. Capacity and time constraints are imposed by (4.7) and (4.8), where Q_{jk} is the capacity of coalition k at node j , t_{ij} is the travel time associated with arc (i, j) , B_{jk} is time at which coalition k starts servicing node j , and M is a sufficiently large constant. t_{ij} includes the service time of each node and obeys the triangle inequality. $q_i \geq 0$ is the load associated with a pickup node $i \in P$; $q_{n+i} = -q_i$ is the load associated with a delivery node $n + i \in D$.

Unlike VRP, in PDP, we need to ensure that a pickup node is visited before a delivery node by adding the constraint in (4.9). Since time windows $[e_i, \ell_i]$ are

associated with each node $i \in N$, the constraint in (4.10) ensures the schedule does not allow coalition k to visit a node i before e_i or after ℓ_i . To ensure the maximum capacity of each coalition is never exceeded, the constraint in (4.11) is imposed. The capacity of a coalition is defined as the sum of the capacities of the vehicles in the coalition, as shown in (4.12). Finally, the integrality of variables is satisfied by adding the constraint in (4.13).

$$\sum_{k \in K} \sum_{j \in N} x_{ijk} = 1 \quad \forall i \in P \quad (4.2)$$

$$\sum_{j \in N} x_{ijk} - \sum_{j \in N} x_{n+i,j,k} = 0 \quad \forall i \in P; k \in K \quad (4.3)$$

$$\sum_{j \in N} x_{0jk} = 1 \quad \forall k \in K \quad (4.4)$$

$$\sum_{i \in N} x_{i,2n+1,k} = 1 \quad \forall k \in K \quad (4.5)$$

$$\sum_{j \in N} x_{jik} - \sum_{j \in N} x_{ijk} = 0 \quad \forall i \in P \cup D; k \in K \quad (4.6)$$

$$Q_{jk} \geq Q_{ik} + q_j - M(1 - x_{ijk}) \quad \forall i \in N; j \in N; k \in K \quad (4.7)$$

$$B_{jk} \geq B_{ik} + t_{ij} - M(1 - x_{ijk}) \quad \forall i \in N; j \in N; k \in K \quad (4.8)$$

$$B_{ik} + t_{i,n+i} \leq B_{n+i,k} \quad \forall i \in P; k \in K \quad (4.9)$$

$$e_i \leq B_{ik} \leq \ell_i \quad \forall i \in N; k \in K \quad (4.10)$$

$$\max\{0, q_i\} \leq Q_{ik} \leq \min\{cap_k, cap_k + q_i\} \quad \forall i \in N; k \in K \quad (4.11)$$

$$cap_k = \sum_{v \in k} cap_v \quad (4.12)$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in N; j \in N; k \in K \quad (4.13)$$

Since we are considering coalitions of cooperating vehicles instead of independent vehicles, we add a set of constraints that ensures coalitions do not overlap, as shown in (4.14). If coalitions k and k' contain common vehicles (their intersection is not empty), then only one of these coalitions can be used. This set of constraints is proportional to the size of the set of coalitions and equal to $|K|^2/2 = 2^{2|V|-1}$.

$$x_{ijk} + x_{ijk'} = 1 \forall k \cap k' \neq \emptyset \quad (4.14)$$

To reduce the size of the search space and number of constraints, we limit the maximum size of a coalition. Instead of considering all subsets of V , we only consider all subsets with a maximum cardinality $k_s \ll m$.

In summary, in this formulation, we introduce three main modifications to the 3-index MIP in [355]:

- The search space of possible solutions is changed from the set of vehicles to the power set of vehicles (set of all subsets).
- The cost function depends on the coalitions, i.e c_{ij} is replaced by c_{ijk} .
- A set of constraints was added to ensure that coalitions do not overlap.

4.4.2 A 2-index PDP-CF Formulation

Lu et al. [346] proposed a compact 2-index MIP by mapping a PDP to a Hamiltonian tour problem. They achieve this by extending the graph to include $m + 1$ artificial nodes, $N_o = 2n + 1, 2n + 2, \dots, 2n + m + 1$, representing start and return locations for each vehicle. The extended graph, $\tilde{G} = (\tilde{N}, \tilde{A})$, $\tilde{N} = P \cup D \cup N_o$, also includes an extended set of arcs \tilde{A} that includes arc connecting all nodes in the pickup and delivery sets, $P \cup D$, cycles in N_o , arcs from nodes in D to nodes in N_o , and arcs from N_o to P . Finding the minimum Hamiltonian tour in \tilde{G} is equivalent to finding the minimum cost route in G (as defined in the previous subsection). The objective function is depicted in (4.15) which clearly shows two indices for the optimization variable, but on a larger graph.

$$\min_x \sum_{i \in \tilde{N}} \sum_{j \in \tilde{N}} c_{ij} x_{ij} \quad (4.15)$$

Multiple constraints are imposed on the objective function and are listed below. Eq. (4.16) and (4.17) ensure that all nodes are visited.

$$\sum_{(i,j) \in \tilde{A}} x_{ij} = 1 \quad \forall i \in \tilde{N} \setminus \{2n + m + 1\} \quad (4.16)$$

$$\sum_{(i,j) \in \tilde{A}} x_{ij} = 1 \quad \forall j \in \tilde{N} \setminus \{2n + 1\} \quad (4.17)$$

Eq. (4.18) and (4.19) copy the value b_{ki} into b_{kj} when a route goes from node i to node j .

$$b_{ki} \leq b_{kj} + (1 - x_{ij}) \quad \forall (i, j) \in \tilde{A} \setminus \{2n + m + 1, 2n + 1\}; k \in \tilde{N} \setminus \{i\} \quad (4.18)$$

$$b_{kj} \leq b_{ki} + (1 - x_{ij}) \quad \forall (i, j) \in \tilde{A} \{2n + m + 1, 2n + 1\}; k \in \tilde{N} \{i\} \quad (4.19)$$

Eq. (4.21) insures that $b_{ij} = 1$ when $x_{ij} = 1$.

$$x_{ij} \leq b_{ij} \quad \forall (i, j) \in \tilde{A} \quad (4.20)$$

Eq. (4.21) states that a node can never precede or succeed itself.

$$b_{ii} = 0 \quad \forall i \in \tilde{N} \quad (4.21)$$

Eq. (4.22) and (4.23) insure that a pickup node is visited before its corresponding delivery node.

$$b_{n+i, i} = 0 \quad \forall i \in \tilde{P} \quad (4.22)$$

$$b_{i, n+i} = 1 \quad \forall i \in \tilde{P} \quad (4.23)$$

Eq. (4.24) guarantees that the same vehicle visits the pickup and delivery nodes.

$$b_{i, 2n+j} = b_{n+i, 2n+j} \quad \forall i \in \tilde{P}; 2n + j \in N_o \quad (4.24)$$

The capacity constraint of each vehicle is imposed by (4.25), where Cap_j is the capacity of vehicle j .

$$q_j + \sum_{i \in \tilde{N}} q_i b_{ij} \leq Cap_j \quad \forall j \in P \quad (4.25)$$

Eq. (4.26) states that node $2n + 1$ is the first node in the Hamiltonian tour.

$$b_{i, 2n+1} = 1 \quad \forall i \in \tilde{N} \quad (4.26)$$

Eq. (4.27) and (4.28) guarantee that the nodes in the Hamiltonian tour are visited in the correct order.

$$b_{2n+k, 2n+j} = 1 \quad \forall k < j; 2n + k \in \tilde{N}; 2n + j \in \tilde{N} \quad (4.27)$$

$$b_{2n+j, 2n+k} = 0 \quad \forall k < j; 2n + k \in \tilde{N}; 2n + j \in \tilde{N} \quad (4.28)$$

Eq. (4.29) states that node $2n + m + 1$ is the last node in the Hamiltonian tour.

$$b_{i, 2n+m+1} = 1 \quad \forall i \in \tilde{N} \{2n + m + 1\} \quad (4.29)$$

Finally, the integrality constraint is imposed by (4.30).

$$x_{ij}, b_{ij} \in \{0, 1\} \quad \forall i, j \in \tilde{N} \quad (4.30)$$

Modifying the presented 2-index MIP formulation to allow coalitions implies that the subset N_o contains $2^m + 1$ artificial nodes and that each occurrence of m should be replaced by 2^m .

4.4.3 Cost Function

c_{ijk} represents the cost of a coalition of vehicles k traversing the path from node i to node j . In this work, we consider three types of cost functions: 1) total traveled distance, 2) delivery time, and 3) energy consumption.

- **Distance traveled** by all vehicles to accomplish delivery tasks which depends on the distance between pickup and delivery nodes. Different distance metrics can be adopted such as Euclidean distance, described by (4.31), or Manhattan distance, described by (4.32). We denote the distance between node i and j as d_{ij} .

$$d_{ij} = (|x_i - x_j|^2 + |y_i - y_j|^2 + |z_i - z_j|^2)^2 \quad (4.31)$$

$$d_{ij} = (|x_i - x_j| + |y_i - y_j| + |z_i - z_j|) \quad (4.32)$$

- **Delivery time** accounts for the handling time and the time to travel from one node to another. It is affected by the maximum speed of a vehicle. Since we assume heterogeneous vehicles, maximum speeds may vary. Therefore, the maximum speed of a coalition is equal to the maximum speed of the slowest vehicle in the coalition.
- **Energy consumption** computes the energy required to complete a delivery task. It includes the energy spent on traveling from one node to another, communicating with other vehicles, performing computations, acquiring sensory data and others.

Each cost function has its advantages and disadvantages. The distance cost function is simple to compute given the coordinates of source and destination locations. In existing PDP formulations, this distance is traveled by a single vehicle; in PDP-CF, it is traveled by all the vehicles in a coalitions. Therefore, it also implicitly penalizes large coalitions since the total distance traveled by the coalition is $d_{ijk} \leq |k|d_{ij}$, where $|k|$ represents the number of vehicles in coalition k . It is an inequality because we may have vehicles capable of transporting smaller vehicles. Therefore, the smaller vehicles wouldn't travel this distance. However, a distance cost function does not account for communication costs of adopting a coalition or idle time (i.e. the time required to load/unload packages

to/from the vehicles). Traveled distance is also independent of the properties of the vehicle traveling this distance which is disadvantageous in a heterogeneous MAS setting. Delivery time incorporates idle time which may increase with the size of coalitions and is dependent on the properties of the vehicles but does not minimize the communication costs of the coalition.

Energy consumption is the most computationally expensive cost function of the three but is the most general. It implicitly minimizes distance traveled and idle time. Communication costs incurred in coalitions are also accounted for. Furthermore, this cost function allows us to abstract from vehicles or robotic agents and consider any type of agent that can positively contribute to the completion of a task. This could be the cloud that allows vehicles to perform more complex computations or sensors in the intelligent transportation infrastructure that provide information to road vehicles. It allows us to quantify the cost of acquiring information to improve cooperation and decision making. Weather conditions can affect the efficiency of various vehicles: strong winds could significantly impede quadrotors even if they would travel the least distance. However, computing the energy consumption is computationally expensive and could significantly slow down the convergence of the optimization problem and the real-time responsiveness of dynamic PDP formulations. It also requires the existence of energy consumption models for all agents (vehicles, sensors, computational resources...). Nevertheless, in this work, we adopt the distance cost function using the Manhattan distance metric due to its simplicity and widespread use in the literature.

4.4.4 Coalition Overlap

Throughout this chapter, we have imposed the constraint of not allowing overlapping coalitions. In this subsection, we will consider the implications of relaxing this condition. However, this would require a slight modification to our definition of a coalition. Defining a coalition as a group of agents that contribute to the delivery of one or more packages, opens the door to non-robotic agents whose contributions enable vehicular agents to physically deliver packages. These contributions could be in the form of sensory information to improve the vehicles' decision making during task execution, or computational resources to allow the execution of more sophisticated algorithms for enhanced performance. As a result, Internet-of-Things and cloud computing technologies can be abstracted to facilitate integration with MRS. Therefore, the set of agents, V , in the PDP-CF formulation would be split into two possibly overlapping subsets: the set of load-bearing agents (or vehicles), V_1 , and the set of non-load-bearing agents, V_2 , i.e. $V = V_1 \cup V_2$ and $V_1 \cap V_2 \neq \emptyset$. The non-overlapping coalitions constraint would still apply to V_1 but not V_2 .

However, the distance cost function would no longer be suitable since it does not account for the cost of utilizing non-load-bearing agents. Instead, the energy

cost function would be more suitable since it can incorporate the work done by the non-load-bearing agents. The cost of acquiring certain information would consist of two components: 1) the energy expended by the device acquiring this information, and 2) the communication cost of acquiring this information from another agent or the cost of transporting the device (i.e. sensor) on-board the vehicle. Similarly, the cost of executing a specific algorithm would be equal to the computational cost, communication cost of connecting to the cloud or the cost of placing the hardware on-board. Communication costs depend on the medium; for example, Bluetooth spends less energy per byte of information but is only suitable for short range communication, compared to WiFi [356].

To obtain an efficient MAS, it is important to quantify the benefits of having agents join a coalition to execute a task, beyond a simple binary assessment of whether the task was successfully completed or not. The benefits of load-bearing agents is straight forward: not incorporating these agents would lead to a failure in executing a task. The benefits of non-load-bearing agents is not as evident. The value of information not only depends on the cost of acquiring this information but also the quality of this information. The more accurate information an agent has, the better its decision will be. Quantifying quality or accuracy can be accomplished by estimating the noisiness of this information: information with more noise is less reliable in decision making. Noise can be introduced to the system from the sensors, the communication hardware, the environment, and other agents, to name a few sources. Similarly, the value of an algorithm’s output depends on the cost of the computational resources and the accuracy of the output. Therefore, combining the cost and benefit of including an agent in a coalition by modifying the objective function would lead to a more faithful representation of the effectiveness of a coalition.

Even though the remainder of this thesis only considers load-bearing agents, this digression sheds some light on future research directions and the importance of the proposed formulation in developing a unified MAS framework that incorporates MRS, Internet-of-Things, intelligent transportation systems and other smart city infrastructure for autonomous package delivery.

4.5 Theoretical Analysis

Table 4.6 summarizes the computational complexity of 3-index and 2-index PDP formulations with and without CF. The graph size is based on the number of nodes and arcs in the graph. The number of optimization variables determines the size of the search space and affects how quickly MIP can converge to a solution, if any. The 2-index MIP formulation is more compact than the 3-index formulation when coalitions are not allowed since it has less optimization variables at the cost of a larger graph; the number of nodes and arcs grow linearly and quadratically, respectively, with the vehicle set cardinality. However, when CF is introduced into

Table 4.6: Graph Complexity

| | Without CF | With CF |
|------------------------|----------------------------------|--------------------------------------|
| 3-index MIP | | |
| Optimization variables | $(2n)^2m$ | $(2n)^22^m$ |
| Nodes | $2n$ | $2n$ |
| Arcs | $(2n)^2$ | $(2n)^2$ |
| 2-index MIP | | |
| Optimization variables | $2(2n + m + 1)^2$ | $2(2n + 2^m + 1)^2$ |
| Nodes | $2n + m + 1$ | $2n + 2^m + 1$ |
| Arcs | $(2n)^2 + 2(m + 1)n + (m + 1)^2$ | $(2n)^2 + 2(2^m + 1)n + (2^m + 1)^2$ |

both 2-index and 3-index formulations, the optimization problem and graph grow exponentially with the vehicle set cardinality. This is problematic because the search space becomes exponentially larger, making convergence to at least sub-optimal solutions more challenging. Limiting the maximum size of the coalition would result in smaller optimization variables (i.e. 2^m would be replaced by 2^{k_s} where $k_s \ll m$) but could still be challenging.

4.6 Conclusion

In this chapter, we introduced the PDP-CF formulation based on optimization theory. Two MIP formulations were derived and compared from a theoretical computational complexity perspective. However, both approaches scaled exponentially with the number of vehicles leading to convergence problems when attempting to find an optimal solution. As a result, we opted to investigate search-based and data driven approaches in the coming chapters.

Nonetheless, it is beneficial to formulate more efficient MIP solutions by imposing additional assumptions such as restricting the number of agents per coalition. Future work would also investigate the performance of the system when allowing coalition overlap while adopting an energy-based cost function. Incorporating non-vehicular agents should also be validated on realistic benchmarks.

Chapter 5

PDP Solver: An Evolutionary Approach

In this chapter, two evolutionary algorithms, GA and QGA, are proposed as solvers for the PDP-CF formulation presented in the previous chapter. The search-based approach attempts to circumvent the exponentially large search space of PDP-CF. To that end, section 5.1 motivates evolutionary algorithms as solvers for PDP-CF. A brief overview of existing GA solvers for PDP formulations is presented in section 5.2. Section 5.3 presents the GA and QGA solver formulations. Finally, section 5.4 reports the empirical performance of evolutionary solvers on PDP-CF scenarios before section 5.5 concludes with final remarks.

5.1 Introduction

Evolutionary algorithms have been adopted in many non-convex optimization problems. Examples of evolutionary algorithms include GA, simulated annealing, ant colony optimization, and particle swarm optimization, to name a few. Their popularity in many domains stems from their simplicity and ease of use. They search the space of solutions using biologically inspired processes such as evolution, mutation, and cross over. Such processes have been shown to enable an efficient traversal of the search space while reducing the probability of getting stuck in local minima. Furthermore, the emergence of quantum computing technology has enabled the introduction of search algorithms capable of exploring exponentially large search spaces in polynomial time. Evolutionary algorithms have been formulated for quantum computer to leverage this exponential speed up [357]. Quantum mechanics' mathematical constructs such as the linear superposition of states, state collapse and others have also been incorporated into these algorithms to further improve performance.

The exponentially large search space of PDP-CF makes it a good candidate to

leverage QGA’s advantages. QGA’s chromosomes would encode the 3-index MIP optimization variables. However, imposing the capacity and overlap constraints becomes more challenging since this encoding does not prevent QGA from exploring the infeasible region; this hinders QGA’s convergence to a solution in a reasonable time. To avoid entering the infeasible region, we propose a GA encoding that inherently guarantees that the explored solutions are within the feasible region; each vehicle in the set is represented by a chromosome with the genes representing packages that could be delivered by the vehicle. Breaking down the packages into virtual packages of unit size allows us to create coalitions without increasing the problem size. Empirical results on multiple PDP scenarios reflect the merits of the proposed approach.

5.2 Literature Review

Before delving into our proposed encodings, we briefly describe the various evolutionary algorithm formulations developed in the literature to solve vehicle routing problems including PDP. Wang et al. [358] formulated a MIP model for simultaneous PDP with time windows and encoded the optimization variables into GA’s chromosomes. To solve PDP with time windows, [359] proposed a grouping GA which implies that genes represent multiple delivery requests as opposed to one request. Xiao et al. [360] proposed a quantum ant colony optimization algorithm to solve vehicle routing problems. Zhang et al. [357] formulated a hybrid quantum evolutionary algorithm for PDP and demonstrated its superior exploration capabilities compared to GA. Dakroub et al. [361] adopted a GA for intelligent carpooling by adopting multiple population threads with each chromosome in a population representing a driver and each gene representing passengers with the driver. Crossover and mutation operations were modified to prevent GA from exiting the feasible region. Ursani et al. [362] developed a two-phase optimization solver based on GA for vehicle routing problems; the algorithm breaks down the original optimization problem into smaller problems, optimizes those smaller problems then combines their solutions and fine tunes the results using a de-optimization workflow. GA encodes the order of customers. Finally, Jia et al. [363] encoded target and UAV information into GA’s chromosomes to assign targets related to flight trajectories to cooperating UAVs.

5.3 Methodology

Due to the larger search space of PDP-CF, we propose a QGA solver to efficiently traverse this exponentially large search space and possibly converge to a better solution than MIP. To achieve further improvements, we propose a GA encoding that does not scales linearly with the vehicle set cardinality while still allowing

the formation of coalitions.

5.3.1 QGA Solver

QGA adopts multiple mathematical constructs from quantum mechanics, including complex probabilities, state superposition and state collapse, to efficiently explore a search space and converge to a solution. In QGA, the 3-index MIP optimization variables, encoded in the chromosome, are viewed as existing in a linear superposition of their basis states, i.e. a binary randomly variable is simultaneously in a state of 0 and 1 until it is “observed”. In this context, “observed” implies that the fitness function is computed; a random collapse is performed before the fitness of the chromosome can be computed. This collapse is influenced by the probability amplitudes of the state. Specifically, a random number, $r \in [0, 1]$, is generated and compared to α^2 . If it is larger, then the corresponding optimization variable is assigned a value of 1; otherwise, it is assigned a value of 0, as shown in (5.1).

$$x_{ijk} = \begin{cases} 1 & \text{if } r > \alpha_{ijk}^2 \\ 0 & \text{if } r < \alpha_{ijk}^2 \end{cases} \quad (5.1)$$

The workflow of QGA is as follows:

1. Initialize the population.
2. Collapse the chromosome to one of the basis states.
3. Compute fitness of the collapsed chromosomes.
4. While we have not converged:
 - (a) Select the rotation angle using Table 5.1,
 - (b) Apply the rotation gate to the chromosomes in the population, shown in (5.2).
 - (c) Apply the mutation operator.
 - (d) Apply the crossover operator.
 - (e) Collapse the chromosome to basis states.
 - (f) Compute fitness of the collapsed chromosomes.

$$\begin{bmatrix} \alpha_{i+1} \\ \beta_{i+1} \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \quad (5.2)$$

Table 5.1 summarizes the rotation gate’s adjustment strategy. x_i denotes the i^{th} bit in the chromosome and b_i denotes the i^{th} bit in the chromosome with

Table 5.1: Quantum rotation gate angle adjustment strategy [2]

| x_i | b_i | $f(x) > f(b)$ | $\delta\theta_i$ | $s(\alpha_i, \beta_i)$ | | | |
|-------|-------|---------------|------------------|------------------------|-----------------------|----------------|---------------|
| | | | | $\alpha_i\beta_i > 0$ | $\alpha_i\beta_i < 0$ | $\alpha_i = 0$ | $\beta_i = 0$ |
| 0 | 0 | True/False | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | False | ω | +1 | -1 | 0 | ± 1 |
| 0 | 1 | True | ω | -1 | +1 | ± 1 | 0 |
| 1 | 0 | False | ω | -1 | +1 | ± 1 | 0 |
| 1 | 0 | True | ω | +1 | -1 | 0 | ± 1 |
| 1 | 1 | True/False | 0 | 0 | 0 | 0 | 0 |

| x_{111} | | x_{ijk} | | $x_{n,2n,2^m}$ | |
|--------------------|-----|--------------------|-----|-------------------------|--|
| $ \alpha_{111} ^2$ | ... | $ \alpha_{ijk} ^2$ | ... | $ \alpha_{n,2n,2^m} ^2$ | |
| $ \beta_{111} ^2$ | ... | $ \beta_{ijk} ^2$ | ... | $ \beta_{n,2n,2^m} ^2$ | |

Figure 5.1: QGA Chromosome Encoding based on 3-index Formulation

the best fitness. $\delta\theta_i$ denotes the adjustment angle step and $s(\alpha_i, \beta_i)$ denotes the rotating angle direction. $f(x)$ and $f(b)$ denote the fitness of a chromosome and that of the best chromosome, respectively.

Encoding

Instead of encoding the binary optimization variables x_{ijk} , as would be done in GA (see Figure 5.2), QGA encodes the probability amplitudes of the optimization variable's states, as shown in Figure 5.1. We can view the binary variables as having one of two states, 0 or 1, with complex probability α and β , respectively, such that $\alpha^2 + \beta^2 = 1$. We view the genes as a 2-dimensional vector with continuous real values representing the magnitude of the complex probabilities. The number of genes in the chromosome is equal to the number of optimization variables.

Initialization

The probability amplitudes are initialized to $\frac{1}{\sqrt{2}}$ for all chromosomes in the population. This implies that the probability of collapsing to state 1 or 0 is equal to 0.5. If we have some prior knowledge of which states would be better than others for each variable, we can initialize α and β accordingly. However, in this work, we assume no such prior knowledge.

Mutation Operation

The mutation operation must not violate the $\alpha^2 + \beta^2 = 1$ condition. Therefore, the mutation operator randomly modifies α only. Then, β is computed to ensure the squared sum is still equal to 1.

Crossover Operation

The crossover operation is similar to traditional GA where a random cutoff point is selected, and the genes from two parent chromosomes would be swapped.

Fitness Function

The fitness function adopted in this formulation to evaluate the quality of a solution is based on the objective function of the PDP-CF formulation. Specifically, the fitness of a chromosome is inversely proportional to the cost function which is the total distance traveled to delivery packages. This not only depends on the locations of the sources and destinations but also on the number of agents in a coalition. Furthermore, to incorporate environmental characteristics, we modified the fitness function to include a cost for traveling in difficult conditions. For example, an aerial vehicle would have a harder time traveling in windy conditions than a ground vehicle; a biped is prone to failures on rocky terrains compared to six-legged vehicles. While some of these additional costs can be reflected in an energy cost function (as in the former), the increased probability of hardware failures (as in the latter) are not reflected. Therefore, we adopt a heuristic function that quantifies the difficulty (denoted by $c(x)$ in (5.3)) a certain agent performing a task in a given set of conditions. We adopt a categorical function with 5 levels of difficulty: “very easy”, “easy”, “moderate”, “difficult”, and “very difficult”. Due to the difference in units between $c(x)$ and the distance cost function ($d(x)$), simply adding both cost functions is not suitable. Instead, we convert the 5 levels of difficulty into 5 values between 0 and 1 and multiply the distance cost function by the difficulty factor, as shown in (5.3), i.e. we assume the distance is much larger due to the additional difficult.

$$f(x) = (1 + c(x)) \times d(x) \quad (5.3)$$

Constraints

Since the PDP-CF formulation is a constrained optimization problem, part of the search space is in the infeasible region. Therefore, we need to insure that the chromosomes in the population are within the feasible region. To do so, we couple (or entangle in quantum mechanical terminology) the genes that would violate the constraints, i.e. if x_{ijk} and $x_{ijk'}$ cannot be 1 simultaneously because

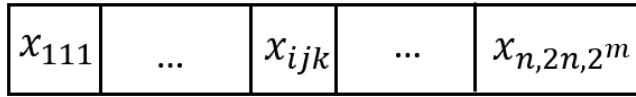


Figure 5.2: GA Chromosome Encoding based on 3-index Formulation

coalitions k and k' would violate the coalition overlap constraint, then these two genes are coupled and the collapse of one would affect the value of the other.

5.3.2 GA Solver

The workflow of GA is as follows:

1. Initialize the population.
2. Compute fitness of the chromosomes in the initial population.
3. While we have not converged:
 - (a) Apply the mutation operator.
 - (b) Apply the crossover operator.
 - (c) Compute fitness of the population.

Encoding

One intuitive approach is to encode the binary optimization variables x_{ijk} , as shown in Figure 5.2. However, for a simple PDP with 5 packages and 3 vehicles, the number of optimization variables is 300; for a PDP-CF of the same size, the number of variables grows to 900, based on (4.1). Furthermore, feasible regions in our constrained PDP and PDP-CF formulations are a small portion of the search space. Therefore, naively traversing this space could result in a large number of chromosomes in the infeasible region, hindering GA and QGA's ability to converge to a solution, let alone an optimal or near-optimal solution. Therefore, it is important to develop an encoding that incorporates the constraints to avoid exiting the feasible region.

One such encoding is illustrated in Figure 5.3 where a collection of chromosomes represents a possible solution (e.g. population 1 is one solution and population 2 is another). The chromosome represents a vehicle; the length of the chromosome represents the capacity of a vehicle. Since vehicles have unequal capacities, the lengths of the chromosomes are not uniform. The number of chromosomes in a population is equal to the number of vehicles. Since packages have unequal demands, we represent a package by multiple virtual packages of uniform demand, i.e. if a package p_i has a demand of three units, we represent it by three virtual packages of with unit demand (see p_3 in Figure 5.3). Therefore, each gene

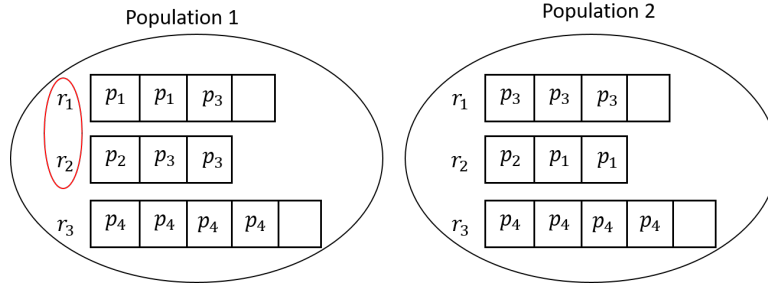


Figure 5.3: GA Chromosome Encoding

of the chromosome represents a package with a demand of one unit. If the virtual packages corresponding to a physical package are split over multiple vehicles, then these vehicles form a coalition. Therefore, we are able to search the space of vehicle coalitions without significantly increasing the number of variables in the problem.

However, this encoding does not solve the routing problem. It only solves the coalition-package assignment problem. Multiple solutions can be adopted but mainly fall under two categories: extending the encoding to solve the routing problem or running a separate algorithm to determine the delivery sequence. The former would expand the search space that GA would traverse by considering the order of packages in the chromosome as the sequence of delivery, i.e. the package in gene 1 would be picked up first and dropped off first. However, this may not result in the most optimal routing solution since all packages are picked up before any one can be dropped off. The latter implies that a routing problem for each coalition would be solved to determine the cost of the package-coalition assignment and can be solved using a traveling salesman problem solver such as GA, particle swarm optimization or other approaches in the literature. However, this approach is computationally expensive, leading to a computationally expensive overall PDP-CF solver.

Initialization

Randomly initializing the population does not guarantee that the chromosomes are in the feasible region. The set of scenarios encountered in PDP can be divided into three subsets based on the relationship between the set of packages and set of vehicles:

- Case 1: the total package demands exceeds the total vehicle capacities,
- Case 2: the total vehicle capacities is less than the total package demand
- Case 3: the total package demands is equal to the total vehicle capacity.

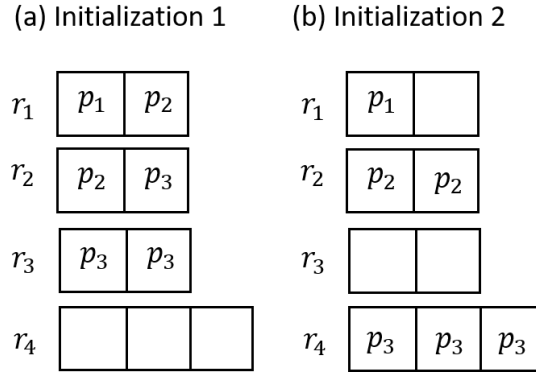


Figure 5.4: GA Initialization

If there are more packages to delivery than vehicles capable of delivering them (case 1), then we pick a subset of packages that would be delivered in the first round (based on priority, time windows or other criteria) and the other packages would be delivered in the second round. On the other hand, if the available packages to delivery do not require all the available vehicles to deliver them (case 2 or case 3), we keep the vehicles in the population but set their initial assignments to 0, i.e. they would not deliver any packages.

Multiple deterministic initialization schemes can be adopted. We can sequentially fill the vehicles to capacity with the available virtual packages until all packages have been assigned to a vehicle (Initialization 2 in Figure 5.4). We can also attempt to minimize the number of coalitions by first assigning packages to vehicles that can handle their demands then spread the remaining virtual packages across multiple vehicles (Initialization 1 in Figure 5.4). A sample delivery problem and initialization is presented in Figure 5.4: given three packages with demands of 1,2, and 3 units, there are four vehicles available each with a capacity of 2 units except the one who has a capacity of 3 units. In this work, we investigate both approaches and study their effect on the convergence of GA.

Crossover Operator

The crossover operation must maintain the validity of the chromosome, i.e. if a chromosome is in the feasible region, the crossover operation must keep the chromosome in the feasible region. Given two chromosomes in the feasible region, a crossover point is randomly selected based on the length of the shorter chromosome (since the chromosomes do not have a uniform length). Then, the genes to the left of the crossover point are swapped to maintain the correct chromosome lengths. Figure 5.5 illustrates this operation.

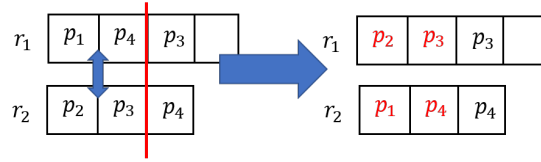


Figure 5.5: GA Crossover Operation

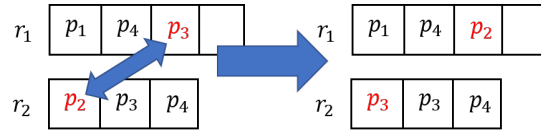


Figure 5.6: GA Mutation Operation

Mutation Operator

The mutation operation is slightly different from generic GA since this operation must insure that the chromosome still satisfies the constraints. In generic GA, mutation is simply changing the value of a random gene in a chromosome. However, doing so in our formulation may result in a package being dropped from the solution or a package being assigned to two vehicles. Therefore, we modify the mutation operator by defining it as being a gene swap between two chromosomes. Therefore, we randomly select two genes from two chromosomes and swap their values, as shown in Figure 5.6. While this may seem very similar to the crossover operation, we note a crucial difference that justifies keeping both operations: in certain scenarios, a portion of the genes would never be modified by the crossover operation. Introducing the mutation operation would allow GA to modify these genes and hence reach all of the search space. For example, if a problem contains four vehicles with capacity 5 and one with capacity 8, the last 3 genes in the vehicle with capacity 8 would never be modified since the crossover operation will always select a crossover point that is less than 5 (to maintain the vehicle capacities).

Fitness Function

GA adopts the same fitness function as QGA, described above.

5.4 Empirical Validation

5.4.1 Experimental Setup

The experiments were run on an Intel Core i7 processor with 8 GB of RAM with a Windows 10 operating system. The code was written in Matlab R2016b. In all experiments, the evolutionary algorithms were allowed to run for a maximum of

Table 5.2: Instance Description

| Instance | Number of Requests | Demand Range | Number of Vehicles | Capacity Range |
|----------|--------------------|--------------|--------------------|----------------|
| ID1 | 5 | {1,2} | 5 | {1} |
| ID2 | 3 | {1,2,3,4,5} | 5 | {1,2,3} |
| ID3 | 10 | {1,2,3,4,5} | 10 | {1,2,3,4,5} |
| ID4 | 10 | {1,2,3,4,5} | 10 | {1,2,3,4,5} |
| ID5 | 10 | {1,2,3,4,5} | 10 | {1,2,3,4,5} |
| ID6 | 30 | {1,2,3,4,5} | 20 | {1,2,3,4,5} |

100 iterations, unless otherwise specified. Also, Manhattan distances were computed in the cost function. To test the proposed solvers, we randomly generate PDP scenarios by specifying the number of vehicles and packages, in addition to their respective capacity and demand ranges. We also specify the size of the grid that includes the pickup and delivery locations. We assumed all vehicles are housed in a depot with a fixed location (at the origin of the 2D grid). PDP scenarios of various sizes were created and summarized in Table 9.3.

5.4.2 QGA Results

Table 5.3 summarizes the performance of QGA with and without CF. We notice that as the problem size increases, the computational time of QGA increases exponentially while that of GA increases linearly. For small problem sizes, it is clear that QGA converges to less costly schedules. For larger problems, QGA struggles to converge to a solution (Inf implies an infinite cost function, i.e. solution is not in feasible region); this is expected given the exponentially larger search space. Therefore, it is important to develop a more computationally efficient search algorithm for PDP-CF.

5.4.3 GA Results

Table 5.4 summarizes the performance of GA. We cannot compare the performance of GA and QGA in their current formulations because the latter finds a static assignment for all packages, whereas the former finds the best packages to deliver given the available robots (i.e. schedules are build incrementally). We notice that, for some instances, GA converged to a solution that contained all vehicles in a single coalition while other instances required multiple coalitions to delivery the packages. The routing approach did not produce a significant difference in most cases with brute force and encoded routing approaches converging to almost the same total distance traveled. Computational costs did not vary significantly. However, as the problem size increases, the brute force approach

Table 5.3: QGA Performance

| Instance | Algorithm | With CF | Total Distance | Running Time (sec) |
|----------|-----------|---------|----------------|--------------------|
| ID1 | QGA | Y | 768 | 0.5142 |
| ID1 | QGA | N | 1047 | 0.2207 |
| ID2 | QGA | Y | 466 | 0.2930 |
| ID2 | QGA | N | 452 | 0.1907 |
| ID3 | QGA | Y | 2487 | 71.3716 |
| ID3 | QGA | N | 2401 | 0.6433 |
| ID4 | QGA | Y | Inf | 69.0503 |
| ID4 | QGA | N | 67329 | 0.6145 |
| ID5 | QGA | Y | Inf | 68.3353 |
| ID5 | QGA | N | 2220 | 0.5559 |

becomes more costly.

5.5 Conclusion

In this chapter, we presented two solvers to find schedules for PDP-CF. QGA attempted to resolve the exponentially large search space of the MIP formulations by leveraging QGA’s speed and efficient search space exploration, a byproduct of the chromosomes’ linearly superposed gene states. GA’s encoding bypassed the exponentially large search space by dividing packages into virtual packages; chromosomes represented vehicles and genes represented these virtual packages, with coalitions formed among vehicles with virtual packages corresponding to the same physical package. Simulations on multiple PDP scenarios exhibited the effectiveness of the proposed encoding. Future work will investigate a more efficient encoding for QGA that does not grow exponentially with the number of vehicles. Applying GA’s encoding to QGA is one approach but would require extending the collapse procedure to non-binary gene. Formulating an online version of the algorithm would be beneficial for dynamic CF. Incorporating additional constraints such as time windows and package priority into the encoding would allow the application of these solvers to a wider range of PDP scenarios. Finally, benchmarking QGA and GA on more realistic scenarios would provide a better idea of their merits.

Table 5.4: GA Performance

| Instance | With CF | Routing | Total Distance | Running Time (sec) | Number of Coalitions (size range) |
|----------|---------|-------------|----------------|--------------------|-----------------------------------|
| ID1 | Y | Encoded | 18,035 | 0.21 | 3 ([1,2]) |
| ID1 | Y | Brute force | 18,035 | 0.16 | 3 ([1,2]) |
| ID2 | Y | Encoded | 28,345 | 0.15 | 2 ([1,5]) |
| ID2 | Y | Brute force | 28,825 | 0.14 | 2 ([1,5]) |
| ID3 | Y | Encoded | 95,448 | 35.1 | 1 (8) |
| ID3 | Y | Brute force | 27,200 | 35.2 | 1 (8) |
| ID4 | Y | Encoded | 35,867 | 34.9 | 5 ([1,3]) |
| ID4 | Y | Brute force | 40,686 | 35.41 | 5([1,3]) |
| ID5 | Y | Encoded | 63,533 | 35.1 | 5([1,3]) |
| ID5 | Y | Brute force | 52,142 | 35.0 | 5([1,3]) |
| ID6 | Y | Encoded | 146,728 | 35.56 | 10 ([1,3]) |
| ID6 | Y | Brute force | 96,063 | 35.36 | 10 ([1,3]) |

Chapter 6

PDP Solver: A Deep Learning Approach

In this chapter, we present a data driven approach to PDP-CF scheduling by training an ANN to solve for PDP-CF schedules. Since ANN are most commonly trained in a supervised framework, a dataset of PDP scenarios and their scheduling solutions must be compiled. Section 6.1 motivates this approach. Section 6.2 summarizes existing work that adopts ANN for decision making problems. Section 6.3 proposes multiple ANN formulations for PDP-CF, then section 6.4 describes the database that must be compiled to train these models. Finally, section 6.5 concludes with future work.

6.1 Introduction

ANN have been adopted to solve many problems from classification and regression to function approximation, optimization and control problems. Their universal approximation capabilities and their uniform structure make them attractive in many applications. Their drawbacks include the need to compile a large set of generally labeled training data, their susceptibility to overfitting and their computationally expensive training algorithms. Nevertheless, adopting ANN to find schedules for PDP-CF would shift the burden of modeling this complex problem from the system designer to the ANN.

Multiple input-output representations can be adopted for the PDP-CF problem, from images to graphs, that would influence how well ANN will learn a model and make predictions. Different ANN architectures have been presented in the literature to optimize the training process for different data types. For example, Convolutional Neural Networks (CNN) were developed to handle images or 2D inputs without resorting to flattening the input, whereas RNN could handle time series data without explicitly representing the time dependencies. Furthermore, a subset of these architectures has mainly targeted irregularly structured data

such as graphs and manifolds [364].

In this work, we develop an ANN solver for PDP-CF that can be trained on a compiled dataset of PDP scenarios. We discuss multiple formulations of ANN to solve PDP-CF. Specifically, one formulation is to represent PDP scenarios as graphs (inputted to the network) where nodes represent vehicles, pickup locations and delivery locations. Arcs represent valid transitions between nodes with arc weights representing the associated cost. The network output is a graph with the arcs that are included in the solution, i.e. the network will prune connections in the graph to obtain a scheduling solution. Another possibility is represent agents, packages and environments by feature vectors and have an ANN learn a model to map the input to an agent-package pair. We also discuss the characteristics of a dataset of PDP scenarios that should be compiled to train the proposed ANN models.

6.2 Literature Review

ANN have been applied to many control problems, optimization problems, and decision making models including MDP and POMDP, as discussed in Chapter 3. In this section, we will focus on ANN applied to graph data.

ANN including feedforward networks, CNN, and RNN have learned from an array of data types including images, speech, text, seismic and others. This data can be represented on regular grids (Euclidean spaces), non-Euclidean manifolds, or graphs. Data represented as graphs includes social networks, brain connectivity, word embeddings and PDP.

Extensions to non-Euclidean data in the literature include geodesic CNN which generalize CNN to non-Euclidean manifolds which are locally Euclidean, based on geodesic or polar coordinate system, that are similar to “patches” in images [364]. Multiple layers perform various functions such as geodesic convolution, Fourier transform [365, 366], linear combination, point-wise and global shape descriptor extraction [367], and other functions to effectively process non-Euclidean data. This approach is a natural way of generalizing CNN to manifolds, where convolutions are fulfilled by sliding a window over the manifold, and local geodesic coordinates are used in place of image “patches”.

CNN have learned affinity graphs for image segmentation in a supervised framework by manually creating ground truth affinity graphs for images [368]. Segmentation significantly improved when combining segmentation algorithms with CNN’s affinity graph on three dimensional segmentation of neuronal process reconstruction. [369] adopt spectral graph theory to extend CNN to irregular representations such as graphs. Spectral graph theory allows the efficient application of convolution filters on graphs by focusing on local neighborhoods and developing a recursive formulation. [370] applied CNN to graph-structured data by generalizing the convolution operator. Their “edge-conditioned convolution”

computes the weighted sum of signals between a node and its neighbors, weighted by the edges, without resorting to spectral graph theory.

Instead of modifying CNN to process graph, [371] preprocessed graphs then applied generic CNN formulations to the modified input. Local neighborhoods in graphs were first selected, then the subgraph was normalized to convert the graph representation to a vector space representation before CNN filters process the graphs.

6.3 Methodology

In this section, we discuss possible ANN formulations. First, we discuss how descriptors can be used to represent agents, packages and environments. Then, we present two possible ANN formulations that will be investigated in future work.

6.3.1 Agent Descriptors

An intelligent agent, whether virtual or physical, exhibits multiple characteristics that allow it to operate in its environment. An agent’s cognitive architecture is the core infrastructure that allows an agent to perceive and act on its environment to obtain a reward. The level of sophistication of the agent’s perception, reasoning, decision making and learning modules, among other factors, determines the effectiveness of the agent at accomplishing tasks with varying complexities.

Robotic agents capable of delivering packages autonomously consist of multiple hardware components, as shown in Figure 6.1. Perception and actuation hardware allow robots to interact with the environment, whereas communication devices enable robots to communicate with other agents or the cloud. Computational resources allow robots to process percepts and make decisions, and power sources provide the necessary energy for the agent to function. Each component influences the ability of each robot to perform package delivery tasks and should be considered when forming delivery schedules. While the state component in Figure 6.2 is not a physical component, it incorporates task dependent features describing the current state of the robot in the environment and provides crucial information to the decision making module.

Figure 6.2 represents a hierarchical list of functions a package delivery robot must possess. A package delivery task consists of two main subtasks: navigation and package handling. Functionality related to navigation and package handling are denoted in blue and red, respectively. To successfully navigate from source to destination, robots should execute multiple lower level functions such as localization, path planning, environment mapping and obstacle avoidance. Furthermore, executing these functions requires motion planning. Handling a package requires the robot to recognize these packages in the environment and displace them

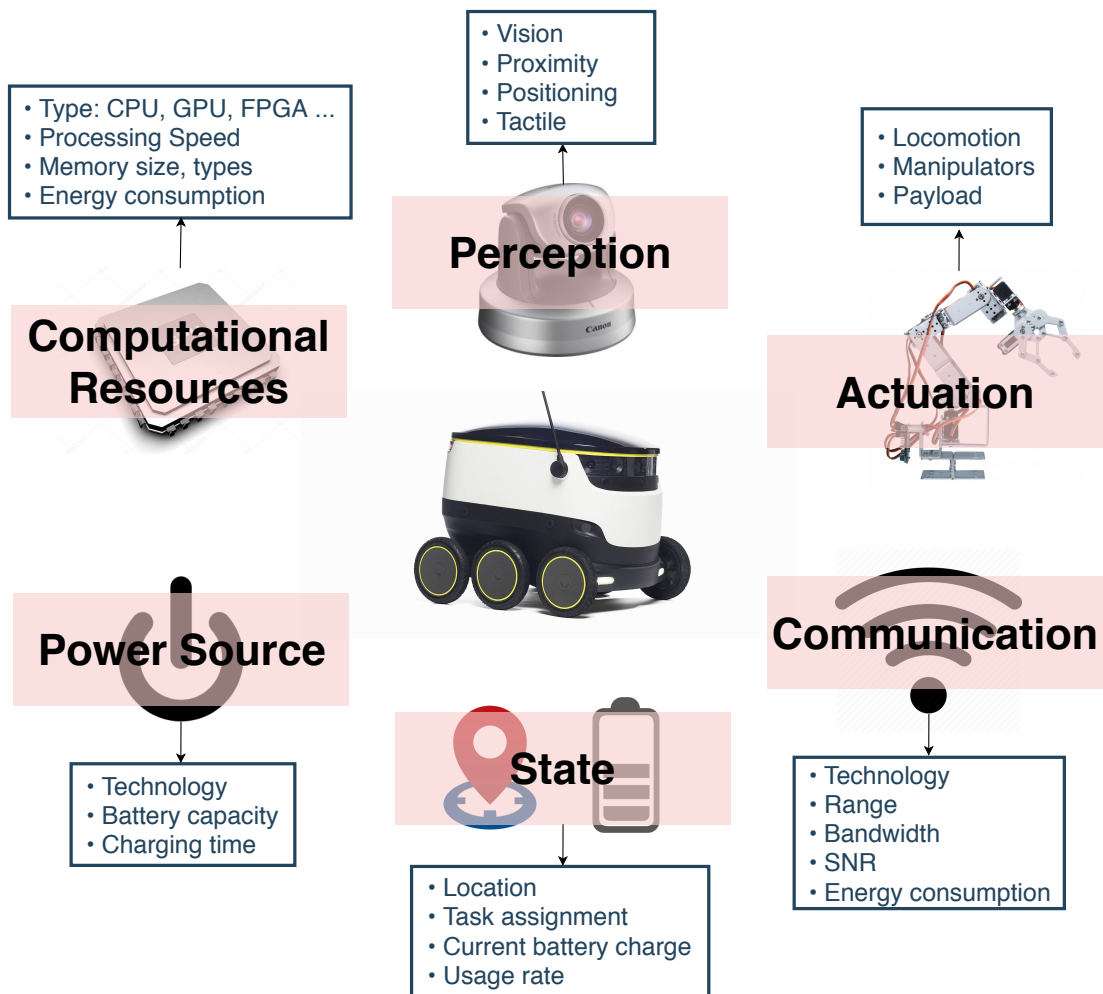


Figure 6.1: The main components of a robotic agent.

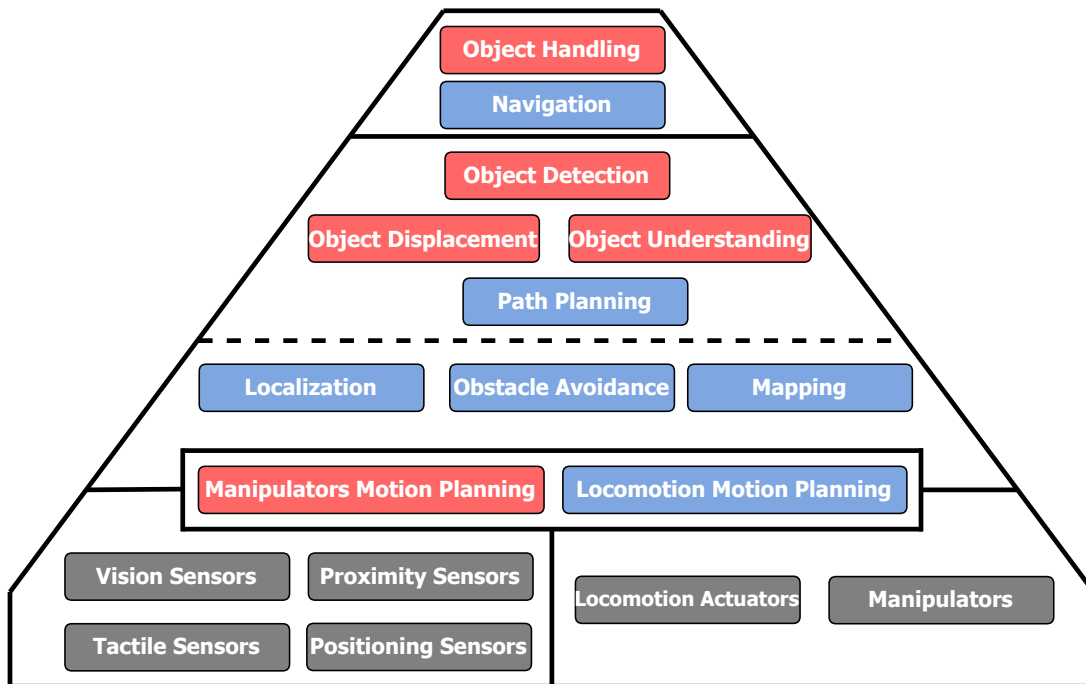


Figure 6.2: The main functions of a package delivery agent.

through lower level functionality like lifting, gripping or pushing. Finally, sensors and actuators provide the physical capabilities that allow robots to perform tasks in real world environments.

6.3.2 Package Descriptors

Packages must be represented by at least their source and destination locations and mass or demand (in capacitated problems). Additional properties can be included in the descriptor vector to create a more realistic representation of packages. For example, priority of delivery, handling requirements (e.g. fragile, flammable...) and pickup and delivery time windows can be included. Combining these features with environmental characteristics would lead to the selection of a more suitable coalition of robots or agents to deliver packages. Table 6.1 summarizes a descriptor of packages.

6.3.3 Environment Descriptors

Environmental descriptors can be generated from maps of given the source and destination locations of the packages and Internet of Things devices that can gather weather, traffic and other relevant information. Describing the environment would lead to a more suitable choice of agent that can handle a given situation. This would reduce the uncertainty and improve the probability of an

Table 6.1: Package Descriptor

| Notation | Description |
|-------------------|-------------------------------|
| (x_i, y_i, z_i) | Source or Initial Location |
| (x_f, y_f, z_f) | Destination or Final Location |
| m | Mass |
| pr | Priority |
| h | Handling requirements |
| TW | time window |

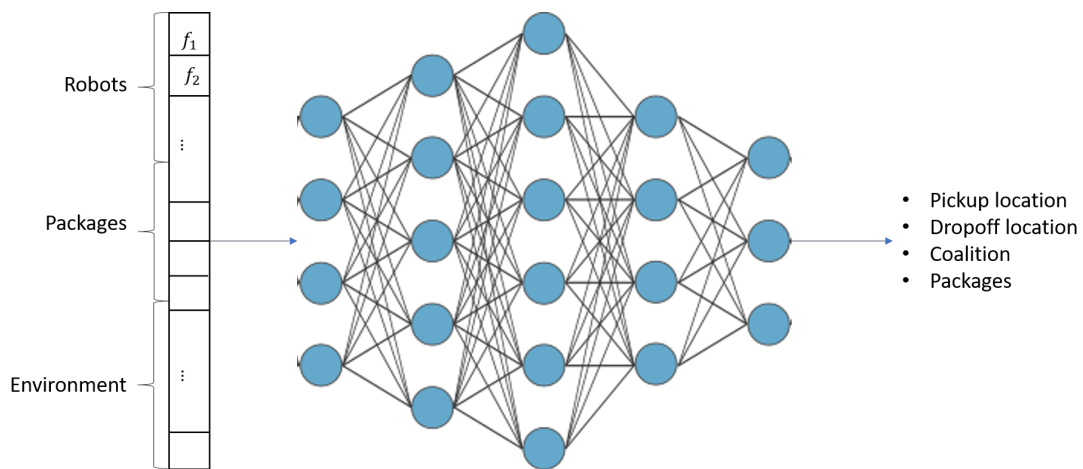


Figure 6.3: A feature based ANN formulation.

agent or coalition completing a task. Features that can be included in the descriptor vector can include: visibility, wind, rain, ice, snow, traffic, terrain and others.

6.3.4 ANN Formulation

Multiple formulation options can be considered when designing ANN. The first option we consider is shown in Figure 6.3 where the agent, package and environmental descriptors are inputted to a network that determines the pickup and dropoff locations of a subset of packages and assigns a coalition of agents to complete the task. This formulation allows transfer of packages at intermediary locations between coalitions. This transfer could result to less costly solutions, reduces the search space by allowing an incremental scheduling of package deliveries and can tolerate agent failures.

Another formulation represents PDP-CF as a graph. Given a graph describing the PDP scenario (Figure 4.2), the network must prune connections from the graph to obtain a cost effective solution that schedules delivery tasks. Re-

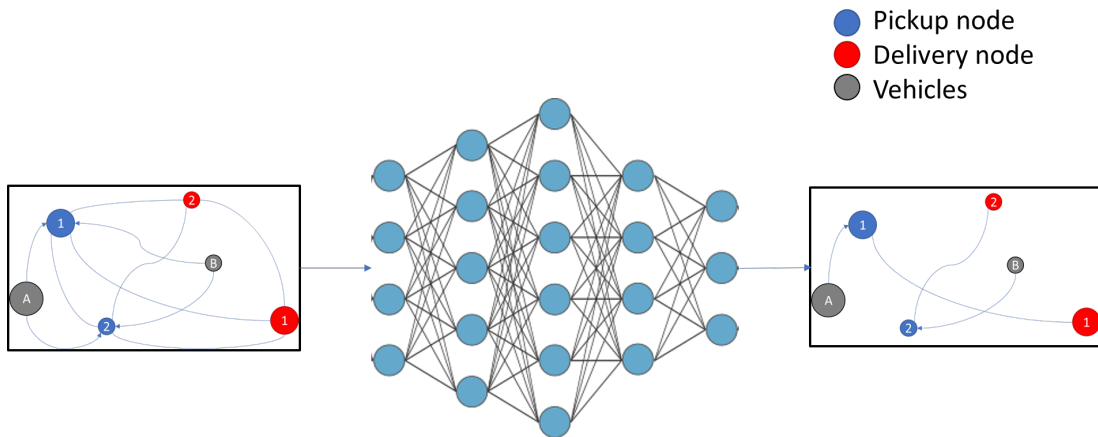


Figure 6.4: A graph based ANN formulation.

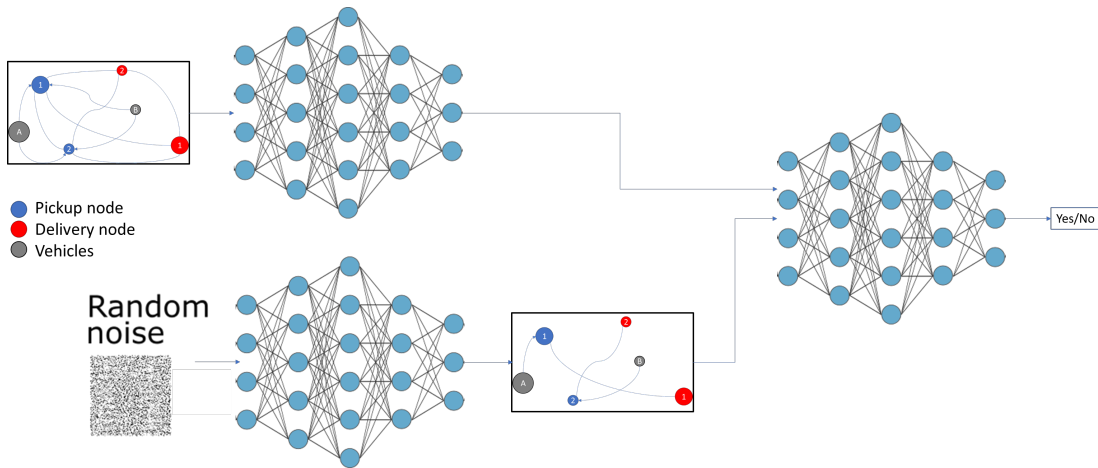


Figure 6.5: A GAN formulation.

maining connections represent which vehicles will visit pickup and delivery nodes associated with specific packages. This formulation is more complex since ANN models that can handle non-Euclidean data must be formulated. Multiple such formulations exist in the literature and can be investigated.

Finally, the last possible formulation adopts a generative adversarial network (GAN) to generate possible solutions to a given PDP scenario. Once a solution is generated, another network determines whether this solution is a valid schedule, i.e. it satisfies the constraints. Many such solutions would be generated and the least costly option would be adopted. However, GANs are challenging to train since they suffer from convergence issues. Nevertheless, this approach may lead to a more efficient search of the very large space and is worth investigating.

6.4 Database Description

6.4.1 Descriptor Representation

Real-world environments present autonomous package delivery systems with many challenges from environmental stochasticity and diversity to agent uncertainty. Therefore, validating the algorithms proposed in this work on artificial data does not guarantee its effective deployment to real-world environments. To validate these algorithms on more realistic scenarios, a more realistic dataset should be built, with characteristics that more faithfully model real-world environments. We identified four main components that should be represented in the database: 1) agent descriptors, 2) package descriptors, 3) delivery routes characteristics, and 4) delivery schedules.

Agents

We consider two types of agents: robots (UGV and UAV) and non-robotic agents. Using Figure 6.1 to characterize and represent agents, we created a set of UGV and UAV based on existing models. Specifically, six UGV models (Pioneer LX, Pioneer 3, Seekur Jr, AmigoBot, Laser PowerBot, and KUKA KMR iiwa) were used as the base models, from which 95 variants were created by modifying hardware specs and accessories. The robot data sheets were used to build the feature vectors for each robot. Multiple features were included based on the six categories in Figure 6.1.

Similarly, five UAV models (Lockheed Martin, DJI S900 Ready To Fly, AscTec Pelican, AGRASMG-1, and FREEFLY ALTA 8) were used to create a set of 20 heterogeneous agents with various capabilities. The non-robotic agents such as cloud resources and Internet-of-Things devices (weather, traffic ...) will be integrated into the set of agents by creating feature vectors based on their data sheets.

Packages

Various package delivery companies such as DHL, FedEx and UPS [372] have made publicly available various datasets about their package delivery activities. Specifically, they have provided information about packages that go through their facilities including destination addresses, location type, and pickup/delivery time windows. Package characteristics such as type, weight and fragility were not provided. We used these datasets to create our set of packages but made some modifications. Instead of using the warehouses as source locations for all package instances, we changed some to other residential or business locations. We added weight, fragility (handling), and priority as features describing the packages. In total, approximately 122,000 packages were compiled.

Delivery Routes

We considered two APIs to retrieve delivery routes and their characteristics: 1) Google Maps, and 2) open street maps [373]. We adopted the open street maps API due to its user-friendliness and its popularity in vehicle route planning literature. Given source and destination locations (their longitudes and latitudes), we retrieved the intermediary points of the shortest route, as determined by open street maps. The route is a valid navigation path along existing roads in cities and urban environments. The distance is computed using the haversine formula [374] in (6.1) (R is the radius of the sphere or Earth in this case). Given the longitudes (ϕ_1 and ϕ_2) and latitudes (λ_1 and λ_2) of two points on the sphere, this formula computes the circle-distance between these two points. To approximate the distance of a route, we sum the haversine values of consecutive intermediary points along the route.

$$d = 2R \cdot \arcsin \left(\sqrt{\sin^2 \left(\frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right) \quad (6.1)$$

Some features are missing from this database but will be included in future versions of this database. Specifically, we did not retrieve any terrain characteristics or weather conditions affecting roads (such as visibility or icy roads). The considered APIs did not allow 3D route planning either; other APIs will be considered to obtain valid routes for UAV.

So far, we have collected approximately 3400 routes in the city of Atlanta, Georgia (USA) by selecting addresses from the DHL dataset []. These addresses were randomly combined in pairs and routes were retrieved for these pairs.

Delivery Schedules

In a supervised learning framework, we generate delivery schedules for PDP scenarios using MIP, GA and QGA solvers. While the solutions may not be optimal, they are generally good enough. For small scale problems, brute force search could also be adopted to find the optimal solution if it exists and measure how far from the optimal is the solution of other solvers. Given the schedules, we can train ANN in a supervised framework.

It is possible to train ANN for PDP in an RL framework by developing a game-like simulation environment agents form coalitions and complete delivery tasks in this environment. Rewards would be awarded to agents based on completing the task which would allow ANN to learn which agents work well in a coalition and for what tasks. However, a survey of existing simulators should be performed to determine the most suitable simulator, if one exists. Otherwise, a simulator should be designed and implemented for this purpose.

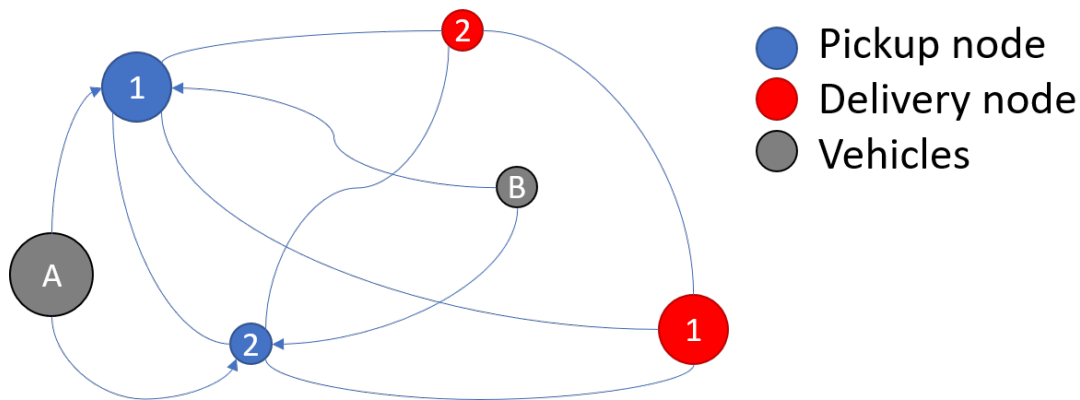


Figure 6.6: An image of the graphical representation of a PDP scenario.

6.4.2 Graphical Representation

A labeled corpus must be compiled to train ANN in a supervised framework. To do so, we can rely on the solvers in Chapter 5 to generate acceptable scheduling solutions to PDP and PDP-CF.

First, we must convert a PDP scenario to its graph theoretical representation and save the graphs as images, as shown in Figure 6.6. Pickup nodes, delivery nodes, and vehicle nodes are plotted in the Cartesian coordinate system. Pickup and delivery nodes represent the packages' source and destination locations; the size of the node represents the demand of packages. To simplify the problem, we bin the demands into three categories: small, medium and large. The vehicle nodes represent the start location of vehicles and the node sizes represent the capacity of vehicles which are also quantized into three bins (small, medium, large).

Arcs between the various nodes are plotted and represent the cost of going from one node to another. The directed graph is not fully connected: connections that are not allowed are eliminated. For example, connections from delivery nodes to pickup nodes corresponding to the same package are not allowed since a package cannot be delivered before it has been picked up.

Then, solutions to the various scenarios are generated using GA, QGA and MIP solvers. Since these solvers generally converge to sub-optimal solutions, we generate multiple acceptable solutions per PDP scenario. We also generate solutions that are not acceptable (i.e. they violate constraints) and label them as incorrect. Solutions are represented by the same graph with the arcs traversed by the vehicles plotted only. A sample solution is shown in Figure 6.7.

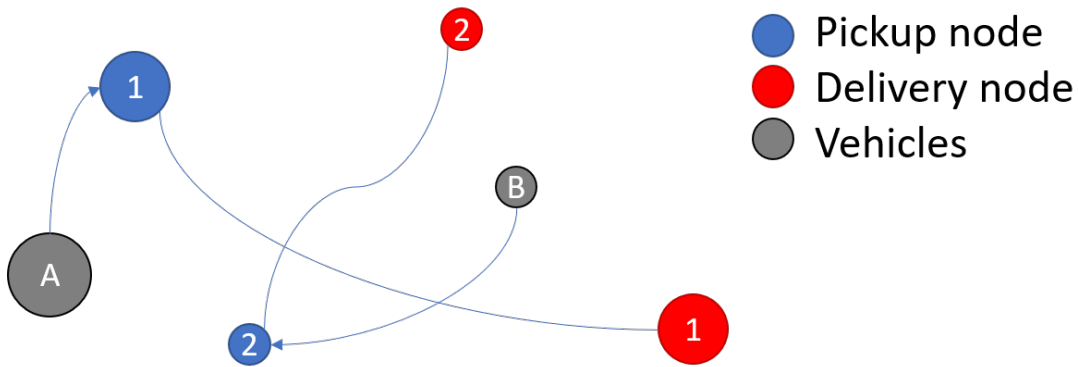


Figure 6.7: An acceptable solution to a PDP scenario.

6.5 Conclusion

In this chapter, ANN models were proposed to determine a suitable package delivery schedule. The characteristics of training data was also discussed. The next step for this work is to compile a large enough dataset and train the proposed ANN models before comparing them to the solvers in Chapter 5. Future work will also investigate training ANN in an RL framework to reduce the burden of compiling labeled data. Transfer learning can also be investigated to reduce the learning overhead before deploying a model by leveraging trained models from similar domains. Incorporating environmental descriptors can further improve performance, especially in outdoor package delivery problems by modeling terrain characteristics, weather conditions and others to form the most suitable coalition for a given environmental state and delivery task. Other types of ANN, such as competitive networks and generative adversarial networks, could be investigated to achieve better performance.

Chapter 7

ANN Training Algorithm: A Non-iterative Approach

RNN are a type of ANN that have been successfully applied to many problems in artificial intelligence. However, they are expensive to train since the number of learned weights grows exponentially with the number of hidden neurons. Non-iterative training algorithms have been proposed to reduce the training time, mainly on feedforward ANN. In this chapter¹, the application of non-iterative randomized training algorithms to various RNN architectures, including Elman RNN, fully connected RNN, and Long Short-Term Memory (LSTM), are investigated. The mathematical formulation and theoretical computational complexity of the proposed algorithms are presented. Finally, their performance is empirically compared to other iterative RNN training algorithms on time series prediction and sequential decision-making problems. Non-iteratively-trained RNN architectures showed promising results as significant training speedup of up to 99%, and improved repeatability were achieved compared to backpropagation-trained RNN. Although the decrease in prediction accuracy was found to be statistically significant based on Friedman and ANOVA testing, some applications like real-time embedded systems can tolerate and make use of that. The rest of this chapter is organized as follows. Section 7.1 motivates the proposed algorithms. Section 7.2 summarizes related work. Section 7.3 presents the proposed training algorithms. Section 7.4 presents a theoretical analysis of the proposed algorithms before discussing the simulation results in section 7.5. Finally, section 7.6 concludes with some final remarks.

¹This chapter appears in **Rizk, Y.**, and Awad, M., “On Extreme Learning Machines in Sequential and Time Series Prediction: A Non-Iterative and Approximate Training Algorithm for Recurrent Neural Networks,” *Elsevier Neurocomputing*, 2018 (in press).

7.1 Introduction

Sequential decision-making and time series prediction problems are an integral part of smart cities with applications in smart electric grids, intelligent transportation systems, robotic systems, Internet of things and others. These applications generally require computationally efficient algorithms that are difficult to deploy on energy aware and computationally limited platforms.

ANN, and RNN specifically, have been successfully applied to many problems in artificial intelligence and machine learning including time series prediction and sequential decision-making problems. While they are the most powerful neural networks and have exceeded human performance on some applications [375], RNN training, which mainly relies on iterative optimization, is computationally expensive and faces difficulty in resource-challenged applications. Thus, non-iteratively trained ANN such as random weight neural networks (RWNN) [376], random vector functional link (RVFL) [377], random activation weight network (RAWN) [378], and ELM [379] have been investigated to reduce training costs.

In this paper, we investigate training various RNN architectures non-iteratively, to reduce the computational complexity of RNN training. Specifically, we apply a non-iterative randomized offline training algorithm based on Moore-Penrose pseudoinverse, which was proposed by Schmidt et al. in 1992 [376], Pao et al. in 1994 [377], Te Braake and Van Straten in 1995 [378] and Huang et al. in 2004 [379], on a variety of RNN architectures. The following RNN architectures are considered in this work: Elman [380], Jordan [381], fully connected [382], non-linear auto regressive moving average with exogenous input (NARMAX) [383], and LSTM [384]. Networks with and without direct links (DL), connections between the input and output layer neurons [385], are also considered.

Since learning the weights of the recurrent connections is not a linear problem, we propose two training algorithms to address this issue: the first, ELM-rand, randomly assigns values to the recurrent connection weights, while the second, ELM-lin, linearizes the non-linear activation functions and learns the recurrent connection weights using one of the non-iterative training algorithms mentioned above. Table 7.1 summarizes the various network architecture/training algorithm combinations investigated in this work (denoted by “X”) and those presented in the literature. In addition, “N” denotes the absence of DL while “Y” denotes their presence.

The proposed algorithms are validated on the double inverted pendulum problem modeled as a POMDP and 14 publicly available time series prediction databases. Compared to RNN trained using the iterative backpropagation algorithm, non-iteratively trained RNN achieved faster training time and better repeatability while incurring an increase in prediction error.

The contributions of this work include:

1. proposing two non-iterative training algorithms for multiple RNN architec-

Table 7.1: ANN architecture and training algorithm combinations proposed in this work (denoted by X)

| Network | DL | Iterative training | Non-iterative training |
|---------------------|----|--------------------|------------------------|
| Elman | N | [380] | X |
| Elman | Y | - | X |
| Jordan | N | [381] | [386] |
| Jordan | Y | [387] | X |
| NARX/NARMAX | N | [383] | X |
| NARX/NARMAX | Y | - | X |
| Fully connected RNN | N | [382] | X |
| Fully connected RNN | Y | - | X |
| LSTM | N | [384] | X |
| LSTM | Y | - | X |
| Feedforward | N | [388] | [376, 378, 379] |
| Feedforward | Y | [385] | [377] |

tures including fully connected RNN and LSTM,

2. comparing the theoretical computational complexity of various RNN architectures when trained non-iteratively,
3. and empirically comparing the proposed algorithms to other algorithms in the literature on multiple publicly available datasets for time series regression and control problems.

7.2 Literature Review

In this section, we survey existing work on RNN architectures and non-iterative training algorithms for various applications including classification, regression and clustering of sequential and non-sequential data.

7.2.1 Recurrent Neural Networks

RNNs are the most powerful ANN and are considered general computers [375]; the network’s weights are its program, and changing them implies changing the program it runs. The main difference between RNN and ANN is the presence of recurrent connections that form cyclic graphs. Many RNN architectures and training algorithms have been proposed in the literature for various applications; we discuss a few in what follows.

Network Architectures

Various methods of introducing cycles in a graph yield different types of RNN. In this work, we consider the six architectures illustrated in Figure 7.1. Some more complex RNN architectures, which are not within the scope of this work, include bidirectional RNN which combine the output of two RNN, each processing the input sequence in a different direction [389], hierarchical RNN which feed the output of one RNN to another RNN [390], continuous time RNN [382], complex-valued RNN [391], and time-delayed RNN with reaction-diffusion terms [392,393].

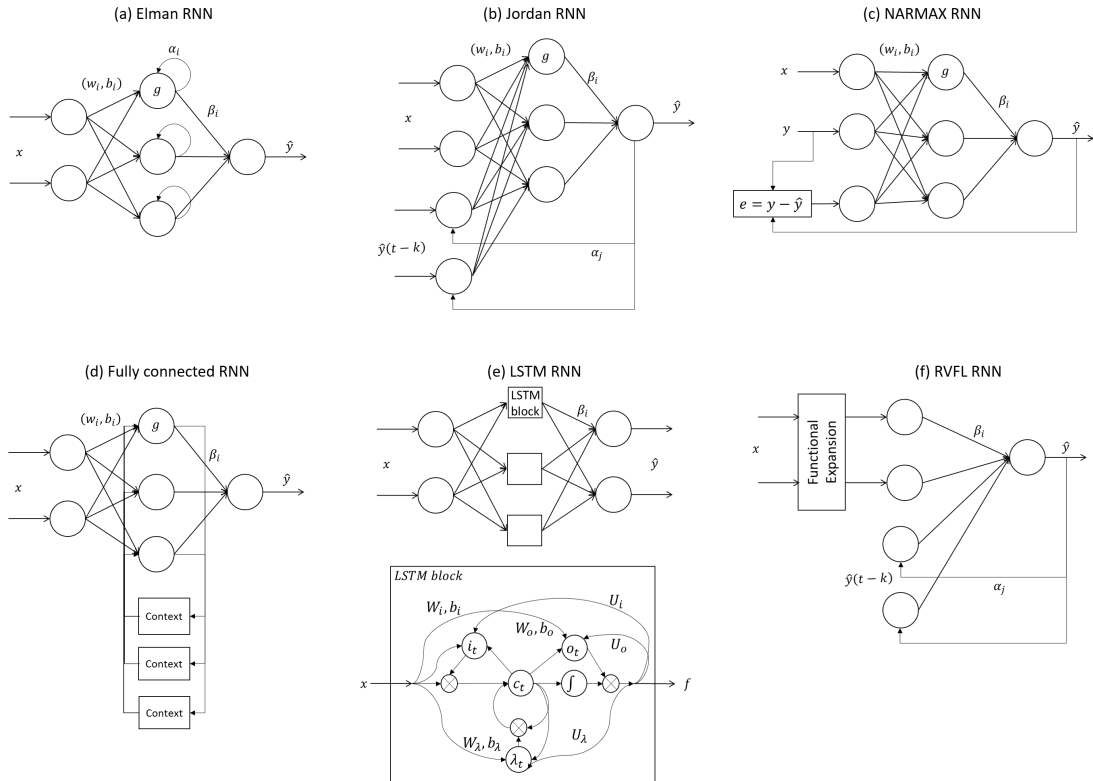


Figure 7.1: Different RNN Architectures

The simplest forms of RNN include Elman [380] and Jordan RNN [381], described by (7.1) and (7.2) respectively, which are single hidden layer networks with context neurons. Context neurons introduce recurrence into the network by feeding back signals in the network. Jordan networks provide context in terms of the predicted output. Although simpler to training using traditional backpropagation, Jordan RNN cannot represent an arbitrary dynamical system [394]. On the other hand, Elman networks provide context in terms of the internal state of the network which allows them to represent any Q^{th} order dynamical system, where Q is the number of context neurons [395]. However, these networks are

difficult to train using backpropagation as the network depth and size increases, due to the vanishing gradient problem [384].

$$\hat{y}(t) = \sum_{i=1}^M \beta_i f_i(t) \quad (7.1)$$

$$f_i(t) = g \left(w_i^T x(t) + \sum_{k=1}^Q \alpha_{ik} f_i(t-k) + b_i \right)$$

$$\hat{y}(t) = \sum_{i=1}^M \beta_i g \left(w_i^T x(t) + \sum_{k=1}^Q \alpha_{ik} \hat{y}(t-k) + b_i \right) \quad (7.2)$$

Recurrent NARMAX networks, described by (7.3) where $e(t) = y(t) - \hat{y}(t)$, have been proposed for non-linear time series prediction using ANN [383]. They were trained using BPTT but proven to be special cases of fully connected RNN architectures [383]. Determining the order of the output and error polynomials is generally based on a grid search and affects the number of input layer neurons, as shown in Figure 7.1(c). In a fully connected RNN architecture, which is described by (7.4), neurons are connected to all other neurons which include neurons in the same, previous and subsequent layers. A fully connected RNN is the most general RNN architecture.

$$\hat{y}(t) = \sum_{i=1}^M \beta_i g \left(\sum_{j=1}^F w_{ij} x_j + \sum_{j=F+1}^{F+Q} w_{ij} y(t-j+F) \right. \quad (7.3)$$

$$\left. + \sum_{j=F+Q+1}^{F+Q+R} w_{ij} e(t-j+F+Q) + b_i \right)$$

$$\hat{y}(t) = \sum_{i=1}^M \beta_i f_i(t) \quad (7.4)$$

$$f_i(t) = g \left(w_i^T x(t) + \sum_{k=1}^Q \sum_{l=1}^M \alpha_{ikl} f_l(t-k) + b_i \right)$$

LSTM networks were proposed to solve the vanishing gradient problem, present when training RNN with backpropagation [384]. LSTM cells replace neurons as the network's basic units, as shown in Figure 7.1(e). An LSTM cell consists of a typical linear neuron and multiplicative gates that feedback outputs and inputs to the neuron while controlling their effects on the linear unit through forgetting factors and weights. The output of the LSTM block is described by (7.5), where \circ represents the Hadamard product of two matrices.

$$\begin{aligned}
\hat{y}(t) &= \sum_{i=1}^M \beta_i f_i(t) \\
\lambda(t) &= g_\lambda (W_\lambda x(t) + U_\lambda f(t-1) + b_\lambda) \\
in(t) &= g_{in} (W_{in} x(t) + U_{in} f(t-1) + b_{in}) \\
o(t) &= g_o (W_o x(t) + U_o f(t-1) + b_o) \\
c(t) &= \lambda(t) \circ c(t-1) + in(t) \circ g_c (W_c x(t) + U_c f(t-1) + b_c) \\
f(t) &= o(t) \circ g_f (c(t))
\end{aligned} \tag{7.5}$$

Finally, converting RVFL ANN from a feedforward to a recurrent RVFL architecture, i.e. RNN with DL, simply requires us to pass the input through a preprocessing phase before applying it to any one of the RNN architectures. Figure 7.1(f) depicts a Jordan RNN architecture with DL, which is described by (7.6). However, similar illustrations can be derived for Elman, fully connected, NARMAX and LSTM networks.

$$\hat{y}(t) = \sum_{i=1}^M \beta_i \phi_i(t) + \sum_{k=1}^Q \alpha_k \hat{y}(t-k) \tag{7.6}$$

Training Algorithms

Many training algorithms have been proposed for RNN. Gradient descent based methods are iterative approaches and include backpropagation through time (BPTT) [396] and real-time recurrent learning [397]. Given a sequence of input-output pairs and initial weight values, BPTT unfolds the recurrence in the network through time to transform an RNN to a feedforward network trained using backpropagation. BPTT performs a forward and backward pass at each time step. However, this method is susceptible to local minima and suffers from the vanishing gradient problem when run on a fully connected RNN. Furthermore, BPTT can only be iteratively applied in batch mode. Therefore, an online alternative, real-time recurrent learning, was introduced which performs one forward pass only and computes RNN's derivatives with respect to its parameters without storing hidden states.

Other algorithms that optimally train RNN include a Hessian free optimization algorithm which reduces the computational complexity of the optimization algorithm by avoiding the expensive computation of the Hessian matrix using the truncated Newton method [398]. [399] adopted the Levenberg-Marquardt optimization algorithm to learn RNN weights. Bayesian filters including the extended Kalman filters [400] have been used to compute RNN weights and improve RNN training convergence. On the other hand, search algorithms have also been

adopted including particle swarm optimization [401], genetic algorithm [402], artificial immune system and ant colony optimization [401].

Even though these iterative training algorithms have performed well in many applications, they remain computationally expensive and require the manual tuning of many hyper-parameters. A non-iteratively trained Jordan RNN was proposed by [386] for electricity load forecasting. The non-iteratively trained network fed back the output layer values to the input layer and was tested on only one-time series prediction problem. The approach was only validated on this simple RNN architecture which cannot learn more complex real-world problems. In our work, we attempt to generalize this non-iterative approach to more powerful RNN architectures.

Reservoir Computing

Reservoir computing [403] is a type of RNN with random connections used to map temporal data to higher dimensions. The readout is then trained using some machine learning algorithm which is generally iterative. One example is the echo state machine, a sparsely connected RNN with randomly assigned weights [404]. Liquid state machines are spiking RNNs with random connections that map time-varying inputs to spatio-temporal neuron activation patterns [405]. Instead, we investigate the effect of random connections on densely connected RNN with deterministic neurons using a non-iterative training algorithm.

7.2.2 Non-Iteratively Trained Artificial Neural Networks

Many researchers have attempted to develop computationally less expensive algorithms to train ANN. Randomized training algorithms, surveyed in [406], are one approach to reduce training complexity, with some references developing non-iterative training algorithms which improved training time significantly since the connection weights are computed once. To that end, RWNN in 1992 [376], RVFL in 1994 [377], RAWN in 1995 [378] and ELM in 2004 [379] were proposed to train single hidden layer feedforward networks by randomly assigning input weights and computing output weights using the least squares method.

Random Weight Neural Networks

To reduce the training time of feedforward ANN, Schmidt et al. [376] applied non-iterative training to the single hidden layer feedforward ANN. This network architecture, shown in Figure 7.2 (without DL), appended the hidden layer with a bias neuron to produce the input-output function in (7.7). The predicted output of the network \hat{y} is a non-linear function of the input due to the presence of $g(\cdot)$ which represents the activation function. Hereafter, Schmidt et al.'s approach is referred to as RWNN.

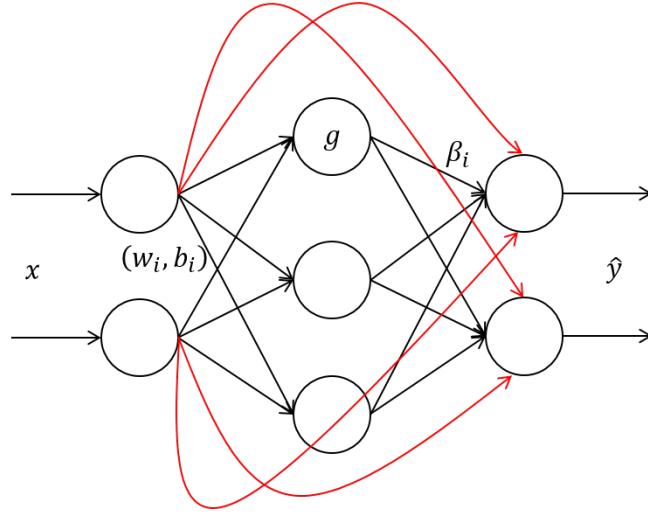


Figure 7.2: Single hidden layer feedforward network trained non-iteratively with randomized input weights

The supervised training of ANN involves minimizing the square of the error between the expected output y and the predicted output \hat{y} : $\min e^2 = (y - \hat{y})^2$. Since the objective function is quadratic, least squares can be used to solve for these variables in one step when the minimization problem is linear in the optimization variables.

The algorithm proposed in [376] randomly assigns input weights w_i from a uniform distribution which leads to a linear optimization problem, as shown in (7.7). Therefore, the output weights β may be analytically computed. Considering the single hidden layer feedforward ANN in [376], the output matrix H is computed using (7.8) where an element of H is $h_{ji} = g(w_i^T x_j)$, $i = 1, \dots, M$, $j = 1, \dots, N$, and the $(M+1)^{th}$ column contains 1 that multiplies the bias neuron. Note that w_i has dimensionality MF , where F is the number of features in the input vector. The output weights form the parameter vector $\theta = \beta$ with dimension $M+1$ and are learned using the generalized Moore-Penrose pseudoinverse, $\theta = (H^T H)^{-1} H^T y$, where y is the target matrix or desired output in a supervised learning paradigm.

$$\hat{y} = \sum_{i=1}^M \beta_i g(w_i^T x) + \beta_{M+1} \quad (7.7)$$

$$H = \begin{bmatrix} g(w_1^T x_1) & \dots & g(w_M^T x_1) & 1 \\ \vdots & \ddots & \vdots & \vdots \\ g(w_1^T x_N) & \dots & g(w_M^T x_N) & 1 \end{bmatrix}_{N \times (M+1)} \quad (7.8)$$

Random Vector Functional Link Networks

RVFL networks added direct connections from the input to the output layer, as shown in Figure 7.2 and described by (7.9). It randomly assigned weights to the input weights and biases from a uniform distribution and learned the output weights, whose dimension is $M + F$, using a non-iterative algorithm to reduce training complexity [377].

$$\hat{y}(t) = \sum_{i=1}^M \beta_i g(w_i^T x(t) + b_i) + \sum_{i=M+1}^{M+F} \beta_i x_{i-M}(t) \quad (7.9)$$

RVFL, which is based on a functional link ANN architecture results in the matrix shown in (7.10). The output weights are computed as in the previous subsection. Therefore, the main differences between RVFL and RWNN are adding a bias to each hidden neuron and modeling a linear dependence between the input and output through direct input-output connections.

$$H = \begin{bmatrix} x_1^T & g(w_1^T x_1 + b_1) & \dots & g(w_M^T x_1 + b_M) \\ \vdots & \vdots & \ddots & \vdots \\ x_N^T & g(w_1^T x_N + b_1) & \dots & g(w_M^T x_N + b_M) \end{bmatrix}_{N \times (M+F)} \quad (7.10)$$

These direct connections have been empirically shown to improve the performance of such networks over networks without these connections. For example, [407] showed the significance of the direct input-output connections when applying an ensemble RVFL learning paradigm, which outperformed ELM networks, to the crude oil price prediction problem. [408] reached a similar conclusion with regard to the effectiveness of DL when they formulated an ensemble method based on randomized networks including RVFL and ELM with successive projection for regression. A comprehensive analysis of RVFL was performed in [409], discussing parameter sensitivity, activation functions and other effects on 121 UCI classification data sets. They demonstrated RVFL's superior performance compared to ELM and empirically illustrated the importance of DL between the input and output, revealing a superposition of linear and non-linear dependence of the output on the input which is not captured in ELM's network architecture. This was also evident [410] who applied RVFL to electricity load prediction.

Random Activation Weight Networks

RAWN, introduced by [378], modified RWNN's training algorithm by randomly assigning the activation or input weights of the network in Figure 7.2 (without the DL), describing the input-output relationship in (7.11), from a normal distribution and performing regularization to ensure that H , described by (7.12), was not singular. The hidden neurons have biases, and the output vector β has

dimension M . As a result, this algorithm was shown to be suitable for control applications and dynamic systems.

$$\hat{y} = \sum_{i=1}^M \beta_i g(w_i^T x + b_i) \quad (7.11)$$

$$H = \begin{bmatrix} g(w_1^T x_1 + b_1) & \dots & g(w_M^T x_1 + b_M) \\ \vdots & \ddots & \vdots \\ g(w_1^T x_N + b_1) & \dots & g(w_M^T x_N + b_M) \end{bmatrix}_{N \times M} \quad (7.12)$$

Extreme Learning Machines

Huang et al. [379] applied non-iterative training to the single hidden layer feed-forward ANN shown in Figure 7.2 (without the red arrows), naming the approach ELM. This single hidden layer architecture relates the input x and output of the network \hat{y} by the function in (7.11) where $g(\cdot)$ represents the activation function, and input weights w_i and biases b_i are randomly sampled from a uniform distribution, performing regularization to H in (7.12) when necessary.

The input weights are an MF -dimension vector and the biases an M -dimension vector. Unlike RWNN, ELM adds a bias term to each hidden neuron. However, [409,410] have shown through extensive experiments with numerous datasets that biases do not contribute much to performance. Compared to RAWN [378], ELM samples the input weights and biases from a uniform distribution as opposed to a normal distribution as in [378]. In addition, this differs from RVFL's formulation which is a function of two summations, a non-linear and a linear weighted sum of the input. These direct input-output connections, omitted in [379], have been empirically shown to improve performance as RVFL has outperformed ELM on various applications in [407–410].

ELM has been generalized to regression problems [411] and was adopted in many applications from modeling ship roll motion [412] to tanker motion dynamics [413]. Online variants [414–417] have also been developed for real-time prediction. ELM has been mainly applied to feedforward ANN, except for [386] which non-iteratively trained a simple Jordan RNN architecture. However, Jordan RNN cannot handle very complex problems. In this work, we investigate the effectiveness of training other more complex RNN architectures non-iteratively that can be applied to a wider range of regression problems.

Instead of performing batch learning, the output weights can be learned in an online fashion by converting the least squares minimization algorithm to the RLS algorithm using the matrix inversion lemma to obtain (7.13) [414]. This allows us to train ANN on real-world problems where data is acquired incrementally. Incorporating a forgetting factor enables the training algorithm to perform better in non-stationary environments and in the presence of slowly varying model parameters.

$$\begin{aligned}
\theta(t) &= \theta(t-1) + K(t) (y(t) - H^T \theta(t-1)) \\
K(t) &= P(t)H(t) = P(t-1)H(t) (I + H^T(t)P(t-1)H(t))^{-1} \\
P(t) &= (I - K(t)H^T(t)) P(t-1)
\end{aligned} \tag{7.13}$$

7.3 Non-Iterative Proximal Randomized Recurrent Neural Networks

In this work, we propose to train RNN using a non-iterative training algorithm. While [386] proposed to apply non-iterative learning to the Jordan RNN architecture, we extend it to other RNN architectures. Based on (7.1) and the new output matrix elements in (7.14), it is evident that the problem of learning the weights of the recurrent connections, α , in Elman RNN is not linear. Similarly, we can train a fully connected RNN non-iteratively based on (7.4) and obtain the output matrix elements in (7.15), which is also not linear in α . The output matrix elements of a NARMAX architecture are described by (7.16), and an LSTM architecture is described by (7.17). As for RVFL networks, a fully connected RNN with DL is described by (7.18), while a NARMAX and LSTM with DL are described by (7.19) and (7.20), respectively.

$$h_{ij}(t) = g \left(w_i^T x_j(t) + b_i + \sum_{k=1}^Q \alpha_{ik} f_i(t-k) \right) \tag{7.14}$$

$$h_{ij}(t) = g \left(w_i^T x_j(t) + b_i + \sum_{k=1}^Q \sum_{l=1}^M \alpha_{ilk} f_l(t-k) \right) \tag{7.15}$$

$$h_{ij}(t) = g \left(w_i^T [x_j^T(t), y_j^T(t), e_j^T(t)]^T + b_i \right) \tag{7.16}$$

$$h_{ij}(t) = o(t) \circ g_f(c(t)) \tag{7.17}$$

$$h_{ij}(t) = \begin{cases} g(w_i^T x_j + b_i) & \text{if } 1 \leq i \leq M \\ \hat{y}_j(t-i+M) & \text{if } M < i \leq M+Q \end{cases} \tag{7.18}$$

$$h_{ij}(t) = \begin{cases} [x_j^T(t), y_j^T(t), e_j^T(t)]^T & \text{if } 1 \leq i \leq F \\ g \left(w_i^T [x_j^T(t), y_j^T(t), e_j^T(t)]^T + b_i \right) & \text{if } F < i \leq M+F \end{cases} \tag{7.19}$$

$$h_{ij}(t) = \begin{cases} x_j^T(t) & \text{if } 1 \leq i \leq F \\ o(t) \circ g_f(c(t)) & \text{if } F < i \leq M+F \end{cases} \tag{7.20}$$

Table 7.2: Pseudocode for ELM-rand training algorithm

-
1. Randomly assign w_i, b_i, α_i
 2. Compute H
 3. Perform regularization on H to avoid singular matrices
 4. Compute β using the generalized Moore-Penrose pseudoinverse
-

7.3.1 Randomized Non-iterative RNN Training Algorithm

Our first attempt to resolve this non-linearity is to randomly assign the values of α , like the input weights and biases. Therefore, the non-iterative training algorithm can be applied as described in [376–379] with the output matrix elements as shown in (7.14) for the Elman RNN and (7.15) for the fully connected RNN architecture, in addition to LSTM and networks with DL, without changing the parameter vector θ , i.e. $\theta = \beta$. Hereafter, RNN architectures trained non-iteratively with randomly assigned α values are referred to as ELM-rand. Table 7.2 summarizes the ELM-rand algorithm for RNN architectures. Input weights, biases and recurrent connections are randomly assigned. The output matrix H corresponding to the network architecture is computed; regularization is performed if necessary. Finally, the output weights are computed using the generalized pseudo-inverse.

7.3.2 Linear Approximation of Non-iterative RNN Training Algorithm

Instead of randomly assigning the recurrent connection weights, these weights can be learned by approximating the activation function with a linear function. More specifically, we consider linearizing the sine and sigmoid activation functions as follows. The sine function is approximated by its Taylor series expansion, shown in (7.21), and results in the output matrix elements in (7.22), when using a first order polynomial. The sigmoid function is approximated by a piecewise linear function using the centered linear approximation method to obtain the function in (7.23) [418]. In this case, we are estimating both β and α , resulting in a parameter vector of $\theta = [\beta, \alpha]^T$. Hereafter, ELM-lin represents RNN architectures trained non-iteratively with a linearized activation function. Table 7.3 summarizes the ELM-lin algorithm for RNN architectures which randomly assigns the input weights and biases. The output matrix H corresponding to the network architecture is computed based on the linearized activation functions. Finally, the output and recurrent weights are computed using the generalized pseudo-inverse.

$$g(z) = \sin(z) = \sum_{j=0}^{\infty} \frac{(-1)^j z^{2j+1}}{(2j+1)!} \approx z \quad (7.21)$$

Table 7.3: Pseudocode for ELM-lin training algorithm

-
1. Randomly assign w_i, b_i
 2. Compute H with a linearized activation function
 3. Perform regularization on H to avoid singular matrices
 4. Compute $\theta = [\beta, \alpha]^T$ using the generalized Moore-Penrose pseudoinverse
-

$$h_{ij}(t) = w_i x_j(t) + b_i + \sum_{k=1}^Q \sum_{l=1}^M \alpha_{ilk} f_l(t - k) \quad (7.22)$$

$$g(z) = \frac{1}{1 + e^{-z}} \approx \frac{1}{2} \left(1 + \frac{z}{2} \right) \quad (7.23)$$

7.3.3 Kaczmarz's Projection Algorithm

To further reduce the computational complexity of RNN-NIPR algorithms, we propose adopting Kaczmarz's approximation of RLS, shown in (7.24) where $\gamma > 0$ and $0 < \xi < 2$ are tunable relaxation parameters, to learn the output weights in an online fashion. Online learning allows us to process data that is obtained in real-time or pseudo real-time, i.e. it is not available offline to train a learning algorithm. First proposed by Kaczmarz in 1937, this algorithm finds the solution of a system of linear equations by iteratively projecting the current estimate onto the equations' solution space [419]. This method is also known as algebraic reconstruction technique, and it is considered a special case of the projection onto convex sets method [420].

While its convergence is slower than RLS, it is computationally more efficient because it avoids updating the covariance matrix which can become computationally expensive as the number of parameters to estimate increases. The computational complexity of this approximation is linear in the number of weights that are learned, compared to quadratic for RLS. Since the number of weights to learn in a fully connected RNN architecture grows exponentially with the number of hidden neurons, training such a network for online applications becomes very expensive when using RLS. Therefore, adopting Kaczmarz's approximation could lead to significant computational savings for energy aware applications.

$$\theta(t) = \theta(t - 1) + \frac{\gamma H(t)}{\xi + H^T(t)H(t)} (y(t) - H^T \theta(t - 1)) \quad (7.24)$$

7.4 Theoretical Analysis

In this section, we perform a theoretical comparison of the various non-iteratively-trained RNN architectures.

7.4.1 Computational Complexity

The computational complexity of training a network and predicting an output given an input vector depends on the network architecture and training algorithm. Therefore, to analyze the computational complexity of various RNN architectures when trained non-iteratively, we first derive the complexity of the non-iterative training algorithm for a generic feedforward single hidden layer network then consider the output matrix dimensions of the different RNNs.

The non-iterative algorithm consists of three main steps: (1) randomly assigning values to the input weights and biases, (2) computing the output matrix H and (3) solving for the parameter vector θ which contains the output weights β , using the Moore-Penrose pseudoinverse. We consider a network with F input neurons, M hidden neurons and 1 output neuron trained on N input vectors. Assigning random values requires a constant amount of time and the memory requirements are linear in the number of assigned values, equal to the sum of input layer weights and biases. The output matrix H is an $N \times M$ matrix and consists of multiplying F -dimensional input vectors by F -dimensional input weights and applying an activation function with a constant number of floating-point computations c . Therefore, computing the H matrix requires a total of $NM(F + c)$ computations. Finally, calculating the pseudoinverse of H and multiplying the result by the target output requires $O(NM)$ operations when $M \leq N$ and $O(NM + N^2M)$ otherwise. Since RNN are trained on a large number of data points compared to the number of hidden neurons, computing θ requires $O(NM)$ operations. On the other hand, testing requires computing the H matrix and multiplying it by β , equivalent $O(M(F + c) + M)$ computations for one test point, to find \hat{y} . Table 7.4 summarizes the time and space complexity of the non-iterative training algorithm. The space complexity indicates how many floating-point numbers are saved in memory.

Table 7.4: Computational complexity of ELM

| | Time complexity | Space complexity |
|------------------------------------|---------------------|-------------------|
| Randomly assign weights and biases | $O(1)$ | $O(FM + M)$ |
| Compute H | $O(NM(F + c))$ | $O(NM)$ |
| Compute θ | $O(NM)$ | $O(M)$ |
| Training Total | $O(NM(F + c) + NM)$ | $O(FM + 2M + NM)$ |

Focusing on the RNN architectures discussed in section 7.3, the number of input layer weights, biases, hidden layer weights and output layer weights will vary based on the architecture under consideration. Table 7.5 summarizes the number of weights for each layer, based on the architecture. When training Elman, fully connected and LSTM RNN using ELM-rand, the hidden recurrent

weights will be randomly assigned and will not affect the size of the H matrix or the number of parameters to estimate. However, for Elman and fully connected RNN trained using ELM-lin, the parameter vector becomes $\theta = [\beta, \alpha]^T$, and the size of the H matrix is also affected. The number of input weights is based on the dimensionality of the feature vector (F) and the number of hidden layer neurons (M). Jordan and NARMAX networks increase the dimensionality of the input vector by feeding back the predicted output which results in a higher number of input weights, as shown in Table 7.5. The LSTM cell contains four gates that contain input weights, recurrent connections and biases, as shown in Figure 7.1, leading to $4MF$ input weights per cell. The number of biases is equal to the number of hidden layer neurons (M), except in the LSTM network where the LSTM cell contains four bias weights. The number of recurrent connections in the hidden layer depends on the number of hidden layers and number of time steps fed back into the neuron. For example, it is possible to feedback only the previous value of the neuron output, which means $Q = 1$. The number of output weights depends on the number of direct connections from the hidden and input layers to the output neurons. Input to output layer connections increase the dimensionality of the parameter vector by the dimensionality of the input vector, i.e. the dimensionality of θ becomes $M + F$.

Finally, the computational complexity of training a network involves combining the number of weights in each architecture to the number of computations per weight for a given training algorithm. For example, a fully connected RNN has M output weights, MQ recurrent weights, NM input weights and M bias weights. Therefore, the ELM-rand algorithm requires a total of $O(NM(F + Q^2 + c) + NM)$ computations and $O(FM + M + MQ + NM + M)$ of space on this network. On the other hand, training an Elman RNN using ELM-lin would require $O(N(MQ + M))$ computations to find β , $O(NM(F + c))$ computations to find H and $O(FM + M + NM + M + MQ)$ of memory.

7.4.2 Model Transformations

In this section, we investigate the relationship between the various RNN architectures. [383] proved that a non-linear autoregressive moving average(NARMA) network is a special case of the fully connected RNN architecture; extending it to NARMAX network simply involves adding the exogenous input to both input layers. Moreover, α_{ikl} in (7.4) is set to $-\alpha_{ik}\beta_i$ and w_{ij} in (7.4) is set to $w_{ij} + \alpha_{ij}$, $j \leq Q$ or w_{ij} , $j \leq F$ for $F > Q$. Jordan RNN are also a special case of fully connected RNN and can be created by setting α_{ikl} in (7.4) to $\alpha_{ik}\beta_i$ from (7.2). Fully connected RNN can be converted to Elman RNN by setting $\alpha_{ikl} = 0$ if $k \neq 1$ and $i \neq l$. Finally, Elman and Jordan RNN are equivalent when α_{ik} in (7.1) is set to $\alpha_{ik}\beta_j$ for $i = j$ where $j = [1, M]$. The model equivalences could help to improve the performance of non-iteratively-trained RNN. For example, one could train a NARMAX network, then convert it to a fully connected RNN

Table 7.5: Number of weights per layer for ANN architectures

| Network | DL | Input Weights: w | Biases: b | Recurrent Weights: α | Output Weights: β |
|-----------------|----|-----------------------|----------------|--------------------------------|----------------------------|
| Feedforward | N | FM | M | 0 | M |
| Feedforward | Y | FM | M | 0 | $M + F$ |
| Elman | N | FM | M | MQ | M |
| Elman | Y | FM | M | MQ | $M + F$ |
| Jordan | N | $(F + Q)M$ | M | 0 | M |
| Jordan | Y | $(F + Q)M$ | M | 0 | $M + F$ |
| Fully connected | N | FM | M | QM^2 | M |
| Fully connected | Y | FM | M | QM^2 | $M + F$ |
| NARMAX | N | $(F+Q+R)M$ | M | 0 | M |
| NARMAX | Y | $(F+Q+R)M$ | M | 0 | $M + F$ |
| LSTM | N | $4MF$ | $4M$ | $4M$ | M |
| LSTM | Y | $4MF$ | $4M$ | $4M$ | $M + F$ |

based on the weight relationships mentioned above. This would result in faster training. Similarly, this methodology could be applied to other architectures as well. However, the empirical performance of these theoretical transformations on real-world problems should be investigated to confirm their merit.

7.5 Empirical Validation

In this section, we present the simulation results of applying non-iteratively-trained RNN on multiple benchmarks compared to other ANN algorithms in the literature, based on prediction accuracy and training time. Before presenting the results, a brief description of the benchmarks and experimental setup is presented.

7.5.1 Experimental Setup

The experiments were run in Matlab R2016b on an Intel Xeon 64-bit processor with 12 cores, 2.0 GHz clock speed and 24 GB of RAM. Matlab's neural networks toolbox was used to train NARMAX-BPTT while the statistics and machine learning toolbox was used to train support vector regression (SVR). The SVR hyperparameters were obtained using Matlab's built-in automatic hyperparameter optimization which search for the optimal kernel, box constraint, epsilon value and other hyperparameters using Bayesian optimization. The ELM code was downloaded from [421], and all other algorithms were written in Matlab R2016b. Hereafter, ELM is referred to as feedforward network without DL trained

using ELM-rand, and RVFL is referred to as feedforward network with DL trained using ELM-rand or simply feedforward networks when no confusion is possible.

Multiple error measures were used to compare the performance of the various algorithms including mean square error (MSE), $E[(y - \hat{y})^2]$, root MSE (RMSE), $\sqrt{E[(y - \hat{y})^2]}$, normalized RMSE (nRMSE), $\frac{1}{y_{max} - y_{min}} \sqrt{E[(y - \hat{y})^2]}$, mean absolute error (MAE), $E[|y - \hat{y}|]$, and normalized MAE (nMAE), $\frac{1}{y_{max} - y_{min}} \sqrt{E[|y - \hat{y}|]}$. Furthermore, the training time and error measures of each algorithm were averaged over 5 independent runs to assess repeatability since the considered algorithms possess random characteristics. A repeatability analysis allows us to determine how sensitive the algorithms' performance is to the random number generator.

Time Series Prediction

The proposed methodology was validated on time series prediction problems. Table 7.6 summarizes the characteristics of the publicly available databases, sorted in ascending order of total database size. The Sante Fe Laser data is stationary [422], while Quebec Births is non-stationary [423]. The data in Sante Fe Laser was normalized to improve the condition number of the input matrix and avoid matrix inversion singularities. For the electricity load balance data set [424], the MT166 substation of the electricity load balance data set was selected for training and MT257 for testing, and the input vector consists of the day of the measurement after normalizing the data. AEMO data [425] reports the electricity load demand in Australia; the first 70% of the data was used for training and the remaining 30% for testing. The same was done for the Atmosfera Temperature, Atmosfera Humidity, Bebida, Consumo, Fortaleza, Ipi, and Lavras, obtained from [426]. Deep tesla is an autonomous driving database whose purpose is to predict the steering angle of the steering wheel given an image of the road ahead as seen from the front window shield of a car [427]. The first 1,500 frames of the database are used in our experiments since they correspond to the a single driving simulation. Japan population tracks the population of various Japanese regions [428]. The SP 500 database records the stock prices since 1950 [429].

POMDP-modeled Decision-Making

The inverted double pendulum on a sliding cart benchmark, also referred to as the double pole balancing problem, is a continuous sequential decision-making problem. The benchmark was run using the Simulink model of an inverted double pendulum on a sliding cart in Matlab R2015b, on an Intel Core i7, 64-bit machine with 2.4 GHz clock speed and 12 GB of RAM. The networks were trained on 1200 data points, then they were tested on 300 data points. Usually, the feature vector contained the cart's position and velocity in addition to the poles' angles and their velocities. The force applied to the cart was predicted. Excluding the cart

Table 7.6: Description of the time series benchmarks

| Database | Number of Training Instances | Number of Testing Instances | Exogenous Input Dimension | Output Mean | Output Standard Deviation | Output Min | Output Max |
|-------------------|------------------------------|-----------------------------|---------------------------|-------------------------|---------------------------|-------------------------|-------------------------|
| Fortaleza | 104 | 45 | 0 | 1445.0 | 496.95 | 468.0 | 2836.0 |
| Consumo | 108 | 46 | 0 | 120.90 | 27.24 | 75.39 | 232.0 |
| Bebida | 130 | 57 | 0 | 92.66 | 20.35 | 75.39 | 232.0 |
| Ipi | 130 | 57 | 0 | 103.60 | 18.28 | 65.81 | 148.3 |
| Atmosfera | 255 | 110 | 0 | 15.35 | 3.16 | 5.6 | 21.6 |
| Tempera- ture | | | | | | | |
| Atmosfera | 255 | 110 | 0 | 81.15 | 7.95 | 53.34 | 95.79 |
| Humidity | | | | | | | |
| Lavras | 268 | 116 | 0 | 127.60 | 122.06 | 0 | 718 |
| Santa Fe | 700 | 300 | 0 | 59.9 | 46.9 | 2 | 255 |
| Laser | | | | | | | |
| Quebec | 1000 | 4113 | 0 | 250.8 | 41.9 | -23.08 | 366.0 |
| Births | | | | | | | |
| Deep | 1050 | 450 | 240 | 0.8523 | 3.2355 | -6 | 6 |
| Tesla | | | | | | | |
| Japan | 1778 | 762 | 0 | 0.1399×10^7 | 0.1384×10^7 | 0.0123×10^7 | 1.3514×10^7 |
| popula- tion | | | | | | | |
| SP500 | 12053 | 5165 | 1 | 0.0899×10^{10} | 0.1533×10^{10} | 0.0001×10^{10} | 1.1456×10^{10} |
| AEMO | 12264 | 5256 | 0 | 7981.57 | 1190.91 | 5113.03 | 13787.85 |
| Electricity | 140257 | 140257 | 1 | 2.7×10^{14} | 2.6×10^{14} | 0 | 9.9×10^{14} |
| Load Bal- ance | | | | | | | |

and poles' velocities makes the problem partially observable and can be modeled using POMDP, which was considered in this work. The data was collected by saving the input and output values of a PID controller designed to control the plant. The controller was run for 10 seconds sampled at approximately $6.67ms$. Performance was measured by computing the MSE of the predicted force that should be applied to the cart and the actual force that was applied by a PID controller.

7.5.2 Non-iteratively-trained RNN Performance Analysis

A grid search was performed over the number of hidden neurons and types of activation functions (sigmoid and sine) to find the best architecture. Figure 7.3 plots the predicted output values for ELM-lin trained fully connected RNN on the AEMO database, which shows that the algorithms could learn the trend in the data relatively well. Tables 7.7-7.17 summarize average (avg) and standard

deviation (std) of the MSE, RMSE, MAE and training time of the various RNN architectures for both ELM-rand and ELM-lin on some of the databases, when sampling the random weights from a uniform distribution. Results of BPTT-trained RNN and SVR are also included. The hyperparameter column reports the number of hidden neurons for ANN algorithms and (kernel,box constraint) for SVR.

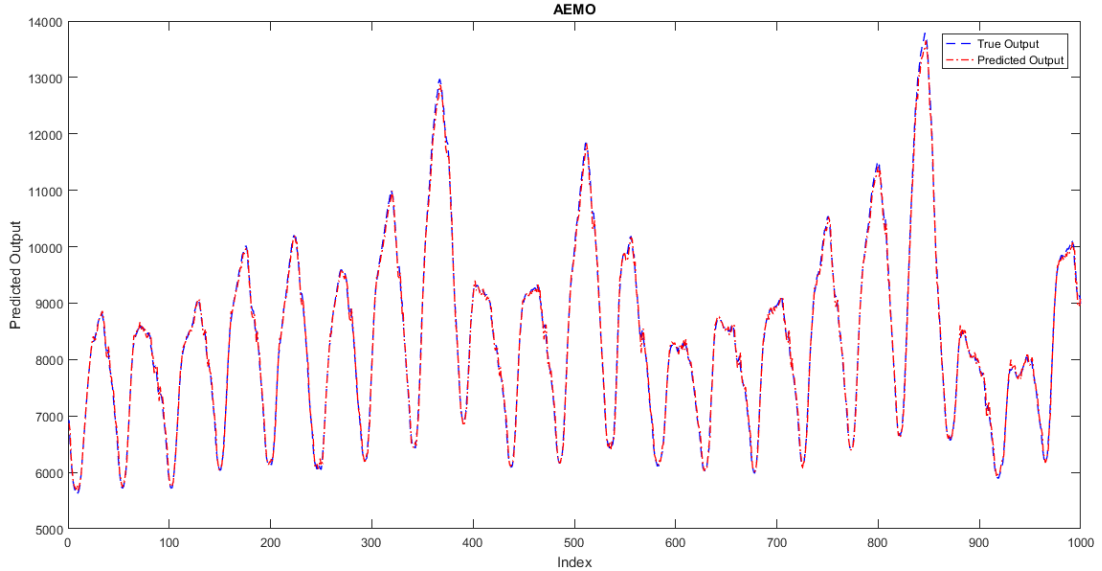


Figure 7.3: Predicted output for AEMO of a fully connected RNN (without DL) trained using ELM-lin

Sampling the random weights from a normal distribution was tested and the results showed either almost identical MSE to uniform distribution or significantly worse values. We also compared the RNN variants derived from the feedforward network in [376] and [379], where the difference was the bias terms. The results showed that the latter performed as well as the former on all RNN variants and databases, which is consistent with [410] who concluded that the bias term was irrelevant.

ELM-lin generally achieved lower MSE but required more time to train than ELM-rand. For example, for the Santa Fe database, ELM-lin had a lower MSE than ELM-rand on almost all RNN architectures. Fully connected RNN with DL was an exception. A similar pattern was observed for other databases as well with some exceptions.

Furthermore, non-iteratively-trained fully connected RNN performed better than non-iteratively-trained Elman RNN but was also more computationally expensive. For example, the inverted pendulum problem experienced a 56% improvement in MSE with the Elman RNN with ELM-lin and fully connected

Table 7.7: Performance on Atmosphere Humidity Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|------------------|-------------------------------|---------------------------------------|--------------------------------------|--------------------------------------|
| Elman | N | ELM-lin | 45 | 3.12E-3 \pm 6.98E-3 | 4.95E+1 \pm 4.03E-12 | 7.04E+0 \pm 2.86E-13 | 5.86E+0 \pm 2.54E-13 |
| Elman | N | ELM-rand | 10 | 3.1E-1 \pm 9.0E-2 | 5.01E+1 \pm 2.3E+0 | 7.08E+0 \pm 1.62E-1 | 5.85E+0 \pm 8.62E-2 |
| Elman | N | BPTT | 10 | 2.18E-1 \pm 2.70E-2 | 7.61E+1 \pm 1.86E+1 | 8.67E+0 \pm 1.06E+0 | 7.21E+0 \pm 7.90E-1 |
| Elman | Y | ELM-lin | 25 | 1.88E-1 \pm 6.3E-2 | 4.95E+1 \pm 4.00E-12 | 7.04E+0 \pm 2.84E-13 | 5.86E+0 \pm 2.49E-13 |
| Elman | Y | ELM-rand | 10 | 3.9E-1 \pm 7.8E-2 | 5.12E+1 \pm 6.87E+0 | 7.14E+0 \pm 4.62E-1 | 5.82E+0 \pm 1.55E-1 |
| Fully connected | N | ELM-lin | 45 | 1.50E-1 \pm 8.54E-3 | 4.95E+1 \pm 6.93E-12 | 7.04E+0 \pm 4.92E-13 | 6.0E+0 \pm 3.5E-1 |
| Fully connected | N | ELM-rand | 10 | 3.12E-3 \pm 6.98E-3 | 5.2E+1 \pm 5.6E+0 | 7.2E+0 \pm 3.7E-1 | 3.70E+1 \pm 6.90E+1 |
| Fully connected | Y | ELM-lin | 45 | 1.50E-1 \pm 8.54E-3 | 4.95E+1 \pm 7.23E-12 | 7.04E+0 \pm 5.13E-13 | 5.86E+0 \pm 4.37E-13 |
| Fully connected | Y | ELM-rand | 10 | 2.6E-1 \pm 7.0E-2 | 5.34E+1 \pm 1.17E+1 | 7.28E+0 \pm 7.55E-1 | 5.84E+0 \pm 1.87E-1 |
| Jordan | N | ELM-rand | 20 | 9.36E-3 \pm 8.54E-3 | 6.83E+1 \pm 1.41E+1 | 8.23E+0 \pm 8.20E-1 | 6.61E+0 \pm 4.68E-1 |
| Jordan | Y | ELM-rand | 50 | 9.36E-3 \pm 8.54E-3 | 6.35E+1 \pm 4.40E-2 | 7.97E+0 \pm 2.76E-3 | 6.39E+0 \pm 1.94E-3 |
| NARMAX | N | ELM-rand | 15 | 9.36E-3 \pm 8.54E-3 | 5.83E+1 \pm 1.06E+1 | 7.61E+0 \pm 6.64E-1 | 6.31E+0 \pm 5.83E-1 |
| NARMAX | Y | ELM-rand | 50 | 2.34E-1 \pm 9.06E-2 | 1.54E+2 \pm 1.37E-1 | 1.24E+1 \pm 5.54E-3 | 1.05E+1 \pm 5.48E-3 |
| NARX | N | BPTT | 15 | 1.99E-1 \pm 1.30E-2 | 9.71E+1 \pm 2.30E+1 | 9.79E+0 \pm 1.21E+0 | 7.73E+0 \pm 8.20E-1 |
| Feedforward [379] | N | ELM-rand | 20 | 3.9E-1 \pm 7.8E-2 | 6.18E+1 \pm 2.87E+0 | 7.86E+0 \pm 1.82E-1 | 6.6E+0 \pm 1.85E-1 |
| Feedforward [376] | N | ELM-rand | 15 | 6.25E-3 \pm 1.40E-2 | 6.28E+1 \pm 3.99E+0 | 7.92E+0 \pm 2.55E-1 | 6.66E+0 \pm 2.16E-1 |
| Feedforward [377] | Y | ELM-rand | 10 | 3.6E-1 \pm 1.1E-1 | 4.75E+1\pm6.69E-1 | 6.89E+0\pm4.8E-2 | 5.73E+0\pm4.4E-2 |
| SVR | - | - | Linear, 593 | 7.71E-2 \pm 0.0E+0 | 4.99E+1 \pm 0.0E+0 | 7.07E+0 \pm 0.0E+0 | 5.83E+0 \pm 0.0E+0 |

Table 7.8: Performance on Atmosphere Temperature Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|------------------|-------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| Elman | N | ELM-lin | 35 | 3.1E-1 \pm 5.9E-2 | 2.48E+0 \pm 3.50E-14 | 1.58E+0 \pm 1.11E-14 | 1.27E+0 \pm 1.28E-14 |
| Elman | N | ELM-rand | 10 | 1.56E-2 \pm 2.70E-2 | 9.17E+0 \pm 8.15E+0 | 2.81E+0 \pm 1.26E+0 | 1.42E+0 \pm 1.57E-1 |
| Elman | N | BPTT | 20 | 2.65E-1 \pm 7.26E-2 | 1.18E+1 \pm 6.93E+0 | 3.3E+0 \pm 1.06E+0 | 2.63E+0 \pm 9.75E-1 |
| Elman | Y | ELM-lin | 10 | 6.24E-3 \pm 8.54E-3 | 2.48E+0 \pm 4.74E-14 | 1.58E+0 \pm 1.51E-14 | 1.27E+0 \pm 1.82E-14 |
| Elman | Y | ELM-rand | 10 | 3.2E-1 \pm 0.0E+0 | 5.05E+0 \pm 2.03E+0 | 2.21E+0 \pm 4.63E-1 | 1.39E+0 \pm 8.28E-2 |
| Fully connected | N | ELM-lin | 25 | 5.62E-2 \pm 8.54E-3 | 2.48E+0 \pm 1.32E-13 | 1.58E+0 \pm 4.20E-14 | 1.27E+0 \pm 4.75E-14 |
| Fully connected | N | ELM-rand | 10 | 3.12E-3 \pm 6.98E-3 | 3.47E+0 \pm 2.49E+0 | 1.79E+0 \pm 5.73E-1 | 1.27E+0 \pm 1.20E-1 |
| Fully connected | Y | ELM-lin | 45 | 2.15E-1 \pm 2.56E-2 | 2.48E+0 \pm 1.35E-13 | 1.58E+0 \pm 4.30E-14 | 1.27E+0 \pm 6.41E-14 |
| Fully connected | Y | ELM-rand | 10 | 2.7E-1 \pm 6.8E-2 | 6.95E+1 \pm 1.49E+2 | 5.05E+0 \pm 7.42E+0 | 1.63E+0 \pm 7.49E-1 |
| Jordan | N | ELM-rand | 40 | 9.36E-3 \pm 8.54E-3 | 1.69E+1 \pm 3.32E+0 | 4.1E+0 \pm 4.11E-1 | 3.58E+0 \pm 3.77E-1 |
| Jordan | Y | ELM-rand | 10 | 6.24E-3 \pm 8.54E-3 | 4.37E+0 \pm 8.45E-2 | 2.09E+0 \pm 2.02E-2 | 1.78E+0 \pm 2.18E-2 |
| LSTM | N | ELM-rand | 10 | 3.74E-2 \pm 8.54E-3 | 1.27E+2 \pm 1.29E+2 | 9.9E+0 \pm 6.04E+0 | 9.53E+0 \pm 6.23E+0 |
| LSTM | Y | ELM-rand | 25 | 1.66E-1 \pm 1.59E-2 | 4.31E+0 \pm 7.33E-2 | 2.08E+0 \pm 1.77E-2 | 1.77E+0 \pm 1.83E-2 |
| NARMAX | N | ELM-rand | 15 | 1.25E-2 \pm 6.98E-3 | 3.28E+1 \pm 6.19E+0 | 5.71E+0 \pm 5.43E-1 | 5.23E+0 \pm 5.82E-1 |
| NARMAX | Y | ELM-rand | 10 | 1.78E-1 \pm 1.25E-2 | 8.19E+0 \pm 2.66E-2 | 2.86E+0 \pm 4.64E-3 | 2.54E+0 \pm 4.31E-3 |
| NARX | N | BPTT | 10 | 1.05E+0 \pm 1.74E+0 | 6.75E+0 \pm 9.66E-1 | 2.59E+0 \pm 1.83E-1 | 1.97E+0 \pm 6.01E-2 |
| Feedforward [379] | N | ELM-rand | 50 | 3.0E-1 \pm 1.2E-1 | 5.05E+0 \pm 1.27E+0 | 2.23E+0 \pm 2.76E-1 | 1.74E+0 \pm 1.61E-1 |
| Feedforward [376] | N | ELM-rand | 50 | 6.25E-3 \pm 1.40E-2 | 5.01E+0 \pm 8.81E-1 | 2.23E+0 \pm 1.99E-1 | 1.73E+0 \pm 1.30E-1 |
| Feedforward [377] | Y | ELM-rand | 10 | 3.1E-1 \pm 1.1E-1 | 2.74E+0 \pm 1.87E-3 | 1.66E+0 \pm 5.64E-4 | 1.31E+0 \pm 1.79E-3 |
| SVR | - | - | Linear, 0.26927 | 1.65E+0 \pm 0.0E+0 | 2.15E+0\pm0.0E+0 | 1.47E+0\pm0.0E+0 | 1.19E+0\pm0.0E+0 |

Table 7.9: Performance on Bebida Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|------------------|-------------------------------|---------------------------------------|---------------------------------------|--------------------------------------|
| Elman | N | ELM-lin | 30 | 3.12E-3 \pm 6.98E-3 | 1.93E+2 \pm 2.11E-12 | 1.39E+1 \pm 7.60E-14 | 1.04E+1 \pm 9.87E-14 |
| Elman | N | ELM-rand | 20 | 2.6E-1 \pm 7.4E-2 | 1.71E+2\pm1.06E+1 | 1.31E+1\pm4.05E-1 | 9.6E+0\pm2.71E-1 |
| Elman | N | BPTT | 10 | 8.94E-1 \pm 1.34E+0 | 6.71E+2 \pm 5.46E+2 | 2.46E+1 \pm 9.04E+0 | 2.02E+1 \pm 8.52E+0 |
| Elman | Y | ELM-lin | 10 | 2.8E-1 \pm 6.3E-2 | 1.93E+2 \pm 3.27E-12 | 1.39E+1 \pm 1.18E-13 | 1.04E+1 \pm 1.72E-13 |
| Elman | Y | ELM-rand | 20 | 3.20E-1 \pm 0.0E+0 | 1.85E+2 \pm 6.11E+0 | 1.36E+1 \pm 2.24E-1 | 9.91E+0 \pm 1.49E-1 |
| Fully connected | N | ELM-lin | 45 | 7.49E-2 \pm 6.98E-3 | 1.93E+2 \pm 7.21E-12 | 1.39E+1 \pm 2.60E-13 | 1.04E+1 \pm 1.40E-13 |
| Fully connected | N | ELM-rand | 10 | 3.20E-1 \pm 0.0E+0 | 2.57E+2 \pm 1.36E+2 | 1.57E+1 \pm 3.79E+0 | 1.26E+1 \pm 3.88E+0 |
| Fully connected | Y | ELM-lin | 35 | 4.37E-2 \pm 1.31E-2 | 1.93E+2 \pm 1.29E-11 | 1.39E+1 \pm 4.65E-13 | 1.04E+1 \pm 1.10E-13 |
| Fully connected | Y | ELM-rand | 10 | 2.9E-1 \pm 5.8E-2 | 2.01E+2 \pm 1.24E+0 | 1.42E+1 \pm 4.38E-2 | 1.03E+1 \pm 2.46E-2 |
| Jordan | N | ELM-rand | 45 | 6.24E-3 \pm 8.54E-3 | 2.29E+2 \pm 4.47E+1 | 1.51E+1 \pm 1.46E+0 | 1.17E+1 \pm 7.57E-1 |
| Jordan | Y | ELM-rand | 45 | 6.24E-3 \pm 8.54E-3 | 2.24E+2 \pm 1.63E-1 | 1.50E+1 \pm 5.46E-3 | 1.06E+1 \pm 4.96E-3 |
| LSTM | N | ELM-rand | 10 | 1.56E-2 \pm 1.10E-2 | 3.46E+3 \pm 6.25E+3 | 4.34E+1 \pm 4.44E+1 | 4.09E+1 \pm 4.42E+1 |
| LSTM | Y | ELM-rand | 45 | 1.69E-1 \pm 1.53E-2 | 2.25E+2 \pm 9.35E-2 | 1.50E+1 \pm 3.12E-3 | 1.06E+1 \pm 1.94E-3 |
| NARMAX | N | ELM-rand | 25 | 6.24E-3 \pm 8.54E-3 | 2.35E+2 \pm 1.13E+2 | 1.50E+1 \pm 3.62E+0 | 1.24E+1 \pm 2.67E+0 |
| NARMAX | Y | ELM-rand | 45 | 1.13E-1 \pm 9.19E-2 | 2.37E+2 \pm 1.98E-1 | 1.54E+1 \pm 6.44E-3 | 1.20E+1 \pm 5.45E-3 |
| NARX | N | BPTT | 15 | 1.22E+0 \pm 2.26E+0 | 4.27E+2 \pm 9.03E+1 | 2.06E+1 \pm 2.08E+0 | 1.67E+1 \pm 1.85E+0 |
| Feedforward [379] | N | ELM-rand | 10 | 2.8E-1 \pm 6.3E-2 | 8.29E+2 \pm 8.36E+1 | 2.88E+1 \pm 1.40E+0 | 2.57E+1 \pm 2.25E-1 |
| Feedforward [376] | N | ELM-rand | 20 | 3.0E-1 \pm 8.4E-2 | 9.02E+2 \pm 1.03E+2 | 3.00E+1 \pm 1.70E+0 | 2.67E+1 \pm 1.04E+0 |
| Feedforward [377] | Y | ELM-rand | 15 | 3.6E-1 \pm 7.4E-2 | 1.78E+2 \pm 3.51E+0 | 1.33E+1 \pm 1.31E-1 | 9.67E+0 \pm 1.03E-1 |
| SVR | - | - | Linear, 1.0514 | 5.33E-2 \pm 0.0E+0 | 2.03E+2 \pm 0.0E+0 | 1.42E+1 \pm 0.0E+0 | 1.11E+1 \pm 0.0E+0 |

Table 7.10: Performance on Consumo Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|------------------|-------------------------------|---------------------------------------|--------------------------------------|--------------------------------------|
| Elman | N | ELM-lin | 10 | 3.12E-3 \pm 6.98E-3 | 2.0E+2 \pm 3.4E-11 | 1.4E+1 \pm 1.2E-12 | 1.05E+1 \pm 1.52E-12 |
| Elman | N | ELM-rand | 20 | 2.7E-1 \pm 1.3E-1 | 2.47E+2 \pm 7.43E+1 | 1.55E+1 \pm 2.29E+0 | 1.24E+1 \pm 2.33E+0 |
| Elman | N | BPTT | 10 | 2.37E-1 \pm 1.27E-1 | 1.37E+3 \pm 9.76E+2 | 3.49E+1 \pm 1.38E+1 | 2.85E+1 \pm 1.06E+1 |
| Elman | Y | ELM-lin | 20 | 1.8E-1 \pm 5.2E-2 | 2.01E+2 \pm 4.26E-11 | 1.42E+1 \pm 1.50E-12 | 1.05E+1 \pm 1.90E-12 |
| Elman | Y | ELM-rand | 30 | 2.3E-1 \pm 7.8E-2 | 2.01E+2 \pm 2.87E+0 | 1.42E+1 \pm 1.01E-1 | 1.03E+1 \pm 5.83E-2 |
| Fully connected | N | ELM-lin | 10 | 3.12E-3 \pm 6.98E-3 | 2.0E+2 \pm 2.3E-11 | 1.4E+1 \pm 8.1E-13 | 1.05E+1 \pm 1.14E-12 |
| Fully connected | N | ELM-rand | 10 | 1.6E-1 \pm 0.0E+0 | 4.74E+2 \pm 1.33E+2 | 2.15E+1 \pm 3.36E+0 | 1.49E+1 \pm 2.44E+0 |
| Fully connected | Y | ELM-lin | 40 | 4.37E-2 \pm 6.98E-3 | 2.0E+2 \pm 2.8E-10 | 1.4E+1 \pm 9.8E-12 | 1.05E+1 \pm 1.22E-11 |
| Fully connected | Y | ELM-rand | 15 | 1.9E-1 \pm 5.1E-2 | 1.99E+2\pm2.83E+0 | 1.41E+1\pm1.0E-1 | 1.03E+1\pm5.9E-2 |
| Jordan | N | ELM-rand | 45 | 3.1E-1 \pm 5.1E-2 | 3.74E+2 \pm 6.44E+1 | 1.93E+1 \pm 1.69E+0 | 1.33E+1 \pm 1.35E+0 |
| Jordan | Y | ELM-rand | 50 | 1.25E-2 \pm 6.98E-3 | 5.12E+2 \pm 2.19E-1 | 2.26E+1 \pm 4.84E-3 | 1.71E+1 \pm 3.97E-3 |
| LSTM | N | ELM-rand | 30 | 1.87E-2 \pm 6.98E-3 | 5.14E+3 \pm 4.21E+3 | 6.62E+1 \pm 3.07E+1 | 6.43E+1 \pm 3.15E+1 |
| LSTM | Y | ELM-rand | 35 | 1.06E-1 \pm 8.75E-2 | 5.13E+2 \pm 1.42E-1 | 2.27E+1 \pm 3.12E-3 | 1.71E+1 \pm 2.42E-3 |
| NARMAX | N | ELM-rand | 10 | 3.12E-3 \pm 6.98E-3 | 2.74E+2 \pm 5.66E+1 | 1.65E+1 \pm 1.64E+0 | 1.32E+1 \pm 1.57E+0 |
| NARMAX | Y | ELM-rand | 50 | 2.59E-1 \pm 8.82E-2 | 3.63E+2 \pm 1.11E-1 | 1.90E+1 \pm 2.92E-3 | 1.62E+1 \pm 2.98E-3 |
| NARX | N | BPTT | 10 | 9.64E-1 \pm 1.77E+0 | 9.46E+2 \pm 7.36E+2 | 2.88E+1 \pm 1.20E+1 | 1.95E+1 \pm 6.12E+0 |
| Feedforward [379] | N | ELM-rand | 20 | 2.7E-1 \pm 6.7E-2 | 7.98E+2 \pm 3.13E+1 | 2.82E+1 \pm 5.53E-1 | 2.54E+1 \pm 4.81E-1 |
| Feedforward [376] | N | ELM-rand | 30 | 2.6E-1 \pm 7.0E-2 | 8.57E+2 \pm 1.09E+2 | 2.92E+1 \pm 1.80E+0 | 2.57E+0 \pm 1.31E+0 |
| Feedforward [377] | Y | ELM-rand | 10 | 3.2E-1 \pm 1.5E-1 | 7.56E+2 \pm 6.32E+2 | 2.55E+1 \pm 1.14E+1 | 1.43E+1 \pm 3.02E+0 |
| SVR | - | - | Linear, 2.1304 | 1.62E+0 \pm 0.0E+0 | 2.01E+2 \pm 0.0E+0 | 1.42E+1 \pm 0.0E+0 | 8.84E+0 \pm 0.0E+0 |

Table 7.11: Performance on IPI Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|------------------|-------------------------------|---------------------------------------|--------------------------------------|--------------------------------------|
| Elman | N | ELM-lin | 20 | 3.0E-1 \pm 4.9E-2 | 1.21E+2 \pm 2.93E-11 | 1.10E+1 \pm 1.33E-12 | 8.9E+0\pm1.0E-12 |
| Elman | N | ELM-rand | 20 | 2.9E-1 \pm 5.4E-2 | 1.76E+2 \pm 4.38E+1 | 1.32E+1 \pm 1.75E+0 | 1.11E+1 \pm 1.66E+0 |
| Elman | N | BPTT | 10 | 3.06E-1 \pm 1.16E-1 | 5.00E+2 \pm 3.92E+2 | 2.10E+1 \pm 8.44E+0 | 1.81E+1 \pm 7.79E+0 |
| Elman | Y | ELM-lin | 20 | 3.1E-1 \pm 0.0E+0 | 1.21E+2 \pm 2.18E-11 | 1.10E+1 \pm 9.90E-13 | 8.9E+0\pm7.6E-13 |
| Elman | Y | ELM-rand | 10 | 2.6E-1 \pm 7.3E-2 | 1.14E+2\pm2.49E+0 | 1.1E+1\pm1.16E-1 | 9.15E+0 \pm 1.79E-1 |
| Fully connected | N | ELM-lin | 45 | 8.42E-2 \pm 8.54E-3 | 1.21E+2 \pm 1.88E-10 | 1.10E+1 \pm 8.53E-12 | 8.92E+0 \pm 6.43E-12 |
| Fully connected | N | ELM-rand | 10 | 3.2E+0 \pm 0.0E+0 | 5.49E+2 \pm 8.10E+1 | 2.34E+1 \pm 1.73E+0 | 1.88E+1 \pm 1.57E+0 |
| Fully connected | Y | ELM-lin | 50 | 1.34E-1 \pm 1.40E-2 | 1.21E+2 \pm 2.17E-10 | 1.10E+1 \pm 9.88E-12 | 8.9E+0\pm8.0E-12 |
| Fully connected | Y | ELM-rand | 10 | 2.7E-1 \pm 6.8E-2 | 1.14E+2 \pm 1.94E+0 | 1.07E+1 \pm 9.08E-2 | 9.12E+0 \pm 1.34E-1 |
| Jordan | N | ELM-rand | 45 | 3.12E-3 \pm 6.98E-3 | 2.37E+2 \pm 5.81E+1 | 1.53E+1 \pm 1.92E+0 | 1.24E+1 \pm 1.55E+0 |
| Jordan | Y | ELM-rand | 50 | 3.0E-1 \pm 5.4E-2 | 1.57E+2 \pm 1.39E-1 | 1.25E+1 \pm 5.55E-3 | 1.02E+1 \pm 4.30E-3 |
| LSTM | N | ELM-rand | 15 | 1.56E-2 \pm 1.10E-2 | 6.29E+3 \pm 6.87E+3 | 6.81E+1 \pm 4.54E+1 | 6.60E+1 \pm 4.62E+1 |
| LSTM | Y | ELM-rand | 25 | 2.81E-2 \pm 5.63E-2 | 1.57E+2 \pm 5.15E-2 | 1.25E+1 \pm 2.06E-3 | 1.02E+1 \pm 1.44E-3 |
| NARMAX | N | ELM-rand | 45 | 9.36E-3 \pm 8.54E-3 | 5.17E+2 \pm 2.13E+2 | 2.24E+1 \pm 4.36E+0 | 1.90E+1 \pm 3.5E+0 |
| NARMAX | Y | ELM-rand | 50 | 1.91E-1 \pm 4.57E-2 | 6.76E+2 \pm 1.53E-1 | 2.60E+1 \pm 2.95E-3 | 2.05E+1 \pm 2.21E-3 |
| NARX | N | BPTT | 10 | 1.02E+0 \pm 1.8E+0 | 7.69E+2 \pm 5.10E+2 | 2.65E+1 \pm 9.29E+0 | 2.22E+1 \pm 8.15E+0 |
| Feedforward [379] | N | ELM-rand | 45 | 3.0E-1 \pm 1.1E-1 | 4.75E+2 \pm 4.96E+1 | 2.18E+1 \pm 1.11E+0 | 2.03E+1 \pm 9.53E-1 |
| Feedforward [376] | N | ELM-rand | 40 | 6.25E-03 \pm 1.40E-2 | 4.58E+2 \pm 3.32E+1 | 2.14E+1 \pm 7.70E-1 | 1.96E+1 \pm 5.60E-1 |
| Feedforward [377] | Y | ELM-rand | 10 | 3.12E-3 \pm 6.98E-3 | 1.34E+2 \pm 2.85E+1 | 1.15E+1 \pm 1.18E+0 | 9.76E+0 \pm 9.99E-1 |
| SVR | - | - | Linear, 897.3 | 1.6E+0 \pm 0.0E+0 | 2.90E+2 \pm 0.0E+0 | 1.70E+1 \pm 0.0E+0 | 1.58E+1 \pm 0.0E+0 |

Table 7.12: Performance on Inverted Pendulum Data Set

| Architecture | DL | Training Algo- rithm | Hyper- parameters | Training Time (avg ± std) | Testing MSE (avg ± std) | Testing RMSE (avg ± std) | Testing MAE (avg ± std) |
|-------------------------|----|----------------------------|----------------------|------------------------------|----------------------------|-----------------------------|----------------------------|
| Elman | N | ELM-lin | 35 | 6.24E-3±8.54E-3 | 1.30E-1±4.34E-16 | 3.61E-1±6.05E-16 | 2.60E-1±3.02E-16 |
| Elman | N | ELM- rand | 10 | 9.36E-3±2.09E-2 | 1.08E-1±1.00E-2 | 3.28E-1±1.52E-2 | 2.32E-1±5.34E-3 |
| Elman | N | BPTT | 10 | 5.00E-1±1.30E-1 | 1.41E-1±2.33E-2 | 3.74E-1±3.00E-2 | 2.73E-1±2.44E-2 |
| Elman | Y | ELM-lin | 15 | 2.5E-1±9.1E-2 | 1.30E-1±9.83E-16 | 3.61E-1±1.35E-15 | 2.60E-1±5.54E-16 |
| Elman | Y | ELM- rand | 10 | 3.12E-3±6.98E-3 | 1.03E-1±6.44E-3 | 3.21E-1±9.90E-3 | 2.31E-1±3.15E-3 |
| Fully con- nected | N | ELM-lin | 25 | 2.84E-1±3.00E-2 | 1.30E-1±8.15E-15 | 3.61E-1±1.13E-14 | 2.60E-1±8.06E-15 |
| Fully con- nected | N | ELM- rand | 30 | 3.12E-3±6.98E-3 | 1.01E-1±7.12E-3 | 3.18E-1±1.11E-2 | 2.30E-1±7.44E-3 |
| Fully con- nected | Y | ELM-lin | 45 | 4.39E+0±3.80E-1 | 1.30E-1±1.09E-14 | 3.61E-1±1.51E-14 | 2.60E-1±9.42E-15 |
| Fully con- nected | Y | ELM- rand | 40 | 6.24E-3±8.54E-3 | 1.04E-1±5.43E-3 | 3.23E-1±8.44E-3 | 2.33E-1±5.79E-3 |
| Jordan | N | ELM- rand | 35 | 5.93E-2±1.31E-2 | 1.45E-1±5.80E-4 | 3.80E-1±7.62E-4 | 2.77E-1±4.72E-4 |
| Jordan | Y | ELM- rand | 10 | 3.74E-2±1.40E-2 | 9.96E-2±1.11E-3 | 3.16E-1±1.75E-3 | 2.31E-1±1.08E-3 |
| LSTM | N | ELM- rand | 45 | 3.34E-1±3.92E-2 | 1.45E-1±6.97E-3 | 3.81E-1±9.30E-3 | 2.76E-1±7.02E-3 |
| LSTM | Y | ELM- rand | 40 | 3.70E+1±1.11E+0 | 1.88E-2±4.35E-5 | 1.37E-1±1.59E-4 | 7.65E-2±5.23E-4 |
| NARMAX | N | ELM- rand | 35 | 6.55E-2±1.71E-2 | 1.45E-1±2.05E-3 | 3.81E-1±2.69E-3 | 2.77E-1±1.82E-3 |
| NARMAX | Y | ELM- rand | 10 | 7.63E-1±8.64E-2 | 1.93E-2±7.60E-5 | 1.39E-1±2.74E-4 | 7.84E-2±9.40E-4 |
| NARX | N | BPTT | 10 | 1.27E+0±1.97E+0 | 7.04E-3±7.82E-3 | 7.61E-2±3.94E-2 | 4.57E-2±1.33E-2 |
| Feedforward [379] | N | ELM- rand | 20 | 3.5E-1±7.9E-2 | 4.24E-3±3.36E-3 | 6.09E-2±2.55E-2 | 3.16E-2±1.26E-2 |
| Feedforward [376] | N | ELM- rand | 20 | 9.38E-3±2.10E-2 | 8.64E-3±9.23E-3 | 7.91E-2±5.46E-2 | 3.40E-2±1.97E-2 |
| Feedforward [377] | Y | ELM- rand | 15 | 3.12E-3±6.98E-3 | 9.51E-2±1.11E-3 | 3.08E-1±1.80E-3 | 2.25E-1±1.45E-3 |
| SVR | - | - | Gaussian, 7.3806 | 2.57E+0±0.0E+0 | 2.63E-4±0.0E+0 | 1.62E-2±0.0E+0 | 1.16E-2±0.0E+0 |

Table 7.13: Performance on Quebec Births Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|---------------------|-------------------------------|---------------------------------------|---------------------------------------|---------------------------------------|
| Elman | N | ELM-lin | 40 | 5.20E-3 \pm 9.01E-3 | 1.15E-2 \pm 3.25E-16 | 1.07E-1 \pm 1.51E-15 | 8.43E-2 \pm 1.32E-15 |
| Elman | N | ELM-rand | 10 | 3.12E-3 \pm 6.98E-3 | 2.15E-2 \pm 1.17E-2 | 1.42E-1 \pm 3.91E-2 | 8.55E-2 \pm 8.01E-4 |
| Elman | N | BPTT | 20 | 6.55E-1 \pm 3.15E-1 | 1.07E-2 \pm 2.99E-3 | 1.03E-1 \pm 1.38E-2 | 8.18E-2 \pm 9.63E-3 |
| Elman | Y | ELM-lin | 35 | 9.36E-3 \pm 8.54E-3 | 1.15E-2 \pm 2.69E-16 | 1.07E-1 \pm 1.25E-15 | 8.43E-2 \pm 1.04E-15 |
| Elman | Y | ELM-rand | 10 | 3.12E-3 \pm 6.98E-3 | 1.39E-2 \pm 2.25E-3 | 1.17E-1 \pm 9.50E-3 | 8.50E-2 \pm 5.67E-4 |
| Fully connected | N | ELM-lin | 15 | 4.37E-2 \pm 6.98E-3 | 1.15E-2 \pm 1.75E-15 | 1.07E-1 \pm 8.16E-15 | 8.43E-2 \pm 6.40E-15 |
| Fully connected | N | ELM-rand | 40 | 1.04E-2 \pm 9.01E-3 | 1.57E-2 \pm 2.33E-3 | 1.25E-1 \pm 9.44E-3 | 9.50E-2 \pm 5.08E-3 |
| Fully connected | Y | ELM-lin | 25 | 2.00E-1 \pm 1.31E-2 | 1.15E-2 \pm 6.84E-16 | 1.07E-1 \pm 3.19E-15 | 8.43E-2 \pm 2.54E-15 |
| Fully connected | Y | ELM-rand | 10 | 2.7E-1 \pm 7.1E-2 | 1.65E-2 \pm 7.15E-3 | 1.26E-1 \pm 2.52E-2 | 8.10E-2 \pm 1.10E-3 |
| Jordan | N | ELM-rand | 20 | 3.43E-2 \pm 6.98E-3 | 4.37E-2 \pm 2.05E-2 | 2.05E-1 \pm 4.72E-2 | 1.75E-1 \pm 5.07E-2 |
| Jordan | Y | ELM-rand | 15 | 4.06E-2 \pm 1.40E-2 | 3.04E-2 \pm 6.17E-3 | 1.74E-1 \pm 1.80E-2 | 1.42E-1 \pm 1.91E-2 |
| LSTM | N | ELM-rand | 20 | 1.37E-1 \pm 6.98E-3 | 1.63E-1 \pm 1.25E-1 | 3.73E-1 \pm 1.71E-1 | 3.51E-1 \pm 1.78E-1 |
| LSTM | Y | ELM-rand | 20 | 2.59E-1 \pm 5.64E-2 | 3.22E-2 \pm 1.40E-2 | 1.75E-1 \pm 3.87E-2 | 1.48E-1 \pm 4.02E-2 |
| NARMAX | N | ELM-rand | 10 | 3.12E-2 \pm 1.59E-15 | 4.27E-2 \pm 2.42E-2 | 2.00E-1 \pm 5.66E-2 | 1.74E-1 \pm 6.07E-2 |
| NARMAX | Y | ELM-rand | 10 | 2.75E-1 \pm 6.96E-2 | 3.08E-2 \pm 2.46E-3 | 1.75E-1 \pm 7.09E-3 | 1.42E-1 \pm 8.53E-3 |
| NARX | N | BPTT | 20 | 6.35E-1 \pm 4.22E-2 | 1.29E-2 \pm 1.74E-3 | 1.14E-1 \pm 7.89E-3 | 8.89E-2 \pm 5.07E-3 |
| Feedforward [379] | N | ELM-rand | 25 | 3.1E-1 \pm 8.3E-2 | 3.89E-3 \pm 9.84E-5 | 6.23E-2 \pm 7.88E-4 | 4.86E-2 \pm 6.44E-4 |
| Feedforward [376] | N | ELM-rand | 35 | 3.5E-1 \pm 1.2E-1 | 3.88E-3\pm6.17E-5 | 6.23E-2\pm4.97E-4 | 4.85E-2\pm3.10E-4 |
| Feedforward [377] | Y | ELM-rand | 20 | 3.9E-1 \pm 7.8E-2 | 1.14E-2 \pm 4.74E-08 | 1.07E-1 \pm 2.22E-07 | 8.37E-2 \pm 3.65E-07 |
| SVR | - | - | Polynomial, 0.16565 | 3.12E-2 \pm 0.0E+0 | 2.06E-1 \pm 0.0E+0 | 4.54E-1 \pm 0.0E+0 | 4.28E-1 \pm 0.0E+0 |

Table 7.14: Performance on Santa Fe Data Set

| Architecture | DL | Training Algo- rithm | Hyper- parameters | Training Time (avg ± std) | Testing MSE (avg ± std) | Testing RMSE (avg ± std) | Testing MAE (avg ± std) |
|-------------------------|----|----------------------------|----------------------|------------------------------|----------------------------|-----------------------------|----------------------------|
| Elman | N | ELM-lin | 40 | 3.12E-3±6.98E-3 | 2.38E-2±5.30E-16 | 1.54E-1±1.72E-15 | 1.20E-1±1.84E-15 |
| Elman | N | ELM- rand | 10 | 9.36E-3±2.09E-2 | 5.73E-2±5.10E-2 | 2.23E-1±9.65E-2 | 1.05E-1±1.14E-2 |
| Elman | N | BPTT | 15 | 5.10E-1±1.28E-1 | 4.86E-2±3.11E-2 | 2.10E-1±7.51E-2 | 1.57E-1±4.71E-2 |
| Elman | Y | ELM-lin | 45 | 3.12E-3±6.98E-3 | 2.38E-2±2.44E-16 | 1.54E-1±7.90E-16 | 1.20E-1±8.38E-16 |
| Elman | Y | ELM- rand | 15 | 3.12E-3±6.98E-3 | 1.28E-1±7.39E-2 | 3.46E-1±1.04E-1 | 1.73E-1±2.13E-2 |
| Fully con- nected | N | ELM-lin | 15 | 3.43E-2±6.98E-3 | 2.38E-2±2.80E-15 | 1.54E-1±9.08E-15 | 1.20E-1±9.30E-15 |
| Fully con- nected | N | ELM- rand | 45 | 3.2E-1±7.5E-2 | 2.92E-2±1.65E-2 | 1.63E-1±5.10E-2 | 1.11E-1±4.50E-2 |
| Fully con- nected | Y | ELM-lin | 30 | 4.49E-1±9.95E-2 | 2.38E-2±3.39E-15 | 1.54E-1±1.10E-14 | 1.20E-1±1.28E-14 |
| Fully con- nected | Y | ELM- rand | 50 | 3.12E-3±6.98E-3 | 1.26E-2±2.58E-3 | 1.11E-1±1.12E-2 | 7.46E-2±8.87E-3 |
| Jordan | N | ELM- rand | 15 | 2.50E-2±8.54E-3 | 6.08E-2±6.12E-3 | 2.46E-1±1.26E-2 | 1.78E-1±6.51E-3 |
| Jordan | Y | ELM- rand | 10 | 2.81E-2±6.98E-3 | 6.01E-2±6.03E-3 | 2.45E-1±1.24E-2 | 1.77E-1±6.05E-3 |
| LSTM | N | ELM- rand | 35 | 1.40E-1±1.56E-2 | 7.57E-2±4.01E-2 | 2.69E-1±6.66E-2 | 2.23E-1±5.83E-2 |
| LSTM | Y | ELM- rand | 10 | 4.38E-1±3.31E-1 | 5.01E-2±1.35E-3 | 2.24E-1±3.03E-3 | 1.51E-1±2.72E-3 |
| NARMAX | N | ELM- rand | 20 | 2.18E-2±8.54E-3 | 6.16E-2±6.41E-3 | 2.48E-1±1.29E-2 | 1.80E-1±6.53E-3 |
| NARMAX | Y | ELM- rand | 25 | 2.84E-1±7.49E-2 | 5.24E-2±1.01E-3 | 2.29E-1±2.21E-3 | 1.57E-1±1.66E-3 |
| NARX | N | BPTT | 15 | 5.63E-1±3.37E-1 | 5.03E-2±6.49E-3 | 2.24E-1±1.41E-2 | 1.75E-1±1.13E-2 |
| Feedforward [379] | N | ELM- rand | 10 | 3.1E-1±1.3E-1 | 1.41E-2±2.52E-3 | 1.18E-1±1.10E-2 | 1.03E-1±1.29E-2 |
| Feedforward [376] | N | ELM- rand | 10 | 2.9E-1±1.1E-1 | 2.34E-2±1.71E-2 | 1.46E-1±5.11E-2 | 1.29E-1±5.43E-2 |
| Feedforward [377] | Y | ELM- rand | 15 | 3.6E-1±7.4E-2 | 3.42E-2±1.97E-4 | 1.85E-1±5.32E-4 | 1.40E-1±6.68E-4 |
| SVR | - | - | Polynomial, 2.949 | 1.77E+0±0.0E+0 | 1.69E-2±0.0E+0 | 1.30E-1±0.0E+0 | 1.22E-1±0.0E+0 |

Table 7.15: Performance on Deep Tesla Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|------------------|-------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Elman | N | ELM-lin | 10 | 6.6E-28 \pm 4.9E-28 | 6.3E-3 \pm 1.3E-2 | 1.0E-1 \pm 2.2E-14 | 1.0E-1 \pm 2.2E-14 |
| Elman | N | ELM-rand | 15 | 1.0E-4 \pm 1.1E-4 | 3.1E-3 \pm 6.3E-3 | 1.0E-1 \pm 2.1E-3 | 9.9E-2 \pm 1.3E-3 |
| Elman | N | BPTT | 15 | 1.1E-2 \pm 1.2E-2 | 1.8E-2 \pm 1.5E-3 | 1.5E-1 \pm 4.7E-2 | 1.4E-1 \pm 4.3E-2 |
| Elman | Y | ELM-lin | 10 | 8.6E-28 \pm 8.3E-28 | 6.3E-3 \pm 1.3E-2 | 1.0E-1 \pm 2.7E-14 | 1.0E-1 \pm 2.7E-14 |
| Elman | Y | ELM-rand | 10 | 2.1E-12 \pm 4.0E-12 | 1.0E-2 \pm 2.0E-12 | 1.0E-1 \pm 1.4E-6 | 1.0E-1 \pm 1.4E-6 |
| Fully connected | N | ELM-lin | 10 | 1.2E-27 \pm 2.0E-27 | 1.0E-2 \pm 8.4E-28 | 1.0E-1 \pm 2.9E-14 | 1.0E-1 \pm 2.9E-14 |
| Fully connected | N | ELM-rand | 10 | 1.9E-3 \pm 1.2E-3 | 6.3E-3 \pm 1.3E-2 | 1.4E-1 \pm 5.5E-2 | 1.1E-1 \pm 1.2E-2 |
| Fully connected | Y | ELM-lin | 15 | 2.3E-28 \pm 1.8E-28 | 6.3E-3 \pm 1.3E-2 | 1.0E-1 \pm 1.6E-14 | 1.0E-1 \pm 1.6E-14 |
| Fully connected | Y | ELM-rand | 10 | 1.6E-12 \pm 3.1E-12 | 1.0E-2 \pm 1.4E-12 | 1.0E-1 \pm 1.2E-6 | 1.0E-1 \pm 1.2E-6 |
| Jordan | N | ELM-rand | 10 | 2.3E-2 \pm 4.6E-4 | 1.9E-2 \pm 1.5E-2 | 1.2E-1 \pm 7.7E-4 | 1.1E-1 \pm 5.0E-4 |
| Jordan | Y | ELM-rand | 45 | 3.4E-7 \pm 8.0E-8 | 3.1E-3 \pm 6.3E-3 | 1.0E-1 \pm 2.6E-5 | 1.0E-1 \pm 3.2E-5 |
| LSTM | N | ELM-rand | 20 | 3.8E-1 \pm 1.6E-1 | 3.1E-3 \pm 6.3E-3 | 4.7E-1 \pm 3.4E-1 | 4.5E-1 \pm 3.5E-1 |
| LSTM | Y | ELM-rand | 15 | 1.1E-8 \pm 7.3E-9 | 3.1E-3 \pm 6.3E-3 | 1.0E-1 \pm 6.9E-5 | 1.0E-1 \pm 6.8E-5 |
| NARMAX | N | ELM-rand | 10 | 2.3E-2 \pm 2.4E-4 | 3.1E-3 \pm 6.3E-3 | 1.2E-1 \pm 2.4E-4 | 1.1E-1 \pm 2.4E-4 |
| NARMAX | Y | ELM-rand | 10 | 1.5E-8 \pm 8.0E-9 | 1.6E-2 \pm 2.0E-2 | 1.0E-1 \pm 1.0E-5 | 1.0E-1 \pm 1.3E-5 |
| NARX | N | BPTT | 30 | 3.3E-4 \pm 1.4E-5 | 4.7E-2 \pm 3.2E-3 | 2.5E-1 \pm 7.7E-2 | 2.3E-1 \pm 7.3E-2 |
| Feedforward [379] | N | ELM-rand | 15 | 6.7E-5 \pm 1.0E-4 | 1.6E-2\pm2.0E-2 | 9.9E-2\pm1.7E-3 | 9.9E-2\pm1.8E-3 |
| Feedforward [377] | Y | ELM-rand | 10 | 1.5E-12 \pm 1.5E-12 | 1.0E-2 \pm 6.6E-13 | 1.0E-1 \pm 8.1E-7 | 1.0E-1 \pm 8.2E-7 |
| SVR | - | - | Linear, 41.44 | 6.2E-7 \pm 0.0E+0 | 6.9E-4 \pm 0.0E+0 | 1.0E-1 \pm 0.0E+0 | 1.0E-1 \pm 0.0E+0 |

Table 7.16: Performance on Japan Population Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|-------------------|-------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Elman | N | ELM-lin | 10 | 1.8E-3 \pm 4.0E-18 | 1.3E-2 \pm 1.5E-2 | 4.8E-2 \pm 2.9E-16 | 1.3E-2 \pm 1.9E-15 |
| Elman | N | ELM-rand | 10 | 1.6E-3 \pm 2.4E-5 | 9.0E-3 \pm 3.4E-3 | 9.5E-2 \pm 5.8E-2 | 1.5E-2 \pm 3.8E-3 |
| Elman | N | BPTT | 30 | 1.3E-2 \pm 1.4E-2 | 2.5E-2 \pm 4.7E-3 | 1.1E-1 \pm 6.4E-2 | 6.7E-2 \pm 5.4E-2 |
| Elman | Y | ELM-lin | 10 | 1.8E-3 \pm 0.0E+0 | 6.3E-3 \pm 1.3E-2 | 4.8E-2 \pm 0.0E+0 | 1.3E-2 \pm 2.0E-16 |
| Elman | Y | ELM-rand | 10 | 1.4E-3 \pm 9.0E-5 | 3.2E+2 \pm 1.2E+3 | 1.8E+1 \pm 3.4E+1 | 7.3E-1 \pm 1.3E+0 |
| Fully connected | N | ELM-lin | 10 | 1.6E-3 \pm 1.9E-5 | 7.9E-3 \pm 1.4E-3 | 8.9E-2 \pm 3.7E-2 | 1.4E-2 \pm 2.0E-3 |
| Fully connected | N | ELM-rand | 10 | 1.8E-3 \pm 4.0E-18 | 6.3E-3 \pm 1.3E-2 | 4.8E-2 \pm 1.6E-15 | 1.3E-2 \pm 1.2E-14 |
| Fully connected | Y | ELM-lin | 10 | 1.8E-3 \pm 4.0E-18 | 1.3E-2 \pm 1.5E-2 | 4.8E-2 \pm 6.7E-16 | 1.3E-2 \pm 4.8E-15 |
| Fully connected | Y | ELM-rand | 35 | 1.1E-3 \pm 1.2E-4 | 1.3E-2 \pm 1.5E-2 | 2.4E-1 \pm 3.2E-1 | 2.3E-2 \pm 1.6E-2 |
| Jordan | N | ELM-rand | 15 | 1.6E-1 \pm 3.9E-3 | 6.3E-3 \pm 7.7E-3 | 4.5E-1 \pm 6.0E-2 | 4.4E-1 \pm 5.9E-2 |
| Jordan | Y | ELM-rand | 10 | 1.5E-1 \pm 8.1E-3 | 1.6E-2 \pm 9.9E-3 | 3.7E-1 \pm 6.8E-2 | 3.6E-1 \pm 6.7E-2 |
| LSTM | N | ELM-rand | 10 | 8.7E-2 \pm 9.5E-2 | 1.3E-2 \pm 6.3E-3 | 3.2E-1 \pm 1.0E-1 | 3.1E-1 \pm 1.0E-1 |
| LSTM | Y | ELM-rand | 10 | 1.3E-1 \pm 3.2E-2 | 1.6E-2 \pm 0.0E+0 | 4.0E-1 \pm 4.3E-2 | 3.9E-1 \pm 4.2E-2 |
| NARMAX | N | ELM-rand | 15 | 1.5E-1 \pm 1.0E-2 | 1.6E-2 \pm 0.0E+0 | 4.3E-1 \pm 3.3E-2 | 4.2E-1 \pm 3.3E-2 |
| NARMAX | Y | ELM-rand | 15 | 1.5E-1 \pm 6.2E-3 | 1.6E-2 \pm 1.7E-2 | 3.9E-1 \pm 2.3E-2 | 3.8E-1 \pm 2.2E-2 |
| NARX | N | BPTT | 20 | 2.1E-3 \pm 7.8E-4 | 6.7E-2 \pm 3.4E-3 | 7.1E-2 \pm 5.9E-3 | 2.5E-2 \pm 4.9E-3 |
| Feedforward [379] | N | ELM-rand | 10 | 1.6E-3 \pm 8.9E-6 | 3.1E-3\pm6.3E-3 | 3.9E-2\pm3.2E-4 | 8.7E-3\pm2.9E-4 |
| Feedforward [377] | Y | ELM-rand | 20 | 1.5E-3 \pm 2.2E-5 | 1.3E-3 \pm 2.7E-7 | 3.7E-2 \pm 5.2E-4 | 9.2E-3 \pm 4.8E-5 |
| SVR | - | - | Linear, 0.0010492 | 1.9E-3 \pm 0.0E+0 | 1.5E-3 \pm 0.0E+0 | 5.0E-2 \pm 0.0E+0 | 8.1E-3 \pm 0.0E+0 |

Table 7.17: Performance on SP500 Data Set

| Architecture | DL | Training Algorithm | Hyper-parameters | Training Time (avg \pm std) | Testing MSE (avg \pm std) | Testing RMSE (avg \pm std) | Testing MAE (avg \pm std) |
|-------------------|----|--------------------|---------------------|-------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Elman | N | ELM-lin | 10 | 2.0E-6 \pm 0.0E+0 | 1.3E-2 \pm 1.2E-2 | 1.0E+0 \pm 3.6E-14 | 9.7E-1 \pm 3.4E-14 |
| Elman | N | ELM-rand | 45 | 7.7E-3 \pm 1.6E-3 | 1.6E-2\pm1.4E-2 | 1.7E-1\pm8.7E-3 | 1.2E-1\pm3.6E-3 |
| Elman | N | BPTT | 50 | 5.8E-4 \pm 2.7E-4 | 1.2E-1 \pm 1.1E-2 | 6.9E-1 \pm 3.3E-1 | 6.2E-1 \pm 3.1E-1 |
| Elman | Y | ELM-lin | 10 | 2.0E-6 \pm 4.1E-21 | 6.3E-3 \pm 7.7E-3 | 1.0E+0 \pm 6.8E-14 | 9.7E-1 \pm 6.4E-14 |
| Elman | Y | ELM-rand | 10 | 2.0E-6 \pm 1.4E-8 | 1.0E+0 \pm 2.6E-6 | 1.0E+0 \pm 1.6E-3 | 9.7E-1 \pm 1.4E-3 |
| Fully connected | N | ELM-lin | 10 | 2.0E-6 \pm 0.0E+0 | 2.8E-2 \pm 6.3E-3 | 1.0E+0 \pm 7.8E-14 | 9.7E-1 \pm 7.3E-14 |
| Fully connected | N | ELM-rand | 10 | 2.3E-2 \pm 4.5E-3 | 1.6E-2 \pm 0.0E+0 | 2.9E-1 \pm 3.8E-2 | 2.4E-1 \pm 4.4E-2 |
| Fully connected | Y | ELM-lin | 10 | 2.0E-6 \pm 4.1E-21 | 5.9E-2 \pm 1.8E-2 | 1.0E+0 \pm 1.8E-13 | 9.7E-1 \pm 1.8E-13 |
| Fully connected | Y | ELM-rand | 10 | 2.0E-6 \pm 1.4E-8 | 1.6E-2 \pm 9.9E-3 | 1.0E+0 \pm 9.6E-4 | 9.7E-1 \pm 7.3E-4 |
| Jordan | N | ELM-rand | 35 | 6.7E-1 \pm 3.7E-3 | 7.2E-2 \pm 1.6E-2 | 5.0E-1 \pm 1.3E-3 | 4.7E-1 \pm 1.1E-3 |
| Jordan | Y | ELM-rand | 15 | 3.5E-6 \pm 8.9E-8 | 9.1E-2 \pm 1.8E-2 | 1.0E+0 \pm 7.7E-5 | 9.6E-1 \pm 6.2E-5 |
| LSTM | N | ELM-rand | 10 | 7.0E-2 \pm 3.5E-2 | 3.0E-1 \pm 2.5E-2 | 5.8E-1 \pm 2.9E-1 | 5.5E-1 \pm 3.0E-1 |
| LSTM | Y | ELM-rand | 10 | 3.8E-6 \pm 7.4E-8 | 2.8E-1 \pm 3.2E-2 | 1.0E+0 \pm 4.2E-5 | 9.6E-1 \pm 3.5E-5 |
| NARMAX | N | ELM-rand | 30 | 6.7E-1 \pm 9.6E-3 | 1.0E-1 \pm 1.6E-2 | 5.0E-1 \pm 2.3E-4 | 4.7E-1 \pm 2.4E-4 |
| NARMAX | Y | ELM-rand | 25 | 3.3E-6 \pm 1.3E-8 | 8.4E-2 \pm 1.3E-2 | 1.0E+0 \pm 5.2E-5 | 9.6E-1 \pm 4.2E-5 |
| NARX | N | BPTT | 20 | 3.2E-6 \pm 2.2E-8 | 3.7E-1 \pm 3.7E-2 | 1.0E+0 \pm 1.5E-1 | 9.8E-1 \pm 1.5E-1 |
| Feedforward [379] | N | ELM-rand | 20 | 9.2E-3 \pm 1.6E-3 | 3.2E-2 \pm 1.4E-4 | 1.8E-1 \pm 1.2E-2 | 1.2E-1 \pm 9.7E-3 |
| Feedforward [377] | Y | ELM-rand | 10 | 1.9E-6 \pm 4.1E-8 | 1.0E+0 \pm 8.4E-6 | 1.0E+0 \pm 2.9E-3 | 9.7E-1 \pm 2.5E-3 |
| SVR | - | - | Polynomial, 0.15218 | 1.0E+4 \pm 0.0E+0 | 1.1E+0 \pm 0.0E+0 | 1.0E+0 \pm 0.0E+0 | 9.9E-1 \pm 0.0E+0 |

RNN with ELM-lin but required 87% more time to train. Comparing non-iteratively-trained Elman and fully connected RNN on the Santa Fe laser data set, non-iteratively-trained fully connected RNN achieved 77% improvement in MSE but resulted in a slight increase in MSE, 1.6%, for the Electricity load balance data set. The training time varied based on the database size but generally reflected an increase in training time for non-iteratively-trained fully connected RNN compared to non-iteratively-trained Elman RNN. Non-iteratively-trained LSTM did not perform well in general on the smaller databases due to the curse of dimensionality. Larger databases are required to learn a better model. Considering the electricity load database, a $3.5\times$ increase in MSE was experienced compared to non-iteratively-trained feedforward networks. Therefore, additional research should be performed to assess the merit of training an LSTM network non-iteratively.

Next, we compared the performance of non-iteratively-trained Elman and fully connected RNN to the Jordan RNN architecture proposed in [386] and non-iteratively-trained feedforward networks based on [376, 377, 379]. Note that the difference between [376] and [379] is the presence of a bias neuron in the hidden layer as opposed to the input layer. Considering the inverted pendulum problem, the feedforward network trained using ELM had the best performance with an MSE of 0.0168 only 4.8% better than the best non-iteratively-trained RNN algorithm which was fully connected RNN trained with ELM-rand. However, the disadvantage of using feedforward networks is that the best autoregressive model with exogenous input, i.e. the order of the autoregressive output, must be obtained by trial and error. Non-iteratively-trained RNN does not require this grid search. In addition, learning the additional weights did not cause a significant reduction in training speed.

Focusing on architectures with and without DL, a consistent pattern was not observed, i.e. RNN architectures with DL sometimes performed better than their counterparts without DL and vice versa. For example, a fully connected RNN trained using ELM-rand performed better without DL on the Atmosphere Temperature database but much worse on the Consumo database. However, ELM-rand trained LSTM networks with DL consistently outperformed LSTM without DL on all the mentioned databases. Therefore, adding DL to non-iteratively-trained RNN architectures should be investigated on a case by case basis.

Compared to NARX network trained using BPTT, a speedup of 69% in training time of non-iteratively-trained RNN was achieved with only a 4.3% degradation in prediction accuracy. NARMAX with ELM achieved faster training time than NARX with BPTT but resulted in larger MSE. In addition, non-iteratively-trained RNN achieved better repeatability given their lower standard deviations compared to BPTT-trained networks. This is due to the lower dimensional search spaces of non-iterative training algorithms compared to iterative algorithms, since the input weights are not learned. Similar results were obtained on the larger data sets where significant speedups, up to 99% for Electricity load balance, were

achieved but at the cost of an increase in MSE.

Compared to SVR, non-iteratively-trained RNN are approximately $7.7\times$ faster than SVR, when averaging across all databases; the speed up was more significant as the size of the database increased and reached almost $37\times$ on the Electricity load database. However, feedforward networks' MSE and MAE were on average $2.2\times$ and $1.03\times$ larger than SVR's MSE and MAE, respectively.

ELM have been known to overfit, especially as the number of hidden layer neurons increases. To minimize the possibility of choosing models that were overfitting, we penalized networks with a large number of hidden layers by choosing less complex models when the performance of both models was arbitrarily close. In future work, we will also investigate incorporating a regularization term in the objective function that has been shown to avoid overfitting [430].

Finally, we note that some data sets resulted in large prediction errors due to data ill conditioning and random initialization. Since the network architecture contains recurrent connections, unsuitable initial conditions can cause the error to increase dramatically and the network is unable to correct it. Therefore, for recurrent architectures, non-iterative training techniques should take certain measures to insure a good initialization and matrix conditioning, which will be investigated in future work.

7.5.3 Non-iteratively-trained RNN Statistical Analysis

We performed a statistical analysis to compare the performance of the variants on all the databases in this work. The tests were performed on 16 non-iteratively-trained RNN variants considered in this work. Specifically, the Friedman ranking and ANOVA tests resulted in p-values equal to 0 and 0.0086, respectively. Both values are less than the significance level set to $\alpha = 0.05$, which implies the presence of a significant difference between the algorithms. Furthermore, the Nemenyi post-hoc was performed once a significant difference was observed, to rank the algorithms. The rankings are shown in Figure 7.4. The critical distance was found to be equal to 0.2.

Based on Nemenyi's ranking algorithm, LSTM trained using ELM-rand without DL was the highest ranked algorithm, even though it did not always have the lowest error measure (as observed in Tables 7.7-7.17). LSTM trained using ELM-rand with DL came in second place. However, NARMAX trained using ELM-rand without DL was the worst algorithm. Furthermore, the difference between Elman trained using ELM-lin with DL and fully connected RNN trained using ELM-rand with DL was less than the critical distance which implies that difference is insignificant. Feedforward networks ranked fourth and ninth which implies that RNN architectures were more suitable for the databases given their time series properties. All algorithms joined by vertical lines in Figure 7.4 were not significantly better than each other. Furthermore, we notice that the fully connected RNN, which is the most general architecture (as discussed in section

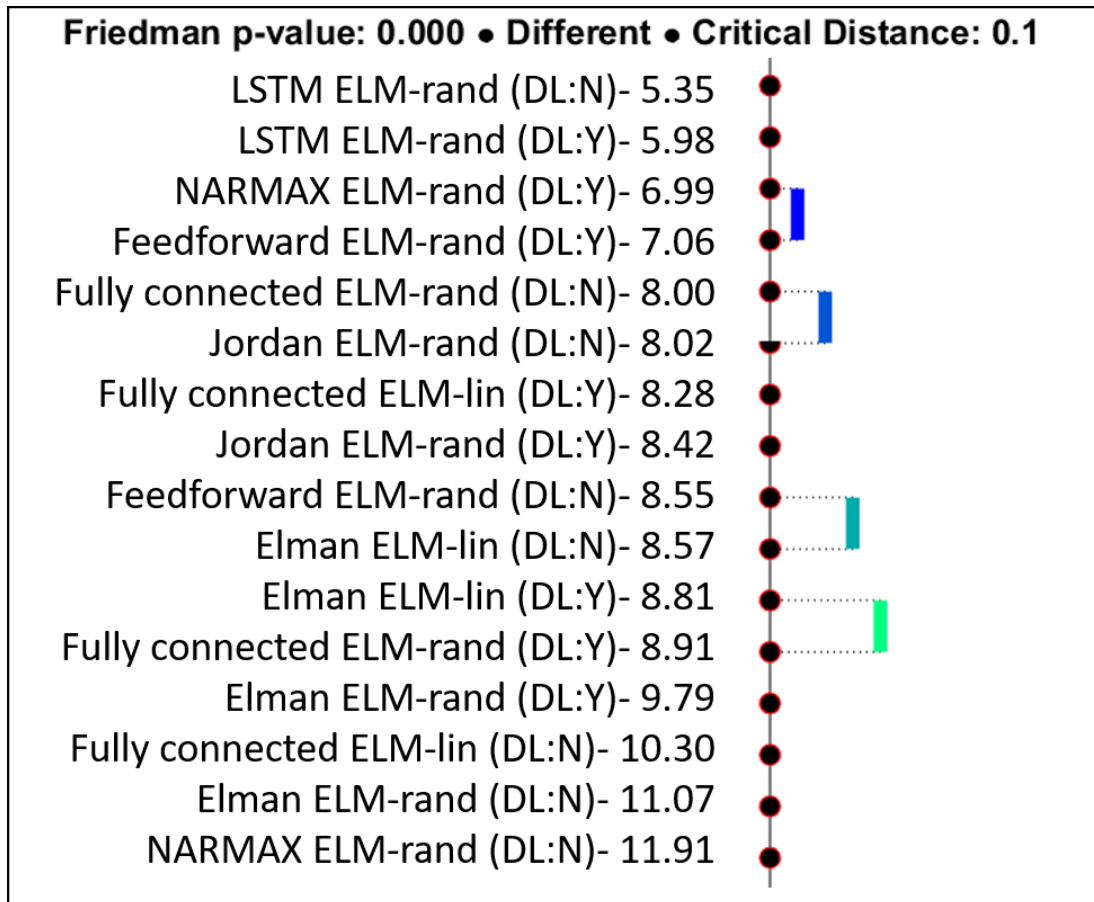


Figure 7.4: Non-iteratively-trained RNN variants ranking based on Nemenyi post-hoc test

7.4.2), is the most difficult to train since it ranked in the bottom half. Its special cases, obtained from fully connected RNN by pruning some connections, had smaller search spaces that allowed the convergence to better solutions on the given datasets. The rankings were based on the databases used in this work which may not have been suitable for the lower ranked architectures. One should keep in mind that these architectures may still be beneficial for other applications since research has shown that the optimal network architecture is data dependent.

7.5.4 Parameter Sensitivity Analysis

Table 7.18 compares the performance of non-iteratively-trained RNN when different activation functions are used. The number of hidden neurons was fixed to 20 for this analysis. We notice that the sine activation function produced less ill-conditioned problems. Fully connected RNN trained using ELM-rand with a

sigmoid activation function would sometimes lead to an ill-conditioned H matrix, resulting in very large MSE. This problem was also aggravated when the number of hidden neurons increased. In addition, the linearized sigmoid activation function resulted in better RNN models than ELM-rand. The sine activation function exhibited the opposite behavior for some data sets. This could be due to the low order approximation and could be remedied by adopting higher order approximations. Furthermore, the approximation might not be valid for certain data distributions. This should also be considered in the approximation procedure.

Table 7.18: Effect of kernel choice on nRMSE, nMAE and training time

| Architecture | DL | Training Algorithm | Training Time (sec) | | nRMSE | | nMAE | |
|--------------------------|----|--------------------|---------------------|-------|---------|--------|---------|-------|
| | | | Sigmoid | Sine | Sigmoid | Sine | Sigmoid | Sine |
| Bebida | | | | | | | | |
| Elman | N | ELM-lin | 0.003 | 0.012 | 0.301 | 0.375 | 0.225 | 0.314 |
| Elman | N | ELM-rand | 0.045 | 0.030 | 0.283 | 1.959 | 0.208 | 1.730 |
| Elman | N | BP ^{TT} | 0.220 | 0.185 | 0.794 | 0.750 | 0.631 | 0.632 |
| Elman | Y | ELM-lin | 0.018 | 0.013 | 0.301 | 0.375 | 0.225 | 0.314 |
| Elman | Y | ELM-rand | 0.032 | 0.012 | 0.295 | 0.337 | 0.215 | 0.251 |
| Fully connected | N | ELM-lin | 0.009 | 0.037 | 0.301 | 0.375 | 0.225 | 0.314 |
| Fully connected | N | ELM-rand | 0.029 | 0.030 | 3.457 | 2.544 | 0.768 | 2.398 |
| Fully connected | Y | ELM-lin | 0.022 | 0.050 | 0.301 | 0.375 | 0.225 | 0.314 |
| Fully connected | Y | ELM-rand | 0.007 | 0.003 | 0.372 | 0.348 | 0.242 | 0.252 |
| Jordan | N | ELM-rand | 0.006 | 0.009 | 0.419 | 2.461 | 0.315 | 2.430 |
| Jordan | Y | ELM-rand | 0.006 | 0.013 | 0.325 | 0.325 | 0.229 | 0.229 |
| LSTM | N | ELM-rand | 0.016 | 0.044 | 2.667 | 2.037 | 2.655 | 2.017 |
| NARMAX | N | ELM-rand | 0.003 | 0.012 | 0.513 | 2.436 | 0.442 | 2.392 |
| NARX | N | BP ^{TT} | 0.254 | 0.186 | 0.796 | 0.732 | 0.572 | 0.567 |
| Feedforward [379] | N | ELM-rand | 0.031 | 0.031 | 1.4441 | 2.585 | 0.8304 | 2.453 |
| Feedforward [377] | Y | ELM-rand | 0.030 | 0.046 | 0.311 | 0.434 | 0.226 | 0.294 |
| IPI | | | | | | | | |
| Elman | N | ELM-lin | 0.047 | 0.046 | 0.183 | 0.436 | 0.148 | 0.388 |
| Elman | N | ELM-rand | 0.032 | 0.032 | 0.219 | 1.067 | 0.185 | 0.810 |
| Elman | N | BP ^{TT} | 0.285 | 0.205 | 0.489 | 0.417 | 0.377 | 0.342 |
| Elman | Y | ELM-lin | 0.030 | 0.032 | 0.183 | 0.436 | 0.148 | 0.388 |
| Elman | Y | ELM-rand | 0.015 | 0.012 | 0.185 | 0.227 | 0.156 | 0.188 |
| Fully connected | N | ELM-lin | 0.016 | 0.050 | 0.183 | 0.436 | 0.148 | 0.388 |
| Fully connected | N | ELM-rand | 0.015 | 0.012 | 0.3775 | 2.207 | 2.608 | 2.026 |
| Fully connected | Y | ELM-lin | 0.016 | 0.062 | 0.183 | 0.436 | 0.148 | 0.388 |
| Fully connected | Y | ELM-rand | 0.031 | 0.012 | 0.1974 | 0.192 | 1.930 | 0.159 |
| Jordan | N | ELM-rand | 0.010 | 0.012 | 0.342 | 2.002 | 0.273 | 1.975 |
| Jordan | Y | ELM-rand | 0.006 | 0.010 | 0.208 | 0.208 | 0.170 | 0.170 |
| LSTM | N | ELM-rand | 0.016 | 0.041 | 1.728 | 1.699 | 1.703 | 1.675 |
| NARMAX | N | ELM-rand | 0.006 | 0.012 | 0.379 | 2.019 | 0.311 | 1.961 |
| NARX | N | BP ^{TT} | 0.212 | 0.186 | 0.606 | 0.570 | 0.513 | 0.458 |
| Feedforward [379] | N | ELM-rand | 0.016 | 0.012 | 0.402 | 0.3875 | 0.356 | 1.964 |
| Feedforward [377] | Y | ELM-rand | 0.050 | 0.047 | 0.240 | 0.4037 | 0.194 | 6.653 |
| Lavras | | | | | | | | |
| Elman | N | ELM-lin | 0.016 | 0.012 | 0.158 | 0.152 | 0.114 | 0.108 |
| Elman | N | ELM-rand | 0.031 | 0.047 | 0.176 | 0.254 | 0.122 | 0.182 |
| Elman | N | BP ^{TT} | 0.432 | 0.267 | 0.543 | 0.178 | 0.443 | 0.125 |
| Elman | Y | ELM-lin | 0.015 | 0.047 | 0.158 | 0.152 | 0.114 | 0.108 |
| Elman | Y | ELM-rand | 0.016 | 0.031 | 0.152 | 0.179 | 0.110 | 0.121 |
| Fully connected | N | ELM-lin | 0.034 | 0.037 | 0.158 | 0.152 | 0.114 | 0.108 |
| Fully connected | N | ELM-rand | 0.031 | 0.015 | 0.1604 | 0.264 | 0.117 | 0.191 |
| Fully connected | Y | ELM-lin | 0.037 | 0.050 | 0.158 | 0.152 | 0.114 | 0.108 |
| Fully connected | Y | ELM-rand | 0.031 | 0.012 | 0.237 | 0.180 | 0.126 | 0.124 |
| Jordan | N | ELM-rand | 0.009 | 0.031 | 0.166 | 0.250 | 0.118 | 0.180 |
| Jordan | Y | ELM-rand | 0.012 | 0.037 | 0.174 | 0.174 | 0.114 | 0.114 |
| LSTM | N | ELM-rand | 0.034 | 0.078 | 0.232 | 0.213 | 0.170 | 0.151 |
| NARMAX | N | ELM-rand | 0.009 | 0.053 | 0.197 | 0.251 | 0.152 | 0.181 |
| NARX | N | BP ^{TT} | 1.617 | 0.208 | 0.152 | 0.150 | 0.111 | 0.110 |
| Feedforward [379] | N | ELM-rand | 0.047 | 0.012 | 0.1345 | 0.257 | 0.0910 | 0.189 |
| Feedforward [377] | Y | ELM-rand | 0.0469 | 0.031 | 0.189 | 0.184 | 0.119 | 0.127 |
| Inverted Pendulum | | | | | | | | |
| Elman | N | ELM-lin | 0.016 | 0.019 | 0.074 | 0.075 | 0.044 | 0.046 |
| Elman | N | ELM-rand | 0.003 | 0.003 | 0.065 | 0.070 | 0.038 | 0.040 |
| Elman | N | BP ^{TT} | 1.788 | 1.780 | 0.083 | 0.117 | 0.047 | 0.068 |
| Elman | Y | ELM-lin | 0.025 | 0.016 | 0.074 | 0.075 | 0.044 | 0.046 |
| Elman | Y | ELM-rand | 0.009 | 0.003 | 0.066 | 0.067 | 0.039 | 0.039 |

| | | | | | | | | |
|-------------------|---|----------|-------|-------|-------|--------|-------|-------|
| Fully connected | N | ELM-lin | 0.568 | 0.512 | 0.074 | 0.075 | 0.044 | 0.046 |
| Fully connected | N | ELM-rand | 0.016 | 0.019 | 0.684 | 0.122 | 0.071 | 0.064 |
| Fully connected | Y | ELM-lin | 0.580 | 0.484 | 0.074 | 0.075 | 0.044 | 0.046 |
| Fully connected | Y | ELM-rand | 0.022 | 0.019 | 0.071 | 0.078 | 0.041 | 0.044 |
| Jordan | N | ELM-rand | 0.296 | 0.306 | 0.097 | 0.100 | 0.063 | 0.062 |
| Jordan | Y | ELM-rand | 0.321 | 0.312 | 0.078 | 0.083 | 0.047 | 0.043 |
| LSTM | N | ELM-rand | 1.822 | 2.065 | 0.099 | 0.096 | 0.064 | 0.059 |
| NARMAX | N | ELM-rand | 0.293 | 0.312 | 0.096 | 0.098 | 0.062 | 0.063 |
| NARX | N | BPTT | 1.837 | 2.012 | 0.013 | 0.014 | 0.005 | 0.005 |
| Feedforward [379] | N | ELM-rand | 0.006 | 0.003 | 0.008 | 0.033 | 0.003 | 0.012 |
| Feedforward [377] | Y | ELM-rand | 0.006 | 0.003 | 0.085 | 0.0801 | 0.041 | 0.502 |

Focusing on the effect of the number of hidden neurons on non-iteratively-trained RNN, Table 7.19 summarizes the obtained results for the inverted pendulum and electricity load data sets. A sigmoid activation function was adopted for this analysis. As expected, the computational cost of non-iteratively-trained fully connected RNN is greater than Elman RNN, especially as the number of hidden neurons increases. As previously mentioned, fully connected RNN with ELM-rand can result in ill-conditioned models that get worse as the network size increases. Comparing ELM-rand to ELM-lin, the former is less computationally expensive than the latter, since the output matrix and parameter vector are smaller in size. For example, considering the Electricity load balance data set, fully connected RNN with ELM-rand is 15% to 49% times faster than fully connected RNN with ELM-lin.

7.5.5 Online Learning: RLS versus Kaczmarz’s Approximation

For online learning applications, ELM networks are trained using RLS which updates the weights as input data is received in real-time or pseudo real-time. To compare the performance of RLS based ELM training and Kaczmarz’s approximation based training, we fixed the network architecture where the number of hidden neurons was set to 30 and a sigmoid activation function was adopted. The same initial random values were used for both RLS and Kaczmarz’s algorithm to have a fairer comparison.

As expected, the latter reduces the training time but at the cost of prediction accuracy. Table 7.20 reports on the average reduction in training time and average increase in MSE when using Kaczmarz’s approximation as opposed to RLS for our proposed RNN-NIPR architectures, in addition to feedforward ELM and JRNN-NIPR. For example, FRNN-NIPR-lin greatly benefited from this approximation, since it averaged 98% reduction in training time when using Kaczmarz’s approximation instead of RLS. For some problems, a negative increase in MSE was observed, i.e. Kaczmarz’s approximation returned a lower MSE than RLS which could be a result of the data distribution or initial conditions. Similarly, a negative decrease in training time implies that Kaczmarz’s approximation training algorithm had a larger running time than the RLS algorithm. However, the

Table 7.19: Effect of number of hidden neurons on MSE and training time

| Architecture | DL | Training Algo- rithm | Number of Hid- den Neu- rons | Test | Training | Test | Training |
|-----------------|----|----------------------------|--|--------------------------|-------------------|------------|---------------|
| | | | | Set MSE | Time (sec) | Set MSE | Time (sec) |
| | | | | Electricity Load Balance | Inverted Pendulum | | |
| Elman | N | ELM-lin | 15 | 0.0686 | 0.720 | 0.0175 | 0.002 |
| | | | 20 | 0.0686 | 1.092 | 0.0175 | 0.004 |
| | | | 25 | 0.0686 | 1.320 | 0.0175 | 0.004 |
| | | | 30 | 0.0686 | 1.660 | 0.0175 | 0.008 |
| Elman | N | ELM-rand | 15 | 0.0694 | 0.204 | 0.0135 | 0.002 |
| | | | 20 | 0.0700 | 0.318 | 0.0135 | 0.016 |
| | | | 25 | 0.0700 | 0.492 | 0.0138 | 0.002 |
| | | | 30 | 0.0695 | 0.790 | 0.0132 | 0.006 |
| Fully connected | N | ELM-lin | 15 | 0.0686 | 14.258 | 0.0175 | 0.072 |
| | | | 20 | 0.0686 | 33.346 | 0.0175 | 0.258 |
| | | | 25 | 0.0686 | 59.298 | 0.0175 | 0.702 |
| | | | 30 | 0.0686 | 83.766 | 0.0175 | 2.646 |
| Fully connected | N | ELM-rand | 15 | 0.0691 | 1.056 | 0.0699 | 0.010 |
| | | | 20 | 0.0679 | 1.262 | 0.0141 | 0.010 |
| | | | 25 | 0.0686 | 1.476 | 0.0485 | 0.010 |
| | | | 30 | 0.0755 | 1.706 | 0.0167 | 0.014 |
| LSTM | N | ELM-rand | 15 | 2.0650 | 6.879 | 0.0314 | 1.323 |
| | | | 20 | 0.2729 | 13.167 | 0.0293 | 1.822 |
| | | | 25 | 0.1438 | 15.025 | 0.0318 | 2.493 |
| | | | 30 | 0.6983 | 18.798 | 0.0293 | 3.179 |

difference was not large (-1.123% for ERNN-NIPR-rand) and could be attributed to the random values chosen in the initialization. Despite the increase in MSE, Kaczmarz’s approximation may be beneficial when it is not feasible to compute the Hessian matrix due to computational resources or time constraints such as in real-time decision-making. Applications such as anytime control that only require an approximate solution can also benefit from this approach.

Table 7.21 summarizes the p-values and CDs of the statistical tests performed to compare RLS to Kaczmarz’s training algorithms. The results are averaged over all the databases included in this work. The results show that there is a statistical difference in the prediction values when using RLS and Kaczmarz, as expected, since $p < 0.05$. LSTM-NIPR and FF-ELM were exceptions where the Friedman test resulted in a p-value greater than 0.05 but the ANOVA test resulted in p-value less than 0.05. On the other hand, RVFL and some of its RNN variants did not show any statistical difference between RLS and its approximation. Furthermore, the p-values are less than the CDs on most tests. However, a statistical difference does not always imply a practical difference; Kaczmarz’s approximation can be used in applications where the discrepancy in accuracy can be tolerated to achieved reduced computational complexity.

7.6 Conclusion

In this paper, we proposed to train RNN using a non-iterative randomized algorithm for time series prediction and sequential decision-making problems, to reduce training time. We focused on Elman, Jordan, fully connected RNN, NAR-MAX and LSTM RNN architectures with and without DL. Since learning the recurrent connections for these architectures is a non-linear problem, we proposed two approaches to train RNN non-iteratively. The first approach randomly assigns the values of the recurrent connections while the second approach linearizes the activation function and learns these weights.

In addition to performing a theoretical computational complexity of various non-iteratively trained RNN architectures, experimental validation was performed on 14 publicly available time series prediction data sets and the double inverted pendulum POMDP sequential decision-making problem. Multiple error measures, training time and repeatability were compared to other methods in the literature including non-iteratively-trained Jordan RNN and feedforward ANN, BPTT trained RNN and SVR. Results showed that even though RNN trained using BPTT and SVR achieve lower MSE, non-iteratively-trained RNN require significantly less time to train. This tradeoff is especially beneficial in real-time decision-making problems on computationally challenged platforms such as in communication challenged robotics, i.e. robots in environments that prevent them from offloading their computations to the cloud. While the Nemenyi post hoc test showed the superiority of the proposed approaches compared to other

Table 7.20: RLS versus Kaczmarz's approximation

| Algorithm | Increase Testing (%) | in MSE | Decrease Training (%) | in Time |
|---------------------------------|-------------------------------------|-------------------|--------------------------------------|--------------------|
| Inverted Pendulum | | | | |
| Elman RELM-lin | 45.648 | | 57.143 | |
| Elman RELM-rand | 1.641 | | 35.556 | |
| Fully connected RELM-lin | 114.065 | | 99.488 | |
| Fully connected RELM-rand | 1.249 | | 39.216 | |
| Jordan RELM | 1.024 | | 31.481 | |
| NARMAX RELM | 0.419 | | 33.929 | |
| FF-ELM | 0.037 | | 35.417 | |
| Quebec Births | | | | |
| Elman RELM-lin | 66.162 | | 57.143 | |
| Elman RELM-rand | 102.171 | | 45.946 | |
| Fully connected RELM-lin | 30.105 | | 99.500 | |
| Fully connected RELM-rand | 486.494 | | 33.333 | |
| Jordan RELM | 123.489 | | 29.545 | |
| NARMAX RELM | -16.977 | | 36.957 | |
| FF-ELM | 50.235 | | 29.730 | |
| Santa Fe Laser | | | | |
| Elman RELM-lin | 43.656 | | 60.000 | |
| Elman RELM-rand | 102.363 | | 92.718 | |
| Fully connected RELM-lin | -9.142 | | 99.386 | |
| Fully connected RELM-rand | -23.129 | | 48.485 | |
| Jordan RELM | 744.342 | | 39.394 | |
| NARMAX RELM | 54.420 | | 47.368 | |
| FF-ELM | -67.924 | | 46.667 | |
| Electricity Load Balance | | | | |
| Elman RELM-lin | -5.029 | | 4.838 | |
| Elman RELM-rand | 71.463 | | -1.123 | |
| Fully connected RELM-lin | 15.863 | | 97.481 | |
| Fully connected RELM-rand | 12307.925 | | 1.111 | |
| Jordan RELM | 12.477 | | 2.065 | |
| NARMAX RELM | 1162.280 | | 1.754 | |
| FF-ELM | 480.007 | | 5.847 | |

Table 7.21: Statistical comparison of RLS and Kaczmarz’s approximation

| Algorithm | Friedman Test | | ANOVA Test | |
|-------------------------------------|---------------|----------|------------|-----------|
| | p-value | CD | p-value | CD |
| Elman RELM-rand | 6.65E-26 | 2.13E-02 | 2.89E-14 | 3.95E-01 |
| Elman RELM-lin | 1.57E-25 | 1.19E-02 | 4.68E-02 | 2.31E+01 |
| Fully connected RELM-rand | 1.27E-17 | 2.04E-02 | 7.03E-02 | 4.76E+02 |
| Fully connected RELM-lin | 9.42E-35 | 1.40E-02 | 5.03E-05 | 2.81E+01 |
| LSTM RELM | 1.54E-80 | 4.06E-03 | 9.35E-159 | 2.56E-01 |
| NARMAX RELM | 6.86E-02 | 4.55E-02 | 2.97E-02 | 9.49E+00 |
| Jordan RELM | 4.64E-03 | 6.86E-02 | 1.66E-05 | 9.73E+00 |
| FF-ELM | 5.34E-20 | 4.93E-02 | 1.72E-20 | 3.57E+00 |
| RVFL | 4.37E-02 | 2.03E-02 | 1.37E-01 | 2.83E+01 |
| Jordan Recurrent RVFL | 1.09E-01 | 7.48E-02 | 1.06E-01 | 7.12E+02 |
| Fully connected Recurrent RVFL-rand | 8.12E-02 | 8.02E-02 | 2.01E-01 | 3.14E+111 |
| Fully connected Recurrent RVFL-lin | 1.27E-24 | 1.46E-02 | 1.06E-01 | 2.15E+01 |
| Elman Recurrent RVFL-rand | 1.04E-02 | 7.66E-02 | 9.13E-02 | 1.15E+01 |
| Elman Recurrent RVFL-lin | 2.11E-27 | 2.28E-02 | 1.40E-01 | 2.36E+01 |

work in the literature, the proposed algorithms can be further improved to avoid diverging solutions when recurrent weights are badly initialized and assess its performance on deeper networks which will be subject of future work. Furthermore, future work will investigate the proposed algorithms in anytime control problems where the computationally efficient learner is used to provide a quick first guess and a more accurate learner enhances this guess.

In addition, non-iteratively-trained Elman RNN required less time to train than non-iteratively-trained fully connected RNN even though it performed worse. ELM-lin performed better than ELM-rand on most databases. This prompts the investigation of higher order approximations of the activation function which will be the subject of future work. Finally, LSTM did not perform well compared to other network architectures. This could be due to the size of the training data and high non-linearity of LSTM cells. Therefore, further research is necessary to optimize the non-iterative training algorithm for LSTM networks.

Furthermore, future work will also include the investigation of other activation functions and the practical implications of the theoretical model transformations on real-world problems. Methods to reduce the incurred loss of accuracy while maintaining the computational gains and LSTM with linearized activation functions will also be investigated.

Chapter 8

ANN Initialization: A Data Dependent Approach for Regression

The training algorithm proposed in Chapter 7 was sensitive to the random initialization that caused it to diverge for some databases. In this chapter¹, we investigate an initialization approach that depends on the training data, in an attempt to reduce ELM’s sensitivity to random initialization.

Specifically, we propose a context dependent input weight selection for regression ELM (CDR-ELM), which is a non-iterative training algorithm for offline supervised regression that computes the input weights and biases by clustering the training data and computing cluster head differences. CDR-ELM is compared to ELM and other algorithms on multiple publicly available regression databases. Experimental results show that the proposed algorithm produced comparable results to existing algorithms while outperforming ELM on some databases.

Next, section 8.1 presents the motivation and overview of the proposed algorithm before section 8.2 presents a survey of existing work on non-iterative training algorithms for ANN. Section 8.3 discusses the proposed methodology. Section 8.5 presents the experimental results before concluding in section 8.6.

8.1 Introduction

ANN have been applied to many artificial intelligence problems such as computer vision, speech recognition, and natural language processing, among others. However, they are mainly trained using iterative algorithms, resulting in slow training. ELM train feedforward ANN non-iteratively by randomly assigning in-

¹Rizk, Y., and Awad, M., “Context Dependent Input Weight Selection for Regression Extreme Learning Machines,” *Int. Conf. Artificial Neural Networks (ICANN) Proceedings (Vol. 10614)*, Springer, 2017.

put weights and biases, then solving a convex optimization function to find the output weights [379]. Their efficient training and good generalization has resulted in the application of ELM to many problems including feature extraction, time series prediction, classification, and regression [431]. However, randomly assigning weights without any considering the training data could result in sub-par performance. Exploiting the characteristics of the available data would produce a more specialized model for the current problem instead of a “one size fits all” approach. Therefore, it is worth investigating methods to extract additional information from this data to influence the choice of input weights and bias and improve ELM’s performance.

In this work, we propose a context dependent input weight selection for regression ELM (CDR-ELM) [432], which is a non-iterative training algorithm based on ELM for regression problems that computes the input weights and biases from the training data, instead of randomly assigning these values. While existing work has investigated explicitly computing these weights [433], [434], their algorithms mainly targeted supervised classification problem. The focus of this work is offline, supervised regression problems. First, the input data is clustered, then the difference between cluster heads is used as input weights. Biases are computed based on the size of the clusters. CDR-ELM’s training time and prediction error are compared to the performance of standard ELM [379], backpropagation trained ANN and Support Vector Regression (SVR) on multiple publicly available regression databases. Furthermore, we extended some of the existing algorithms that target classification problems to regression and compared to our proposed algorithm. Results showed that CDR-ELM outperformed ELM on some databases and achieved comparable results to other algorithms. Therefore, CDR-ELM is not suitable for very noisy data since the lack of randomness can lead to overfitting or for data that is sparsely distributed since the cluster heads will not faithfully represent the data.

8.2 Literature Review

To reduce the computational complexity of training ANN, randomized algorithms have been developed [406]. A subset of these randomized algorithms is non-iterative training algorithms that achieve significant speedups by computing network weights once. More specifically, Schmidt et al. [376] and Huang et al. [379] randomly assigned the input weights and biases and solving for the output weights in a single hidden layer feedforward network using Linear Least Squares (LLS). RVFL applied the same algorithm to a functional link network, a flattened ANN that allowed connections from the input layer to the output layer [377,409]. While initially formulated for offline, supervised, single hidden layer ANN learning, the LLS-based algorithm has been extended to semi-supervised and unsupervised learning [435,436], online learning [414] and deep learning [437–439], among oth-

ers. However, the input weights and biases are randomly sampled from a uniform distribution.

Some work has investigated constraining the sample space from which random weight vectors are selected to improve generalization while maintaining fast training. Zhu et al. randomly selected the input weight and bias vectors from a set of the normalized, pair-wise inter-class differences of training vectors [440]. This algorithm, termed constrained ELM (C-ELM), outperformed normalized ELM, deep ELM and Support Vector Machines (SVM) on multiple classification data sets. While training times were not reported, the authors noted that generating the input weight vectors was time consuming, especially as the number of hidden neurons increases. Liu et al. proposed a class-constrained ELM (C²ELM) which computed the input weight vectors using an ELM auto encoder (ELM-AE) [441]. An ELM-AE was trained for each class to learn the variance in the training data. The output layer weights of the ELM-AE were used as input layer weights for the ELM that will learn the classification model. The bias vectors were randomly assigned, as in ELM. The performance of C²ELM was compared to ELM, ELM-AE, CIW-ELM, C-ELM and receptive field ELM (RF-ELM). Results showed that C²ELM generally outperformed all the methods on MNIST and CIFAR-10 but achieved a slightly higher training time than ELM and C-ELM.

On the other hand, Tapson et al. proposed computed input weights ELM (CIW-ELM), where the input weights and biases were equal to a randomly weighted sum of training samples [433]. The output weights of the single hidden layer network were computed using the Moore-Penrose pseudoinverse. Training samples were normalized to have zero mean and unity standard deviation and the weights of the weighted sum were normalized to unity magnitude. CIW-ELM outperformed ELM on most classification databases and was significantly faster than MLP. The proposed algorithm was derived for classification problems but could be extended to regression by binning all samples into one class. However, the authors noted that this approach may not improve performance on regression problems. McDonnell et al. proposed multiple algorithms based on C-ELM and CIW-ELM for classification, namely receptive field ELM (RF-ELM) and 2-layered ELM [434]. Tissera et al. extended these algorithms presented in [434] to deep networks [439]. They proposed stacked auto encoders with a smaller number of hidden neurons [439]. Input weights were selected using C-ELM and trained each auto encoder separately. Cervellera et al. proposed to compute ELM's input weights and biases from low-discrepancy sequences (LDS) [442]. Henriquez et al. applied this method to parallel layer ELM (PL-ELM) [443]. These deterministic sequences cover a space without clustering or gaps. Figure 8.1 summarizes the presented literature.

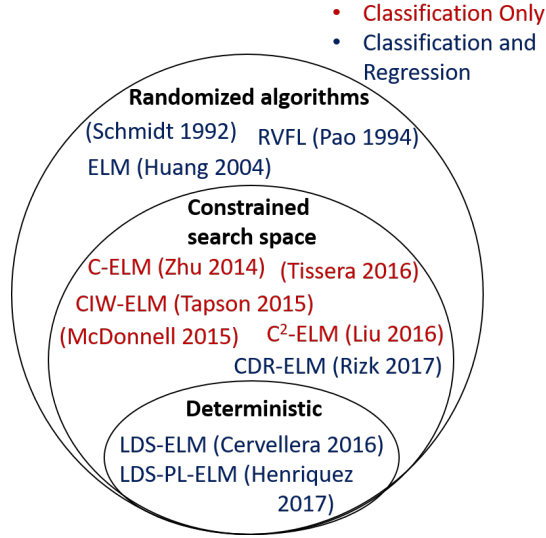


Figure 8.1: Summary of works in the literature

8.3 Proposed Methodology

8.3.1 Extreme Learning Machines Overview

ELM [379], shown in Figure 8.2, are single hidden layer feedforward neural networks trained in one step by randomly assigning input weights, w_i , and biases, b_i in (8.1) from a uniform distribution. This converts the problem of learning the output weights, β , to a linear problem since the output matrix, H , which contains the non-linear activation function no longer contains optimization variables and can be computed using (8.2) when the input, x , is available.

$$\hat{y} = H\beta = \sum_{i=1}^M \beta_i g(w_i^T x + b_i) \quad (8.1)$$

$$H = \begin{bmatrix} g(w_1^T x_1 + b_1) & \dots & g(w_M^T x_1 + b_M) \\ \vdots & \ddots & \vdots \\ g(w_1^T x_N + b_1) & \dots & g(w_M^T x_N + b_M) \end{bmatrix} \quad (8.2)$$

The generalized Moore-Penrose pseudoinverse is used to compute the output weights, $\beta = (H^T H)^{-1} H^T y$, where y is the target matrix or desired output in a supervised learning paradigm. Therefore, the ELM training algorithm performs the following steps:

1. Randomly assign values to the input weights and biases.
2. Compute the H matrix using (7.12).

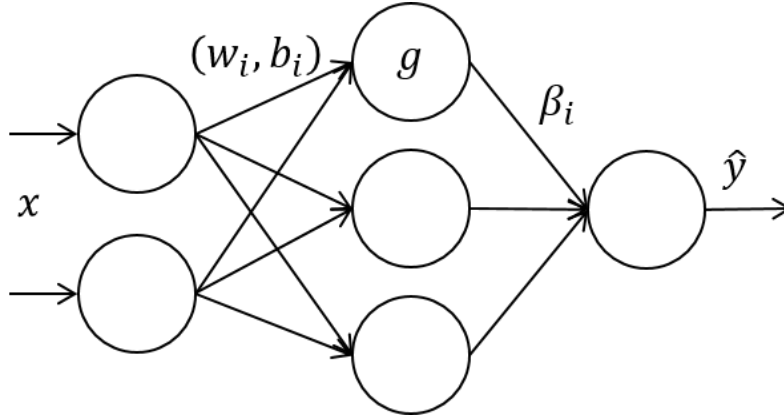


Figure 8.2: ELM Architecture

3. Compute the β vector: $\beta = (H^T H)^{-1} H^T y$.

Table 8.1 summarizes the nomenclature adopted in this paper.

8.3.2 Context Dependent ELM

Since it is not possible in regression to sample the space of inter-class difference vectors and assign input weight vectors, we propose to cluster the training data into a finite number of clusters based on the number of hidden neurons to determine the input weights and biases. Specifically, input weights are computed by taking the pair-wise difference of cluster heads. Biases are proportional to the cluster size.

1. Compute the number of clusters, P , such that $M = \frac{P^2 - P}{2}$.
2. Cluster the training data using k-means into P clusters.
3. Compute the cluster head of each cluster.
4. Compute the input weights using (8.3).
5. Compute the hidden neuron biases, $b_i = \frac{N_j}{N_k}$.
6. Train ELM.

$$w_i = \frac{|E[x_{\in cluster_i}] - E[x_{\in cluster_j}]|}{|E[y_{\in cluster_i}] - E[y_{\in cluster_j}]|} \quad (8.3)$$

Table 8.1: Nomenclature

| Variable | Definition |
|------------|---|
| x | Input matrix |
| y | Expected output |
| \hat{y} | Predicted output |
| $g(\cdot)$ | Activation function |
| w_i | Input weights vector |
| N_j | Number of training samples in cluster j |
| b_i | Neuron bias |
| M | Number of hidden layer neurons |
| β | Output weights vector |
| N | Number of training samples |
| H | Output matrix |
| F | Number of features in input vector |
| P | Number of clusters |

8.4 Theoretical Computational Complexity

Next, we discuss the computational complexity of determining the input weights and biases, summarized in Table 8.2. We do not derive the computational complexity of the remaining steps in the ELM algorithm since they are the same for all the ELM variants we are comparing.

The computational complexity of the proposed algorithm is based on the complexity of k-means, averaging cluster points and computing cluster head differences. The computational complexity of k-means is $O(N_{max}NP)$ where N_{max} is the number of iterations for k-means to converge. Computing the cluster heads requires computing P averages of $F \times 1$ vectors and M pair-wise differences. Averaging training samples in the cluster and computing differences to obtain the input weights requires $O(3MF)$, whereas computing the biases requires $O(2M)$ operations. Therefore, the total time complexity is $O(O(N_{max}NP + 3MF + 2M))$. The space complexity of the algorithm is $O(M(F + 1) + P(N + 1))$ to save the input layer weights, biases and cluster points and heads. Selecting a random value from a uniform distribution requires $O(1)$ computations and $O(1)$ memory. Considering a network with M hidden neurons and F -dimension input vectors, a total of MF input weights and M biases should be assigned leading to $M(F + 1)$ time and space complexity.

CIW-ELM computes the input weights and biases from the weighted sum of the training samples. A total of $M^2F(2N - 1) + 3F$ operations are required to compute the weighted sum ($W_{M \times F} = R_{M \times N}X_{N \times F}$) and normalize the resulting weights. For regression problems, we assume all points belong to one class. Furthermore, the algorithm formulation assumes the biases are part of the input

Table 8.2: Computational Complexity of the considered ELM variants

| Algorithm | Time Complexity | Space Complexity |
|---------------------|-----------------------------|--------------------------|
| CDR-ELM | $O(N_{max}NP + 3MF + 2M)$ | $O(M(F + 1) + P(N + 1))$ |
| ELM | $O(MF)$ | $O(MF)$ |
| CIW-ELM | $O(M^2FN + F)$ | $O(MF)$ |
| C-ELM | $O(MF)$ | $O(MF)$ |
| C ² -ELM | $O(NMF(F + c) + NMF)$ | $O(F^2M + 2MF + NMF)$ |
| LDS-Halton-ELM | $O(MF)$ | $O(MF)$ |
| LDS-Sobol-ELM | $O(F\lceil\log_2(M)\rceil)$ | $O(MF)$ |

matrix.

C-ELM requires the computation of M differences between two vectors of dimension $F \times 1$. Each difference requires F floating point operations. Furthermore, the resulting difference vectors are normalized by the magnitude of the difference which requires $((F + 1) + (F - 1))$ operations. Computing the bias consists of the difference the norms of two training vectors divided by the norm of the hidden neuron’s input weight vector; this results in $3(F + 1 + F - 1) + F$ operations. Therefore, a total of $9FM$ operations are required by C-ELM to assign input weight and bias values for a single hidden layer network with M hidden neurons. The space complexity equal to the number of weights and biases that should be computed.

C²ELM assigns the output weights of an ELM-AE to the input weights of an another ELM network. In the regression formulation, we assume only one ELM-AE is trained with M hidden neurons which requires $O(NM(F + c) + NMF)$ operations and $O(2FM + M + NM)$ of memory.

The computational complexity of PL-ELM depends on the computational complexity of the LDS generator. In this derivation, we assume that a PL-ELM network has a total of M hidden neurons per subnetwork. Therefore, a PL-ELM network contains $2MF$ input weights and $2M$ biases. A Sobol sequence generator requires $O(F\lceil\log_2(M)\rceil)$ to generate and $M \times F$ matrix [444], while a Halton sequence generator requires $O(MF)$ operations [445].

8.5 Empirical Validation

8.5.1 Experimental Setup

Multiple publicly available databases from the UCI machine learning repository [424], summarized in Table 8.3, were used to benchmark CDR-ELM in MATLAB 2016b on Intel Core i7 processor. Four-fold cross validation was adopted and the results of randomized algorithms were averaged over five independent runs. Mean

Table 8.3: Database Characteristics

| Database | Number of Instances | Number of Features | Output Mean | Output Standard Deviation | Output Range |
|----------|---------------------|--------------------|-------------|---------------------------|---------------|
| CASP | 45730 | 9 | 7.749 | 6.118 | [0, 20.9] |
| Concrete | 1030 | 8 | 35.818 | 16.706 | [2.3, 82.6] |
| Compress | | | | | |
| Housing | 506 | 13 | 22.533 | 9.197 | [5, 50] |
| Istanbul | 536 | 8 | 0.0016 | 0.016 | [-0.06, 0.07] |
| Stock | | | | | |
| Servo | 167 | 4 | 1.3897 | 1.559 | [0.13, 7.1] |
| Slump | 103 | 9 | 36.039 | 7.838 | [17.2, 58.5] |

square error (MSE), $E[(y - \hat{y})^2]$, and mean absolute error (MAE), $E[|y - \hat{y}|]$, were used to compare the performance of the various algorithms.

8.5.2 Regression Performance Analysis

CDR-ELM [432] is compared to backpropagation trained ANN (ANN-BP), LLS, SVR, standard ELM [379], and LDS-based ELM [442]. C-ELM [440], C²ELM [441], CIW-ELM [433] were extended to regression by considering that all training point belong to one class. A summary of the results is presented in Table 8.4. CDR-ELM achieved a lower testing MSE than ELM on the concrete compress, housing and slump data sets. For example, CDR-ELM achieved a 19.4% reduction in MSE compared to ELM but required twice as long to train the model. CDR-ELM also achieved 11% reduction in MAE compared to ELM on the housing data and was 1.23 times slower.

On the other hand, it did not perform well on Istanbul stock or CASP. For example, CDR-ELM resulted in a 5.8% MAE increase and 2.3 times slower than ELM. ANN-BP achieved the lowest MSE but was 107 times slower than CDR-ELM. While C²ELM achieved 28% decrease in MSE compared to CDR-ELM on the servo data set, it also required 2.4 times longer to train the model. In general, the additional computations required by CDR-ELM slowed down training but still resulted in faster training time than SVR, ANN-BP and CIW-ELM, especially on larger data sets.

Considering the repeatability of the various algorithms, we notice that CDR-ELM had a high variance, meaning it was sensitive to the initial conditions of the k-means algorithm. While it had the lowest variance on the slump data (25% lower than C-ELM and 50% lower than ELM), it had higher variances on the remaining data sets. For example, it had 60% higher variance than ELM. Note that the variance of CIW-ELM, LDS-Halton-ELM, LDS-Sobol-ELM, LLS

and SVR stems from averaging over the multiple folds. They were not run 5 independent times since they are not inherently randomized algorithms.

Table 8.4: Summary of results

| Algorithm | Hidden Neurons | Training Time (avg±std) (in seconds) | Testing MSE (avg±std) | Testing MAE (avg±std) |
|--------------------------|----------------|--------------------------------------|-----------------------|-----------------------|
| CASP | | | | |
| CDR-ELM [432] | 90 | 2.1305±0.3128 | 24.8556±0.6097 | 4.0432±0.0324 |
| ELM [379] | 150 | 0.9242±0.0579 | 22.7199±0.3307 | 3.8223±0.0264 |
| C ² ELM [441] | 70 | 1.0148±0.0720 | 25.1677±0.6189 | 4.0694±0.0395 |
| C-ELM [440] | 150 | 0.4102±0.0336 | 24.5077±0.2850 | 4.0338±0.0277 |
| CIW-ELM [433] | 70 | 9.3461±2.9974 | 24.8624±0.5008 | 4.0604±0.0388 |
| LDS-Halton-ELM [442] | 150 | 0.9883±0.0896 | 24.5895±0.3285 | 4.0319±0.0257 |
| LDS-Sobol-ELM [442] | 150 | 0.9648±0.0430 | 24.2259±0.2626 | 4.0027±0.0253 |
| ANN-BP | 70 | 228.666±98.64 | 20.0305±0.5991 | 3.4002±0.0980 |
| LLS | 0 | 0.0097±0.0002 | 86.9483±0.7996 | 7.8451±0.0541 |
| SVR | 0 | 65.4661±2.738 | 28.0038±0.3460 | 4.2246±0.0282 |
| Concrete Compress | | | | |
| CDR-ELM [432] | 130 | 0.0563±0.0199 | 41.9042±6.7595 | 4.6234±0.3062 |
| ELM [379] | 130 | 0.0203±0.0203 | 51.7742±5.8053 | 5.4326±0.2202 |
| C ² ELM [441] | 150 | 0.0914±0.0239 | 41.4455±5.7780 | 4.6178±0.2103 |
| C-ELM [440] | 150 | 0.0219±0.0179 | 69.7120±12.9374 | 6.5284±0.6144 |
| CIW-ELM [433] | 70 | 0.0164±0.0193 | 73.4289±3.3873 | 6.6319±0.1741 |
| LDS-Halton-ELM [442] | 130 | 0.0234±0.0156 | 46.4158±5.0346 | 5.0230±0.1089 |
| LDS-Sobol-ELM [442] | 110 | 0.0078±0.0156 | 53.6112±4.9201 | 5.4114±0.1445 |
| ANN-BP | 30 | 0.3256±0.0421 | 41.5115±14.3298 | 4.4741±0.2693 |
| LLS | 0 | 0.0017±0.0024 | 1426.128±27.77 | 36.1824±0.299 |
| SVR | 0 | 0.0431±0.0075 | 114.60±11.56 | 8.2306±0.2728 |
| Housing | | | | |
| CDR-ELM [432] | 50 | 0.0203±0.0210 | 18.4246±7.3442 | 2.7149±0.2620 |
| ELM [379] | 110 | 0.0164±0.0172 | 19.6721±5.1287 | 3.0349±0.2091 |
| C ² ELM [441] | 30 | 0.0328±0.0123 | 21.7419±4.6381 | 3.1992±0.1843 |
| C-ELM [440] | 150 | 0.0164±0.0172 | 18.0919±5.4313 | 2.8219±0.2333 |
| CIW-ELM [433] | 70 | 0.0125±0.0180 | 22.0912±6.4946 | 3.0561±0.2640 |

| Algorithm | Hidden Neurons | Training Time (avg±std) (in seconds) | Testing MSE (avg±std) | Testing MAE (avg±std) |
|--------------------------|----------------|--------------------------------------|------------------------|-----------------------|
| LDS-Halton-ELM [442] | 110 | 0.0273±0.0197 | 20.6313±1.6359 | 3.1805±0.1576 |
| LDS-Sobol-ELM [442] | 30 | 0.0273±0.0197 | 29.1856±9.0897 | 3.5612±0.2833 |
| ANN-BP | 30 | 0.3897±0.0786 | 24.2276±9.2753 | 3.2496±0.4588 |
| LLS | 0 | 0.0005±0.0001 | 562.913±18.51 | 23.0678±0.3198 |
| SVR | 0 | 0.0222±0.0077 | 26.1085±7.1033 | 3.2581±0.2992 |
| Istanbul Stock | | | | |
| CDR-ELM [432] | 30 | 0.0281±0.0207 | 0.0002±3.23E-5 | 0.0094±0.0008 |
| ELM [379] | 50 | 0.0086±0.0186 | 3.29E-5±8.25E-6 | 0.0043±0.0005 |
| C ² ELM [441] | 30 | 0.0555±0.0266 | 2.73E-5±5.34E-6 | 0.0040±0.0004 |
| C-ELM [440] | 150 | 0.0172±0.0202 | 2.8E-5±6.65E-6 | 0.0040±0.0005 |
| CIW-ELM [433] | 30 | 0.0063±0.0155 | 0.0001±4.63E-5 | 0.0065±0.0017 |
| LDS-Halton-ELM [442] | 30 | 0.0391±0.0271 | 4.71E-5±2.15E-5 | 0.0050±0.0008 |
| LDS-Sobol-ELM [442] | 70 | 0.0000±0.0000 | 7.36E-5±2.35E-5 | 0.0061±0.0006 |
| ANN-BP | 30 | 0.2451±0.0413 | 3.56E-5±9.56E-6 | 0.0044±0.0004 |
| LLS | 0 | 0.0003±0.0001 | 2.79E-5±5.63E-6 | 0.0040±0.0003 |
| SVR | 0 | 4.9083±0.4293 | 2.52E-5±5.27E-6 | 0.0038±0.0003 |
| Servo | | | | |
| CDR-ELM [432] | 50 | 0.0125±0.0180 | 0.7058±0.2495 | 0.5932±0.0942 |
| ELM [379] | 50 | 0.0055±0.0170 | 0.6053±0.1557 | 0.5186±0.0669 |
| C ² ELM [441] | 30 | 0.0297±0.0086 | 0.5024±0.1656 | 0.4870±0.0584 |
| C-ELM [440] | 130 | 0.0125±0.0165 | 0.7106±0.2014 | 0.6008±0.0815 |
| CIW-ELM [433] | 30 | 0.0000±0.0000 | 3.0099±1.9306 | 1.1602±0.1719 |
| LDS-Halton-ELM [442] | 30 | 0.0156±0.0180 | 0.9054±0.2479 | 0.7113±0.0847 |
| LDS-Sobol-ELM [442] | 30 | 0.0133±0.0198 | 1.0155±0.4015 | 0.7254±0.1471 |
| ANN-BP | 30 | 0.1695±0.0143 | 1.0309±0.3729 | 0.7044±0.1777 |
| LLS | 0 | 0.0044±0.0046 | 3.4718±0.2580 | 1.4536±0.1045 |
| SVR | 0 | 0.0125±0.0018 | 1.8725±0.4575 | 0.7644±0.1016 |
| Slump | | | | |
| CDR-ELM [432] | 30 | 0.0102±0.0170 | 6.0396±1.9512 | 1.8471±0.3334 |
| ELM [379] | 150 | 0.0047±0.0144 | 7.4950±3.9178 | 1.8282±0.4218 |
| C ² ELM [441] | 150 | 0.0375±0.0147 | 7.6987±4.5734 | 1.8769±0.3966 |
| C-ELM [440] | 150 | 0.0094±0.0147 | 8.1308±2.6088 | 2.1849±0.2985 |
| CIW-ELM [433] | 30 | 0.0047±0.0144 | 10.1248±4.0657 | 2.3593±0.5033 |

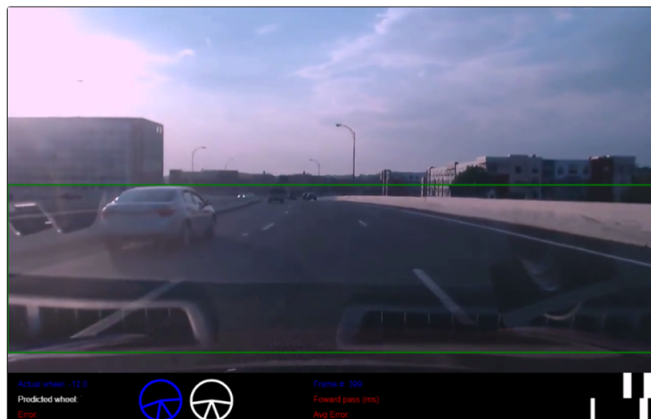


Figure 8.3: Deep Tesla Data Sample (adapted from [1])

| Algorithm | Hidden Neurons | Training Time (avg±std) (in seconds) | Testing MSE (avg±std) | Testing MAE (avg±std) |
|----------------------|----------------|--------------------------------------|-----------------------|-----------------------|
| LDS-Halton-ELM [442] | 30 | 0.0078±0.0156 | 9.4276±2.2604 | 2.3424±0.3589 |
| LDS-Sobol-ELM [442] | 150 | 0.0117±0.0234 | 12.6901±2.5740 | 2.4416±0.1993 |
| ANN-BP | 30 | 0.1639±0.0066 | 20.7354±8.4033 | 3.2563±0.7767 |
| LLS | 0 | 0.0335±0.0566 | 1800.08±160.86 | 41.1354±1.6454 |
| SVR | 0 | 0.0101±0.0017 | 7.1495±1.8567 | 2.1505±0.2915 |

8.5.3 Autonomous Car Steering Wheel Controller

Autonomous driving requires real-time decision making based on sensory input to avoid car accidents with potentially catastrophic consequences. Steering is one task of autonomous driving systems; the system takes as input an image of the road ahead, paying particular attention to its curvature, and determines the most appropriate steering wheel angle to avoid driving off the road or colliding with other vehicles on a multi-lane road.

Instead of designing a steering angle controller using traditional control theory approaches, we train ELM on the DeepTesla dataset [1] which provides the optimal input-output pairs. An example of the input and output of the DeepTesla data is shown in Figure 8.3. The steering angles ranged from -6 to 6 in increments 0.5. The input was RGB (red-green-blue) images with dimensions 1280×720 but their resolution was reduced by a factor of 64 to obtain 240 features. We took the first 1500 frames from the DeepTesla recordings. 80% of the data was used for training and 20% for testing.

The best algorithm was CDR-ELM which outperformed ELM by 43%, ANN by 23% and SVR by 85% while taking significantly less time to make a decision

Table 8.5: Deep Tesla Results

| Algorithm | Hidden Neurons | Training Time (in seconds) | Testing MSE | Testing MAE |
|----------------------|----------------|----------------------------|----------------|---------------|
| CDR-ELM [432] | 30 | 0.1594 | 10.9218 | 3.1950 |
| ELM [379] | 130 | 1.7899 | 15.5676 | 3.7846 |
| C-ELM [440] | 30 | 1.1936 | 11.9129 | 3.2962 |
| LDS-Halton-ELM [442] | 30 | 3.2966 | 10.9219 | 3.1950 |
| LDS-Sobol-ELM [442] | 30 | 3.2966 | 10.9219 | 3.1950 |
| ANN-BP | 30 | 467.00 | 13.4700 | 3.2100 |
| LLS | 0 | 0.0380 | 11.8000 | 2.7900 |
| SVR | 0 | 40.730 | 20.2200 | 3.6900 |

($11\times$, $2930\times$, $256\times$, less than ELM, ANN, and SVM, respectively). However, if we use the same number of neurons for both ELM and CDR-ELM, ELM is about 10 times faster. ANNs required approximately 25 epochs to converge.

8.6 Conclusion

In this paper, we presented a non-iterative training algorithm for supervised regression where input weights and biases are computed from clustered the training data instead of random assignment. Experimental results show that CDR-ELM outperforms ELM and other algorithms in the literature on some databases while slightly under performing on other databases. Even though the proposed algorithm achieved fast training and low error rates, it exhibited sensitivity to initial conditions of the clustering algorithm due to the high standard deviation when performing the repeatability analysis. Therefore, future work will investigate methods to reduce the its sensitivity to the initial conditions of the cluster algorithms.

Chapter 9

ANN Initialization: A Least Squares Approach

In Chapter 8, we proposed a data dependent initialization algorithm for non-iteratively trained ANN. Iteratively-trained ANN have outperformed their non-iteratively trained counterparts but have also been sensitive to initialization conditions. In this chapter, we propose to combine both training approaches by replacing random initialization in iteratively-trained ANN by the weights of a non-iteratively trained ANN.

Specifically, we investigate initializing iteratively-trained ANN using the non-iteratively trained ELM autoencoder algorithm, then train using backpropagation. Tested on multiple publicly available classification and regression datasets, we observed improvements in generalization error and training convergence. Next, section 9.1 motivates the proposed algorithm. Section 9.2 briefly summarizes work in the literature on initializing ANN, before section 9.3 introduces our proposed methodology. Sections 9.4 and 9.5 analyze the proposed approach from a theoretical and empirical perspective, respectively. Finally, section 9.6 concludes with some final remarks.

9.1 Introduction

ANN have been successfully applied to many pattern recognition, function estimation, control and optimization problems. Shallow beginnings morphed into deep architectures with millions of parameters that required hours of training on powerful clusters of GPUs and millions of training samples. Various training algorithms have been proposed to improve training efficiency; they consist of three main components: initialization, weight update rule and termination conditions. At one end of the spectrum is the backpropagation training algorithm [446] which iteratively learns randomly initialized weights. At the other end of the spectrum are non-iterative training algorithms such as randomized

ANN [376] and ELM [379] that learn network weights in one step. The former approach is most suitable for problems that require an accuracy prediction model and have the time and resources to train this model, while the latter is suitable for problems that require approximate solutions in at least pseudo-real time with minimal computational resources. An integral aspect of efficient training is good initialization that allows algorithms to converge faster to better models.

In this work, we investigate combining both ends of the spectrum by initializing the weights using a non-iterative training algorithm, then applying a few iterations of backpropagation to obtain a final model. Specifically, multi-layer ANN are initialized by applying the ELM autoencoders based on least-squares [447], [438] to compute the initial weight values of all layers. Then, a traditional backpropagation algorithm updates the weights. The proposed algorithm is compared to iterative and non-iterative training algorithms from the literature on multiple publicly available classification and regression datasets. Experiments show that the proposed algorithm achieved comparable accuracies while converging faster.

9.2 Related Work

The many ANN generations, from the shallow beginnings to the current deep architectures, have been mainly initialized by randomly sampling from the weight space. With the large influence of initialization on convergence, many researchers have investigated methods to initialize ANN which can be mainly categorized into two bins: 1) computing the weights or pretraining, and 2) randomly sampling weights from a constrained space.

Jammett et al. [448] used interval arithmetic to find the best interval for each weight instead of solving for the weight value. Le et al. [449] initialized the recurrent connections using a scaled identity matrix. Chen et al. [450] pretrained deep conditional random fields by first training restricted Boltzmann machines. Pretraining is a common initialization method adopted in deep learning where the network is first trained from scratch before network parameters are fine-tuned. Pretraining is generally done in an unsupervised fashion and has been proven effective in efficiently training deep networks [451].

Input weights have been selected from a constrained weight space that consists of the set of the normalized, pair-wise inter-class differences of training vectors [440] or using ELM autoencoders when training ANN non-iteratively. Such networks have been also initialized by computing the weights using various methods such as setting the input weights equal to the random weighted sum of the training samples [433]. Input weights were also sampled from low-discrepancy sequences, which are deterministic sequences that sample a space without clustering or gaps [442]. The difference between cluster heads was used in [432] to initialize the input layer weights for regression problems.

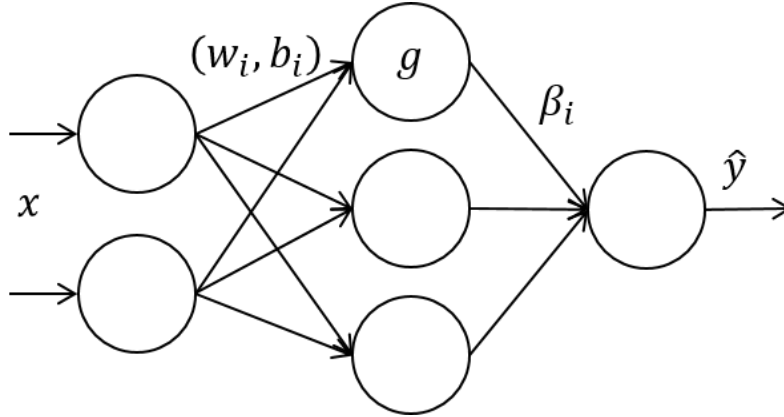


Figure 9.1: ELM Architecture

9.3 Methodology

Before discussing the details of the proposed algorithm, we briefly present an overview of ELM and backpropagation.

9.3.1 Extreme Learning Machines Overview

ELM are single hidden layer feedforward neural networks, as shown in Figure 9.1, trained in one step by randomly assigning input weights, w_i , and biases, b_i in (9.1) from a uniform distribution [379]. As a result, learning the output weights, β , becomes a linear and convex optimization problem. This is due to the fact that the non-linear output matrix, H , does not contain optimization variables (w_i and b_i are randomly assigned) and can be computed using (9.2) when the input, x , is available.

$$\hat{y} = H\beta = \sum_{i=1}^M \beta_i g(w_i^T x + b_i) \quad (9.1)$$

$$H = \begin{bmatrix} g(w_1^T x_1 + b_1) & \dots & g(w_M^T x_1 + b_M) \\ \vdots & \ddots & \vdots \\ g(w_1^T x_N + b_1) & \dots & g(w_M^T x_N + b_M) \end{bmatrix} \quad (9.2)$$

The output weights are computed using the generalized Moore-Penrose pseudoinverse, $\beta = (H^T H)^{-1} H^T y$, where y is the true output or target for supervised learning. Therefore, the ELM training algorithm performs the following steps:

1. Randomly assign values to the input weights and biases.
2. Compute H using (7.12).

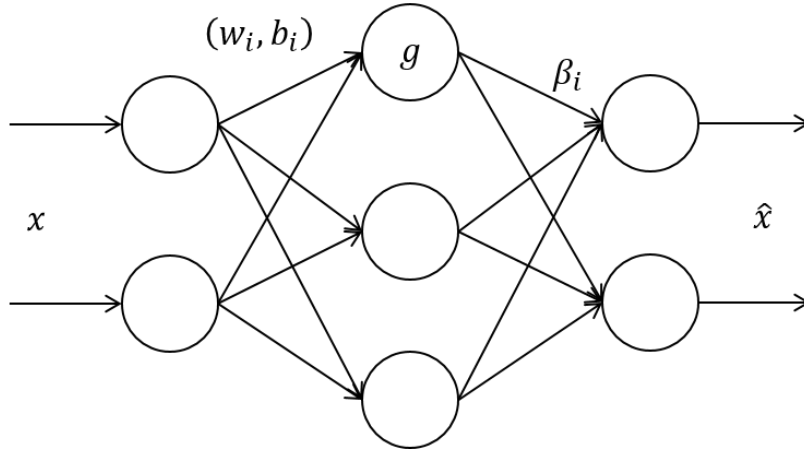


Figure 9.2: ELM Autoencoder Architecture

3. Compute β .

ELM were extended to multi-layer ANN by using ELM autoencoders that computed the weights of the hidden layers, then the output layer weights were computed using the single hidden layer approach described above [447], [438]. An ELM autoencoder is defined as a single hidden layer feedforward network that reconstructs the input, as shown in Figure 9.2. It randomly initializes the input layer weights and solves a least squares problem to find the output layer weights that minimize the error between the actual input, x , and the reconstructed input, \hat{x} . Each autoencoder extracts features from the input before the last layer learns a classification model in a supervised framework.

9.3.2 Backpropagation Overview

Backpropagation (hereafter referred to as BP-rand), the most common ANN training algorithm, was first proposed by Werbos (1974) [452], then further developed by Rumelhart, Hinton, and Williams (1986) [446]. It iteratively learns the network weights by minimizing the prediction error, which is shown in (9.3) and defined as the difference between the predicted and desired outputs. The algorithm relies on gradient descent (GD) to find the network weights that minimize this error measure. First, training data is propagated through the network to compute the predicted output, in what is called the feedforward phase. Then, in the feedback phase, the gradient of the error function is used to update the network weights, using the weight update rule in (9.4).

$$E = \sum_i e(n)^2 = \sum_i (\hat{y}(n) - y(n))^2 \quad (9.3)$$

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \frac{\partial E}{\partial w_{ij}} \quad (9.4)$$

In summary, the backpropagation training algorithm performs the following steps:

1. Randomly initialize all the weights.
2. While termination condition has not been met
 - (a) Feedforward phase: compute the predicted output by propagating the input through the network.
 - (b) Feedback phase: update the weights, using the weight update rule, based on the prediction error.

9.3.3 Least Squares Initialized Backpropagation Training Algorithm

While training ANN non-iteratively produces significant computational gains, such models still lag behind their iteratively trained counterparts when considering prediction accuracy. Therefore, we propose to initialize ANN using ELM’s non-iterative training algorithm, then fine-tune the network weights using backpropagation. This allows us to leverage both algorithms’ strengths: fast training time and good generalization performance.

If the ANN has multiple hidden layers, we initialize it using the multi-layer ELM algorithm [438]. First, the weights of each hidden layer are computed using an autoencoder trained using least squares. Next, the weights of the output layer are learned in a supervised fashion using least squares, as described in section 9.3.1. Finally, backpropagation is used to fine-tune the weights in the entire network. The workflow of the proposed algorithm, hereafter referred to as least squares backpropagation (BP-LS), is summarized in Table 9.1.

9.4 Theoretical Computational Complexity Analysis

The non-iterative algorithm consists of three main steps: (1) randomly assigning values to the input weights and biases, (2) computing the output matrix H and (3) solving for the output weights β , using the Moore-Penrose pseudoinverse. We consider a network with F input neurons, M hidden neurons and C output neurons trained on N input vectors.

Assigning random values requires a constant amount of time and the memory requirements are linear in the number of assigned values, equal to the sum of input

Table 9.1: BP-LS

1. For each hidden layer
 - a. Randomly initialize input weights of autoencoder
 - b. Compute the output weights of the autoencoder
 - c. Set the weights of the hidden layer equal to the output weights of the autoencoder
2. Compute H
3. Compute β
4. While termination condition has not been met
 - a. Feedforward phase
 - b. Feedback phase

layer weights and biases. The output matrix H is an $N \times M$ matrix and consists of multiplying F -dimensional input vectors by F -dimensional input weights and applying an activation function with a constant number of floating-point computations k . Therefore, computing the H matrix requires a total of $NM(F + k)$ computations. Finally, calculating the pseudoinverse of H and multiplying the result by the target output requires $O(N^3)$ operations when $M, C \leq N$.

On the other hand, testing requires computing the H matrix and multiplying it by β , equivalent $O(MC(F+k)+MC)$ computations for one test point, to find \hat{y} . Table 9.2 summarizes the time and space complexity of the non-iterative training algorithm. The space complexity indicates how many floating-point numbers are saved in memory.

Table 9.2: Computational complexity of a single hidden layer ELM

| | Time complexity | Space complexity |
|------------------------------------|----------------------|-----------------------|
| Randomly assign weights and biases | $O(FM + M)$ | $O(FM + M)$ |
| Compute H | $O(NM(F + k))$ | $O(NM)$ |
| Compute β | $O(N^3)$ | $O(MC)$ |
| Training Total | $O(NM(F + k) + N^3)$ | $O(FM + M + NM + MC)$ |

Initializing a single hidden layer network with random numbers requires $O(FM + M + MC)$ time and space complexity. If we consider a multi-layer ANN with L layers and M neurons per layer, L ELM autoencoders are needed and one supervised ELM. The computational complexity of an ELM autoencoder is the same as derived above, except that we replace C output neurons by F . The time complexity is $O((L + 1)(NM(F + k) + N^3))$ and the space complexity is $O(L(2FM + M + NM) + FM + M + NM + MC)$. Therefore, it is computational more expensive to initialize the network using ELM. However, with computa-

tions performed on GPUs, significant speedups can be achieved in future work to reduce the additional overhead of initializing ANN using ELM.

9.5 Empirical Validation

9.5.1 Experimental Setup

In this section, we evaluate BP-LS and compare it to its randomly initialized counterpart BP-rand. Experiments were run on a machine equipped with an Intel Xeon 64-bit 12-core processor, an NVIDIA GTX 1080 GPU and NVIDIA Tesla K20m GPU, with Windows 10 and RedHat 6.5 operating systems, respectively. The algorithms were written in Python based on the Tensorflow 1.3 package. Table 9.3 summarizes the properties of the adopted databases publicly available on the UCI Machine Learning Repository [424] and Kaggle [453]. We report the mean absolute error (MAE) and the testing accuracy for regression and classification datasets, respectively.

9.5.2 Performance Evaluation

Table 9.4 and 9.5 report on the performance of BP-LS compared to BP-rand, and ELM. Due to the random nature of the algorithms, we performed a repeatability analysis, where each experiment was repeated 5 times while fixing all hyperparameters. The average and standard deviations are reported. The termination condition stated that if the validation error did not vary for more than 5 epochs, the training terminated. We trained multiple network architectures and reported results of the best architecture. BP-LS converged to lower MAE and higher testing accuracies on most datasets but generally required more epochs to converge. This prevented BP-LS's initialization from getting stuck in local minima. For example, on Boston housing, BP-LS achieved an MAE 400 times better than BP-rand but required 11 times more epochs to converge. For the Combined cycle power plant dataset, BP-LS achieved an MAE that was 98% lower than BP-rand and only required one additional epoch. On average, for all the regression datasets, BP-LS achieved 98% reduction in MAE while requiring almost 4 times more epochs to converge. Considering the classification datasets, BP-LS achieved 13% higher accuracy while taking approximately twice as many epochs to converge as BP-rand. Compared to ELM, BP-LS achieved 30% lower MAE and 14% higher accuracy. As for the repeatability analysis, the standard deviation of BP-LS was generally lower than BP-rand, which implies that a good initialization helped ANN be less sensitive to randomly initialized weights.

Table 9.3: Database Description

| Database | Type | Number of Instances | Number of Features | Number of Classes | Output Range |
|--|----------------|---------------------|--------------------|-------------------|-----------------|
| Sports Articles for Objectivity Analysis [454] | Classification | 1000 | 59 | 2 | - |
| Diabetic Retinopathy Debrecen [424] | Classification | 1151 | 19 | 2 | - |
| Banknote Authentication [424] | Classification | 1372 | 4 | 2 | - |
| EEG Eye State [424] | Classification | 14980 | 14 | 2 | - |
| HTRU2 [424] | Classification | 17898 | 8 | 2 | - |
| Boston Housing [424] | Regression | 506 | 13 | - | [5, 50] |
| Airfoil Self-Noise [424] | Regression | 1503 | 5 | - | [103, 141] |
| Combined Cycle Power Plant [424] | Regression | 9568 | 3 | - | [420,496] |
| Appliances Energy Prediction [424] | Regression | 19735 | 27 | - | [10, 1080] |
| California Housing Prices [453] | Regression | 20640 | 8 | - | [14999, 500001] |

9.5.3 Hyperparameter Sensitivity Analysis

In this section, we consider the effect of the maximum number of epochs and the activation functions on the performance of the various algorithms. Tables 9.6 and 9.7 report the performance on some datasets when varying the activation functions of the various hidden layers. We notice that changing the activation function did not significantly affect the performance of BP-LS, especially for the classification dataset.

Next, instead of adopting a stopping criterion based on the validation error, we allow training to run for a pre-determined number of epochs. This allows us to study whether BP-rand and BP-LS can escape a local minimum if given enough time. Table 9.8 reports the results for two datasets using the best network architecture for each dataset (reported in Tables 9.4 and 9.5). Focusing on the classification dataset, we notice that BP-LS starts with an accuracy of 62.7% after 5 epochs of training and reaches 80% when trained for 100 epochs. However, BP-rand performs worse than BP-LS; it achieves 52.7% accuracy within 5 epochs, improves to 58.2% after 20 epochs, then begins to overfit after 100 epochs of training. For the regression dataset, BP-LS begins to overfit after 5 epochs but still outperforms BP-rand.

9.5.4 Training Error Convergence Analysis

Plotting the training error values as they change through the epochs allows us to see the effect of initialization on the speed of convergence and overall performance. Figure 9.3 plots the error curve, averaged over 50 iterations, for the best BP-LS and BP-rand models, for the *Sports Articles for Objectivity Analysis* classification database. We notice that, on average, both approaches start with an error that is almost equal. Both algorithms exhibit approximately the same standard deviation (approximately equal to 0.15). The error decreases faster in BP-LS compared to BP-rand, emphasizing the importance of a good initialization.

9.5.5 Weight Distribution Analysis

Next, we compare the network weight distribution of BP-rand and BP-LS. For this analysis, we train a network with 3 hidden layers and 10, 20, and 5 neurons per hidden layer on the *Sports Articles for Objectivity Analysis* classification database. Figure 9.4 and 9.5 plot the histograms of the network weights at initialization and after training the networks for 100 epochs, respectively. BP-LS's initial weights were slightly skewed towards negative values while the randomly initialized weights in BP-rand were skewed towards positive values. After 100 epochs of training, the distribution for both algorithms became centered around 0. BP-rand converged to a larger number of negative weights (less than -0.5) compared to BP-LS. Therefore, the initialization affected the evolution of the

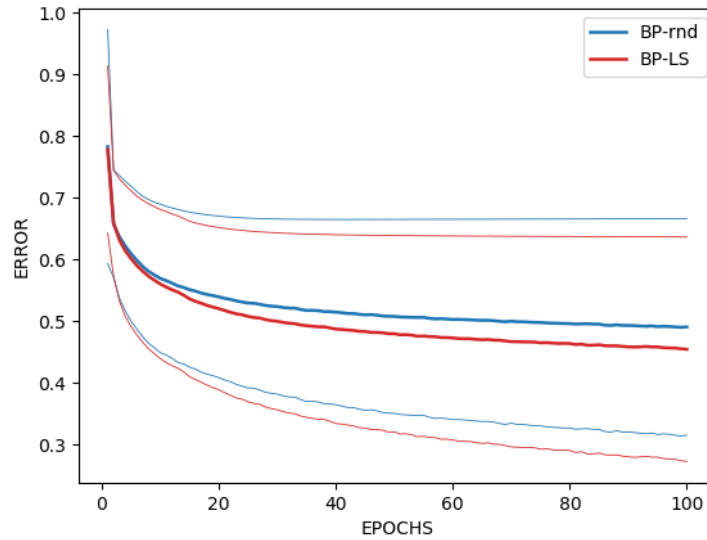


Figure 9.3: Training Error as a Function of Training Epochs for Sports Classification

network weights which led to a difference in performance, in favor of BP-LS.

9.6 Conclusion

The success of ANN led to its application in a large set of problems with a wide spectrum of computational resources. Since initialization greatly affects the quality of the prediction model, we propose, in this work, to initialize the multi-layer ANN using ELM's non-iterative least squares training algorithm. Then, we fine-tuned the network parameters using backpropagation. The theoretical computational complexity of the proposed algorithm was compared to that of randomly initialized backpropagation. Experiments on multiple classification datasets showed promising results that motivate further investigation. Future work will apply the proposed initialization algorithm to other types of ANN architectures such as recurrent neural networks and validate the approach on additional benchmarks.

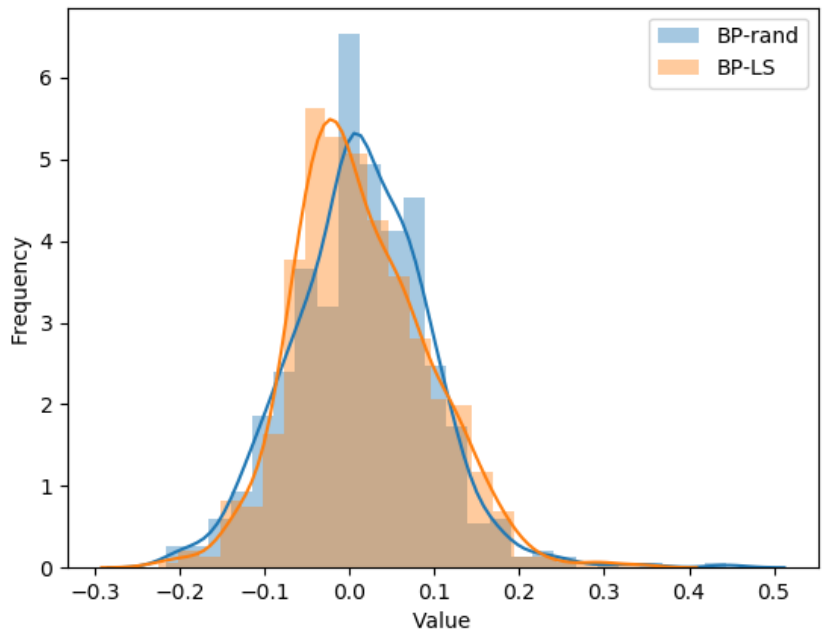


Figure 9.4: Weight Distribution of BP-rand and BP-LS after Initialization

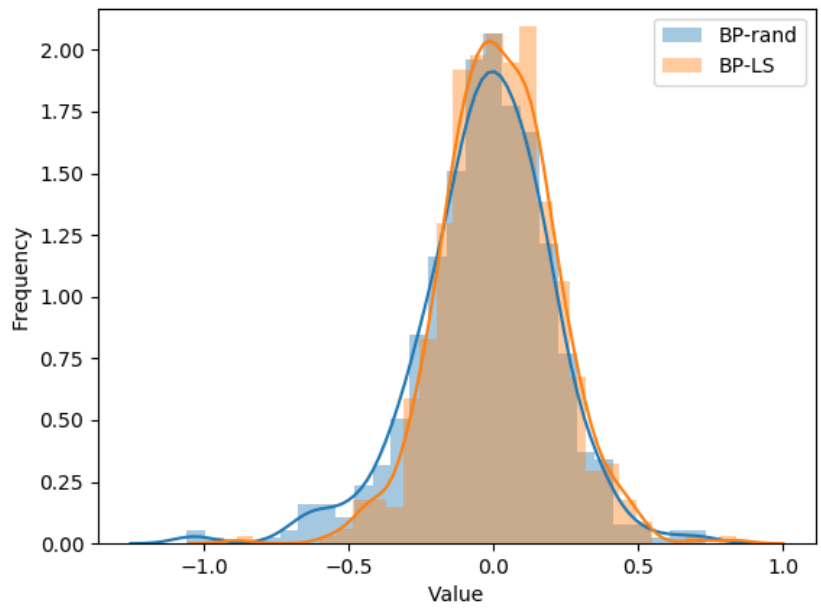


Figure 9.5: Weight Distribution of BP-rand and BP-LS after Training

Table 9.4: Performance of BP-rand and BP-LS on Regression Datasets

| Database | Algorithm | MAE (avg \pm std) | Epochs (avg \pm std) | Network Architecture |
|--------------------|-----------|----------------------------------|------------------------|----------------------|
| Boston Housing | ELM | 0.74 \pm 0.69 | - | [10, 10, 10] |
| | BP-LS | 0.09\pm0.006 | 93.2 \pm 30.9 | [10, 10, 10] |
| | BP-rand | 36.10 \pm 6.6 | 8.4 \pm 1.36 | [50, 10] |
| Airfoil Self-Noise | ELM | 0.70\pm0.71 | - | [10, 10, 10] |
| | BP-LS | 1.4 \pm 0.84 | 16 \pm 11.78 | [10, 10, 10] |
| | BP-rand | 38.0 \pm 23.08 | 10.2 \pm 2.64 | [10, 10, 10] |
| Combined Cycle | ELM | 2.46 \pm 3.08 | - | [10, 10, 10] |
| | BP-LS | 1.1\pm0.89 | 9.6 \pm 3.77 | [10, 10, 10] |
| Power Plant | BP-rand | 73.3 \pm 48.07 | 8.8 \pm 2.86 | [10, 10, 10] |
| Appliances | ELM | 0.97 \pm 0.86 | - | [10, 20, 5] |
| Energy Prediction | BP-LS | 0.05\pm0.004 | 26 \pm 20.76 | [10, 10, 10] |
| | BP-rand | 20.41 \pm 12.38 | 10.4 \pm 4.13 | [10, 10, 10] |
| California Housing | ELM | 0.64 \pm 0.94 | - | [10, 20, 5] |
| | BP-LS | 0.6\pm0.67 | 9.2 \pm 4.45 | [10, 10, 10] |
| Prices | BP-rand | 23.32 \pm 13.35 | 7.8 \pm 2.32 | [10, 10, 10] |

Table 9.5: Performance of BP-rand and BP-LS on Classification Datasets

| Database | Algorithm | Accuracy (avg \pm std) | Epochs (avg \pm std) | Network Architecture |
|---------------------------------|-----------|-------------------------------------|------------------------|----------------------|
| Sports Articles for Objectivity | ELM | 77.63 \pm 0.85 | - | [200, 50] |
| | BP-LS | 81.66\pm0.60 | 33 \pm 7.29 | [10, 50, 100] |
| Analysis | BP-rand | 61.96 \pm 2.09 | 18 \pm 12.85 | [10, 10, 10] |
| Diabetic Retinopathy | ELM | 53.97 \pm 3.81 | - | [10, 10, 10] |
| Debrecen | BP-LS | 70.23\pm1.15 | 1 \pm 0 | [10, 20, 5] |
| | BP-rand | 51.42 \pm 2.27 | 10.8 \pm 5.3 | [10, 10, 10] |
| Banknote Authentication | ELM | 61.86 \pm 8.15 | - | [10, 10, 10] |
| | BP-LS | 99.95 \pm 0.097 | 252.8 \pm 212.71 | [10, 20, 5] |
| | BP-rand | 100\pm0 | 77.2 \pm 154.4 | [10, 20, 5] |
| EEG Eye | ELM | 53.84 \pm 4.11 | - | [50, 10, 50, 15] |
| Stately | BP-LS | 55.13\pm8.9E-05 | 13.8 \pm 4.79 | [50, 10, 50, 15] |
| | BP-rand | 52.05 \pm 4.11 | 8.6 \pm 3.14 | [10, 10, 10] |
| HTRU2 | ELM | 94.01 \pm 2.91 | - | [10, 10, 10] |
| | BP-LS | 96.48\pm2.86 | 37.6 \pm 17.23 | [10, 20, 5] |
| | BP-rand | 90.76 \pm 0 | 9.8 \pm 2.48 | [50, 10, 50, 15] |

Table 9.6: Performance of BP-rand and BP-LS on Regression Datasets with Different Activation Functions

| Database | Activation function | Algorithm | MAE (avg \pm std) | Epochs (avg \pm std) |
|-------------------|---------------------|-----------|------------------------------------|------------------------|
| Boston Housing | elu-elu-lin | BP-LS | 0.41\pm0.8 | 100 \pm 67 |
| | elu-elu-lin | BP-rand | 1870 \pm 1176 | 10.8 \pm 6.4 |
| | tanh-elu-lin | BP-LS | 0.38\pm0.74 | 92 \pm 43 |
| | tanh-elu-lin | BP-rand | 769 \pm 164 | 98 \pm 1.6 |
| Airfoil | elu-elu-lin | BP-LS | 3.67\pm4.54 | 21 \pm 15 |
| Self-Noise | elu-elu-lin | BP-rand | 2105 \pm 2168 | 9.4 \pm 3.6 |
| | tanh-elu-lin | BP-LS | 1.97\pm1.82 | 1 \pm 0 |
| | tanh-elu-lin | BP-rand | 769 \pm 758 | 8.8 \pm 3.4 |
| Power Plant | elu-elu-lin | BP-LS | 1.84\pm2.59 | 10.6 \pm 2.8 |
| Appliances | elu-elu-lin | BP-rand | 6441 \pm 3482 | 8.8 \pm 2.1 |
| Energy | tanh-elu-lin | BP-LS | 1.17\pm1.26 | 9 \pm 4.24 |
| | tanh-elu-lin | BP-rand | 1487 \pm 825 | 11.2 \pm 3.18 |
| Energy Prediction | elu-elu-lin | BP-LS | 1.84\pm2.59 | 10.6 \pm 2.8 |
| | elu-elu-lin | BP-rand | 577 \pm 538 | 9.4 \pm 3.2 |
| | tanh-elu-lin | BP-LS | 0.008\pm0.0006 | 56.2 \pm 43.9 |
| | tanh-elu-lin | BP-rand | 12.57 \pm 10.03 | 11 \pm 5.25 |

Table 9.7: Performance of BP-rand and BP-LS on Classification Datasets with Different Activation Functions

| Network Architecture | Activation function | Algorithm | Accuracy (avg \pm std) | Epochs (avg \pm std) |
|----------------------|---------------------|-----------|-----------------------------------|----------------------------------|
| EEG Eye State | | | | |
| [10, 20, 5] | soft-sig-soft | BP-LS | 55.1\pm0.0001 | 1 \pm 0 |
| [50, 10, 50, 15] | soft-sig-soft | BP-rand | 53.7 \pm 0.025 | 1 \pm 0 |
| [10, 20, 5] | elu-elu-sig | BP-LS | 55.1\pm0.0001 | 1 \pm 0 |
| [50, 10, 50, 15] | elu-elu-sig | BP-rand | 53.7 \pm 0.025 | 1 \pm 0 |
| HTRU2 | | | | |
| [10, 20, 5] | soft-sig-soft | BP-LS | 97.8 \pm 0.0002 | 59.8\pm 31.9 |
| [10, 20, 5] | soft-sig-soft | BP-rand | 97.8 \pm 0.0001 | 66 \pm 34.3 |
| [10, 20, 5] | elu-elu-sig | BP-LS | 97.8\pm0.0006 | 48 \pm 16 |
| [50, 10, 50, 15] | elu-elu-sig | BP-rand | 90.7 \pm 0 | 8.2 \pm 3.3 |

Table 9.8: Effect of Epochs on Performance

| Database | Number of Epochs | BP-rand | BP-LS |
|--|------------------|---------------------|-------------------|
| Accuracy (avg \pm std)(%) | | | |
| Sports Articles for Objectivity Classification | 5 | 52.7 \pm 13.4 | 62.7 \pm 17.8 |
| | 10 | 52.7 \pm 13.4 | 67.2 \pm 16.9 |
| | 20 | 58.2 \pm 10.9 | 81.6 \pm 1.0 |
| | 50 | 52.7 \pm 13.4 | 58.8 \pm 19.2 |
| | 100 | 52.7 \pm 13.4 | 80.3 \pm 0.3 |
| MAE (avg \pm std) | | | |
| Boston Housing | 5 | 61.081 \pm 21.411 | 0.170 \pm 0.053 |
| | 10 | 58.758 \pm 21.276 | 0.840 \pm 1.234 |
| | 20 | 59.206 \pm 30.148 | 0.471 \pm 0.437 |
| | 50 | 55.990 \pm 16.736 | 0.595 \pm 0.590 |
| | 100 | 59.322 \pm 22.800 | 0.893 \pm 0.755 |

Chapter 10

ANN Regularization: An Information Theoretic Approach

Deep neural networks, like other machine learning algorithms, have suffered from overfitting. This causes reduced performance at testing time and wasted resources during training. In this work, we aim to reduce the probability of overfitting by pruning a trained fully connected network using information theoretic approaches including transfer entropy, Kullback-Leibler divergence and correlation. A two phase training approach is adopted. Once a dense network is trained using traditional methods, one of the aforementioned pruning criteria is computed for each pair of connected neurons in the network. Connections with a metric value below a predefined threshold are pruned. Then, the pruned network is trained to fine tune the remaining weights. The metrics are compared on multiple classification benchmarks from the literature that motivated follow on research. Next, we motivate our approach in section 10.1. Then, we briefly present existing work on DL regularization in section 10.2 before discussing our proposed approach in section 10.3. An empirical evaluation of this approach is discussed in section 10.4, before we conclude with some final remarks in section 10.5.

10.1 Introduction

DL has achieved remarkable results in many applications such as object recognition, image captioning and automatic speech recognition due to the abundance of training data and computational resources. However, these Deep Neural Networks (DNN) are vulnerable to overfitting. Many regularization techniques for machine learning algorithms have been proposed to reduce the likelihood of overfitting and improve performance from penalizing complex models to stopping training early. Determining the best model complexity of DNN is still an open research problem, ensemble learning is expensive and monitoring validation set accuracy has not been enough to avoid overfitting. Two methods that have found

success in DL have been dropout which is essentially averaging multiple network models while training [455] and batch normalization which normalizes layer inputs [456].

In this work, we adopt a two-phase training approach. Instead of sparsifying the network based on the connection weights as in [457], we compare multiple information theoretic metrics to eliminate paths with low information propagation. Specifically, we greedily prune individual connections along a path with low transfer entropy, low correlation or low Kullback-Leibler (KL) divergence. Reducing the complexity of the model by pruning connections help the network avoid overfitting and hence has a regularizing effect. Empirical results on classification datasets showed that our proposed approach outperformed dropout [458] and batch normalization [456] regularization techniques while exhibiting similar performance to [457].

10.2 Literature Review

Overfitting severely undermines the exerted effort on training by preventing models to perform well on unseen data. DNN are especially vulnerable to overfitting because of the large number of parameters that is learned. One common approach to overcome overfitting in DL is dropout [455, 458]; during training, neurons are randomly (with a probability of 0.5) eliminated from the network at each iteration when a training sample is passed through the network. Such a training approach prevents neurons from co-adapting since a subset of the networks overall neurons are trained on a given input.

Many variants of dropout have been proposed in the literature. Instead of randomly selecting neurons to drop out of the network, [459] proposed to dropout individual connections, outperforming it on some datasets. Fraternal dropout apply the dropout approach to recurrent neural network to improve their performance by training two RNNs, each using a different dropout mask [460]. [461] adaptively modified the dropout hyper-parameter using the Rademacher complexity value. This approach outperformed the fix-valued dropout on MNIST, CIFAR-10 and text classification.

Dense-sparse-dense (DSD) neural networks [457] were trained using a three phase training algorithm where a dense network is trained first, then pruned based on weight values to produce a sparse network. This sparse network is further training before the network is made dense again and trained to obtain the final model. Tested image classification, caption generation and speech recognition using CNN, LSTM and RNN, DSD improved on state of the art results in all domains.

Another common approach is batch normalization which removes the mean and standard deviation of random batches of data from the input [456]. Furthermore, the inputs to the hidden layers were also normalized and improved

generalization performance. Batch normalization improved the generalization accuracy while reducing the number of iterations till convergence for feedforward networks. However, this approach did not result in similar gains for RNNs [462]. Furthermore, batch normalization was counterproductive when combined with weight decay (L_2 regularization) [463].

10.3 Methodology

We first present a brief overview of some information theoretic concepts used in this work, before presenting the proposed regularization algorithm.

10.3.1 Transfer Entropy Overview

In information theory, entropy is viewed as a measure of the amount of information conveyed by a discrete random variable and computed by (10.1) [464]. A larger entropy implies larger information content. When considering the relationship between two random variables, mutual information (MI) quantifies the mutual dependence between these random variables, using (10.2) for discrete cases. Correlation is also a measure of dependence but is considered a special case of MI that only applies to continuous random variables. The correlation is computed using (10.3).

$$H(X) = - \sum_{x \in X} p(x) \log(p(x)) \quad (10.1)$$

$$I(X; Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log\left(\frac{p(x, y)}{p(x)p(y)}\right) \quad (10.2)$$

$$\rho_{X,Y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \quad (10.3)$$

Knowing one random variable (X), we sometimes want to quantify the information that can be inferred about a second random variable (Y). In such cases, conditional entropy is adopted and is based on the conditional probability distribution, as shown in (10.4). Similarly, the conditional MI quantifies the dependence between two random variables (X and Y) knowing a third (Z), as shown in (10.5). When considering time series random variables, transfer entropy denotes the amount of information transferred from one to another. More formally, transfer entropy measures information inferred about the time series random variable Y at time t (Y_t) knowing previous values of the Y ($Y_{t-1:t-T}$) and knowing the values of a second time series random variable X ($X_{t-1:t-T}$). As shown in (10.6), transfer entropy is equal to the conditional MI. Transfer entropy has been used in neuroscience to characterize the functional connectivity of the

brain [465], in artificial neural networks to insert feedback connections in the network [466] and other complex systems to estimate information flow [467].

$$H(Y|X) = - \sum_{x \in X, y \in Y} p(y, x) \log\left(\frac{p(y, x)}{p(x)}\right) \quad (10.4)$$

$$I(X; Y|Z) = \sum_{z \in Z} p(z) \sum_{x \in X} \sum_{y \in Y} p(x, y|z) \log\left(\frac{p(x, y|z)}{p(x|z)p(y|z)}\right) \quad (10.5)$$

$$T_{X \rightarrow Y} = H(Y_t|Y_{t-1:t-T}) - H(Y_t|Y_{t-1:t-T}, X_{t-1:t-T}) = I(Y_t; X_{t-1:t-T}|Y_{t-1:t-T}) \quad (10.6)$$

The KL divergence between two distributions X and Y is an asymmetric probability distance measure computed using (10.7).

$$D_{KL}(X||Y) = - \sum_i X(i) \log \frac{Y(i)}{X(i)} \quad (10.7)$$

10.3.2 Artificial Neural Network Model

We consider a fully connected multi-hidden layer feedforward artificial neural network (ANN) where the weights, w_j^i , and biases, b_j^i are learned, as shown in Figure 10.1. We assume $L - 1$ hidden layers ($i = 1, \dots, L$ with the $i = L$ representing the output layer) and each hidden layer contains M_i hidden neurons. The F -dimensional input to the network is denoted by (10.8) and the output of the intermediate hidden layers is denoted by (10.9). The output of the network, shown in (10.10), is assumed to belong to one of C labels.

We assume the output of each neuron in the network, excluding input and output layer neurons, is a random variable X that depends on the input to the network. For example, the output of neuron j in layer i is represented by the random variable h_j^i ; we have N samples of this random variable: h_{j1}^i to h_{jN}^i . Therefore, for a given set of inputs \mathcal{X} , the matrix \mathcal{H} holds the values of the M_i random variables corresponding to the neurons in hidden layer i .

$$\mathcal{X} = \begin{bmatrix} x_{11} & \dots & x_{iF} \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{NF} \end{bmatrix} \quad (10.8)$$

$$\mathcal{H}^i = \begin{bmatrix} h_{11}^i & \dots & h_{1M_i}^i \\ \vdots & \ddots & \vdots \\ h_{N1}^i & \dots & h_{NM_i}^i \end{bmatrix} \quad (10.9)$$

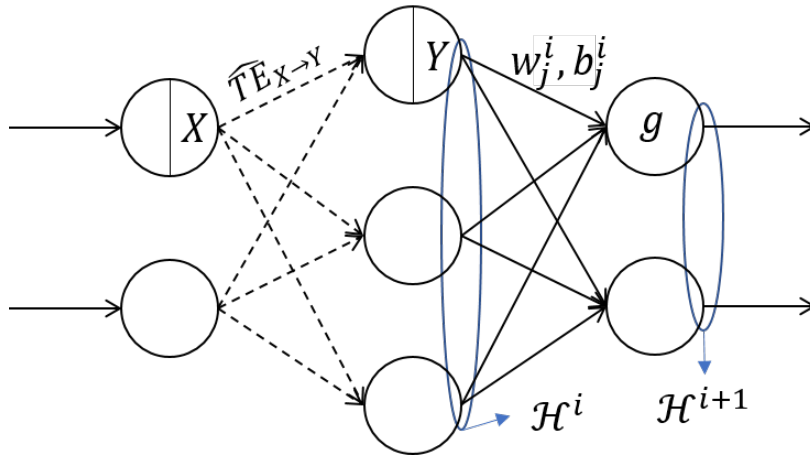


Figure 10.1: ANN Architecture

$$\mathcal{Y} = \begin{bmatrix} y_{11} & \cdots & y_{1C} \\ \vdots & \ddots & \vdots \\ y_{N1} & \cdots & y_{NC} \end{bmatrix} \quad (10.10)$$

10.3.3 Two-phase Training Algorithm

Many training algorithms have been proposed to train DNN while minimizing overfitting. One class of algorithms trains DNN in one phase where supervised or unsupervised learning is adopted. Another class of algorithms adopt a two-phase approach where pre-training is performed first then fine-tuning improves the network parameters. Recently, a three-phase approach was proposed by [457] where a dense, sparse then dense networks are trained sequentially. In our work, we propose a two-phase training algorithm that prunes network connections based on information theoretic metrics instead of the absolute values of the weights, as in [457]. Specifically, the steps in Table 10.1 are performed.

The value of n that determines the degree of sparsification is a hyper-parameter than can be tuned based on the prediction performance-computational complexity trade-off. If the network is overfitting, eliminating some connections would reduce the model complexity and help the network avoid overfitting while reducing the computational complexity. More details on the transfer entropy computations are presented in the next section.

Instead of deterministically pruning connections in the bottom n^{th} percentile, we also consider a pseudo-random pruning approach. Connections in the bottom n^{th} percentile are eliminated with a certain probability $p > 0.5$, i.e. a biased coin is flipped and the connection is eliminated with a probability p .

Table 10.1: Workflow of the Proposed Algorithm

-
1. Train a dense (fully connected) network.
 2. Create a sparse network:
 - a. Randomly sample a subset of the training data.
 - b. Extract the intermediary layer outputs for this subset to form the random variables associated with each neuron in the hidden layers.
 - c. Compute the transfer entropy, KL divergence or correlation of each pair of consecutive neurons.
 - d. Eliminate connections between neuron pairs with an information theoretic metric value in the bottom n^{th} percentile.
 3. Train the sparse network.
-

10.3.4 Information Theoretic Sparsification

Instead of pruning the network based on the values of the network weights [457], we compare multiple information theoretic metrics: 1) transfer entropy, 2) correlation, and 3) KL divergence.

Transfer entropy allows us to identify the network pathways with a high transfer of information. Computing the transfer entropy of all possible paths in the network by considering the non-consecutive source and destination neurons is computationally expensive, especially in deep networks with millions of network connections. Therefore, we approximate the transfer entropy of each path by the sum of transfer entropy of consecutive neurons. Therefore, if a network contains a total of $M = \sum_{i=1}^L M_i$ neurons, we are only required to compute $\sum_{i=1}^L M_i \times M_{i+1}$ instead of $M \times M$ values.

However, the computation of the transfer entropy for one neuron pair is computationally expensive since it requires the computation of the joint entropy and MI over multiple layers L . One approximation is to assume the Markov property where Y_i only depends on Y_{i-1} instead of $Y_{i-1:i-L}$ and X_{t-1} instead of $X_{i-1:i-L}$, as shown in (10.11) [468]. i represents the layer in which the neuron belongs to, i.e. the output (Y_i) of a neuron in layer i only depends on the output of a neuron in the previous layer ($i - 1$) and not all previous layers. Furthermore, since our random variables are continuous with unknown distributions, we need to approximate their transfer entropy using the kernel density estimation method [469].

$$T_{X \rightarrow Y} = H_{Y_i, Y_{i+1}} + H_{Y_i, X_i} - H_{Y_i} - H_{Y_i, Y_{i+1}, X_i} \quad (10.11)$$

Nevertheless, this is still computationally expensive. Therefore, we investigate adopting the correlation as a measure of dependency between two neurons instead of the transfer entropy. Computing the correlation is equivalent to the matrix dot product of the intermediate layer outputs, as shown in (10.12). \mathcal{H}^i represents the output of hidden layer i , as shown in Figure 10.1.

$$Cor(\mathcal{H}^i, \mathcal{H}^{i+1}) = (\mathcal{H}^i)^T(\mathcal{H}^{i+1}) \quad (10.12)$$

Finally, we also consider KL divergence since it is a non-symmetrical (directed) distance metric which may be a more accurate reflection of directed information flow in networks. We adopt the same workflow as for transfer entropy but compute KL divergence using (10.13).

$$D_{KL}(Y_{i+1}||X_i) = -Y_{i+1} \log \frac{X_i}{Y_{i+1}} \quad (10.13)$$

10.4 Empirical Validation

10.4.1 Experimental Setup

The algorithms, written in Python 2.7 using the Keras, Tensorflow and Java Information Dynamics Toolkit (JIDT) packages, were run on a machine with an Intel Xeon 64-bit 12-core processor and a Quadro K2000 NVIDIA GPU. Transfer entropy was computed using the kernel estimators implementation in Python [469].

The proposed algorithm is compared to other regularization techniques on multi-layer perceptrons (MLP) for supervised classification tasks. Specifically, the MNIST dataset with 60,000 training samples and 10,000 testing samples was adopted. The data consisted of feature vectors with a dimension of 784 that could belong to one of 10 possible classes. The skin segmentation dataset contains 245,057 samples represented by 4 features and belong to one of two classes. The Higgs Boson dataset contains 250,000 training instances, represented by 30 features, and 550,000 testing instances. Unless specified otherwise, a 4-hidden layer network architecture is adopted with 512 neurons per layer. In pseudo-random pruning, connections in the bottom n^{th} percentile are eliminated with a probability of $p = 0.85$.

10.4.2 Performance Analysis

Table 10.2 summarizes the performance of the pseudo-random pruning approach with the various pruning metrics. Our pruning metrics all lead to an improved prediction accuracy before (untrained sparse network) and after fine-tuning the weights while reducing the number of connections by an average of almost 50% across databases. While the accuracy increase for skin segmentation and MNIST were minor (0.18% and 0.5%, respectively), Higgs boson benefited from KL divergence with an 2.84% improvement in prediction accuracy.

Even though DSD achieved slightly higher accuracies (less than 0.3%), the reduction in network complexity was consistently lower than our proposed metrics. However, its computational complexity was significantly less than our proposed

Table 10.2: Pseudo-random Pruning Results

| Network | Training | Pruning | Sparsity | Epoch | Accuracy (%) | Training Time (sec) | Pruning Time (sec) | Connectivity Reduction (%) |
|--------------------------|---------------------|------------------|----------|-------|--------------|---------------------|--------------------|----------------------------|
| Skin Segmentation | | | | | | | | |
| Dense | Batch Normalization | - | 20 | 10 | 99.04 | 277.84 | 0.00 | 0.00 |
| Dense | Dropout | - | 20 | 10 | 99.75 | 195.18 | 0.00 | 0.00 |
| Sparse | - | Transfer Entropy | 20 | 10 | 99.78 | 0.00 | 314.19 | 48.76 |
| Sparse | Dropout | Transfer Entropy | 20 | 10 | 99.93 | 191.45 | 0.00 | 48.76 |
| Sparse | - | KL Divergence | 20 | 10 | 99.78 | 0.00 | 16532.61 | 43.74 |
| Sparse | Dropout | KL Divergence | 20 | 10 | 99.88 | 190.62 | 0.00 | 43.74 |
| Sparse | - | Correlation | 20 | 10 | 99.78 | 0.00 | 15.39 | 48.17 |
| Sparse | Dropout | Correlation | 20 | 10 | 99.89 | 192.07 | 0.00 | 48.17 |
| Sparse | - | Weights [457] | 20 | 10 | 99.78 | 0.00 | 0.02 | 20.00 |
| Sparse | Dropout | Weights [457] | 20 | 10 | 99.94 | 192.67 | 0.00 | 20.00 |
| Redense | Dropout | Weights [457] | 20 | 10 | 99.87 | 177.53 | 0.00 | 12.40 |
| Higgs Boson | | | | | | | | |
| Dense | Batch Normalization | - | 20 | 10 | 93.03 | 287.27 | 0.00 | 0.00 |
| Dense | Dropout | - | 20 | 10 | 95.50 | 203.49 | 0.00 | 0.00 |
| Sparse | - | Transfer Entropy | 20 | 10 | 95.52 | 0.00 | 313.53 | 73.45 |
| Sparse | Dropout | Transfer Entropy | 20 | 10 | 97.56 | 199.04 | 0.00 | 73.45 |
| Sparse | - | KL Divergence | 20 | 10 | 95.52 | 0.00 | 16063.55 | 64.44 |
| Sparse | Dropout | KL Divergence | 20 | 10 | 98.34 | 198.31 | 0.00 | 64.44 |
| Sparse | - | Correlation | 20 | 10 | 95.52 | 0.00 | 15.86 | 67.33 |
| Sparse | Dropout | Correlation | 20 | 10 | 95.55 | 200.34 | 0.00 | 67.33 |
| Sparse | - | Weights [457] | 20 | 10 | 95.52 | 0.00 | 0.02 | 20.00 |
| Sparse | Dropout | Weights [457] | 20 | 10 | 98.63 | 200.55 | 0.00 | 20.00 |
| Redense | Dropout | Weights [457] | 20 | 10 | 98.39 | 184.87 | 0.00 | 15.09 |
| MNIST | | | | | | | | |
| Dense | Batch Normalization | - | 20 | 10 | 97.22 | 108.67 | 0.00 | 0 |
| Dense | Dropout | - | 20 | 10 | 98.15 | 80.66 | 0.00 | 0 |
| Sparse | - | Transfer Entropy | 20 | 10 | 98.28 | 0.00 | 295.44 | 19.18 |
| Sparse | Dropout | Transfer Entropy | 20 | 10 | 98.61 | 83.68 | 0.00 | 19.18 |
| Sparse | - | KL Divergence | 20 | 10 | 98.28 | 0.00 | 17479.56 | 19.33 |
| Sparse | Dropout | KL Divergence | 20 | 10 | 98.6 | 83.49 | 0.00 | 19.33 |
| Sparse | - | Correlation | 20 | 10 | 98.28 | 0.00 | 8.91 | 19.21 |
| Sparse | Dropout | Correlation | 20 | 10 | 98.57 | 84.07 | 0.00 | 19.21 |
| Sparse | - | Weights [457] | 20 | 10 | 98.28 | 0.00 | 0.02 | 20 |
| Sparse | Dropout | Weights [457] | 20 | 10 | 98.59 | 84.27 | 0.00 | 20 |
| Redense | Dropout | Weights [457] | 20 | 10 | 98.16 | 75.34 | 0.00 | 1.86 |

metrics. The pruning time represents the amount of time required to determine which connections should be eliminated. KL divergence was the most expensive, requiring on average approximately 16,700 seconds, $55\times$ more time than transfer entropy. Correlation was the least computationally expensive information theoretic metric, requiring tens of seconds compared to transfer entropy’s hundreds of seconds, but needed $1000\times$ longer than pruning based on weight values as in [457].

Table 10.3 summarizes the results of varying the number of epochs used to train the networks. We notice that the pruning algorithms’ performance was mainly affected by the dense model than by the number of epochs it was trained for. For example, fine-tuned sparse networks based on transfer entropy improved on the dense networks’ prediction by 0.96% when trained for 60 epochs but converged to the lowest accuracy compared to corresponding sparse networks trained for 10 and 200 epochs. This is due to its corresponding dense network converging to the lowest accuracy among the three networks. All proposed pruning approaches improved on the dense network by an average of 0.93% for all epochs

values.

Table 10.3: Varying the number of epochs on MNIST

| Network | Training | Pruning | Sparsity | Epoch | Accuracy (%) | Training Time (sec) | Pruning Time (sec) | Connectivity Reduction (%) |
|---------|---------------------|------------------|----------|-------|--------------|---------------------|--------------------|----------------------------|
| Dense | Batch Normalization | - | 30 | 10 | 97.25 | 57.97 | 0.00 | 0 |
| Dense | Dropout | - | 30 | 10 | 97.23 | 42.71 | 0.00 | 0 |
| Sparse | - | Transfer Entropy | 30 | 10 | 97.66 | 0.00 | 20.61 | 10.03 |
| Sparse | Dropout | Transfer Entropy | 30 | 10 | 98.03 | 12.10 | 0.00 | 10.03 |
| Sparse | - | KL Divergence | 30 | 10 | 97.66 | 0.00 | 1084.17 | 10.03 |
| Sparse | Dropout | KL Divergence | 30 | 10 | 98.03 | 23.71 | 0.00 | 10.03 |
| Sparse | - | Correlation | 30 | 10 | 97.66 | 0.00 | 4.25 | 10.03 |
| Sparse | Dropout | Correlation | 30 | 10 | 97.93 | 12.30 | 0.00 | 10.03 |
| Sparse | - | Weights [457] | 30 | 10 | 97.66 | 0.00 | 0.01 | 29.99 |
| Sparse | Dropout | Weights [457] | 30 | 10 | 98.08 | 18.32 | 0.00 | 29.99 |
| Redense | Dropout | Weights [457] | 30 | 10 | 97.54 | 23.02 | 0.00 | 3.99 |
| Dense | Batch Normalization | - | 30 | 60 | 97.38 | 39.30 | 0.00 | 0 |
| Dense | Dropout | - | 30 | 60 | 96.09 | 19.07 | 0.00 | 0 |
| Sparse | - | Transfer Entropy | 30 | 60 | 96.71 | 0.00 | 20.80 | 10.03 |
| Sparse | Dropout | Transfer Entropy | 30 | 60 | 97.50 | 12.28 | 0.00 | 10.03 |
| Sparse | - | KL Divergence | 30 | 60 | 96.71 | 0.00 | 1066.47 | 10.03 |
| Sparse | Dropout | KL Divergence | 30 | 60 | 97.71 | 23.68 | 0.00 | 10.03 |
| Sparse | - | Correlation | 30 | 60 | 96.71 | 0.00 | 4.34 | 10.03 |
| Sparse | Dropout | Correlation | 30 | 60 | 97.87 | 23.95 | 0.00 | 10.03 |
| Sparse | - | Weights [457] | 30 | 60 | 96.71 | 0.00 | 0.01 | 29.69 |
| Sparse | Dropout | Weights [457] | 30 | 60 | 97.72 | 18.37 | 0.00 | 29.69 |
| Redense | Dropout | Weights [457] | 30 | 60 | 97.51 | 17.47 | 0.00 | 14.59 |
| Dense | Batch Normalization | - | 30 | 200 | 95.77 | 39.13 | 0.00 | 0.00 |
| Dense | Dropout | - | 30 | 200 | 97.52 | 54.90 | 0.00 | 0.00 |
| Sparse | - | Transfer Entropy | 30 | 200 | 97.84 | 0.00 | 20.66 | 10.03 |
| Sparse | Dropout | Transfer Entropy | 30 | 200 | 98.08 | 41.27 | 0.00 | 10.03 |
| Sparse | - | KL Divergence | 30 | 200 | 97.84 | 0.00 | 1102.75 | 10.03 |
| Sparse | Dropout | KL Divergence | 30 | 200 | 98.10 | 23.85 | 0.00 | 10.03 |
| Sparse | - | Correlation | 30 | 200 | 97.84 | 0.00 | 3.75 | 10.03 |
| Sparse | Dropout | Correlation | 30 | 200 | 98.14 | 23.82 | 0.00 | 10.03 |
| Sparse | - | Weights [457] | 30 | 200 | 97.84 | 0.00 | 0.01 | 29.99 |
| Sparse | Dropout | Weights [457] | 30 | 200 | 98.26 | 18.27 | 0.00 | 29.99 |
| Redense | Dropout | Weights [457] | 30 | 200 | 97.75 | 11.99 | 0.00 | 3.99 |

10.4.3 Repeatability Analysis

Since the proposed algorithm samples the input space to reduce the computational complexity, we perform a repeatability study to assess how this sampling affects the algorithm’s performance. Figure 10.2 plots the accuracy for each repetition, in addition to the mean and standard deviation over five repetitions. We notice that adopting transfer entropy has a low standard deviation (in the order of 10^{-4}) and hence is repeatable. It was not significantly affected by the input subsampling step when computing the transfer entropy.

10.5 Conclusion

With many deep networks suffering from overfitting, regularization techniques have abounded in the literature. In this work, we compare multiple information theoretic metrics to perform regularization by eliminating paths in the network

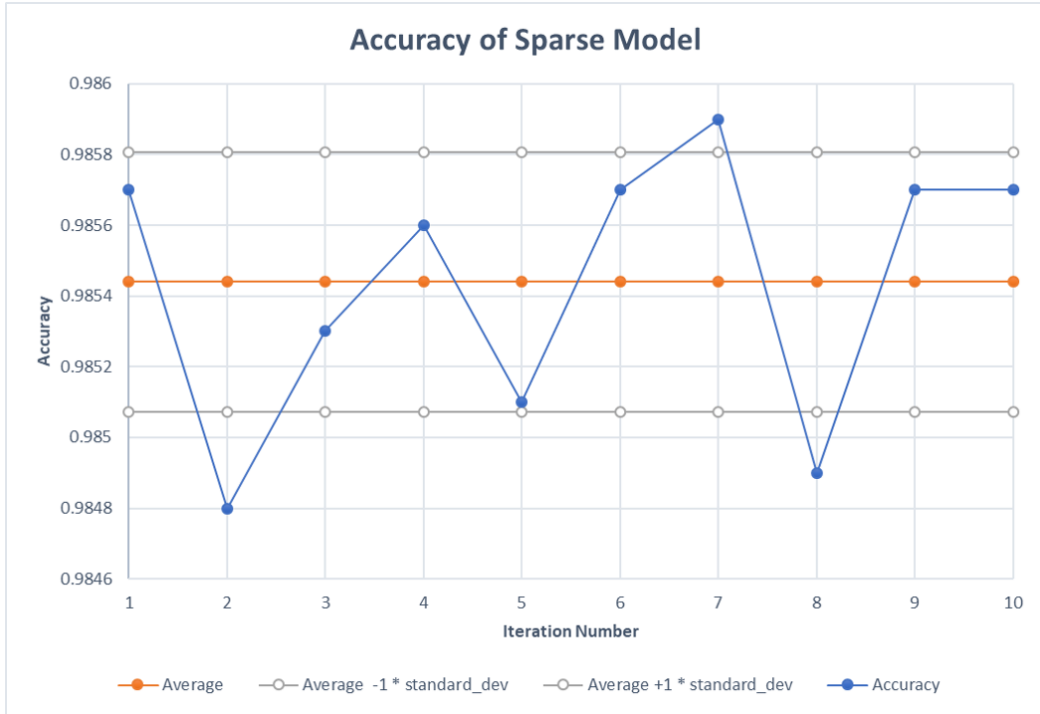


Figure 10.2: Repeatability Analysis on MNIST for $n = 30$

with low information transfer. This helps the network avoid overfitting by reducing its model complexity. Approximations of transfer entropy and KL divergence were adopted to reduce the computational complexity overhead incurred by the algorithm and compared to correlation-based pruning. Empirical results on multiple classification datasets showed that our proposed approach outperformed some regularization techniques in the literature and achieved comparable performance to others. Future work will investigate more accurate approximates of transfer entropy and KL divergence. The regularization technique will be applied to other network architectures such as convolutional neural networks and recurrent neural networks on larger datasets. We will also investigate deeper pre-trained networks such as VGG, Inception and GoogLeNet.

Chapter 11

Conclusion

In this chapter, we summarize the main contributions of this work and briefly discuss future research directions.

11.1 Summary

In this thesis, we investigated the possibility of incorporating MAS into the autonomous package delivery problem, modeled as a PDP. CF was incorporated into the optimization formulation for PDP by modifying the objective function and constraints. Multiple cost functions were investigated including distance, time and energy. The possibility of allowing overlapping coalitions was also discussed. A 3-index and 2-index MIP were derived for PDP-CF and were shown to grow exponentially with the linear increase in vehicle fleet. Since this is not scalable, we investigated search based approaches such as GA and QGA to improve the formulation's scalability. Introducing the concept of virtual packages, we were able to create a GA encoding that allowed the chromosome population to grow linearly with the vehicle fleet size, rendering the formulation more scalable. Furthermore, we investigated a data-driven approach which adopted ANN as a PDP-CF solver. Since the ANN was trained in a supervised fashion, we compiled a corpus for autonomous package delivery to train the ANN.

Even though existing ANN training algorithms have been effective in many applications, we looked to further improve their performance on our application by proposing a more computationally efficient training algorithm for RNN. Specifically, we proposed a non-iterative training algorithm (R-ELM) based on least squares and a computationally efficient approximation of RLS, known as Kaczmarsz's approximation. Furthermore, we proposed two context-dependent initialization algorithms for non-iteratively and iteratively trained ANN, CDR-ELM and BP-LS, respectively. Finally, we investigated an information theoretic pruning approach to help DL avoid overfitting. Once dense networks are trained, the transfer entropy of network connections are computed to prune connections

with low transfer entropy.

11.2 Future Research Directions

Many research directions can be investigated based on the work presented. We briefly discuss a few next.

- **Other domains:** While we focused on autonomous package delivery, it is possible to apply the proposed algorithms to other applications. For example, PDP-CF can be applied to the resource allocation problem in heterogeneous cloud computing services; heterogeneous cloud computing resources, which include CPUs, GPUs, FPGAs, and other computing devices with different capabilities, must be assigned to various jobs dynamically to maximize the utility of the hardware. Smart farming can also benefit from MAS to reduce the effects of soil compaction by heavy machinery which would be replaced by cooperating light-weight autonomous vehicles.
- **PDP-CF with time windows:** Our PDP-CF was derived from the simplest PDP formulation that did not assign time windows for package delivery. Extending PDP-CF to include time windows would make the formulation applicable to a wider set of real-world delivery problems.
- **Repeated CF:** The proposed formulation did not improve its choice of coalition based on the performance of coalitions in previously assigned tasks. Since the nature of our case study (and other real-world problems) is inherently repeated, learning from previous coalition assignments could significantly improve the performance of PDP-CF by reducing the search space (eliminating weak coalitions from consideration) and converging to better coalitions. However, this would require the quantification of synergy within a coalition, trust and reliability of agents within a coalition.
- **Decentralized solvers:** The solvers in this work were mainly centralized, with access to knowledge about all agents and tasks in the system. However, developing a decentralized PDP-CF solver could improve scalability issues and allow agents to join or leave coalitions on the fly. Disconnecting from a centralized source would also allow the application of this formulation to communication constrained problems (e.g. search and rescue in disaster-stricken environments), where only peer-to-peer communication is possible.
- **RL-based solvers:** With a decentralized formulation, lifelong learning becomes possible when a RL framework is considered. Agents can improve their decision making through experience by obtaining rewards or punishments from the environment. Furthermore, this would reduce the need to

create a labeled database for supervised learning and facilitate the deployment of PDP-CF to other domains. Deep RL and deep Q-learning are fields that can be investigated to train the ANN solver in a RL domain. Furthermore, transfer learning can be incorporated to reduce the learning overhead in new domains.

- **Real-world simulations:** The experiments in this work were performed on synthetic benchmarks that did not incorporate many of the real-world constraints such as environmental stochasticity, terrain characteristics, vehicle or robot failures, and others. To that end, environmental descriptors can be included in the dataset to influence decision making. Furthermore, environmental stochasticity can be modeled by noise in the system to make simulations more realistic and study the robustness of the proposed algorithms.
- **Non-vehicle agent incorporation:** The introduction of the descriptors and energy cost function (including information acquisition costs) allow the abstraction of vehicles or robots to agents and the incorporation of non-vehicle agents that can contribute information to improve task execution beyond aiding in the physical transportation of packages. However, the experiments in this work did not consider such a scenario. Therefore, the dataset could be extended to allow for such scenarios to investigate the effectiveness of our proposed framework in such domains.

Appendix A

Abbreviations

ANN Artificial Neural Networks

CF Coalition Formation

CNN Convolutional Neural Networks

DL Deep Learning

DNN Deep Neural Networks

ELM Extreme Learning Machines

GA Genetic Algorithm

LLS Linear Least Squares

LSTM Long Short-Term Memory

MAS Multi-Agent System

MDP Markov Decision Process

MIP Mixed Integer Programming

MRS Multi-Robot System

PDP Pickup and Delivery Problem

PDP-CF Pickup and Delivery Problem with Coalition Formation

POMDP Partially Observable Markov Decision Process

QGA Quantum Genetic Algorithm

RL Reinforcement Learning

RLS Recursive Least Squares
RNN Recurrent Neural Networks
SLAM Simultaneous Localization and Mapping
SVM Support Vector Machines
SVR Support Vector Regression
UAV Unmanned Aerial Vehicle
UGV Unmanned Ground Vehicle
UUV Unmanned Underwater Vehicle
VRP Vehicle Routing Problem

Appendix B

Heterogeneous MAS Workflow Literature Review

The main components of the workflow to automate MRS (in Figure 2.2) are: task decomposition, coalition formation, task allocation, perception, and MAS planning and control. We survey existing work in each of these areas in sections B.1 to B.4, then identify some remaining challenges and possible future research directions in section B.5.

B.1 Task Decomposition

Task decomposition, the first step in the MAS workflow for complex task automation, divides a complex task into a set of simpler or more primitive sub-tasks that are either independent or sequentially dependent on each other. For example, mapping a building can be divided into mapping of individual floors and rooms in the building.

Planners for task decomposition problems can be general [470] or domain specific [471, 472]. An example of the latter is soccer task decomposition where covering the playing field is divided among robots based on relative position of the ball and players [472]. The ball, viewed as a gravitational source, creates a gravitational field around it and affects the sub-task assignments. While many systems require the designer to manually decompose complex tasks to a sequence of simpler sub-tasks [32, 52], some work have attempted to automate this process and can be divided into three main categories: decompose-then-allocate, allocate-then-decompose and simultaneous decomposition and allocation [473].

Decompose-then-allocate algorithms first decompose a complex task into a list of sub-tasks in a centralized fashion then allocate the various sub-tasks to available agents. Task tree decomposition divided tasks based on logistic relationships in the battlefield [474]. Automatic decomposition and abstraction learned to divide complex decision making tasks into sub-tasks from human demonstrations,

using mutual information measures [475]. Tracking people in an environment was dynamically divided among robots based on geographical proximity [476]. Allocate-then-decompose algorithms, such as the M+ algorithm [470], first allocate a list of tasks to agents and then each agent divides this task to more primitive sub-tasks. Finally, simultaneous task decomposition and allocation algorithms opted not to decouple the task decomposition and allocation steps and proposed a solution based on task trees and auctioning [471, 477].

B.2 Coalition Formation and Task Allocation

After decomposing a complex task into a list of sub-tasks, these sub-tasks should be allocated to a robot or group of robots for execution. Since some of the sub-tasks are multi-robot tasks, they should be assigned to groups of cooperating robots. Next, we discuss research on forming coalitions of robots (coalition formation) and assigning to them sub-tasks (task allocation) before task execution can be performed.

B.2.1 Coalition Formation

Coalition or team formation divides agents into coalitions or groups. These agents may be non-cooperative [478] or cooperative. In this work, we focus on cooperative coalition formation. Coalition formation can be performed offline to form static coalitions or online to form dynamic teams that can adjust to the environment [479]. Agents are categorized into single-task and multi-task agents [30], i.e. agents that can perform a single task versus those that can perform multiple tasks. As mentioned previously, tasks are either single-robot or multi-robot tasks. Finally, the task to agent mapping or assignment is categorized into instantaneous and time-extended [30]. Architectures to represent agent capabilities and task requirements have been developed including numeric representations [480] and behavior based representations like schema theory [481]. Many search algorithms have been adopted to find the best robot teams including ant colony optimization [482], particle swarm optimization [483], and evolutionary algorithms [484].

Repeated coalition formation under uncertainty deals with forming time varying team when agents have partial information about other agents' capabilities, resulting in uncertainty. Furthermore, information can be heterogeneous, i.e. from different sources [485]. Dynamic, repeated coalition formation with robot type and other uncertainties was performed by incorporating an agent modeling algorithm with game theory [486]. Bayesian reinforcement learning (RL) allowed agents to learn other agents' capabilities through their interactions and transformed the repeated coalition formation problem into a sequential decision making problem [252]. This approach was validated on a football team formation problem [250]. Dynamic robot coalition formation for area coverage problems

was modeled using weighted voting games and Q-learning [487], and extended to formation-based navigation problems [488]. Coalition structures were pruned using the Shapley value and marginal contributions and the transition of agents from one coalition to another was represented by a Markov process [251]. This model searched the coalitions space for the best structure using Markov probability distributions. Self-adapting coalition formation dynamically changed coalition memberships using electric grid specific heuristics [489].

Focusing on MRS coalition formation, ASyMTRE facilitated coalition formation for tightly-coupled tasks [490] where time and space constraints were added to coalition formation formulations [491]. Other approaches include swarm intelligence in RoboCup Rescue [44], ant colony optimization and genetic algorithm [492], decentralized ant colony optimization for modular robotics [493], and greedy approximate algorithms that converged in polynomial time [494]. Furthermore, the MuRoCo algorithm used market-based optimization to return optimal coalition formation for MRS, validated on a drink serving scenario [495]. Coalition pruning was proposed to form approximate coalitions in real-time [46]. Modeling MRS coalition formation as a multi-objective optimization problem allowed the development of CUDA algorithms to speed up processing and was validated on navigation and box pushing tasks [496]. Decentralized optimal and sub-optimal coalition formation algorithms were also developed for UAVs based on particle swarm optimization and validated on UAV search and prosecute mission simulations [45]. Hierarchical optimization solvers searched for sub-optimal, computationally efficient coalitions to improve UAV search and prosecute [47]. Finally, a multi-criteria decision making algorithm was proposed based on influence diagrams to select the best coalition formation algorithm for a given real world scenario [253]. Coalition formation algorithms were classified using domain and mission dependent features. Over 100,000 mission scenarios were developed under various conditions including coalition overlap and communication constraints.

B.2.2 Task Allocation

Task allocation assigns tasks to an agent or a group of agents, i.e. it aims to find an optimal or near-optimal mapping between agents and tasks. A comprehensive taxonomy for task allocation can be found in [497]. Many approaches have been adopted to produce such a mapping in MRS and have been surveyed in [498, 499]. Auctioning or market-based approaches, common in multi-robot task allocation problems, [111, 112] include ASyMTRE-D [490], Murdoch [500], M+ [470] and TraderBots [501].

Since MRS face additional constraints compared to MAS including spatial, temporal, sensing and actuation constraints, MRS specific task allocation algorithms have been developed. Examples include swarm intelligence [502], particle swarm optimization with graph theory for UAV military [503], Sandholm algorithm with K-means clustering [504], utility-based task allocation [41], max-

sum [42], resource welfare [43] and semantic maps [505].

Auctioning algorithms are another class of algorithm popularly adopted in task allocation problems applied in health care facility scenarios capable of handling heterogeneous tasks and robots and task priorities [38], multi-robot traveling, UUV cooperation and object construction [39], and patrolling tasks [506]. Since environments are dynamic and partially observable, task allocation algorithms should preempt and re-allocate tasks. Dynamical re-planning in MAS was achieved using an auction-based decentralized method [507]. Single-robot task preemption while minimizing unnecessary reallocation was developed using simultaneous descending auctioning [508].

B.2.3 Simultaneous Coalition Formation and Task Allocation

Performing coalition formation and task allocation simultaneously has been proposed to improve performance in MRS. IQ-ASyMTRE interleaved both problems to solve a broader range of complex tasks using MRS [509]. Combining task priority ranking and resource constraints improved coalition formation for UAVs [510]. Multi-robot task assignments in UAV search tasks were performed by combining dynamic ANT coalition formation and memetic local search task allocation algorithms based on robot, task and environment information [35].

Auctioning algorithms were also adopted in simultaneous decentralized coalition formation and task allocation. Applications included 2-robot box pushing and transportation, obstacle avoidance and surveillance [511]. Auctioning allowed heterogeneous coalition formation with coalition re-adjustment before task completion to improve performance [512]. Constrained coalition formation and task allocation problems could also be solved using auction algorithms to minimize the time to complete tasks and the distance traveled by robot [513]. Such algorithms have been applied to reconfigurable robot coalition formation by splitting or merging robots while distributing tasks related to navigation, exploration and surveillance sensors' placement [514].

B.3 MAS Planning and Control

MAS planning and control or decision making, is a main module in MAS. It determines the sequence of actions, or policy, that agents should perform to complete their assigned task, once the complex tasks have been decomposed to sub-tasks and allocated to cooperating groups of agents. Decision making models have been applied to a wide spectrum of fields including robotics [118], wireless sensor networks [295], cognitive radio networks [312], intelligent transportation systems [271] and electric grids [319]. Decision making algorithms are generally evaluated based on policy optimality and their time and space complexity [15].

Multiple frameworks have been proposed to model and solve decision making problems including RL, game theory, swarm intelligence and graph theoretic models. While more detailed surveys on multi-agent decision making models can be found in [25,95], we will briefly cover some of the models mentioned above.

Swarm intelligence, inspired by social animals, models the behavior of many decentralized cooperative autonomous agents [200]. Such models are mainly characterized by self-organized and distributed behavior of locally aware and locally interacting agents [18]. Particle swarm optimization [203] is inspired by flocks of birds and schools of fish. Pigeon inspired optimization algorithm rely on the magnetic field, sun and landmarks to achieve path planning [204]. Bee colony optimization is based on the behavior of bees and relies on direct communication between agents [201]. While swarm based systems are robust, flexible, scalable, computationally inexpensive and fault tolerant [52,99], robots are generally homogeneous or can be divided into a small number of clusters of homogeneous robots which greatly restricts MRS applications [52]. More details on swarm robotics can be found in these recent surveys [99,101,102,118].

Game theoretic models include partially observable stochastic games which are sequential probabilistic games where payoffs are unknown to players and depend on their actions, and the game’s state depends on the previous state and the players’ actions [188]. Their sub-classes include MDPs and partially observable MDPs (POMDPs). MDPs assume that decisions satisfy the Markov property, i.e. decisions at the current time step only depend on decisions of the previous time step. They are described by a set of environment states which are fully observable, actions, transition probabilities, reward and discount factor. A policy maps states to actions by maximizing the total reward, measured by a value function. Multi-agent MDP and decentralized MDP extended MDP to MAS, which assumes jointly fully observable environments [515]. POMDPs extend MDPs to partially observable environments by adding a set of observations and an observation function to the model representation. MAS extensions include decentralized POMDP [97], multi-agent POMDP [179] and interactive POMDP [183]. Recent work on decentralized POMDP focused on the sub-classes of MDP models and their computational complexity, briefly mentioned existing solutions [97].

RL allows agents to learn a policy by rewarding “good” behavior and punishing “bad” behavior through a reward signal. RL is one approach to find optimal or sub-optimal policies for game theoretic models. Multi-agent RL allows cooperative MAS to complete tasks with minimal communication overhead by using the global immediate reward instead of the individual agent immediate reward in the Q-learning algorithm to solve repeated games [516]. Validation on box pushing and sensor distribution demonstrated the superior performance of this algorithm compared to other approached. Sparse interaction to negotiate equilibrium sets and transfer knowledge in multi-agent RL reduced computational complexity and led to better coordination and scalability, as shown by simulations on grid world games and robots shelving items in a warehouse [517]. Information sharing has

been modeled as a MDP to reduce communication overhead without affecting performance [518].

Sub-optimal policies for decentralized POMDP were computed using a factored forward-sweep policy computation algorithm that reduced computational complexity and improved scalability [176]. Simulations on hundreds of agents showed the improved scalability with minimal loss in accuracy. An RL inference model that learned MRS configurations and allowed robot systems to complete a task knowing the intermediary robot states and transitions was also developed [519]. Other approaches include dynamic programming [515], expectation maximization [520], heuristic search algorithms [521], temporal difference learning [130], policy search [131], evolutionary computing [137], genetic algorithms [162], neural networks [143], optimization algorithms [133], Monte Carlo methods [158] and deep RL approaches [145].

B.4 Perception

Perception is a crucial component of successful MRS deployment that allows robots to model their environment from sensory information and obtain knowledge of how their actions are affecting the environment and whether they are successfully completing their tasks; it is a sub-block of task execution in Figure 2.2. Without this capability, task execution would be near impossible in real-world environments. Sensors measure variables in the environment, allowing robots to observe how their actions have affected the environment, which leads to more effective task execution. From the low level sensory information, robots need to learn higher level information such as the location obstacles, their locations within a map, and the types of objects in an environment, among others.

SLAM allows robots to simultaneously generate a map of the environment and localize themselves within this map, a vital aspect of any task that involves navigation [522, 523]. Many algorithms have been proposed with varying degrees of computational complexity [524], using a wide range of sensors including cameras [525], acoustic sensors [526], structured light [527] and electromagnetic signals [528]. SLAM has also adopted sensor fusion [529] to benefit from diverse signals such as sonars with laser range finders [530], WiFi, Bluetooth, LTE and magnetic signals [528]. Distributed, decentralized, cooperative or multi-robot SLAM has been developed to leverage multi-robot cooperation and tackle complex environments [531], in communication constrained environments [532], or where direct communication is not possible [533]. Distributed SLAM with sparse robot networks [534] and decentralized active SLAM that forced robot teams to efficiently traverse and map the environment [535] were also investigated.

Scene understanding allows robots to extract general principles from visual cues [536]. It includes computer vision problems of image segmentation, object recognition, event recognition, human activity and behavior recognition [537],

semantic annotation [538], and others. Scene understanding has been applied to pedestrian [539], traffic [540], urban [541], video surveillance [542] and underwater scenes [543]. Multi-agent or distributed computer vision algorithms have been developed to improve scene understanding in MAS applications [544]. Deep neural networks performed multi-object classification and scene understanding by extracting features from raw images and incorporating context through conditional latent tree probability models [545]. Markov random fields performed modeling, inference and learning tasks on visual inputs by representing the inputs' conditional probabilistic dependence with undirected graphs [546]. Time-dependent correlation rules were adopted for dynamic scene understanding in traffic surveillance problems where motion patterns were detected using object tracking, spectral clustering and Allen's interval-based temporal logic [547]. Traffic patterns were learned using hierarchical pattern mining based on latent Dirichlet allocation; traffic states were learned from activities which were modeled from spatial location and velocity [540].

Object motion tracking is another important aspect of scene understanding that helps robots achieve their goals by tracking objects of interest. Developed systems were based on received signal strength variations on wireless links [548], and Kalman filter based SLAM with laser-based occupancy grids [549], to name a few. Multi-robot or cooperative tracking [550] has been developed to track pedestrians [551] and other objects, using particle filters [552], RL [553] and least squares minimization [554]. More information can be found in the recent survey on MRS object detection and tracking [555].

Automatic speech recognition is important in human-robot interactions [556]. Approaches include Hidden Markov models [557], deep neural networks [558] and support vector machines [559]. While deep neural networks have had the best performance to date, they are computationally expensive and require many data points to achieve good performance, making integration with robotic systems for real-time applications a challenge. Some work has attempted to address the computational complexity of automatic speech recognition [560], but there is still a lot of room for improvement. Algorithms that are robust to noise have been proposed to handle noisy environments [561]. In addition, distant talking [562] and the noise from a robot's hardware [563] add to the difficulty of deploying speech recognition algorithms in MRS.

B.5 Challenges and Insights

Allowing heterogeneous agents to cooperate increases the scope of solvable tasks. It introduces parallelism and robustness, leading to better performance with simpler agents than having a single powerful but complex agent performing the same task [26]. However, it also increases the complexity of the design process. Many challenges still face the research community before effective deployment of MRS

performing complex tasks can be achieved. In this section, we discuss some challenges faced in designing MRS and some insights on each of the research areas identified in Figure 2.2.

B.5.1 Big Data

Perception problems, including object recognition, speech recognition and natural language processing, have greatly benefited from hardware advancements that allowed machine learning algorithms to develop models from big data. However, perception algorithms deployed on robot platform do not have the computational resources to leverage these advancements and improve the robots' models of their environments and cloud accessibility is an issue in some robotics applications. In addition, decision making algorithms would greatly benefit from the developed models to improve decisions. However, these models are computationally expensive for robotics applications even if they are trained offline since the most successful models derived from deep neural networks. Therefore, future research should investigate methods to incorporate big data models into computationally constrained MRS applications to improve task planning and execution.

B.5.2 Internet of Things

While state of the art algorithms in perception and scene understanding have seen significant improvements, out-performing humans in some scenarios, integration with robotics applications is still in its early stages. Furthermore, there is still plenty of information to extract from the environment, especially in this era of IoT. Sensor fusion and distributed sensing from heterogeneous sources is one area that can help improve perception for robotics applications.

B.5.3 Task Complexity

As tasks become more complex, decision making algorithms struggle to recognize their complexity and decompose them to simpler tasks that can be solved efficiently. To aid MRS in completing complex tasks in uncertain environments, the task decomposition step should be automated to allow re-planning as conditions change. Furthermore, automated task decomposition could make use of existing ontology and domain specific dictionaries in natural language processing to decompose tasks to sub-tasks. The decomposition could consider the available agents' capabilities and the model of the environment, beyond the workflow depicted in Figure 2.2.

B.5.4 Autonomous Machine Learning

Many machine learning algorithms still rely on human intervention to manually tune algorithm parameters. Autonomous machine learning is a machine learning sub-field striving to develop learning algorithms that do not require a human expert to select a learning algorithm, manually tune parameters and select data for training [564–566]. Incorporating such algorithms into MRS would result in general agents that can better handle dynamic environments.

B.5.5 Scalability and Heterogeneity Trade off

To effectively operate in smart cities, MRS need to be scalable, adaptable and generalizable to successfully cope with the dynamic environment and complexity of their tasks. Having multiple robots act on the environment simultaneously further increases the uncertainty in the system. Many decentralized MRS planning and control algorithms have been proposed but still face challenges when dealing with the tradeoff between scalability and robot heterogeneity in highly dynamic environments. Therefore, developing efficient planning algorithms that strikes a task appropriate balance between scalability and heterogeneity will take MRS a step closer to more ubiquitous existence in smart cities.

B.5.6 Coalition Formation and Task Allocation

Simultaneous coalition formation and task allocation could lead to more optimal mappings and should be investigated further; only a few works have considered this approach but obtained promising results [509]. The coalition and task assignments should be dynamic and time variant to better cope with task complexity and environment variability. Therefore, coalitions might have to be dynamically altered and assigned new tasks before the completion of their assigned tasks to achieve successful task execution. Coalition formation and task allocation algorithms that allow repeated, dynamic coalition formation and task exemption have been developed but still face limitations especially in highly dynamic environments. They should also consider the tradeoff between agent capabilities' redundancy within a coalition and fault tolerance or robustness to agent failures.

B.5.7 Other Challenges

Communication constraints and connectivity uncertainty further complicate things for cooperative MRS, especially for tightly coordinated problems. While connecting MRS to the cloud also allows us to reduce the computational load on these mobile devices and improve their performance [567], the existence and stability of this connection is uncertain and may sometimes cripple the system instead of improving its performance. The time sensitivity of certain tasks and lim-

ited hardware resources of robots requires the development of efficient algorithms for decision making, perception, coalition formation and task decomposition and allocation. Finally, evaluation standards are needed to effectively compare the performance of MRS, as they are still underdeveloped.

Appendix C

ANN History

In this chapter¹, we will go over the evolution of neural networks from their shallow beginnings to the complex structures that have recently become popular.

C.1 Concepts from Neuroscience

Despite the advances in neuroscience and technology that have allowed for a detailed description of the structure of the brain, the learning process in the brain is yet to be completely understood. Biologically, the brain mainly consists of the cerebrum, the cerebellum, and the brain stem [568].

The cerebral cortex, biologically defined as the outer layer of tissue in the cerebrum and believed to be responsible for higher order functioning, is an association of an estimated 25 billion neurons interconnected through thousands of kilometers of axons propagating and spreading about 10^{14} synapses simultaneously [569], arranged in six layers and divided into regions, each performing a specific task [570].

Though it is not very clear how certain areas in the brain become specialized, it is known that multiple factors affect the functional specialization of the brain areas such as structure, connectivity, physiology, development and evolution [571]. Neurons, considered the basic element in the brain, have different shapes and sizes but are all variations of the same underlying scheme, i.e. they start the same general-purpose function but become specialized with training [572]. While dendrites are the site of reception of synaptic inputs, axons convey electrical signals over long distances. Inputs to neurons cause a slow potential change in the state of the neuron; its characteristics are determined by the membrane capacitance and resistance allowing temporal summation [573].

Studies showed that the organization of the cortex can be regarded as an

¹The contents of this chapter appear in section 2 of **Rizk, Y.**, Hajj, N., Mitri, N., and Awad, M., “A comparative study between Deep Neural Networks and Cortical Algorithms,” *Elsevier Applied Computing & Informatics*, 2018 (in press).

association of columnar units [574], [575], each column being a group of nodes sharing the same properties. Learning in the human brain is mainly performed using plastic connections, repeated exposures and firing and inhibition of neurons. In a simplified manner, information flowing in the cortex causes connections in the brain to become active, over time, with repeated exposures these connections are strengthened creating a representation of the information processed in the brain. Moreover, inhibition of neurons - physically defined as prohibiting neurons from firing - partly account for the forgetting process [576].

C.2 Shallow Beginnings

At a nodal level, ANN started with the simplified McCulloch-Pitts neural model (1943) [577], which was composed of a basic summation unit with a deterministic binary activation function. Successors added complexity with every iteration. At the level of activation functions, linear, sigmoid, and Gaussian functions came into use. Outputs were no longer restricted to real values and extended to the complex domain. Deterministic models gave way to stochastic neurons and spiking neurons which simulated ionic exchanges. All these additions were made to achieve more sophisticated learning models.

At the network level, topologies started out with single layered architectures such as Rosenblatt's perceptron (1957) [578], Widrow and Hoff's ADALINE network (1960) [579] and Aizerman's kernel perceptron (1964) [580]. These architectures suffered from poor performance and could not learn the XOR problem, a simple but non-linear binary classification problem. This led to the introduction of more complex networks starting with the multilayer perceptron (Rumelhart, 1986) [581], self-recurrent Hopfield networks (1986) [582], self-organizing maps (SOM or Kohonen networks, 1986) [583], adaptive resonance theory (ART) networks (1980s) [584] and various others which are considered shallow architectures due to the small number of hidden layers.

Successive iterations incrementally improved on their predecessors' shortcomings and promised higher levels of intelligence, a claim that was made partially feasible due to the hardware's improved computational capabilities [585] and due to the development of faster and more efficient training and learning algorithms. Learning mechanics, whether supervised (back propagation) or unsupervised (feed forward algorithms), matured in parallel and allowed for better performance in a varied set of specific tasks. Nonetheless, the compound effect of the innovation targeting all aspects of these shallow networks was not enough to capture true human intelligence while large computational needs throttled the progress of deeper networks.

C.3 Shallow Networks' Limitations

Supervised learning presents many challenges including the curse of dimensionality [586] where the increase in the number of features and training samples makes learning more computationally demanding. Furthermore, non-linear data is more difficult to divide into classes due to the inherent feature overlap. Unable to position themselves as strong AI models - general intelligent acts as defined by Kurzweil - which can faithfully emulate human intelligence, ANN lagged Support Vector Machines (SVM) [587] in the 1990s-2000s.

C.4 Deep Architectures

The early 2000s saw a resurgence in ANN research due to increased processing power and the introduction of more efficient training algorithms which made training deep architectures feasible. Hinton et al.'s greedy training algorithm [588] simplified the training procedure of Boltzmann machines while deep stacking networks broke down training to the constituting blocks of the deep network to reduce the computational burden. Furthermore, Schmidhuber's long short-term memory architecture [589] allowed the training of deeper recurrent neural networks. While these architectures do not borrow biological properties from the brain beyond the neuron, deep architectures with neural network topologies that adhere more faithfully to neuro-scientific theories of the human brain's topology are gaining traction in the connectionist community due in part to the momentum achieved in computational neuroscience.

One of the major and most relevant contributions in that field was made by Edelman and Mountcastle [590]. Their findings lead to a shift from positioning simplified neuron models as fundamental functional units of an architecture to elevating that role to cortical columns, collections of cells characterized by common feed-forward connections and strong inhibitory inter connections. This provided a biologically feasible mechanism for learning and forming invariant representations of sensory patterns that earlier ANN did not.

Additionally, two supplementary discoveries were believed to be key in emulating human intelligence. The first was the suspected existence of a common computational algorithm in the neocortex [572]. This algorithm is pervasive throughout these regions irrespective of the underlying mental faculty. Whether the task is visual, auditory, olfactory, or other, the brain seems to deal with sensory information in very similar ways. The second was the hierarchical structure of the human neocortex [572]. The brain's regions are hierarchically connected so that the bidirectional flow of information merges into more complex representations with every layer, further abstracting the sensory stimuli.

The combination of these two findings forms potential grounds for building a framework that replicates human intelligence; a hierarchy of biologically inspired

functional units that implement a common algorithm. These novel insights from neuroscience have been reflected in the machine learning (ML) and AI fields and have been implemented to varying layers in several algorithms.

While CA restructured the neurons and their connections as well as the learning algorithm [591] based on Edelman and Mountcastle's finding [590], other algorithms modeled other biological theories of the brain's workings. Symbolic architectures such as Adaptive Character of Thought (ACT-R) [592] modeled working memory coupled with centralized control that refers to long term memory when needed. Emergentist architectures such as Hierarchical Temporal Memory (HTM) [593] are based on globalist memory models and use reinforcement or competitive learning schemes to generate their models. Integrating both classes of architectures to form hybrid architectures also exist and include Learning Intelligent Distribution Agent (LIDA) [594].

Bibliography

- [1] L. Fridman, “Deep tesla,” 2017.
- [2] X. Zhang, D. Jiang, T. Han, and N. Wang, “Feature dimension reduction method of rolling bearing based on quantum genetic algorithm,” in *Prognostics and System Health Management Conference (PHM-Chengdu), 2016*, pp. 1–5, IEEE, 2016.
- [3] Statista, “Statistics and facts about global e-commerce,” 2015.
- [4] Reuters, “How amazon is making package delivery even cheaper,” 2016.
- [5] TriviaHive, “On average, how many packages does amazon ship per day?,” 2016.
- [6] B. Dennis, “Nearly 60 percent of americans — the highest ever — are taking prescription drugs,” 2015.
- [7] MUST, “Facts: Older adults and medicine use,” 2013.
- [8] M. Clinic, “Nearly 7 in 10 americans take prescription drugs, mayo clinic, olmsted medical center find,” 2013.
- [9] H. Yen and H. Dreier, “Census: Rural us loses population for first time,” 2013.
- [10] EDGE, “Aging in place in rural communities,” 2015.
- [11] M. Baernholdt, G. Yan, I. Hinton, K. Rose, and M. Mattos, “Quality of life in rural and urban adults 65 years and older: findings from the national health and nutrition examination survey,” *The Journal of Rural Health*, vol. 28, no. 4, pp. 339–347, 2012.
- [12] AHA, “Fast facts on us hospitals,” 2017.
- [13] R. Rahman, M. A. Faiz, S. Selim, B. Rahman, A. Basher, A. Jones, C. d’Este, M. Hossain, Z. Islam, H. Ahmed, *et al.*, “Annual incidence of snake bite in rural bangladesh,” *PLoS Neglected Tropical Diseases*, vol. 4, no. 10, p. e860, 2010.

- [14] B. Mohapatra, D. A. Warrell, W. Suraweera, P. Bhatia, N. Dhingra, R. M. Jotkar, P. S. Rodriguez, K. Mishra, R. Whitaker, P. Jha, *et al.*, “Snakebite mortality in india: a nationally representative mortality survey,” *PLoS Neglected Tropical Diseases*, vol. 5, no. 4, p. e1018, 2011.
- [15] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, 2009.
- [16] P. Langley, J. E. Laird, and S. Rogers, “Cognitive architectures: Research issues and challenges,” *Cognitive Syst. Research*, vol. 10, no. 2, pp. 141–160, 2009.
- [17] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.
- [18] G. Weiss, ed., *Multiagent Systems*. MIT Press, 2013.
- [19] T. Arai, E. Pagello, and L. E. Parker, “Editorial: Advances in multi-robot systems,” *IEEE Trans. Robot. Autom.*, vol. 18, no. 5, pp. 655–661, 2002.
- [20] A. Bond and L. Gasser, “A survey of distributed artificial intelligence,” *Readings in Distributed Artificial Intell.*, 1988.
- [21] L. E. Parker, “Distributed intelligence: Overview of the field and its application in multi-robot systems,” *J. Physical Agents*, vol. 2, no. 1, pp. 5–14, 2008.
- [22] IBM, “1st workshop on cognitive architectures,” 2015.
- [23] P. Davidsson, S. Johansson, and M. Svahnberg, “Characterization and evaluation of multi-agent system architectural styles,” in *Software Eng. for Multi-Agent Syst. IV*, pp. 179–188, Springer, 2005.
- [24] A. Farinelli, G. Grisetti, L. Iocchi, S. L. Cascio, and D. Nardi, “Design and evaluation of multi agent systems for rescue operations,” in *Proc. IEEE Int. Conf. Intell. Robots and Syst.*, vol. 4, pp. 3138–3143, 2003.
- [25] P. Stone and M. Veloso, “Multiagent systems: A survey from a machine learning perspective,” *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.
- [26] L. E. Parker, “Multiple mobile robot systems,” in *Springer Handbook of Robotics*, pp. 921–941, Springer, 2008.
- [27] M. Ngobye, W. T. De Groot, and T. P. Van Der Weide, “Types and priorities of multi-agent system interactions,” *Interdisciplinary Description of Complex Syst.*, vol. 8, no. 1, pp. 49–58, 2010.

- [28] J. Ferber, *Multi-agent systems: an introduction to distributed artificial intelligence*, vol. 1. Addison-Wesley Reading, 1999.
- [29] R. M. Murray, “Recent research in cooperative control of multivehicle systems,” *J. Dynamic Syst., Measurement, and Control*, vol. 129, no. 5, pp. 571–583, 2007.
- [30] B. P. Gerkey and M. J. Matarić, “A formal analysis and taxonomy of task allocation in multi-robot systems,” *Int. J. Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.
- [31] P. Caloud, W. Choi, J.-C. Latombe, C. Le Pape, and M. Yim, “Indoor automation with many mobile robots,” in *Proc. IEEE Int. Workshop on Intell. Robots and Syst. Towards a New Frontier of Applicat.*, pp. 67–72, 1990.
- [32] J. Kiener and O. Von Stryk, “Towards cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots,” *Robotics and Autonomous Syst.*, vol. 58, no. 7, pp. 921–929, 2010.
- [33] E. G. Jones, B. Browning, M. B. Dias, B. Argall, M. Veloso, and A. Stentz, “Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks,” in *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 570–575, 2006.
- [34] M. B. Dias, *Traderbots: A new paradigm for robust and efficient multi-robot coordination in dynamic environments*. PhD thesis, Carnegie Mellon University, Pittsburgh, 2004.
- [35] M. Padmanabhan and G. Suresh, “Coalition formation and task allocation of multiple autonomous robots,” in *IEEE 3rd Int. Conf. Signal Process., Commun. and Networking*, pp. 1–5, 2015.
- [36] O. Arslan, D. P. Guralnik, and D. E. Koditschek, “Coordinated robot navigation via hierarchical clustering,” *IEEE Trans. Robotics*, vol. 32, no. 2, pp. 352–371, 2016.
- [37] X. Dai, L. Jiang, and Y. Zhao, “Cooperative exploration based on supervisory control of multi-robot systems,” *Applied Intell.*, pp. 1–12, 2016.
- [38] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera, “A distributed task allocation algorithm for a multi-robot system in healthcare facilities,” *J. Intell. & Robotic Syst.*, vol. 80, no. 1, pp. 33–58, 2015.
- [39] S. Sariel-Talay, T. R. Balch, and N. Erdogan, “A generic framework for distributed multirobot cooperation,” *J. Intell. & Robotic Syst.*, vol. 63, no. 2, pp. 323–358, 2011.

- [40] J. Baca, P. Pagala, C. Rossi, and M. Ferre, “Modular robot systems towards the execution of cooperative tasks in large facilities,” *Robotics and Autonomous Syst.*, vol. 66, pp. 159–174, 2015.
- [41] T. Abukhalil, M. Patil, S. Patel, and T. Sobh, “Coordinating a heterogeneous robot swarm using robot utility-based task assignment (ruta),” in *IEEE 14th Int. Workshop on Advanced Motion Control*, pp. 57–62, 2016.
- [42] M. Pujol-Gonzalez, J. Cerquides, A. Farinelli, P. Meseguer, and J. A. Rodriguez-Aguilar, “Efficient inter-team task allocation in robocup rescue,” in *Proc. Int. Conf. Autonomous Agents and Multiagent Syst.*, pp. 413–421, Int. Found. for Autonomous Agents and Multiagent Syst., 2015.
- [43] M.-H. Kim, H. Baik, and S. Lee, “Resource welfare based task allocation for uav team with resource constraints,” *J. Intell. & Robotic Syst.*, vol. 77, no. 3-4, pp. 611–627, 2015.
- [44] F. Dos Santos and A. L. Bazzan, “Towards efficient multiagent task allocation in the robocup rescue: a biologically-inspired approach,” *Autonomous Agents and Multi-Agent Syst.*, vol. 22, no. 3, pp. 465–486, 2011.
- [45] J. G. Manathara, P. Sujit, and R. W. Beard, “Multiple uav coalitions for a search and prosecute mission,” *J. Intell. & Robotic Syst.*, vol. 62, no. 1, pp. 125–158, 2011.
- [46] Z. Liu, X.-g. Gao, and X.-w. Fu, “Coalition formation for multiple heterogeneous uavs cooperative search and prosecute with communication constraints,” in *Chinese Control and Decision Conf.*, pp. 1727–1734, IEEE, 2016.
- [47] L. Zhong, G. Xiao-Guang, and F. Xiao-Wei, “Coalition formation for multiple heterogeneous uavs in unknown environment,” in *IEEE 5th Int. Conf. Instrumentation and Measurement, Comput., Commun. and Control*, pp. 1222–1227, 2015.
- [48] P. Dasgupta, J. Baca, K. Guruprasad, A. Muñoz-Meléndez, and J. Jumadinova, “The comrade system for multirobot autonomous landmine detection in postconflict regions,” *J. Robotics*, vol. 2015, 2015.
- [49] S. Barrett and P. Stone, “Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork,” in *AAAI*, pp. 2010–2016, 2015.
- [50] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, “First results in the coordination of heterogeneous robots for large-scale assembly,” in *Experimental Robotics VII*, pp. 323–332, Springer, 2001.

- [51] B. Fernandez-Gauna, I. Etxeberria-Agiriano, and M. Graña, “Learning multirobot hose transportation and deployment by distributed round-robin q-learning,” *PloS one*, vol. 10, no. 7, p. e0127129, 2015.
- [52] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, *et al.*, “Swarmanoid: a novel concept for the study of heterogeneous robotic swarms,” *IEEE Robot. Autom. Mag.*, vol. 20, no. 4, pp. 60–71, 2013.
- [53] R. R. Murphy, K. L. Dreger, S. Newsome, J. Rodocker, B. Slaughter, R. Smith, E. Steimle, T. Kimura, K. Makabe, K. Kon, *et al.*, “Marine heterogeneous multirobot systems at the great eastern japan tsunami recovery,” *J. Field Robotics*, vol. 29, no. 5, pp. 819–831, 2012.
- [54] J. Gregory, J. Fink, E. Stump, J. Twigg, J. Rogers, D. Baran, N. Fung, and S. Young, “Application of multi-robot systems to disaster-relief scenarios with limited communication,” in *Field and Service Robotics*, pp. 639–653, Springer, 2016.
- [55] C. Lesire, G. Infantes, T. Gateau, and M. Barbier, “A distributed architecture for supervision of autonomous multi-robot missions,” *Autonomous Robots*, vol. 40, no. 7, pp. 1343–1362, 2016.
- [56] C. Pippin, G. Gray, M. Matthews, D. Price, A.-P. Hu, W. Lee, M. Novitzky, and P. Varnell, “The design of an air-ground research platform for cooperative surveillance,” tech. rep., Tech. Rep. 112010, Georgia Tech Research Institute, 2010.
- [57] A. Ollero, S. Lacroix, L. Merino, J. Gancet, J. Wiklund, V. Remuß, I. V. Perez, L. G. Gutiérrez, D. X. Viegas, M. A. G. Benitez, *et al.*, “Multiple eyes in the skies: architecture and perception issues in the comets unmanned air vehicles project,” *IEEE Robot. Autom. Mag.*, vol. 12, no. 2, pp. 46–57, 2005.
- [58] L. Sabattini, C. Secchi, and C. Fantuzzi, “Coordinated dynamic behaviors for multirobot systems with collision avoidance,” *IEEE Trans. Cybern.*, 2016.
- [59] D. Portugal and R. P. Rocha, “Cooperative multi-robot patrol with bayesian learning,” *Autonomous Robots*, vol. 40, no. 5, pp. 929–953, 2016.
- [60] A. Howard, L. E. Parker, and G. S. Sukhatme, “Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection,” *The Int. J. Robotics Research*, vol. 25, no. 5-6, pp. 431–447, 2006.

- [61] R. T. Vaughan, G. S. Sukhatme, F. J. Mesa-Martinez, and J. F. Montgomery, “Fly spy: Lightweight localization and target tracking for cooperating air and ground robots,” in *Distributed autonomous robotic systems 4*, pp. 315–324, Springer, 2000.
- [62] F. Morbidi, C. Ray, and G. L. Mariottini, “Cooperative active target tracking for heterogeneous robots with application to gait monitoring,” in *IEEE/RSJ Int. Conf. Intelligent Robots and Syst.*, pp. 3608–3613, IEEE, 2011.
- [63] L. Rosa, M. Cagnetti, A. Nicasastro, P. Alvarez, and G. Oriolo, “Multi-task cooperative control in a heterogeneous ground-air robot team,” *IFAC*, vol. 48, no. 5, pp. 53–58, 2015.
- [64] P. Stegagno, M. Cagnetti, L. Rosa, P. Peliti, and G. Oriolo, “Relative localization and identification in a heterogeneous multi-robot system,” in *IEEE Int. Conf. Robotics and Automation*, pp. 1857–1864, 2013.
- [65] J. J. Roldán, P. Garcia-Aunon, M. Garzón, J. de León, J. del Cerro, and A. Barrientos, “Heterogeneous multi-robot system for mapping environmental variables of greenhouses,” *Sensors*, vol. 16, no. 7, p. 1018, 2016.
- [66] A. Benzerrouk, L. Adouane, and P. Martinet, “Stable navigation in formation for a multi-robot system based on a constrained virtual structure,” *Robotics and Autonomous Syst.*, vol. 62, no. 12, pp. 1806–1815, 2014.
- [67] S. C. Nagavarapu, L. Vachhani, and A. Sinha, “Multi-robot graph exploration and map building with collision avoidance: A decentralized approach,” *J. Intelligent & Robotic Syst.*, pp. 1–21, 2015.
- [68] J. Kim, “Cooperative exploration and networking while preserving collision avoidance,” *IEEE Trans. Cybern.*, 2016.
- [69] P. Chand and D. A. Carnegie, “Mapping and exploration in a hierarchical heterogeneous multi-robot system using limited capability robots,” *Robotics and Autonomous Syst.*, vol. 61, no. 6, pp. 565–579, 2013.
- [70] R. Rahimi, F. Abdollahi, and K. Naqshi, “Time-varying formation control of a collaborative heterogeneous multi agent system,” *Robotics and Autonomous Syst.*, vol. 62, no. 12, pp. 1799–1805, 2014.
- [71] Z. Liu, W. Chen, J. Lu, H. Wang, and J. Wang, “Formation control of mobile robots using distributed controller with sampled-data and communication delays,” *IEEE Trans. Control Syst. Technology*, vol. 24, no. 6, pp. 2125–2132, 2016.

- [72] Q. Wang, M. Wu, Y. Huang, and L. Wang, "Formation control of heterogeneous multi-robot systems," *IFAC Proc. Volumes*, vol. 41, no. 2, pp. 6596–6601, 2008.
- [73] E. Hernandez-Martinez, E. Ferreira-Vazquez, A. Lopez-Gonzalez, J. Flores-Godoy, G. Fernandez-Anaya, and P. Paniagua-Contro, "Formation control of heterogeneous robots using distance and orientation," in *IEEE Int. Conf. Control Applicat.*, pp. 507–512, 2016.
- [74] J. M. Evans, "Helpmate: An autonomous mobile robot courier for hospitals," in *Intelligent Robots and Systems' 94. Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*, vol. 3, pp. 1695–1700, IEEE, 1994.
- [75] M. D. Rossetti and F. Selandari, "Multi-objective analysis of hospital delivery systems," *Computers & Industrial Engineering*, vol. 41, no. 3, pp. 309–333, 2001.
- [76] R. Murai, T. Sakai, H. Kawano, Y. Matsukawa, Y. Kitano, Y. Honda, and K. C. Campbell, "A novel visible light communication system for enhanced control of autonomous delivery robots in a hospital," in *System Integration (SII), 2012 IEEE/SICE International Symposium on*, pp. 510–516, IEEE, 2012.
- [77] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan, "A layered architecture for office delivery robots," in *Proceedings of the first international conference on Autonomous agents*, pp. 245–252, ACM, 1997.
- [78] N. Mathew, S. L. Smith, and S. L. Waslander, "Planning paths for package delivery in heterogeneous multirobot teams," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 4, pp. 1298–1308, 2015.
- [79] H. Surmann and A. Morales, "Scheduling tasks to a team of autonomous mobile service robots in indoor environments," *Journal of Universal Computer Science*, vol. 8, no. 8, pp. 809–833, 2002.
- [80] J. Bohren, R. B. Rusu, E. G. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mösenlechner, W. Meeussen, and S. Holzer, "Towards autonomous robotic butlers: Lessons learned with the pr2," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5568–5575, IEEE, 2011.
- [81] J. Kantorovitch, J. Väre, V. Pehkonen, A. Laikari, and H. Seppälä, "An assistive household robot—doing more than just cleaning," *Journal of Assistive Technologies*, vol. 8, no. 2, pp. 64–76, 2014.

- [82] M. M. Veloso, J. Biswas, B. Coltin, and S. Rosenthal, “Cobots: Robust symbiotic autonomous mobile service robots,” in *IJCAI*, p. 4423, 2015.
- [83] A. J. Scarfe, R. C. Flemmer, H. Bakker, and C. L. Flemmer, “Development of an autonomous kiwifruit picking robot,” in *Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on*, pp. 380–384, IEEE, 2009.
- [84] T. T. Nguyen, E. Kayacan, J. De Baedemaeker, and W. Saeys, “Task and motion planning for apple harvesting robot,” *IFAC Proceedings Volumes*, vol. 46, no. 18, pp. 247–252, 2013.
- [85] J. Hemming, C. Bac, B. van Tuijl, R. Barth, J. Bontsema, E. Pekkeriet, and E. Van Henten, “A robot for harvesting sweet-pepper in greenhouses,” 2014.
- [86] T. A. Permadi, J. Halomoan, and S. Hadiyoso, “Balancing system of tray on waiter robot using complementary filter and fuzzy logic,” in *Industrial Automation, Information and Communications Technology (IAICT), 2014 International Conference on*, pp. 15–21, IEEE, 2014.
- [87] M. Asif, M. Sabeel, and K. Mujeeb-ur Rahman, “Waiter robot—solution to restaurant automation,” in *Proceedings of the 1st student multi disciplinary research conference (MDSRC). University of Wah, Pakistan*, 2015.
- [88] A.-a. Agha-mohammadi, N. K. Ure, J. P. How, and J. Vian, “Health aware stochastic planning for persistent package delivery missions using quadrotors,” in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 3389–3396, IEEE, 2014.
- [89] M. Van De Vissel and J. Anderson, “Coalition formation in multi-agent systems under real-world conditions,” *Proceedings of association for the advancement of artificial intelligence*, 2004.
- [90] D. Landén, F. Heintz, and P. Doherty, “Complex task allocation in mixed-initiative delegation: a uav case study,” in *International Conference on Principles and Practice of Multi-Agent Systems*, pp. 288–303, Springer, 2010.
- [91] N. Mathew, S. L. Smith, and S. L. Waslander, “Optimal path planning in cooperative heterogeneous multi-robot delivery systems,” in *Algorithmic Foundations of Robotics XI*, pp. 407–423, Springer, 2015.
- [92] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, “A survey of research on cloud robotics and automation,” *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

- [93] Gartner, “Gartner says 6.4 billion connected ”things” will be in use in 2016, up 30 percent from 2015,” 2015.
- [94] CNBC, “An internet of things that will number ten billions,” 2016.
- [95] L. Panait and S. Luke, “Cooperative multi-agent learning: The state of the art,” *AAMAS*, vol. 11, no. 3, pp. 387–434, 2005.
- [96] Z. Yan, N. Jouandeau, and A. A. Cherif, “A survey and analysis of multi-robot coordination,” *Int. J. Adv. Robotic Syst.*, vol. 10, p. 399, 2013.
- [97] C. Amato, G. Chowdhary, A. Geramifard, N. K. Ure, and M. J. Kochenderfer, “Decentralized control of partially observable markov decision processes,” in *IEEE 52nd Annu. Conf. Decis. & Control*, pp. 2398–2405, 2013.
- [98] L. Bayindir and E. Şahin, “A review of studies in swarm robotics,” *Turkish J. Elect. Eng. & Comput. Sci.*, vol. 15, no. 2, pp. 115–147, 2007.
- [99] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, “Swarm robotics: a review from the swarm engineering perspective,” *Swarm Intell.*, vol. 7, no. 1, pp. 1–41, 2013.
- [100] Y. Tan and Z.-y. Zheng, “Research advance in swarm robotics,” *Defence Technol.*, vol. 9, no. 1, pp. 18–39, 2013.
- [101] J. C. Barca and Y. A. Sekercioglu, “Swarm robotics reviewed,” *Robotica*, vol. 31, no. 03, pp. 345–359, 2013.
- [102] L. Bayındır, “A review of swarm robotics tasks,” *Neurocomputing*, vol. 172, pp. 292–321, 2016.
- [103] Y. Shoham, R. Powers, and T. Grenager, “Multi-agent reinforcement learning: a critical survey,” tech. rep., Stanford University, 2003.
- [104] L. Busoniu, R. Babuska, and B. De Schutter, “A comprehensive survey of multiagent reinforcement learning,” *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 2, pp. 156–172, 2008.
- [105] M. Reyes-Sierra and C. C. Coello, “Multi-objective particle swarm optimizers: A survey of the state-of-the-art,” *Int. J. Computational Intell. Res.*, vol. 2, no. 3, pp. 287–308, 2006.
- [106] D. Karaboga and B. Akay, “A survey: algorithms simulating bee swarm intelligence,” *Artif. Intell. Rev.*, vol. 31, no. 1-4, pp. 61–85, 2009.
- [107] R. S. Parpinelli and H. S. Lopes, “New inspirations in swarm intelligence: a survey,” *Int. J. Bio-Inspired Computation*, vol. 3, no. 1, pp. 1–16, 2011.

- [108] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, “A comprehensive survey: artificial bee colony (abc) algorithm and applications,” *Artif. Intell. Rev.*, vol. 42, no. 1, pp. 21–57, 2014.
- [109] Y. Zhang and H. Mehrjerdi, “A survey on multiple unmanned vehicles formation control and coordination: normal and fault situations,” in *IEEE Int. Conf. Unmanned Aircraft Syst.*, pp. 1087–1096, 2013.
- [110] K.-K. Oh, M.-C. Park, and H.-S. Ahn, “A survey of multi-agent formation control,” *Automatica*, vol. 53, pp. 424–440, 2015.
- [111] S.-Y. Tang, Y.-F. Zhu, Q. Li, and Y.-L. Lei, “Survey of task allocation in multi agent systems,” *Syst. Eng. & Electron.*, vol. 32, no. 10, pp. 2155–2161, 2010.
- [112] A. Campbell and A. S. Wu, “Multi-agent role allocation: issues, approaches, and multiple perspectives,” *AAMAS*, vol. 22, no. 2, pp. 317–355, 2011.
- [113] C. Koliass, G. Kambourakis, and M. Maragoudakis, “Swarm intelligence in intrusion detection: A survey,” *Comput. & Security*, vol. 30, no. 8, pp. 625–642, 2011.
- [114] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, “Intrusion detection system: A comprehensive review,” *J. Netw. & Comput. Appl.*, vol. 36, no. 1, pp. 16–24, 2013.
- [115] S. Elsayed, R. Sarker, and D. Essam, “Survey of uses of evolutionary computation algorithms and swarm intelligence for network intrusion detection,” *Int. J. Computational Intell. & Appl.*, vol. 14, no. 04, p. 1550025, 2015.
- [116] H. Bai and B. Zhao, “A survey on application of swarm intelligence computation to electric power system,” in *The 6th World Congr. Intell. Control & Autom.*, vol. 2, pp. 7587–7591, IEEE, 2006.
- [117] E. Yang and D. Gu, “Multiagent reinforcement learning for multi-robot systems: A survey,” tech. rep., 2004.
- [118] N. S. Pal and S. Sharma, “Robot path planning using swarm intelligence: A survey,” *Int. J. Comput. Appl.*, vol. 83, no. 12, 2013.
- [119] A. Esmin, R. Coelho, and S. Matwin, “A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data,” *Artif. Intell. Rev.*, vol. 44, no. 1, pp. 23–45, 2015.
- [120] D. Martens, B. Baesens, and T. Fawcett, “Editorial survey: swarm intelligence for data mining,” *Mach. Learn.*, vol. 82, no. 1, pp. 1–42, 2011.

- [121] M. Bratman, *Intention, plans, and practical reason*. 1987.
- [122] D. Poole, “Exploiting the rule structure for decision making within the independent choice logic,” in *Proc. 11th Conf. Uncertainty in Artif. Intell.*, pp. 454–463, Morgan Kaufmann Publishers Inc., 1995.
- [123] J. Lee, M. J. Huber, E. H. Durfee, and P. G. Kenny, “UM-PRS: An implementation of the procedural reasoning system for multirobot applications,” in *Nasa Conf. Publication*, pp. 842–842, 1994.
- [124] D. Poole, “The independent choice logic for modelling multiple agents under uncertainty,” *Artif. Intell.*, vol. 94, no. 1, pp. 7–56, 1997.
- [125] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.
- [126] J. A. Boyan and M. L. Littman, “Exact solutions to time-dependent MDPs,” *Work*, vol. 2, no. 12, p. 10, 2000.
- [127] K. V. Delgado, S. Sanner, and L. N. De Barros, “Efficient solutions to factored mdps with imprecise transition probabilities,” *Artif. Intell.*, vol. 175, no. 9, pp. 1498–1527, 2011.
- [128] R. S. Sutton *et al.*, “Fast gradient-descent methods for temporal-difference learning with linear function approximation,” in *Int. Conf. Mach. Learn.*, pp. 993–1000, ACM, 2009.
- [129] R. S. Sutton, H. R. Maei, and C. Szepesvári, “A convergent $o(n)$ temporal-difference algorithm for off-policy learning with linear function approximation,” in *NIPS*, pp. 1609–1616, 2009.
- [130] P. Escandell-Montero, J. M. Martínez-Martínez, J. D. Martín-Guerrero, E. Soria-Olivas, and J. Gómez-Sanchis, “Least-squares temporal difference learning based on an extreme learning machine,” *Neurocomputing*, vol. 141, pp. 37–45, 2014.
- [131] D. S. Bernstein, C. Amato, E. A. Hansen, and S. Zilberstein, “Policy iteration for decentralized control of markov decision processes,” *J. Artif. Intell. Res.*, vol. 34, no. 1, p. 89, 2009.
- [132] A. Ghate and R. L. Smith, “A linear programming approach to nonstationary infinite-horizon markov decision processes,” *Operations Res.*, vol. 61, no. 2, pp. 413–425, 2013.
- [133] F. Dufour and T. Prieto-Rumeau, “Finite linear programming approximations of constrained discounted markov decision processes,” *SIAM J. Control and Optimization*, vol. 51, no. 2, pp. 1298–1324, 2013.

- [134] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.
- [135] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, vol. 37. 1994.
- [136] J. H. Metzen, M. Edgington, Y. Kassahun, and F. Kirchner, “Analysis of an evolutionary reinforcement learning method in a multiagent domain,” in *AAMAS - Vol. 1*, pp. 291–298, 2008.
- [137] S. Whiteson, “Evolutionary computation for reinforcement learning,” in *Reinforcement Learning*, pp. 325–355, Springer, 2012.
- [138] J. Schmidhuber, D. Wierstra, and F. Gomez, “Evolino: Hybrid neuroevolution/optimal linear search for sequence learning,” in *Proc. 19th Int. Joint Conf. Artif. Intell.*, pp. 853–858, Morgan Kaufmann Publishers Inc., 2005.
- [139] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, “Training recurrent networks by evolino,” *Neural computation*, vol. 19, no. 3, pp. 757–779, 2007.
- [140] J. H. Metzen, F. Kirchner, M. Edgington, and Y. Kassahun, “Towards efficient online reinforcement learning using neuroevolution,” in *Proc. 10th Annu. Conf. Genetic & Evol. Computation*, pp. 1425–1426, 2008.
- [141] R. Koppejan and S. Whiteson, “Neuroevolutionary reinforcement learning for generalized helicopter control,” in *ACM Proc. 11th Annu. Conf. Genetic & Evol. Computation*, pp. 145–152, 2009.
- [142] B. Li and J. Si, “Approximate robust policy iteration using multilayer perceptron neural networks for discounted infinite-horizon markov decision processes with uncertain correlated transition matrices,” *IEEE Trans. Neural Netw.*, vol. 21, no. 8, pp. 1270–1280, 2010.
- [143] A. Hans and S. Udluft, “Ensembles of neural networks for robust reinforcement learning,” in *IEEE Int. Conf. Mach. Learn. & Appl.*, pp. 401–406, 2010.
- [144] A. Nair *et al.*, “Massively parallel methods for deep reinforcement learning,” *arXiv preprint arXiv:1507.04296*, 2015.
- [145] V. Mnih *et al.*, “Asynchronous methods for deep reinforcement learning,” *arXiv preprint arXiv:1602.01783*, 2016.
- [146] R. Tutunov, J. E. Zini, H. Bou-Ammar, and A. Jadbabaie, “Distributed lifelong reinforcement learning with sub-linear regret,” in *56th IEEE Annu. Conf. Decision & Control*, pp. 2254–2259, 2017.

- [147] J. Fu, S. Han, and U. Topcu, “Optimal control in markov decision processes via distributed optimization,” in *54th IEEE Annu. Conf. Decision & Control*, pp. 7462–7469, 2015.
- [148] C. Boutilier, “Sequential optimality and coordination in multiagent systems,” in *IJCAI*, vol. 99, pp. 478–485, 1999.
- [149] C. Guestrin, D. Koller, and R. Parr, “Multiagent planning with factored mdps,” in *NIPS*, vol. 1, pp. 1523–1530, 2001.
- [150] D. S. Bernstein, S. Zilberstein, and N. Immerman, “The complexity of decentralized control of markov decision processes,” in *Proc. 16th Conf. Uncertainty in Artif. Intell.*, pp. 32–37, Morgan Kaufmann Publishers Inc., 2000.
- [151] S. Brechtel *et al.*, “Solving continuous pomdps: Value iteration with incremental learning of an efficient space representation,” in *Int. Conf. Mach. Learn.*, pp. 370–378, 2013.
- [152] X. P. Wan and S. Y. Li, “Shp-vi method of solving dec-pomdp problem,” in *Adv. Materials Res.*, vol. 926, pp. 3245–3249, Trans. Tech. Publ., 2014.
- [153] F. Wu, S. Zilberstein, and N. R. Jennings, “Monte-carlo expectation maximization for decentralized pomdps,” in *IJCAI*, pp. 397–403, 2013.
- [154] Z. Song, X. Liao, and L. Carin, “Solving dec-pomdps by expectation maximization of value function,” in *AAAI Spring Symposium Series*, 2016.
- [155] C. Amato, D. S. Bernstein, and S. Zilberstein, “Optimizing memory-bounded controllers for decentralized pomdps,” in *Proc. 23rd Conf. Uncertainty in Artif. Intell.*, 2007.
- [156] C. Amato, D. S. Bernstein, and S. Zilberstein, “Solving POMDPs using quadratically constrained linear programs,” in *AAMAS*, pp. 341–343, ACM, 2006.
- [157] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo, “Monte carlo value iteration for continuous-state pomdps,” in *Algorithmic Found. of Robotics IX*, pp. 175–191, Springer, 2010.
- [158] D. Silver and J. Veness, “Monte-carlo planning in large pomdps,” in *NIPS*, pp. 2164–2172, 2010.
- [159] S. Seuken and S. Zilberstein, “Memory-bounded dynamic programming for dec-pomdps,” in *IJCAI*, pp. 2009–2015, 2007.

- [160] S. Seuken and S. Zilberstein, “Improved memory-bounded dynamic programming for decentralized pomdps,” in *Proc. 23rd Conf. Uncertainty in Artif. Intell.*, 2007.
- [161] D. Szer, F. Charpillet, and S. Zilberstein, “Maa*: A heuristic search algorithm for solving decentralized pomdps,” in *21st Conf. Uncertainty in Artif. Intell.*, 2005.
- [162] B. Eker and H. L. Akin, “Solving decentralized pomdp problems using genetic algorithms,” *AAMAS*, vol. 27, no. 1, pp. 161–196, 2013.
- [163] F. J. Gomez and J. Schmidhuber, “Co-evolving recurrent neurons learn deep memory POMDPs,” in *Proc. 7th Annu. Conf. Genetic & Evol. Computation*, pp. 491–498, ACM, 2005.
- [164] L. Dung, T. Komeda, and M. Takagi, “Knowledge-based recurrent neural networks in reinforcement learning,” in *Proc. 11th Int. Conf. Artif. Intell. & Soft Computing*, pp. 169–174, ACTA Press, 2007.
- [165] Z. Liu and I. Elhanany, “A scalable model-free recurrent neural network framework for solving POMDPs,” in *IEEE Int. Symp. Approximate Dynamic Programming & Reinforcement Learning*, pp. 119–126, 2007.
- [166] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, “Solving deep memory POMDPs with recurrent policy gradients,” in *Artif. Neural Netw.*, pp. 697–706, Springer, 2007.
- [167] M. Liu, X. Liao, and L. Carin, “The infinite regionalized policy representation,” in *Int. Conf. Mach. Learn.*, pp. 769–776, 2011.
- [168] G. Contardo, L. Denoyer, T. Artieres, and P. Gallinari, “Learning states representations in POMDP,” in *Int. Conf. Learn. Represent.*, pp. 120–122, 2014.
- [169] M. Lauri and R. Ritala, “Planning for robotic exploration based on forward simulation,” *Robot. Auton. Syst.*, 2016.
- [170] D. S. Bernstein, E. A. Hansen, and S. Zilberstein, “Bounded policy iteration for decentralized POMDPs,” in *Proc. 19th Int. Joint Conf. Artif. Intell.*, pp. 52–57, 2005.
- [171] F. A. Oliehoek, M. T. Spaan, and N. A. Vlassis, “Optimal and approximate Q-value functions for decentralized POMDPs,” *J. Artif. Intell. Res.*, vol. 32, pp. 289–353, 2008.
- [172] D. Szer and F. Charpillet, “Point-based dynamic programming for dec-POMDPs,” in *AAAI*, vol. 6, pp. 1233–1238, 2006.

- [173] E. A. Hansen, D. S. Bernstein, and S. Zilberstein, “Dynamic programming for partially observable stochastic games,” in *AAAI*, vol. 4, pp. 709–715, 2004.
- [174] A. Kumar and S. Zilberstein, “Dynamic programming approximations for partially observable stochastic games,” in *Twenty-Second International FLAIRS Conference*, 2009.
- [175] M. Liu, C. Amato, X. Liao, L. Carin, and J. P. How, “Stick-breaking policy learning in dec-pomdps,” in *Proc. 24th Int. Conf. Artif. Intell.*, pp. 2011–2017, AAAI Press, 2015.
- [176] F. A. Oliehoek, S. Whiteson, and M. T. Spaan, “Approximate solutions for factored dec-pomdps with many agents,” in *Proc. Int. Conf. Auton. Agents & Multiagent Syst.*, pp. 563–570, 2013.
- [177] D. V. Pynadath and M. Tambe, “The communicative multiagent team decision problem: Analyzing teamwork theories and models,” *J. Artif. Intell. Res.*, pp. 389–423, 2002.
- [178] S. Seuken and S. Zilberstein, “Formal models and algorithms for decentralized decision making under uncertainty,” *AAMAS*, vol. 17, no. 2, pp. 190–250, 2008.
- [179] C. Amato, F. A. Oliehoek, and E. Shyu, “Scalable bayesian reinforcement learning for multiagent POMDPs,” in *Proc. 1st Multidisciplinary Conf. Reinforcement Learn. & Decis. Making*, 2013.
- [180] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo, “Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs,” in *AAAI*, vol. 5, pp. 133–139, 2005.
- [181] C. Zhang and V. R. Lesser, “Coordinated multi-agent reinforcement learning in networked distributed pomdps,” in *AAAI*, 2011.
- [182] A. Kumar and S. Zilberstein, “Constraint-based dynamic programming for decentralized pomdps with structured interactions,” in *AAMAS - Vol. 1*, pp. 561–568, 2009.
- [183] P. J. Gmytrasiewicz and P. Doshi, “Interactive POMDPs: Properties and preliminary results,” in *AAMAS - Vol. 3*, pp. 1374–1375, 2004.
- [184] P. Doshi and P. J. Gmytrasiewicz, “A particle filtering based approach to approximating interactive POMDPs,” in *AAAI*, pp. 969–974, 2005.
- [185] P. Doshi and D. Perez, “Generalized point based value iteration for interactive POMDPs,” in *AAAI*, pp. 63–68, 2008.

- [186] P. Doshi and P. J. Gmytrasiewicz, “Monte carlo sampling methods for approximating interactive POMDPs,” *J. Artif. Intell. Res.*, pp. 297–337, 2009.
- [187] R. B. Myerson, “Game theory: analysis of conflict,” *Harvard University*, 1991.
- [188] L. S. Shapley, “Stochastic games,” *Proc. Nat. Academy of Sci. of the United States of Amer.*, vol. 39, no. 10, p. 1095, 1953.
- [189] A. Condon, “The complexity of stochastic games,” *Inf. & Computation*, vol. 96, no. 2, pp. 203–224, 1992.
- [190] J. Hu and M. P. Wellman, “Nash Q-learning for general-sum stochastic games,” *The J. Mach. Learn. Res.*, vol. 4, pp. 1039–1069, 2003.
- [191] J. F. Nash *et al.*, “Equilibrium points in n-person games,” *Proc. Nat. Academy of Sci.*, vol. 36, no. 1, pp. 48–49, 1950.
- [192] J. M. Smith and G. R. Price, “The logic of animal conflict,” *Nature*, vol. 246, no. 5427, pp. 15–18, 1973.
- [193] J. Flesch, T. Parthasarathy, F. Thuijssman, and P. Uyttendaele, “Evolutionary stochastic games,” *Dynamic Games & Appl.*, vol. 3, no. 2, pp. 207–219, 2013.
- [194] S. Loertscher, “Rock–scissors–paper and evolutionarily stable strategies,” *Econ. Lett.*, vol. 118, no. 3, pp. 473–474, 2013.
- [195] R. Emery, *Game theoretic control for robot teams*. PhD thesis, 2005.
- [196] A. Guo and V. R. Lesser, “Planning for weakly-coupled partially observable stochastic games,” in *IJCAI*, pp. 1715–1716, 2005.
- [197] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun, “Approximate solutions for partially observable stochastic games with common payoffs,” in *AAMAS*, pp. 136–143, 2004.
- [198] P. Paruchuri *et al.*, “Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games,” in *AAMAS - Vol. 2*, pp. 895–902, 2008.
- [199] C. Kiekintveld, J. Marecki, and M. Tambe, “Approximation methods for infinite bayesian stackelberg games: Modeling distributional payoff uncertainty,” in *AAMAS - Vol. 3*, pp. 1005–1012, 2011.

- [200] G. Beni and J. Wang, “Swarm intelligence in cellular robotic systems,” in *Robots & Biological Syst.: Towards a New Bionics?*, pp. 703–712, Springer, 1993.
- [201] D. Teodorović and M. Dell’Orco, “Bee colony optimization—a cooperative learning approach to complex transportation problems,” in *Adv. OR and AI Methods in Transportation: Proc. 16th Mini-Eur. Conf. & 10th Meeting of EWGT*, pp. 51–60, 2005.
- [202] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. No. 1, Oxford university press, 1999.
- [203] J. Kennedy and R. C. Eberhart, “Particle swarm optimization,” in *Proc. IEEE Int. Conf. Neural Netw.*, pp. 1942–1948, 1995.
- [204] H. Duan and P. Qiao, “Pigeon-inspired optimization: a new swarm intelligence optimizer for air robot path planning,” *Int. J. Intell. Computing & Cybern.*, vol. 7, no. 1, pp. 24–37, 2014.
- [205] S. Ilie and C. Bădică, “Multi-agent approach to distributed ant colony optimization,” *Sci. of Comput. Programming*, vol. 78, no. 6, pp. 762–774, 2013.
- [206] J. Collings and E. Kim, “A distributed and decentralized approach for ant colony optimization with fuzzy parameter adaptation in traveling salesman problem,” in *IEEE Symp. Swarm Intell.*, pp. 1–9, 2014.
- [207] I. Navarro, E. Di Mario, and A. Martinoli, “Distributed vs. centralized particle swarm optimization for learning flocking behaviors,” in *Proc. Eur. Conf. Synthesis & Simulat. of Living Syst.*, pp. 302–309, 2015.
- [208] M. Mesbahi and M. Egerstedt, *Graph theoretic methods in multiagent networks*. Princeton University Press, 2010.
- [209] R. A. Howard and J. E. Matheson, “Influence diagrams,” *Decis. Anal.*, vol. 2, no. 3, pp. 127–143, 2005.
- [210] J. Tatman *et al.*, “Dynamic programming and influence diagrams,” *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 365–379, 1990.
- [211] Y. Zeng, P. Doshi, and Q. Chen, “Approximate solutions of interactive dynamic influence diagrams using model clustering,” in *Proc. Nat. Conf. Artif. Intell.*, vol. 22, p. 782, 2007.
- [212] D. Koller and B. Milch, “Multi-agent influence diagrams for representing and solving games,” *Games & Econ. Behavior*, vol. 45, no. 1, pp. 181–221, 2003.

- [213] Y. Gal and A. Pfeffer, “Networks of influence diagrams: A formalism for representing agents’ beliefs and decision-making processes,” *J. Artif. Intell. Res.*, vol. 33, no. 1, pp. 109–147, 2008.
- [214] D. K. Molzahn, F. Dörfler, H. Sandberg, S. H. Low, S. Chakrabarti, R. Baldick, and J. Lavaei, “A survey of distributed optimization and control algorithms for electric power systems,” *IEEE Trans. Smart Grid*, vol. 8, no. 6, pp. 2941–2962, 2017.
- [215] A. Bidram, A. Davoudi, F. L. Lewis, and Z. Qu, “Secondary control of microgrids based on distributed cooperative control of multi-agent systems,” *IET Generation, Transmission & Distribution*, vol. 7, no. 8, pp. 822–831, 2013.
- [216] J. Ni, L. Liu, C. Liu, X. Hu, and S. Li, “Secondary voltage control for microgrids based on fixed-time distributed cooperative control of multi-agent systems,” in *IEEE Amer. Control Conf.*, pp. 761–766, 2017.
- [217] Y.-J. Pan, H. Werner, Z. Huang, and M. Bartels, “Distributed cooperative control of leader–follower multi-agent systems under packet dropouts for quadcopters,” *Syst. & Control Lett.*, vol. 106, pp. 47–57, 2017.
- [218] W. Wang, J. Huang, and C. Wen, “Distributed adaptive control of multi-agent systems under directed graph for asymptotically consensus tracking,” in *14th Int. Conf. Control, Autom., Robot. & Vision*, pp. 1–5, 2016.
- [219] H. Zhang, T. Feng, G.-H. Yang, and H. Liang, “Distributed cooperative optimal control for multiagent systems on directed graphs: An inverse optimal approach,” *IEEE Trans. Cybern.*, vol. 45, no. 7, pp. 1315–1326, 2015.
- [220] W. He, G. Chen, Q.-L. Han, and F. Qian, “Network-based leader-following consensus of nonlinear multi-agent systems via distributed impulsive control,” *Inf. Sci.*, 2015.
- [221] W. He *et al.*, “Leader-following consensus of nonlinear multiagent systems with stochastic sampling,” *IEEE Trans. Cybern.*, vol. 47, no. 2, pp. 327–338, 2017.
- [222] W. He *et al.*, “Quasi-synchronization of heterogeneous dynamic networks via distributed impulsive control: Error estimation, optimization and design,” *Automatica*, vol. 62, pp. 249–262, 2015.
- [223] R. Yang, H. Zhang, H. Yan, and F. Yang, “Cooperative control of heterogeneous multi-agent systems via distributed adaptive output regulation under switching topology,” in *42nd Annu. IEEE Conf. Ind. Electron. Soc.*, pp. 6770–6775, 2016.

- [224] G. Wang, C. Wang, Q. Du, L. Li, and W. Dong, “Distributed cooperative control of multiple nonholonomic mobile robots,” *J. Intell. & Robotic Syst.*, vol. 83, no. 3-4, pp. 525–541, 2016.
- [225] V. Indelman, P. Gurfil, E. Rivlin, and H. Rotstein, “Graph-based distributed cooperative navigation for a general multi-robot measurement model,” *The Int. J. Robotics Res.*, vol. 31, no. 9, pp. 1057–1080, 2012.
- [226] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun, “Game theoretic control for robot teams,” in *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 1163–1169, 2005.
- [227] C. Murray and G. Gordon, “Multi-robot negotiation: approximating the set of subgame perfect equilibria in general-sum stochastic games,” tech. rep., DTIC Document, 2006.
- [228] Y.-C. Choi and H.-S. Ahn, “A survey on multi-agent reinforcement learning: Coordination problems,” in *IEEE/ASME Int. Conf. Mechatronics & Embedded Syst. & Appl.*, pp. 81–86, 2010.
- [229] S. Ragi and E. K. Chong, “Uav path planning in a dynamic environment via partially observable markov decision process,” *IEEE Trans. Aerosp. Electron. Syst.*, vol. 49, no. 4, pp. 2397–2412, 2013.
- [230] C. Amato *et al.*, “Probabilistic planning for decentralized multi-robot systems,” in *AAAI Fall Symp. Series*, 2015.
- [231] C. Amato *et al.*, “Planning for decentralized control of multiple robots under uncertainty,” in *IEEE Int. Conf. Robot. Autom.*, pp. 1241–1248, 2015.
- [232] C. Amato *et al.*, “Policy search for multi-robot coordination under uncertainty,” in *The Robotics: Sci. & Syst. Conf.*, 2015.
- [233] D. K. Grady, M. Moll, and L. E. Kavraki, “Extending the applicability of pomdp solutions to robotic tasks,” *IEEE Trans. Robot.*, vol. 31, no. 4, pp. 948–961, 2015.
- [234] O. Aşık and H. L. Akın, “Solving multi-agent decision problems modeled as dec-pomdp: A robot soccer case study,” in *RoboCup 2012: Robot Soccer World Cup XVI*, pp. 130–140, Springer, 2013.
- [235] R. Olfati-Saber, A. Fax, and R. M. Murray, “Consensus and cooperation in networked multi-agent systems,” *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

- [236] M. Ji and M. Egerstedt, “Distributed coordination control of multiagent systems while preserving connectedness,” *IEEE Trans. Robot.*, vol. 23, no. 4, pp. 693–703, 2007.
- [237] F. Xiao, L. Wang, J. Chen, and Y. Gao, “Finite-time formation control for multi-agent systems,” *Automatica*, vol. 45, no. 11, pp. 2605–2611, 2009.
- [238] M. Joordens *et al.*, “Consensus control for a system of underwater swarm robots,” *IEEE Syst. J.*, vol. 4, no. 1, pp. 65–73, 2010.
- [239] R. M. de Mendonça, N. Nedjah, and L. de Macedo Mourelle, “Efficient distributed algorithm of dynamic task assignment for swarm robotics,” *Neurocomputing*, vol. 172, pp. 345–355, 2016.
- [240] A. O. de Sá, N. Nedjah, and L. de Macedo Mourelle, “Distributed efficient localization in swarm robotic systems using swarm intelligence algorithms,” *Neurocomputing*, vol. 172, pp. 322–336, 2016.
- [241] S. Nouyan, R. Groß, M. Bonani, F. Mondada, and M. Dorigo, “Teamwork in self-organized robot colonies,” *IEEE Trans. Evol. Comput.*, vol. 13, no. 4, pp. 695–711, 2009.
- [242] S. Alers, D. Claes, K. Tuyls, and G. Weiss, “Biologically inspired multi-robot foraging,” in *AAMAS*, pp. 1683–1684, 2014.
- [243] K. Ohkura, T. Yasuda, Y. Matsumura, and M. Kadota, “Gpu implementation of food-foraging problem for evolutionary swarm robotics systems,” in *Swarm Intell.*, pp. 238–245, Springer, 2014.
- [244] T. Schmickl *et al.*, “Get in touch: cooperative decision making based on robot-to-robot collisions,” *AAMAS*, vol. 18, no. 1, pp. 133–155, 2009.
- [245] H. Wei, Y. Cai, H. Li, D. Li, and T. Wang, “Sambot: A self-assembly modular robot for swarm robot,” in *IEEE Int. Conf. Robot. Autom.*, pp. 66–71, 2010.
- [246] C. Moeslinger, T. Schmickl, and K. Crailsheim, “A minimalist flocking algorithm for swarm robots,” in *Adv. Artif. Life. Darwin Meets von Neumann*, pp. 375–382, Springer, 2011.
- [247] F. Mondada *et al.*, “The cooperation of swarm-bots: Physical interactions in collective robotics,” *IEEE Robot. Autom. Mag.*, vol. 12, no. 2, pp. 21–28, 2005.
- [248] R. O’Grady *et al.*, “Swarm-bots to the rescue,” in *Adv. Artif. Life. Darwin Meets von Neumann*, pp. 165–172, Springer, 2011.

- [249] S. Kernbach *et al.*, “Adaptive collective decision-making in limited robot swarms without communication,” *The Int. J. Robotics Res.*, vol. 32, no. 1, pp. 35–55, 2013.
- [250] T. Matthews, S. D. Ramchurn, and G. Chalkiadakis, “Competing with humans at fantasy football: Team formation in large partially-observable domains,” in *AAAI*, 2012.
- [251] S. S. Liao, J.-D. Zhang, R. Lau, and T. Wu, “Coalition formation based on marginal contributions and the markov process,” *Decis. Support Syst.*, vol. 57, pp. 355–363, 2014.
- [252] G. Chalkiadakis and C. Boutilier, “Sequentially optimal repeated coalition formation under uncertainty,” *AAMAS*, vol. 24, no. 3, pp. 441–484, 2012.
- [253] S. Sen and J. Adams, “An influence diagram based multi-criteria decision making framework for multirobot coalition formation,” *AAMAS*, vol. 29, no. 6, pp. 1061–1090, 2015.
- [254] D. S. Dos Santos and A. L. Bazzan, “Distributed clustering for group formation and task allocation in multiagent systems: A swarm intelligence approach,” *Appl. Soft Computing*, vol. 12, no. 8, pp. 2123–2131, 2012.
- [255] S. Jeong and Y. Shoham, “Bayesian coalitional games,” in *AAAI*, pp. 95–100, 2008.
- [256] K. Akkarajitsakul, E. Hossain, and D. Niyato, “Coalition-based cooperative packet delivery under uncertainty: A dynamic bayesian coalitional game,” *IEEE Trans. Mobile Comput.*, vol. 12, no. 2, pp. 371–385, 2013.
- [257] Y. Zeng, K. Xiang, D. Li, and A. V. Vasilakos, “Directional routing and scheduling for green vehicular delay tolerant networks,” *Wireless Netw.*, vol. 19, no. 2, pp. 161–173, 2013.
- [258] S. Ulbrich and M. Maurer, “Probabilistic online pomdp decision making for lane changes in fully automated driving,” in *16th Int. IEEE Conf. Intell. Transportation Syst.*, pp. 2063–2067, 2013.
- [259] S. A. Kulkarni and G. R. Rao, “Vehicular ad hoc network mobility models applied for reinforcement learning routing algorithm,” in *Contemporary Computing*, pp. 230–240, Springer, 2010.
- [260] C. Wu, S. Ohzahata, and T. Kato, “Flexible, portable, and practicable solution for routing in vanets: a fuzzy constraint q-learning approach,” *IEEE Trans. Veh. Technol.*, vol. 62, no. 9, pp. 4251–4263, 2013.

- [261] M. Boushaba, A. Hafid, A. Belbekkouche, and M. Gendreau, “Reinforcement learning based routing in wireless mesh networks,” *Wireless Netw.*, vol. 19, no. 8, pp. 2079–2091, 2013.
- [262] S. Hosseininezhad, G. N. Shirazi, and V. Leung, “Rlab: Reinforcement learning-based adaptive broadcasting for vehicular ad-hoc networks,” in *IEEE 73rd Veh. Technol. Conf.*, pp. 1–5, 2011.
- [263] A. Soua and H. Afifi, “Adaptive data collection protocol using reinforcement learning for vanets,” in *9th Int. Wireless Commun. & Mobile Computing Conf.*, pp. 1040–1045, 2013.
- [264] A. L. Bazzan, “Opportunities for multiagent systems and multiagent reinforcement learning in traffic control,” *AAMAS*, vol. 18, no. 3, pp. 342–375, 2009.
- [265] D. Houli, L. Zhiheng, and Z. Yi, “Multiobjective reinforcement learning for traffic signal control using vehicular ad hoc network,” *EURASIP J. Adv. Signal Process.*, vol. 2010, p. 7, 2010.
- [266] P. Balaji, X. German, and D. Srinivasan, “Urban traffic signal control using reinforcement learning agents,” *IET Intell. Transport Syst.*, vol. 4, no. 3, pp. 177–188, 2010.
- [267] M. Abdoos, N. Mozayani, and A. L. Bazzan, “Traffic light control in non-stationary environments based on multi agent q-learning,” in *14th Int. IEEE Conf. Intell. Transportation Syst.*, pp. 1580–1585, 2011.
- [268] I. Arel, C. Liu, T. Urbanik, and A. Kohls, “Reinforcement learning-based multi-agent system for network traffic signal control,” *IET Intell. Transport Syst.*, vol. 4, no. 2, pp. 128–135, 2010.
- [269] S. El-Tantawy and B. Abdulhai, “Multi-agent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc),” in *Int. Conf. Intell. Transportation Syst.*, pp. 319–326, 2012.
- [270] M. A. Khamis and W. Gomaa, “Adaptive multi-objective reinforcement learning with hybrid exploration for traffic signal control based on cooperative multi-agent framework,” *Eng. Appl. Artif. Intell.*, vol. 29, pp. 134–151, 2014.
- [271] K. Prabuchandran, A. Hemanth Kumar, and S. Bhatnagar, “Multi-agent reinforcement learning for traffic signal control,” in *IEEE 17th Int. Conf. Intell. Transportation Syst.*, pp. 2529–2534, 2014.

- [272] J. García-Nieto, E. Alba, and A. C. Olivera, “Swarm intelligence for traffic light scheduling: Application to real urban areas,” *Eng. Appl. Artif. Intell.*, vol. 25, no. 2, pp. 274–283, 2012.
- [273] D. Teodorovic, “Transport modeling by multi-agent systems: a swarm intelligence approach,” *Transportation planning & Technol.*, vol. 26, no. 4, pp. 289–312, 2003.
- [274] B. Tatomir and L. Rothkrantz, “Hierarchical routing in traffic using swarm-intelligence,” in *IEEE Intell. Transportation Syst. Conf.*, pp. 230–235, 2006.
- [275] D. Teodorović, “Swarm intelligence systems for transportation engineering: Principles and applications,” *Transportation Res. Part C: Emerg. Technol.*, vol. 16, no. 6, pp. 651–667, 2008.
- [276] R. Muraleedharan and L. A. Osadciw, “Cognitive security protocol for sensor based vanet using swarm intelligence,” in *43rd Asilomar Conf. Signals, Syst. & Comput.*, pp. 288–290, IEEE, 2009.
- [277] S. Manvi, M. Kakkasageri, and C. Mahapurush, “Performance analysis of aodv, dsr, and swarm intelligence routing protocols in vehicular ad hoc network environment,” in *IEEE Int. Conf. Future Comput. and Commun.*, pp. 21–25, 2009.
- [278] J. Toutouh and E. Alba, “Parallel swarm intelligence for vanets optimization,” in *7th IEEE Int. Conf. P2P, Parallel, Grid, Cloud & Internet Computing*, pp. 285–290, 2012.
- [279] P. Wararkar and S. Dorle, “Vehicular adhoc networks handovers with meta-heuristic algorithms,” in *IEEE Int. Conf. Electronic Syst., Signal Process. & Computing Technol.*, pp. 160–165, 2014.
- [280] S. Medetov, M. Bakhouya, J. Gaber, K. Zinedine, and M. Wack, “A bee-inspired approach for information dissemination in vanets,” in *IEEE Int. Conf. Multimedia Computing & Syst.*, pp. 849–854, 2014.
- [281] M. Abu Alsheikh, D. T. Hoang, D. Niyato, H.-P. Tan, and S. Lin, “Markov decision processes with applications in wireless sensor networks: A survey,” *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1239–1267, 2015.
- [282] D. L. Kovacs, W. Li, N. Fukuta, and T. Watanabe, “Mixed observability markov decision processes for overall network performance optimization in wireless sensor networks,” in *IEEE Int. Conf. Adv. Inf. Netw. & Appl.*, pp. 289–298, 2012.

- [283] J. Pajarinen, A. Hottinen, and J. Peltonen, “Optimizing spatial and temporal reuse in wireless networks by decentralized partially observable markov decision processes,” *IEEE Trans. Mobile Comput.*, vol. 13, no. 4, pp. 866–879, 2014.
- [284] J. Zhang, L. Xu, S. Zhou, and X. Ye, “A novel sleep scheduling scheme in green wireless sensor networks,” *The J. Supercomputing*, vol. 71, no. 3, pp. 1067–1094, 2015.
- [285] J. S. Sandhu *et al.*, “Wireless sensor networks for commercial lighting control: decision making with multi-agent systems,” in *AAAI Workshop sensor netw.*, vol. 10, pp. 131–140, 2004.
- [286] S. S. Iyengar, H.-C. Wu, N. Balakrishnan, and S. Y. Chang, “Biologically inspired cooperative routing for wireless mobile sensor networks,” *IEEE Syst. J.*, vol. 1, no. 1, pp. 29–37, 2007.
- [287] M. Farooq, “A comprehensive survey of nature-inspired routing protocols,” in *Bee-Inspired Protocol Eng.*, pp. 19–52, Springer, 2009.
- [288] A. Zengin *et al.*, “A survey on swarm intelligence based routing protocols in wireless sensor networks,” *Int. J. Physical Sci.*, vol. 5, no. 14, pp. 2118–2126, 2010.
- [289] F. Ducatelle, G. Di Caro, and L. Gambardella, “Principles and applications of swarm intelligence for adaptive routing in telecommunications networks,” *Swarm Intell.*, vol. 4, no. 3, pp. 173–198, 2010.
- [290] M. Saleem, G. A. Di Caro, and M. Farooq, “Swarm intelligence based routing protocol for wireless sensor networks: Survey and future directions,” *Inform. Sci.*, vol. 181, no. 20, pp. 4597–4624, 2011.
- [291] A. M. Zungeru, L.-M. Ang, and K. P. Seng, “Classical and swarm intelligence based routing protocols for wireless sensor networks: A survey and comparison,” *J. Netw. & Comput. Appl.*, vol. 35, no. 5, pp. 1508–1536, 2012.
- [292] C. Öztürk, D. Karaboğa, and B. Görkemli, “Artificial bee colony algorithm for dynamic deployment of wireless sensor networks,” *Turkish J. Elect. Eng. & Comput. Sci.*, vol. 20, no. 2, pp. 255–262, 2012.
- [293] H. Ren and M. Q.-H. Meng, “Biologically inspired approaches for wireless sensor networks,” in *Proc. IEEE Int. Conf. Mechatronics & Autom.*, pp. 762–768, 2006.

- [294] B. Singh and D. K. Lobiyal, “A novel energy-aware cluster head selection based on particle swarm optimization for wireless sensor networks,” *Human-Centric Computing & Inf. Sci.*, vol. 2, no. 1, pp. 1–18, 2012.
- [295] P. K. Krishnappa and B. P. Babu, “Investigating open issues in swarm intelligence for mitigating security threats in manet,” *Int. J. Elect. & Comput. Eng.*, vol. 5, no. 5, 2015.
- [296] R. V. Kulkarni and G. K. Venayagamoorthy, “Particle swarm optimization in wireless-sensor networks: A brief survey,” *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 41, no. 2, pp. 262–267, 2011.
- [297] W.-H. Liao, Y. Kao, and Y.-S. Li, “A sensor deployment approach using glowworm swarm optimization algorithm in wireless sensor networks,” *Expert Syst. with Appl.*, vol. 38, no. 10, pp. 12180–12188, 2011.
- [298] S. Roy *et al.*, “A survey of game theory as applied to network security,” in *43rd Hawaii Int. Conf. Syst. Sci.*, pp. 1–10, IEEE, 2010.
- [299] A. A. Strikos, “A full approach for intrusion detection in wireless sensor networks,” *School of Inf. & Commun. Technol.*, 2007.
- [300] K. Premkumar and A. Kumar, “Optimal sleep-wake scheduling for quickest intrusion detection using wireless sensor networks,” in *IEEE 27th Conf. Comput. Commun.*, pp. 1400–1408, 2008.
- [301] Y. Liu, C. Comaniciu, and H. Man, “A bayesian game approach for intrusion detection in wireless ad hoc networks,” in *Proc. Workshop Game theory for commun. & netw.*, p. 4, ACM, 2006.
- [302] A. Servin and D. Kudenko, “Multi-agent reinforcement learning for intrusion detection: A case study and evaluation,” in *Multiagent Syst. Technol.*, pp. 159–170, Springer, 2008.
- [303] F. Kuang, S. Zhang, Z. Jin, and W. Xu, “A novel svm by combining kernel principal component analysis and improved chaotic particle swarm optimization for intrusion detection,” *Soft Computing*, pp. 1–13, 2015.
- [304] L. Wang, C. Dong, J. Hu, and G. Li, “Network intrusion detection using support vector machine based on particle swarm optimization,” *Int. Conf. Appl. Sci. & Eng. Innovation*, 2015.
- [305] A. A. Aburomman and M. B. I. Reaz, “A novel svm-knn-pso ensemble method for intrusion detection system,” *Appl. Soft Computing*, vol. 38, pp. 360–372, 2016.

- [306] B. M. Hosseini, B. Amiri, M. Mirzabagheri, and Y. Shi, “A new intrusion detection approach using pso based multiple criteria linear programming,” *Procedia Comput. Sci.*, vol. 55, pp. 231–237, 2015.
- [307] P. Wang, H.-T. Lin, and T.-S. Wang, “An improved ant colony system algorithm for solving the ip traceback problem,” *Inf. Sci.*, vol. 326, pp. 172–187, 2016.
- [308] C. Wu, K. Chowdhury, M. Di Felice, and W. Meleis, “Spectrum management of cognitive radio using multi-agent reinforcement learning,” in *AAMAS: Industry track*, pp. 1705–1712, 2010.
- [309] H. Li, “Multiagent q-learning for aloha-like spectrum access in cognitive radio systems,” *EURASIP J. Wireless Commun. & Netw.*, vol. 2010, p. 56, 2010.
- [310] K.-L. A. Yau, P. Komisarczuk, and P. D. Teal, “Applications of reinforcement learning to cognitive radio networks,” in *IEEE Int. Conf. Commun. Workshops*, pp. 1–6, 2010.
- [311] A. Galindo-Serrano and L. Giupponi, “Distributed q-learning for aggregated interference control in cognitive radio networks,” *IEEE Trans. Veh. Technol.*, vol. 59, no. 4, pp. 1823–1834, 2010.
- [312] J. Lunden, S. R. Kulkarni, V. Koivunen, and H. V. Poor, “Multiagent reinforcement learning based spectrum sensing policies for cognitive radio networks,” *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 5, pp. 858–868, 2013.
- [313] Q. Zhu, H. Li, Z. Han, and T. Basar, “A stochastic game model for jamming in multi-channel cognitive radio systems,” in *IEEE Int. Conf. Commun.*, pp. 1–6, 2010.
- [314] B. Wang, Y. Wu, and K. R. Liu, “Game theory for cognitive radio networks: An overview,” *Comput. Netw.*, vol. 54, no. 14, pp. 2537–2561, 2010.
- [315] Q. Zhu, Z. Han, and T. Başar, “A differential game approach to distributed demand side management in smart grid,” in *IEEE Int. Conf. Commun.*, pp. 3345–3350, 2012.
- [316] F. S. Melo, A. Sardinha, S. Witwicki, L. M. Ramirez-Elizondo, and M. T. Spaan, “Decentralized multiagent planning for balance control in smart grids,” in *Proc. 1st Int. Workshop Inf. Technol. for Energy Appl.*, vol. 923, 2012.

- [317] J. Wu and X. Guan, “Coordinated multi-microgrids optimal control algorithm for smart distribution management system,” *IEEE Trans. Smart Grid*, vol. 4, no. 4, pp. 2174–2181, 2013.
- [318] F. Lauri *et al.*, “Managing power flows in microgrids using multi-agent reinforcement learning,” *Agent Technol. in Energy Syst.*, 2013.
- [319] M. Peters, W. Ketter, M. Saar-Tsechansky, and J. Collins, “A reinforcement learning approach to autonomous decision-making in smart electricity markets,” *Mach. Learn.*, vol. 92, no. 1, pp. 5–39, 2013.
- [320] Y. Huang, A. Brocco, P. Kuonen, M. Courant, and B. Hirsbrunner, “Smart-grid: A fully decentralized grid scheduling framework supported by swarm intelligence,” in *7th IEEE Int. Conf. Grid & Cooperative Computing*, pp. 160–168, 2008.
- [321] P. Faria, Z. A. Vale, J. Soares, and J. Ferreira, “Particle swarm optimization applied to integrated demand response resources scheduling,” in *IEEE Symp. Computational Intell. Appl. in Smart Grid*, pp. 1–8, 2011.
- [322] Q. Kang, M. Zhou, J. An, and Q. Wu, “Swarm intelligence approaches to optimal power flow problem with distributed generator failures in power networks,” *IEEE Trans. Autom. Sci. Eng*, vol. 10, no. 2, pp. 343–353, 2013.
- [323] M. Hemmati, A. Yassine, and S. Shirmohammadi, “A dec-pomdp model for congestion avoidance and fair allocation of network bandwidth in rate-adaptive video streaming,” in *IEEE Symp. Series on Computational Intell.*, pp. 1182–1189, 2015.
- [324] F. Teng and F. Magoulès, “A new game theoretical resource allocation algorithm for cloud computing,” in *Adv. Grid & Pervasive Computing*, pp. 321–330, Springer, 2010.
- [325] J. Wu, X. Xu, P. Zhang, and C. Liu, “A novel multi-agent reinforcement learning approach for job scheduling in grid computing,” *Future Generation Comput. Syst.*, vol. 27, no. 5, pp. 430–439, 2011.
- [326] S. Yang, Q. Liu, and J. Wang, “Distributed optimization based on a multi-agent system in the presence of communication delays,” *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 5, pp. 717–728, 2017.
- [327] W. Stadler, “Fundamentals of multicriteria optimization,” in *Multicriteria Optimization in Eng. & in the Sci.*, pp. 1–25, Springer, 1988.
- [328] T. L. Vincent, “Game theory as a design tool,” *J. Mechanisms, Transmissions, & Autom. in Des.*, vol. 105, no. 2, pp. 165–170, 1983.

- [329] J. Zhang, D. Qi, and G. Zhao, “A game theoretical formulation for distributed optimization problems,” in *10th IEEE Int. Conf. Control & Automation*, pp. 1939–1944, 2013.
- [330] K. Hasegawa and M. Noto, “Swarm intelligence algorithm for optimality discovery in distributed constraint optimization,” in *IEEE Int. Conf. Syst., Man, Cybern.*, pp. 3611–3616, 2014.
- [331] B. Cao *et al.*, “Distributed parallel particle swarm optimization for multi-objective and many-objective large-scale optimization,” *IEEE Access*, 2017.
- [332] K. M. Passino, “Biomimicry of bacterial foraging for distributed optimization and control,” *IEEE Control Syst.*, vol. 22, no. 3, pp. 52–67, 2002.
- [333] A. Nedić and A. Olshevsky, “Distributed optimization over time-varying directed graphs,” *IEEE Trans. Autom. Control*, vol. 60, no. 3, pp. 601–615, 2015.
- [334] B. Gharesifard and J. Cortés, “Distributed continuous-time convex optimization on weight-balanced digraphs,” *IEEE Trans. Autom. Control*, vol. 59, no. 3, pp. 781–786, 2014.
- [335] L. Braubach, A. Pokahr, and W. Lamersdorf, “A universal criteria catalog for evaluation of heterogeneous agent development artifacts,” *From Agent Theory to Agent Implementation*, pp. 19–28, 2008.
- [336] R. N. Lass, E. A. Sultanik, and W. C. Regli, “Metrics for multiagent systems,” in *Performance evaluation & benchmarking of Intell. syst.*, pp. 1–19, Springer, 2009.
- [337] P. Di Bitonto, M. Laterza, T. Roselli, and V. Rossano, “An evaluation method for multi-agent systems,” *Agent & Multiagent Syst.: Technol. & Appl.*, pp. 32–41, 2010.
- [338] T. Marir, F. Mokhati, H. Bouchelaghem-Seridi, and B. Benaissa, “Dynamic metrics for multi-agent systems using aspect-oriented programming,” in *German Conf. Multiagent Syst. Technol.*, pp. 58–72, Springer, 2016.
- [339] S. N. Parragh, K. F. Doerner, and R. F. Hartl, “A survey on pickup and delivery problems part i: transportation between customers and depot,” *Journal für Betriebswirtschaft*, vol. 58, no. 1, pp. 21–51, 2008.
- [340] S. N. Parragh, K. F. Doerner, and R. F. Hartl, “A survey on pickup and delivery models part ii: Transportation between pickup and delivery locations,” *Journal für Betriebswirtschaft*, vol. 58, no. 2, pp. 81–117, 2008.
- [341] B. Coltin, “Multi-agent pickup and delivery planning with transfers,” 2014.

- [342] M. W. Savelsbergh and M. Sol, “The general pickup and delivery problem,” *Transportation science*, vol. 29, no. 1, pp. 17–29, 1995.
- [343] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, “Static pickup and delivery problems: a classification scheme and survey,” *Top*, vol. 15, no. 1, pp. 1–31, 2007.
- [344] G. Berbeglia, J.-F. Cordeau, and G. Laporte, “Dynamic pickup and delivery problems,” *European J. operational research*, vol. 202, no. 1, pp. 8–15, 2010.
- [345] B. Coltin and M. Veloso, “Online pickup and delivery planning with transfers for mobile robots,” in *IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 5786–5791, IEEE, 2014.
- [346] Q. Lu and M. Dessouky, “An exact algorithm for the multiple vehicle pickup and delivery problem,” *Transportation Science*, vol. 38, no. 4, pp. 503–514, 2004.
- [347] M. G. S. Furtado, P. Munari, and R. Morabito, “Pickup and delivery problem with time windows: a new compact two-index formulation,” *Operations Research Letters*, vol. 45, no. 4, pp. 334–341, 2017.
- [348] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to NP-completeness*. WH Freeman and Company, San Francisco, 1979.
- [349] C. E. Cortés, M. Matamala, and C. Contardo, “The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method,” *European J. Operational Research*, vol. 200, no. 3, pp. 711–724, 2010.
- [350] A. Rais, F. Alvelos, and M. S. Carvalho, “New mixed integer-programming model for the pickup-and-delivery problem with transshipment,” *European J. Operational Research*, vol. 235, no. 3, pp. 530–539, 2014.
- [351] B. Coltin and M. Veloso, “Optimizing for transfers in a multi-vehicle collection and delivery problem,” in *Distributed Autonomous Robotic Systems*, pp. 91–103, Springer, 2014.
- [352] V. Kachitvichyanukul *et al.*, “A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery,” *Computers & Operations Research*, vol. 36, no. 5, pp. 1693–1702, 2009.
- [353] R. Bent and P. Van Hentenryck, “A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows,” *Computers & Operations Research*, vol. 33, no. 4, pp. 875–893, 2006.

- [354] K. Ganesh and T. Narendran, “Taste: a two-phase heuristic to solve a routing problem with simultaneous delivery and pick-up,” *The Int. J. Advanced Manufacturing Technology*, vol. 37, no. 11-12, pp. 1221–1231, 2008.
- [355] S. Ropke and J.-F. Cordeau, “Branch and cut and price for the pickup and delivery problem with time windows,” *Transportation Science*, vol. 43, no. 3, pp. 267–286, 2009.
- [356] G. P. Perrucci, F. H. Fitzek, and J. Widmer, “Survey on energy consumption entities on the smartphone platform,” in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pp. 1–6, IEEE, 2011.
- [357] J.-L. Zhang, Y.-W. Zhao, D.-J. Peng, and W.-L. Wang, “A hybrid quantum-inspired evolutionary algorithm for capacitated vehicle routing problem,” in *International Conference on Intelligent Computing*, pp. 31–38, Springer, 2008.
- [358] H.-F. Wang and Y.-Y. Chen, “A genetic algorithm for the simultaneous delivery and pickup problems with time window,” *Computers & Industrial Engineering*, vol. 62, no. 1, pp. 84–95, 2012.
- [359] G. Pankratz, “A grouping genetic algorithm for the pickup and delivery problem with time windows,” *Or Spectrum*, vol. 27, no. 1, pp. 21–41, 2005.
- [360] H. Xiao-feng and M. Liang, “Quantum-inspired ant colony algorithm for vehicle routing problem with time windows,” *Systems Engineering-Theory & Practice*, vol. 5, no. 33, pp. 1255–1261, 2013.
- [361] O. Dakroub, C. M. Boukhater, F. Lahoud, M. Awad, and H. Artail, “An intelligent carpooling app for a green social solution to traffic and parking congestions,” in *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pp. 2401–2408, IEEE, 2013.
- [362] Z. Ursani, D. Essam, D. Cornforth, and R. Stocker, “Localized genetic algorithm for vehicle routing problem with time windows,” *Applied Soft Computing*, vol. 11, no. 8, pp. 5375–5390, 2011.
- [363] Z. Jia, J. Yu, X. Ai, X. Xu, and D. Yang, “Cooperative multiple task assignment problem with stochastic velocities and time windows for heterogeneous unmanned aerial vehicles using a genetic algorithm,” *Aerospace Science and Technology*, vol. 76, pp. 112–125, 2018.
- [364] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, “Geodesic convolutional neural networks on riemannian manifolds,” in *Proceedings of the IEEE international conference on computer vision workshops*, pp. 37–45, 2015.

- [365] I. Kokkinos and A. Yuille, “Scale invariance without scale selection,” 2008.
- [366] I. Kokkinos, M. M. Bronstein, R. Litman, and A. M. Bronstein, “Intrinsic shape context descriptors for deformable shapes,” in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 159–166, IEEE, 2012.
- [367] O. Tuzel, F. Porikli, and P. Meer, “Region covariance: A fast descriptor for detection and classification,” in *European conference on computer vision*, pp. 589–600, Springer, 2006.
- [368] S. C. Turaga, J. F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H. S. Seung, “Convolutional networks can learn to generate affinity graphs for image segmentation,” *Neural computation*, vol. 22, no. 2, pp. 511–538, 2010.
- [369] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.
- [370] M. Simonovsky and N. Komodakis, “Dynamic edgeconditioned filters in convolutional neural networks on graphs,” in *Proc. CVPR*, 2017.
- [371] M. Niepert, M. Ahmed, and K. Kutzkov, “Learning convolutional neural networks for graphs,” in *International conference on machine learning*, pp. 2014–2023, 2016.
- [372] data.world, “Fedex facilities,” 2013.
- [373] M. Haklay and P. Weber, “Openstreetmap: User-generated street maps,” *Ieee Pervas Comput*, vol. 7, no. 4, pp. 12–18, 2008.
- [374] G. Van Brummelen, *Heavenly mathematics: The forgotten art of spherical trigonometry*. Princeton University Press, 2012.
- [375] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Netw.*, vol. 61, pp. 85–117, 2015.
- [376] W. F. Schmidt, M. A. Kraaijveld, and R. P. Duin, “Feedforward neural networks with random weights,” in *Proceedings on the 11th IAPR International Conference on Pattern Recognition, Vol. II. Conference B: Pattern Recognition Methodology and Systems*, pp. 1–4, IEEE, 1992.
- [377] Y.-H. Pao, G.-H. Park, and D. J. Sobajic, “Learning and generalization characteristics of the random vector functional-link net,” *Neurocomputing*, vol. 6, no. 2, pp. 163–180, 1994.

- [378] H. A. Te Braake and G. Van Straten, “Random activation weight neural net (rawn) for fast non-iterative training,” *Engineering Applications of Artificial Intelligence*, vol. 8, no. 1, pp. 71–80, 1995.
- [379] G. B. Huang, Q. Y. Zhu, and C. K. Siew, “Extreme learning machine: a new learning scheme of feedforward neural networks,” in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 985–990, IEEE, 2004.
- [380] J. L. Elman, “Finding structure in time,” *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.
- [381] M. Jordan, “Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986,” tech. rep., California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986.
- [382] R. J. Williams and D. Zipser, “A learning algorithm for continually running fully recurrent neural networks,” *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [383] J. T. Connor, R. D. Martin, and L. E. Atlas, “Recurrent neural networks and robust time series prediction,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240–254, 1994.
- [384] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [385] Y.-H. Pao and Y. Takefuji, “Functional-link net computing: theory, system architecture, and functionalities,” *Computer*, vol. 25, no. 5, pp. 76–79, 1992.
- [386] Ö. F. Ertugrul, “Forecasting electricity load by a novel recurrent extreme learning machines approach,” *International Journal of Electrical Power & Energy Systems*, vol. 78, pp. 429–435, 2016.
- [387] H. Zhao and J. Zhang, “Nonlinear dynamic system identification using pipelined functional link artificial recurrent neural network,” *Neurocomputing*, vol. 72, no. 13, pp. 3046–3054, 2009.
- [388] D. E. Rumelhart, J. L. McClelland, P. R. Group, *et al.*, “Parallel distributed processing: Explorations in the microstructures of cognition. volume 1: Foundations,” 1986.
- [389] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

- [390] S. El Hahi and Y. Bengio, “Hierarchical recurrent neural networks for long-term dependencies,” in *NIPS*, pp. 493–499, Citeseer, 1995.
- [391] J. Hu and J. Wang, “Global stability of complex-valued recurrent neural networks with time-delays,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 6, pp. 853–865, 2012.
- [392] Z. Tu, N. Ding, L. Li, Y. Feng, L. Zou, and W. Zhang, “Adaptive synchronization of memristive neural networks with time-varying delays and reaction–diffusion term,” *Applied Mathematics and Computation*, vol. 311, pp. 118–128, 2017.
- [393] L. Duan, L. Huang, Z. Guo, and X. Fang, “Periodic attractor for reaction–diffusion high-order hopfield neural networks with time-varying delays,” *Computers & Mathematics with Applications*, vol. 73, no. 2, pp. 233–245, 2017.
- [394] D. Pham and S. Oh, “A recurrent back propagation neural network for dynamic system identification,” *J. of Systems Eng.*, vol. 2, no. 4, pp. 213–223, 1992.
- [395] D. Pham and X. Liu, “Dynamic system identification using partially recurrent neural networks,” *J. Syst. Eng*, vol. 2, no. 2, pp. 90–97, 1992.
- [396] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
- [397] R. J. Williams and D. Zipser, “Gradient-based learning algorithms for recurrent networks and their computational complexity,” *Back-propagation: Theory, architectures and applications*, pp. 433–486, 1995.
- [398] J. Martens and I. Sutskever, “Learning recurrent neural networks with hessian-free optimization,” in *Proceedings of the 28th International Conference on Machine Learning*, pp. 1033–1040, 2011.
- [399] X. Fu, S. Li, and I. Jaithwa, “Implement optimal vector control for lcl-filter-based grid-connected converters by using recurrent neural networks,” *IEEE Transactions on Industrial Electronics*, vol. 62, no. 7, pp. 4443–4454, 2015.
- [400] X. Wang and Y. Huang, “Convergence study in extended kalman filter-based training of recurrent neural networks,” *IEEE Transactions on Neural Networks*, vol. 22, no. 4, pp. 588–600, 2011.
- [401] S. Omkar, D. Mudigere, J. Senthilnath, and M. V. Kumar, “Identification of helicopter dynamics based on flight data using nature inspired techniques,”

- International Journal of Applied Metaheuristic Computing (IJAMC)*, vol. 6, no. 3, pp. 38–52, 2015.
- [402] A. Blanco, M. Delgado, and M. C. Pegalajar, “A real-coded genetic algorithm for training recurrent neural networks,” *Neural networks*, vol. 14, no. 1, pp. 93–105, 2001.
- [403] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, “An overview of reservoir computing: theory, applications and implementations,” in *Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007*, pp. 471–482, 2007.
- [404] H. Jaeger, “Adaptive nonlinear system identification with echo state networks,” *networks*, vol. 8, no. 9, p. 17, 2003.
- [405] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: A new framework for neural computation based on perturbations,” *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.
- [406] L. Zhang and P. Suganthan, “A survey of randomized algorithms for training neural networks,” *Information Sciences*, vol. 364, pp. 146–155, 2016.
- [407] L. Tang, Y. Wu, and L. Yu, “A non-iterative decomposition-ensemble learning paradigm using rvfl network for crude oil price forecasting,” *Applied Soft Computing*, 2017.
- [408] D. P. Mesquita, J. P. P. Gomes, L. R. Rodrigues, S. A. Oliveira, and R. K. Galvão, “Building selective ensembles of randomization based neural networks with the successive projections algorithm,” *Applied Soft Computing*, 2017.
- [409] L. Zhang and P. Suganthan, “A comprehensive evaluation of random vector functional link networks,” *Information Sciences*, vol. 367, pp. 1094–1105, 2016.
- [410] Y. Ren, P. Suganthan, N. Srikanth, and G. Amaratunga, “Random vector functional link network for short-term electricity load demand forecasting,” *Information Sciences*, vol. 367, pp. 1078–1093, 2016.
- [411] N. Wang, M. J. Er, and M. Han, “Generalized single-hidden layer feed-forward networks for regression problems,” *IEEE transactions on neural networks and learning systems*, vol. 26, no. 6, pp. 1161–1176, 2015.
- [412] J.-C. Yin, A. N. Perakis, and N. Wang, “A real-time ship roll motion prediction using wavelet transform and variable rbf network,” *Ocean Engineering*, vol. 160, pp. 10–19, 2018.

- [413] N. Wang, M. Han, N. Dong, and M. J. Er, “Constructive multi-output extreme learning machine with application to large tanker motion dynamics identification,” *Neurocomputing*, vol. 128, pp. 59–72, 2014.
- [414] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, and N. Sundararajan, “On-line sequential extreme learning machine,” *Computational Intelligence*, vol. 2005, pp. 232–237, 2005.
- [415] N. Wang, M. J. Er, X.-Y. Meng, and X. Li, “An online self-organizing scheme for parsimonious and accurate fuzzy neural networks,” *International Journal of Neural Systems*, vol. 20, no. 05, pp. 389–403, 2010.
- [416] Y. Gu, J. Liu, Y. Chen, X. Jiang, and H. Yu, “Toselm: timeliness online sequential extreme learning machine,” *Neurocomputing*, vol. 128, pp. 119–127, 2014.
- [417] N. Wang, M. J. Er, and M. Han, “Parsimonious extreme learning machine using recursive orthogonal least squares,” *IEEE transactions on neural networks and learning systems*, vol. 25, no. 10, pp. 1828–1841, 2014.
- [418] K. Basterretxea, J. Tarela, and I. Del Campo, “Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons,” *IEEE Proceedings-Circuits, Devices and Systems*, vol. 151, no. 1, pp. 18–24, 2004.
- [419] D. Needell, “Randomized kaczmarz solver for noisy linear systems,” *BIT Numerical Mathematics*, vol. 50, no. 2, pp. 395–403, 2010.
- [420] T. Strohmer and R. Vershynin, “A randomized kaczmarz algorithm with exponential convergence,” *Journal of Fourier Analysis and Applications*, vol. 15, no. 2, pp. 262–278, 2009.
- [421] M. Q.-Y. ZHU and D. G.-B. HUANG, “Basic elm algorithms,” 2004.
- [422] A. Weigend and N. Gershenfeld, “Time series prediction: Forecasting the future and understanding the past. 1994,” in *Proceedings of a NATO Advanced Research Workshop on Comparative Time Series Analysis, held in Santa Fe, New Mexico*.
- [423] DataMarket, “Time series data library,” 2016.
- [424] M. Lichman, “UCI machine learning repository,” 2013.
- [425] “Australian energy market operator,” 2015.
- [426] P. A. Morettin and C. Toloï, *Analise de series temporais*. Blucher, 2006.

- [427] L. Friedman, “End-to-end learning from tesla autopilot driving data,” 2018.
- [428] Kaggle, “Japan population data,” 2018.
- [429] Kaggle, “Sp500 since 1950,” 2018.
- [430] Y. Cui, J. Zhai, and X. Wang, “Extreme learning machine based on cross entropy,” in *Machine Learning and Cybernetics (ICMLC), 2016 International Conference on*, vol. 2, pp. 1066–1071, IEEE, 2016.
- [431] G. Huang, G.-B. Huang, S. Song, and K. You, “Trends in extreme learning machines: a review,” *Neural Networks*, vol. 61, pp. 32–48, 2015.
- [432] Y. Rizk and M. Awad, “Context dependent input weight selection for regression extreme learning machines,” in *Artificial Neural Networks and Machine Learning–ICANN 2017: 26th International Conference on Artificial Neural Networks, Alghero, Italy, September 11-14, 2017, Proceedings (Vol. 10614)*, pp. 749–750, Springer, 2017.
- [433] J. Tapson, P. de Chazal, and A. van Schaik, “Explicit computation of input weights in extreme learning machines,” in *Proceedings of ELM-2014 Volume 1*, pp. 41–49, Springer, 2015.
- [434] M. D. McDonnell, M. D. Tissera, T. Vladusich, A. van Schaik, and J. Tapson, “Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the ‘extreme learning machine’ algorithm,” *PloS one*, vol. 10, no. 8, p. e0134254, 2015.
- [435] G. Huang, S. Song, J. N. Gupta, and C. Wu, “Semi-supervised and unsupervised extreme learning machines,” *IEEE Transactions on cybernetics*, vol. 44, no. 12, pp. 2405–2417, 2014.
- [436] Y. Rizk and M. Awad, “On the distributed implementation of unsupervised extreme learning machines for big data,” *Procedia Computer Science*, vol. 53, pp. 167–174, 2015.
- [437] H. Zhou, G.-B. Huang, Z. Lin, H. Wang, and Y. C. Soh, “Stacked extreme learning machines,” *IEEE transactions on cybernetics*, vol. 45, no. 9, pp. 2013–2025, 2015.
- [438] J. Tang, C. Deng, and G.-B. Huang, “Extreme learning machine for multilayer perceptron,” *IEEE transactions on neural networks and learning systems*, vol. 27, no. 4, pp. 809–821, 2016.
- [439] M. D. Tissera and M. D. McDonnell, “Deep extreme learning machines: supervised autoencoding architecture for classification,” *Neurocomputing*, vol. 174, pp. 42–49, 2016.

- [440] W. Zhu, J. Miao, and L. Qing, “Constrained extreme learning machine: a novel highly discriminative random feedforward neural network,” in *International Joint Conference on Neural Networks*, pp. 800–807, IEEE, 2014.
- [441] X. Liu, J. Miao, L. Qing, and B. Cao, “Class-constrained extreme learning machine,” in *Proceedings of ELM-2015 Volume 1*, pp. 521–530, Springer, 2016.
- [442] C. Cervellera and D. Macciò, “Low-discrepancy points for deterministic assignment of hidden weights in extreme learning machines,” *IEEE Trans. on neural networks and learning systems*, vol. 27, no. 4, pp. 891–896, 2016.
- [443] P. A. Henríquez and G. A. Ruz, “Extreme learning machine with a deterministic assignment of hidden weights in two parallel layers,” *Neurocomputing*, vol. 226, pp. 109–116, 2017.
- [444] P. Bratley and B. L. Fox, “Algorithm 659: Implementing sobol’s quasirandom sequence generator,” *ACM Trans. on Mathematical Software*, vol. 14, no. 1, pp. 88–100, 1988.
- [445] J. H. Halton, “Algorithm 247: Radical-inverse quasi-random point sequence,” *Communications of the ACM*, vol. 7, no. 12, pp. 701–702, 1964.
- [446] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [447] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong, “Representational learning with extreme learning machine for big data,” *IEEE intelligent systems*, vol. 28, no. 6, pp. 31–34, 2013.
- [448] M. Jamett and G. Acuña, “An interval approach for weight’s initialization of feedforward neural networks,” in *Mexican International Conference on Artificial Intelligence*, pp. 305–315, Springer, 2006.
- [449] Q. V. Le, N. Jaitly, and G. E. Hinton, “A simple way to initialize recurrent networks of rectified linear units,” *arXiv preprint arXiv:1504.00941*, 2015.
- [450] G. Chen, R. Xu, and S. N. Srihari, “Sequential labeling with online deep learning: Exploring model initialization,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 772–788, Springer, 2016.
- [451] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, “Why does unsupervised pre-training help deep learning?,” *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

- [452] P. Werbos, “Beyond regression: new fools for prediction and analysis in the behavioral sciences,” *PhD thesis, Harvard University*, 1974.
- [453] Kaggle, “California housing dataset,” 2018.
- [454] N. Hajj, Y. Rizk, and M. Awad, “A subjectivity classification framework for sports articles using cortical algorithms for feature selection,” *Springer Neural Computing and Applications*, 2018.
- [455] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.
- [456] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning*, pp. 448–456, 2015.
- [457] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, *et al.*, “Dsd: Dense-sparse-dense training for deep neural networks,” *International Conference on Learning Representations*, pp. 1–13, 2017.
- [458] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [459] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, “Regularization of neural networks using dropconnect,” *International Conference on Machine Learning*, pp. 1058–1066, 2013.
- [460] K. Zolna, D. Arpit, D. Suhubdy, and Y. Bengio, “Fraternal dropout,” *International Conference on Learning Representations*, pp. 1–14, 2018.
- [461] K. Zhai and H. Wang, “Adaptive dropout with rademacher complexity regularization,” *International Conference on Learning Representations*, pp. 1–17, 2018.
- [462] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, “Batch normalized recurrent neural networks,” in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 2657–2661, IEEE, 2016.
- [463] T. van Laarhoven, “L2 regularization versus batch and weight normalization,” *Neural Information Processing Systems*, 2017.

- [464] C. E. Shannon, “A mathematical theory of communication,” *ACM SIG-MOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.
- [465] R. Vicente, M. Wibral, M. Lindner, and G. Pipa, “Transfer entropy—a model-free measure of effective connectivity for the neurosciences,” *Journal of computational neuroscience*, vol. 30, no. 1, pp. 45–67, 2011.
- [466] S. Herzog, C. Tetzlaff, and F. Wörgötter, “Transfer entropy-based feedback improves performance in artificial neural networks,” *arXiv preprint arXiv:1706.04265*, 2017.
- [467] T. Bossomaier, L. Barnett, M. Harré, and J. T. Lizier, “Miscellaneous applications of transfer entropy,” in *An Introduction to Transfer Entropy*, pp. 139–165, Springer, 2016.
- [468] C. Wang, H. Yu, R. W. Grout, K.-L. Ma, and J. H. Chen, “Analyzing information transfer in time-varying multivariate data,” in *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pp. 99–106, IEEE, 2011.
- [469] J. T. Lizier, “Jidt: An information-theoretic toolkit for studying the dynamics of complex systems,” *Frontiers in Robotics and AI*, vol. 1, p. 11, 2014.
- [470] S. C. Botelho and R. Alami, “M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement,” in *Proc. IEEE Int’l Conf. Robotics and Automation*, vol. 2, pp. 1234–1239, 1999.
- [471] R. Zlot and A. Stentz, “Market-based multirobot coordination for complex tasks,” *Int. J. Robotics Research*, vol. 25, no. 1, pp. 73–101, 2006.
- [472] J. Chen, Y. Yang, and L. Wei, “Research on the approach of task decomposition in soccer robot system,” in *IEEE Int. Conf. Digital Manufacturing and Automation*, vol. 2, pp. 284–289, 2010.
- [473] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, “Market-based multirobot coordination: A survey and analysis,” *Proc. IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.
- [474] X. Li, S. Dang, K. Li, and Q. Liu, “Multi-agent-based battlefield reconnaissance simulation by novel task decomposition and allocation,” in *5th IEEE Int. Conf. Comput. Sci. and Educ.*, pp. 1410–1414, 2010.
- [475] L. C. Cobo, C. L. Isbell Jr, and A. L. Thomaz, “Automatic task decomposition and state abstraction from demonstration,” in *Proc. 11th Int. Conf. Autonomous Agents & Multiagent Syst.-Volume 1*, pp. 483–490, 2012.

- [476] T. Hu, S. Messelodi, and O. Lanz, “Dynamic task decomposition for probabilistic tracking in complex scenes,” in *22nd IEEE Int. Conf. Pattern Recognition*, pp. 4134–4139, 2014.
- [477] R. Zlot and A. Stentz, “Multirobot control using task abstraction in a market framework,” in *Collaborative Technology Alliances Conf.*, 2003.
- [478] W. Wang, J. Jiang, B. An, Y. Jiang, and B. Chen, “Toward efficient team formation for crowdsourcing in noncooperative social networks,” *IEEE Trans. Cybern.*, 2016.
- [479] T. Gunn and J. Anderson, “Dynamic heterogeneous team formation for robotic urban search and rescue,” *Procedia Computer Science*, vol. 19, pp. 22–31, 2013.
- [480] M. Klusch and A. Gerber, “Dynamic coalition formation among rational agents,” *IEEE Intell. Syst.*, no. 3, pp. 42–47, 2002.
- [481] D. M. Lyons, M. Arbib, *et al.*, “A formal model of computation for sensory-based robotics,” *IEEE Trans. Robot. Autom.*, vol. 5, no. 3, pp. 280–293, 1989.
- [482] Z. Yu, L. Shuhua, F. Shuai, and W. Di, “A quantum-inspired ant colony optimization for robot coalition formation,” in *Chinese Control and Decision Conf.*, pp. 626–631, IEEE, 2009.
- [483] B. Xu, Z. Yang, Y. Ge, and Z. Peng, “Coalition formation in multi-agent systems based on improved particle swarm optimization algorithm,” *Int. J. Hybrid Inform. Technology*, vol. 8, no. 3, 2015.
- [484] Z. Li, B. Xu, L. Yang, J. Chen, and K. Li, “Quantum evolutionary algorithm for multi-robot coalition formation,” in *Proceedings of the 1st ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp. 295–302, 2009.
- [485] S. Kraus, O. Shehory, and G. Taase, “Coalition formation with uncertain heterogeneous information,” in *Proc. 2nd Int. Joint Conf. Autonomous agents and multiagent syst.*, pp. 1–8, ACM, 2003.
- [486] D. J. Hooper, G. L. Peterson, and B. J. Borghetti, “Dynamic coalition formation under uncertainty,” in *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, pp. 4799–4804, 2009.
- [487] P. Dasgupta, K. Cheng, and B. Banerjee, “Adaptive multi-robot team re-configuration using a policy-reuse reinforcement learning approach,” in *Advanced Agent Technology*, pp. 330–345, Springer, 2012.

- [488] P. Dasgupta and K. Cheng, “Dynamic multi-robot team reconfiguration using weighted voting games,” *J. Experimental & Theoretical Artificial Intell.*, pp. 1–22, 2015.
- [489] G. de O Ramos, J. C. B. Rial, and A. L. Bazzan, “Self-adapting coalition formation among electric vehicles in smart grids,” in *IEEE Int. Conf. Self-Adaptive and Self-Organizing Syst.*, pp. 11–20, 2013.
- [490] L. E. Parker and F. Tang, “Building multirobot coalitions through automated task solution synthesis,” *Proc. IEEE*, vol. 94, no. 7, pp. 1289–1305, 2006.
- [491] S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. R. Jennings, “Coalition formation with spatial and temporal constraints,” in *Proc. 9th Int. Conf. Autonomous Agents and Multiagent Syst.: volume 3*, pp. 1181–1188, 2010.
- [492] B. Qian and H. H. Cheng, “A bio-inspired mobile agent-based coalition formation system for multiple modular-robot systems,” in *IEEE/ASME 10th Int. Conf. Mechatronic and Embedded Syst. and Applicat.*, pp. 1–6, 2014.
- [493] B. Qian and H. H. Cheng, “A mobile agent-based coalition formation system for multi-robot systems,” in *12th IEEE/ASME Int. Conf. Mechatronic and Embedded Syst. and Applicat.*, pp. 1–6, 2016.
- [494] J. A. Adams *et al.*, “Coalition formation for task allocation: theory and algorithms,” *Autonomous Agents and Multi-Agent Syst.*, vol. 22, no. 2, pp. 225–248, 2011.
- [495] F. Rohrmüller, D. Wollherr, and M. Buss, “Muroco: A framework for capability- and situation-aware coalition formation in cooperative multi-robot systems,” *J. Intell. & Robotic Syst.*, vol. 67, no. 3-4, pp. 339–370, 2012.
- [496] M. Agarwal, N. Agrawal, S. Sharma, L. Vig, and N. Kumar, “Parallel multi-objective multi-robot coalition formation,” *Expert Syst. with Applicat.*, vol. 42, no. 21, pp. 7797–7811, 2015.
- [497] G. A. Korsah, A. Stentz, and M. B. Dias, “A comprehensive taxonomy for multi-robot task allocation,” *Int. J. Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.
- [498] B. Choudhury and B. Biswal, “Task allocation methodologies for multi-robot systems,” 2008.

- [499] X. Jia and M. Q.-H. Meng, “A survey and analysis of task allocation algorithms in multi-robot systems,” in *IEEE Int. Conf. Robotics and Biomimetics*, pp. 2280–2285, 2013.
- [500] B. P. Gerkey and M. J. Matari, “Sold!: Auction methods for multirobot coordination,” *IEEE Trans. Robot. Autom.*, vol. 18, no. 5, pp. 758–768, 2002.
- [501] R. Zlot and A. Stentz, “Complex task allocation for multiple robots,” in *Proc. IEEE Int’l Conf. Robotics and Automation*, pp. 1515–1522, 2005.
- [502] S. Liu, Y. Zhang, H. Wu, and J. Liu, “Multi-robot task allocation based on swarm intelligence,” *Journal of Jilin University*, vol. 40, no. 1, pp. 123–129, 2010.
- [503] G. Oh, Y. Kim, J. Ahn, and H.-L. Choi, “Pso-based optimal task allocation for cooperative timing missions,” *IFAC-PapersOnLine*, vol. 49, no. 17, pp. 314–319, 2016.
- [504] M. Elango, G. Kanagaraj, and S. Ponnambalam, “Sandholm algorithm with k-means clustering approach for multi-robot task allocation,” in *Swarm, Evolutionary, and Memetic Computing*, pp. 14–22, Springer, 2013.
- [505] C. Galindo, J.-A. Fernández-Madrigal, J. González, and A. Saffiotti, “Robot task planning using semantic maps,” *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 955–966, 2008.
- [506] C. Pippin, H. Christensen, and L. Weiss, “Performance based task assignment in multi-robot patrolling,” in *Proc. 28th Annu. ACM Symp. Applied Computing*, pp. 70–76, ACM, 2013.
- [507] S. Ponda, J. Redding, H.-L. Choi, J. P. How, M. Vavrina, and J. Vian, “Decentralized planning for complex missions with dynamic communication constraints,” in *IEEE Amer. Control Conf.*, pp. 3998–4003, 2010.
- [508] T. C. Service, S. D. Sen, and J. A. Adams, “A simultaneous descending auction for task allocation,” in *IEEE Int. Conf. Syst. Man Cybern.*, pp. 379–384, 2014.
- [509] Y. Zhang and L. E. Parker, “Iq-asymtre: Forming executable coalitions for tightly coupled multirobot tasks,” *IEEE Trans. Robot.*, vol. 29, no. 2, pp. 400–416, 2013.
- [510] B. Tang, Z. Zhu, H.-S. Shin, and A. Tsourdos, “Task-priority based task allocation of multiple uavs with resource constraint,” in *23th Mediterranean Conf. Control and Automation*, pp. 8–13, IEEE, 2015.

- [511] P. M. Shiroma and M. F. Campos, “Comutar: A framework for multi-robot coordination and task allocation,” in *IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, pp. 4817–4824, 2009.
- [512] M. Irfan and A. Farooq, “Auction-based task allocation scheme for dynamic coalition formations in limited robotic swarms with heterogeneous capabilities,” in *IEEE International Conference on Intelligent Systems Engineering*, pp. 210–215, 2016.
- [513] G. P. Das, T. M. McGinnity, and S. A. Coleman, “Simultaneous allocations of multiple tightly-coupled multi-robot tasks to coalitions of heterogeneous robots,” in *IEEE International Conference on Robotics and Biomimetics*, pp. 1198–1204, 2014.
- [514] Z. Butler and J. Hays, “Task allocation for reconfigurable teams,” *Robotics & Autonomous Systems*, vol. 68, pp. 59–71, 2015.
- [515] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- [516] Z. Zhang, D. Zhao, J. Gao, D. Wang, and Y. Dai, “Fmrq-a multiagent reinforcement learning algorithm for fully cooperative tasks,” *IEEE Trans. Cybern.*, 2016.
- [517] L. Zhou, P. Yang, C. Chen, and Y. Gao, “Multiagent reinforcement learning with sparse interactions by negotiation and knowledge transfer,” *IEEE Trans. Cybern.*, 2016.
- [518] O. Amir, B. J. Grosz, and R. Stern, “To share or not to share? the single agent in a team decision problem,” 2014.
- [519] X. Sun, T. Mao, and L. E. Ray, “Inference model for heterogeneous robot team configuration based on reinforcement learning,” in *IEEE Int. Conf. Technologies for Practical Robot Applicat.*, pp. 55–60, 2009.
- [520] Z. Song, X. Liao, and L. Carin, “Solving dec-pomdps by expectation maximization of value functions,” 2015.
- [521] M. T. Spaan, F. A. Oliehoek, and C. Amato, “Scaling up optimal heuristic search in dec-pomdps via incremental expansion,” in *22nd Int. Joint Conf. Artificial Intell.*, 2011.
- [522] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping: part i,” *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, 2006.

- [523] T. Bailey and H. Durrant-Whyte, “Simultaneous localization and mapping (slam): Part ii,” *IEEE Robot. Autom. Mag.*, vol. 13, no. 3, pp. 108–117, 2006.
- [524] N. K. Dhiman, D. Deodhare, and D. Khemani, “Where am i? creating spatial awareness in unmanned ground robots using slam: A survey,” *Sadhana*, vol. 40, no. 5, pp. 1385–1433, 2015.
- [525] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, “Visual simultaneous localization and mapping: a survey,” *Artificial Intell. Review*, vol. 43, no. 1, pp. 55–81, 2015.
- [526] S. E. Webster, J. M. Walls, L. L. Whitcomb, and R. M. Eustice, “Decentralized extended information filter for single-beacon cooperative acoustic navigation: Theory and experiments,” *IEEE Trans. Robot.*, vol. 29, no. 4, pp. 957–974, 2013.
- [527] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, “Kinectfusion: Real-time dense surface mapping and tracking,” in *Int. Symp. Mixed and augmented reality*, pp. 127–136, IEEE, 2011.
- [528] P. Mirowski, T. K. Ho, S. Yi, and M. MacDonald, “Signalslam: Simultaneous localization and mapping with mixed wifi, bluetooth, lte and magnetic signals,” in *Int. Conf. Indoor Positioning and Indoor Navigation*, pp. 1–10, IEEE, 2013.
- [529] J. M. Santos, M. S. Couceiro, D. Portugal, and R. P. Rocha, “A sensor fusion layer to cope with reduced visibility in slam,” *J. Intell. & Robotic Syst.*, vol. 80, no. 3-4, pp. 401–422, 2015.
- [530] J. M. Santos, M. S. Couceiro, D. Portugal, and R. P. Rocha, “Fusing sonars and lrf data to perform slam in reduced visibility scenarios,” in *IEEE Int. Conf. Autonomous Robot Syst. & Competitions*, pp. 116–121, 2014.
- [531] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marques, “Computation sharing in distributed robotic systems: a case study on slam,” *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 410–422, 2015.
- [532] L. Paull, G. Huang, M. Seto, and J. J. Leonard, “Communication-constrained multi-auv cooperative slam,” in *IEEE Int. Conf. Robotics & Automation*, pp. 509–516, 2015.
- [533] A. Kleiner and D. Sun, “Decentralized slam for pedestrians without direct communication,” in *IEEE/RSJ Int. Conf. Intell. Robots & Syst.*, pp. 1461–1466, 2007.

- [534] K. Y. Leung, T. D. Barfoot, and H. H. Liu, “Decentralized cooperative simultaneous localization and mapping for dynamic and sparse robot networks,” in *IEEE/RSJ Int. Conf. Intell. Robots & Syst.*, pp. 3554–3561, 2010.
- [535] N. Atanasov, J. Le Ny, K. Daniilidis, and G. J. Pappas, “Decentralized active information acquisition: theory and application to multi-robot slam,” in *IEEE Int. Conf. Robotics & Automation*, pp. 4775–4782, 2015.
- [536] R. Guo, *Scene understanding with complete scenes and structured representations*. PhD thesis, University of Illinois at Urbana-Champaign, 2014.
- [537] S. Vishwakarma and A. Agrawal, “A survey on activity recognition and behavior understanding in video surveillance,” *Visual Comput.*, vol. 29, no. 10, pp. 983–1009, 2013.
- [538] K. Zhou, K. M. Varadarajan, M. Zillich, and M. Vincze, “Web mining driven semantic scene understanding and object localization,” in *IEEE Int. Conf. Robotics and Biomimetics*, pp. 2824–2829, 2011.
- [539] D. Llorca, M. Sotelo, A. Hellín, A. Orellana, M. Gavilán, I. Daza, and A. Lorente, “Stereo regions-of-interest selection for pedestrian protection: A survey,” *Transportation research part C: emerging technologies*, vol. 25, pp. 226–237, 2012.
- [540] L. Song, F. Jiang, Z. Shi, R. Molina, and A. K. Katsaggelos, “Toward dynamic scene understanding by hierarchical motion pattern mining,” *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 3, pp. 1273–1285, 2014.
- [541] A. Geiger, M. Lauer, and R. Urtasun, “A generative model for 3d urban scene understanding from movable platforms,” in *IEEE Conf. Comput. Vision and Pattern Recognition*, pp. 1945–1952, 2011.
- [542] I. S. Kim, H. S. Choi, K. M. Yi, J. Y. Choi, and S. G. Kong, “Intelligent visual surveillance—a survey,” *Int. J. Control, Automation & Syst.*, vol. 8, no. 5, pp. 926–939, 2010.
- [543] D. Moroni, M. A. Pascali, M. Reggiannini, and O. Salvetti, “Underwater scene understanding by optical and acoustic data integration,” in *Proc. of Meetings on Acoust.*, vol. 17, p. 070085, Acoustical Soc. of America, 2013.
- [544] R. J. Radke, “A survey of distributed computer vision algorithms,” in *Handbook of Ambient Intell. & Smart Environments*, pp. 35–55, Springer, 2010.

- [545] T. Nimmagadda and A. Anandkumar, “Multi-object classification and unsupervised scene understanding using deep learning features and latent tree probabilistic models,” *arXiv preprint:1505.00308*, 2015.
- [546] C. Wang, N. Komodakis, and N. Paragios, “Markov random field modeling, inference & learning in computer vision & image understanding: A survey,” *Comput. Vision and Image Understanding*, vol. 117, no. 11, pp. 1610–1627, 2013.
- [547] A. M. Talha and I. N. Junejo, “Dynamic scene understanding using temporal association rules,” *Image and Vision Computing*, vol. 32, no. 12, pp. 1102–1116, 2014.
- [548] J. Wilson and N. Patwari, “See-through walls: Motion tracking using variance-based radio tomography networks,” *IEEE Trans. Mobile Comput.*, vol. 10, no. 5, pp. 612–621, 2011.
- [549] K. Kakinuma, M. Ozaki, M. Hashimoto, T. Yokoyama, and K. Takahashi, “Laser-based pedestrian tracking with multiple mobile robots using outdoor slam,” in *IEEE Int. Conf. Robotics and Biomimetics*, pp. 998–1003, 2011.
- [550] C. T. Chou, J.-Y. Li, M.-F. Chang, and L. C. Fu, “Multi-robot cooperation based human tracking system using laser range finder,” in *Int. Conf. Robotics & Automation*, pp. 532–537, IEEE, 2011.
- [551] N. A. Tsokas and K. J. Kyriakopoulos, “Multi-robot multiple hypothesis tracking for pedestrian tracking,” *Autonomous Robots*, vol. 32, no. 1, pp. 63–79, 2012.
- [552] A. Ahmad and P. Lima, “Multi-robot cooperative spherical-object tracking in 3d space based on particle filters,” *Robotics & Autonomous Syst.*, vol. 61, no. 10, pp. 1084–1093, 2013.
- [553] Z. Liu, M. H. Ang Jr, and W. K. G. Seah, “Reinforcement learning of cooperative behaviors for multi-robot tracking of multiple moving targets,” in *IEEE/RSJ Int. Conf. Intell. Robots & Syst.*, pp. 1289–1294, 2005.
- [554] A. Ahmad, G. D. Tipaldi, P. Lima, and W. Burgard, “Cooperative robot localization and target tracking based on least squares minimization,” in *Int. Conf. Robotics & Automation*, pp. 5696–5701, IEEE, 2013.
- [555] C. Robin and S. Lacroix, “Multi-robot target detection and tracking: taxonomy and survey,” *Autonomous Robots*, vol. 40, no. 4, pp. 729–760, 2016.
- [556] S. Heinrich and S. Wermter, “Towards robust speech recognition for human-robot interaction,” in *Proc. IROS Workshop on Cognitive Neuroscience Robotics*, pp. 29–34, 2011.

- [557] H. Jiang, “Discriminative training of hmms for automatic speech recognition: A survey,” *Comput. Speech & Language*, vol. 24, no. 4, pp. 589–608, 2010.
- [558] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, “Convolutional neural networks for speech recognition,” *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 22, no. 10, pp. 1533–1545, 2014.
- [559] R. Solera-Urena, A. I. Garcia-Moral, C. Peláez-Moreno, M. Martinez-Ramon, and F. Diaz-de Maria, “Real-time robust automatic speech recognition using compact support vector machines,” *IEEE Trans. Audio, Speech, & Language Process.*, vol. 20, no. 4, pp. 1347–1361, 2012.
- [560] Z.-H. Tan and B. Lindberg, “Low-complexity variable frame rate analysis for speech recognition and voice activity detection,” *IEEE J. Sel. Topics Signal Process.*, vol. 4, no. 5, pp. 798–807, 2010.
- [561] J. Li, L. Deng, Y. Gong, and R. Haeb-Umbach, “An overview of noise-robust automatic speech recognition,” *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 22, no. 4, pp. 745–777, 2014.
- [562] R. Gomez, T. Kawahara, K. Nakamura, and K. Nakadai, “Multi-party human-robot interaction with distant-talking speech recognition,” in *Proc. 7th Annu. ACM/IEEE Int. Conf. Human-Robot Interaction*, pp. 439–446, 2012.
- [563] G. Ince, K. Nakadai, T. Rodemann, H. Tsujino, and J.-I. Imura, “Whole body motion noise cancellation of a robot for improved automatic speech recognition,” *Advanced Robotics*, vol. 25, no. 11-12, pp. 1405–1426, 2011.
- [564] A. Roy, “Connectionism, controllers, and a brain theory,” *IEEE Trans. Syst. Man Cybern. A., Syst. Humans*, vol. 38, no. 6, pp. 1434–1441, 2008.
- [565] T. Tabuchi, S. Ozawa, and A. Roy, “An autonomous learning algorithm of resource allocating network,” in *Int. Conf. Intell. Data Eng. & Automated Learning*, pp. 134–141, Springer, 2009.
- [566] A. Roy, “On nsf “open questions,” some external properties of the brain as a learning system and an architecture for autonomous learning,” in *Int. Joint Conf. Neural Networks*, pp. 1–8, IEEE, 2010.
- [567] L. Wang, M. Liu, and M. Meng, “A hierarchical auction-based mechanism for real-time resource allocation in cloud robotic systems,” *IEEE Trans. Cybern.*, 2016.
- [568] R. J. Baron, *The cerebral computer: An introduction to the computational structure of the human brain*. Psychology Press, 2013.

- [569] J. Nolte, “The human brain: An introduction to its functional anatomy,” 2002.
- [570] J. M. DeSesso, “Functional anatomy of the brain,” in *Metabolic Encephalopathy*, pp. 1–14, Springer, 2009.
- [571] N. Geschwind, “Specializations of the human brain,” *Scientific American*, vol. 241, no. 3, pp. 180–201, 1979.
- [572] R. C. O’Reilly and Y. Munakata, *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain*. MIT press, 2000.
- [573] M. Catani, D. K. Jones, R. Donato, *et al.*, “Occipito-temporal connections in the human brain,” *Brain*, vol. 126, no. 9, pp. 2093–2107, 2003.
- [574] J. Szentagothai, “The ferrier lecture, 1977: the neuron network of the cerebral cortex: a functional interpretation,” *Proc. Royal Society of London. Series B. Biological Sciences*, vol. 201, no. 1144, pp. 219–248, 1978.
- [575] V. B. Mountcastle, “The columnar organization of the neocortex,” *Brain*, vol. 120, no. 4, pp. 701–722, 1997.
- [576] A. S. Benjamin, J. S. de Belle, B. Etnyre, and T. A. Polk, “The role of inhibition in learning,” *Human Learning: Biology, Brain, and Neuroscience: Biology, Brain, and Neuroscience*, vol. 139, p. 7, 2008.
- [577] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
- [578] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [579] B. Widrow *et al.*, *Adaptive adaline Neuron Using Chemical memistors*. 1960.
- [580] A. Aizerman, E. M. Braverman, and L. Rozoner, “Theoretical foundations of the potential function method in pattern recognition learning,” *Automation and remote control*, vol. 25, pp. 821–837, 1964.
- [581] J. L. McClelland, D. E. Rumelhart, P. R. Group, *et al.*, “Parallel distributed processing,” *Explorations in the microstructure of cognition*, vol. 2, p. 184, 1986.

- [582] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proc. National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.
- [583] T. Kohonen, “Self-organized formation of topologically correct feature maps,” *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.
- [584] S. Grossberg, “Competitive learning: From interactive activation to adaptive resonance,” *Cognitive science*, vol. 11, no. 1, pp. 23–63, 1987.
- [585] J. Misra and I. Saha, “Artificial neural networks in hardware: A survey of two decades of progress,” *Neurocomputing*, vol. 74, no. 1, pp. 239–255, 2010.
- [586] L. Arnold, S. Rebecchi, S. Chevallier, and H. Paugam-Moisy, “An introduction to deep learning,” in *ESANN*, 2011.
- [587] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2000.
- [588] G. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [589] J. Schmidhuber, “Learning complex, extended sequences using the principle of history compression,” *Neural Computation*, vol. 4, no. 2, pp. 234–242, 1992.
- [590] G. M. Edelman and V. B. Mountcastle, *The mindful brain: Cortical organization and the group-selective theory of higher brain function*. Massachusetts Inst of Technology Pr, 1978.
- [591] A. G. Hashmi and M. H. Lipasti, “Cortical columns: Building blocks for intelligent systems,” in *IEEE Symposium on Computational Intelligence for Multimedia Signal and Vision Processing*, pp. 21–28, IEEE, 2009.
- [592] J. R. Anderson, “Act: A simple theory of complex cognition,” *American Psychologist*, vol. 51, no. 4, p. 355, 1996.
- [593] J. Hawkins and S. Blakeslee, *On intelligence*. Macmillan, 2007.
- [594] S. Franklin and F. Patterson Jr, “The lida architecture: Adding new modes of learning to an intelligent, autonomous, software agent,” *pat*, vol. 703, pp. 764–1004, 2006.