Random Neural Networks and Optimisation

Stelios Timotheou

A thesis submitted for the degree of

Doctor of Philosophy of Imperial College London

Department of Electrical and Electronic Engineering

Imperial College London

United Kingdom

August 2010

I dedicate this thesis to my father, Andreas Timotheou,
who passed away on the 26th October 2007, after a stoical battle with cancer.
May God rest his soul.

Abstract

In this thesis we introduce new models and learning algorithms for the Random Neural Network (RNN), and we develop RNN-based and other approaches for the solution of emergency management optimisation problems.

With respect to RNN developments, two novel supervised learning algorithms are proposed. The first, is a gradient descent algorithm for an RNN extension model that we have introduced, the RNN with synchronised interactions (RNNSI), which was inspired from the synchronised firing activity observed in brain neural circuits. The second algorithm is based on modelling the signal-flow equations in RNN as a nonnegative least squares (NNLS) problem. NNLS is solved using a limited-memory quasi-Newton algorithm specifically designed for the RNN case.

Regarding the investigation of emergency management optimisation problems, we examine combinatorial assignment problems that require fast, distributed and close to optimal solution, under information uncertainty. We consider three different problems with the above characteristics associated with the assignment of emergency units to incidents with injured civilians (AEUI), the assignment of assets to tasks under execution uncertainty (ATAU), and the deployment of a robotic network to establish communication with trapped civilians (DRNCTC).

AEUI is solved by training an RNN tool with instances of the optimisation problem and then using the trained RNN for decision making; training is achieved using the developed learning algorithms. For the solution of ATAU problem, we introduce two different approaches. The first is based on mapping parameters of the optimisation problem to RNN parameters, and the second on solving a sequence of minimum cost flow problems on appropriately constructed networks with estimated arc costs. For the exact solution of DRNCTC problem, we develop a mixed-integer linear programming formulation, which is based on network flows. Finally, we design and implement distributed heuristic algorithms for the deployment of robots when the civilian locations are known or uncertain.

Acknowledgements

First and foremost I would like to thank my supervisor, Professor Erol Gelenbe, for his constant support, not only academically and financially, but also on a personal level. I deeply appreciate his persistent care and efforts to address my needs and problems, and I am grateful for the opportunities he provided me to travel and interact with members of the academia and industry worldwide. His expertise and guidance proved invaluable during my first steps as a junior researcher. He will always be an inspirational figure to me.

I am also thankful to my family for their encouragement throughout this demanding period of my life, and especially my wife Maria. If she had not unconditionally offered her love, patience, and support in every possible way to keep me focussed, often at the expense of her own work, this thesis would have been unattainable.

Moreover I would like to thank my colleagues at the Intelligent Systems & Networks Group, who created a sincerely supportive working environment. Special acknowledgements go to George, Avgoustinos, and Georgia, for eagerly sharing their knowledge and time whenever I asked them. Also, I would like to express my gratitude to all my good friends, and particularly Evelina for her insights into combinatorial optimisation, and Yannis for his heartening support and spot-on advice on any concern I confided to him.

Finally, I would like to acknowledge the Cyprus State Scholarships Foundation and the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Networks) project, for funding my studies.

Contents

List of Figures **10**

List of Tables **13**

Abbreviations and Acronyms **14**

List of main symbols **16**

List of Author Publications **20**

1. Introduction **23**
1.1. Application context: disaster management 24
1.2. Review of examined problems 25
1.3. Summary of contributions 26
1.4. Thesis outline 28

2. The random neural network **31**
2.1. Introduction 31
2.2. The random neural network model 32
2.2.1. Mathematical model 32
2.2.2. Network behaviour in steady-state 33
2.2.3. Network Stability 35
2.2.4. Analogy with the formal neural networks 36
2.2.5. Function approximation 37
2.2.6. Hardware implementations 38
2.3. RNN extension models 38
2.3.1. Bipolar random neural network 39
2.3.2. RNN with state-dependent firing 39
2.3.3. Multiple class random neural network 40

2.4. Learning algorithms . . . 40
2.4.1. Gradient descent supervised learning algorithm for the RNN model . . . 41
2.4.2. Alternative RNN supervised learning algorithms . . . 44
2.4.3. Reinforcement learning in RNN . . . 45
2.5. Applications . . . 48
2.5.1. Solution of optimisation problems . . . 48
2.5.2. Modelling applications . . . 52
2.5.3. Learning applications . . . 53
2.6. Conclusions . . . 54

3. Learning extensions of the random neural network model 56
3.1. Introduction . . . 57
3.2. Gradient descent learning in the RNN with synchronised interactions 60
3.2.1. Synchronised interactions in biological neural networks . . . 60
3.2.2. RNNSI mathematical model . . . 61
3.2.3. Steady-state solution . . . 62
3.2.4. RNNSI gradient descent supervised learning . . . 64
3.2.5. Computational complexity . . . 67
3.3. RNN supervised learning using nonnegative least squares . . . 71
3.3.1. Problem formulation . . . 72
3.3.2. Solution approach . . . 75
3.3.3. Efficient computation of NNLS costly functions . . . 88
3.3.4. Computational complexity . . . 96
3.4. Conclusions . . . 98

4. Assignment of emergency units to incidents 100
4.1. Problem description . . . 101
4.2. Supervised learning solution approach . . . 103
4.3. Performance evaluation of the RNNSI learning algorithm . . . 105
4.4. Performance evaluation of the RNN-NNLS algorithm . . . 112
4.4.1. Preliminary results . . . 113
4.4.2. Solving the AEUI problem . . . 116
4.5. Using RNN-NNLS algorithm for weight initialisation . . . 121
4.6. Conclusions . . . 123

5. Asset-task assignment under execution uncertainty 127
5.1. Introduction . . . 128
5.2. Problem description and mathematical formulation . . . 130
5.3. Related problems . . . 131
5.4. The RNN parameter association approach . . . 134
5.5. Network flow algorithms . . . 137
5.6. Obtaining tight lower bounds . . . 142
5.7. Evaluation . . . 147
5.8. Conclusions . . . 153

6. Connecting trapped civilians to a wireless ad-hoc robotic network 154
6.1. Introduction . . . 155
6.2. Related work . . . 157
6.3. Assumptions . . . 157
6.4. A centralised approach . . . 158
6.4.1. Formulation . . . 158
6.4.2. Numerical results using the general centralised approach . . 161
6.4.3. Numerical results for uncertain civilian locations . . . 162
6.4.4. Minimising the number of robots and the energy consumption 164
6.5. A distributed heuristic . . . 167
6.5.1. Simulation results for the distributed algorithm . . . 170
6.5.2. Introducing uncertainty . . . 171
6.5.3. MST-based modification . . . 174
6.6. Conclusions . . . 177

7. Conclusions and future work 178
7.1. Thesis contributions . . . 178
7.2. Future work . . . 182

Bibliography 185

Appendices 204

A. Derivative calculations of RNNSI gradient descent algorithm 205

B. Derivation of expressions associated with the efficient computation of NNLS costly functions **208**
B.1. The first approach 208
B.1.1. Derivation of Eq. (3.60) 208
B.1.2. Derivation of Eq. (3.64) 209
B.2. The second approach 210
B.2.1. Derivation of $\mathbf{\Gamma}_{lm}$ 211
B.2.2. Derivation of $\mathbf{\Gamma w}$ 214

List of Figures

4.1. Dispatching of emergency units to locations of injured civilians . . . 101
4.2. Performance of the RNNSI gradient descent algorithm for the "collective" neural network architecture . . . 107
4.3. Performance of the RNN gradient descent algorithm for the "collective" neural network architecture . . . 108
4.4. Performance of the RNNSI gradient descent algorithm for the "individual" neural network architecture . . . 109
4.5. Performance of the RNN gradient descent algorithm for the "individual" neural network architecture . . . 110
4.6. Performance evaluation of the four architectures considered: (a) "Collective" RNNSI , (b) "Collective" RNN, (c) "Individual" RNNSI, and (d) "Individual" RNN . . . 111
4.7. RNNSI to RNN execution time ratio . . . 113
4.8. Performance of approaches for computing the objective and gradient NNLS function compared to a "naive" one; the metric used is the ratio of execution times between the naive and another approach . . 114
4.9. Comparison of line search procedures lineSearchLin and lineSearchHE in terms of matrix vector products (the y-axis is their ratio) for $N_L = 3$ and $N_L = 5$ when $N_H = N_O$ . . . 115
4.10. Comparison of convergence between algorithms gradNNLS, PQN-SPG and PGNNLS with respect to iterations and execution time . . 117
4.11. Convergence of RNN-NNLS algorithm for $N_L = 3$ and $N_L = 5$ when $N_H = 2N_O$ . . . 118
4.12. Performance of the NNLS-RNN algorithm for the "collective" neural network architecture . . . 119
4.13. Performance of the NNLS-RNN algorithm for the "individual" neural network architecture . . . 120

4.14. Comparison between the RNN-NNLS and RNN learning algorithms. The four architectures considered are: (a) "Collective" RNN-NNLS, (b) "Collective" RNN, (c) "Individual" RNN-NNLS, and (d) "Individual" RNN 122
4.15. Performance of the RNN learning algorithm with random or NNLS initialisation for the "collective" NN architecture, with no hidden neurons. 124
4.16. Execution times of the RNN learning algorithm with random or NNLS initialisation 125

5.1. Flow network for the solution of problem (5.2) 137
5.2. Piecewise linear approximation 145

6.1. Example scenario: A group of robots establish communication with trapped civilians 156
6.2. Robot allocations according to the centralised solutions for (i)$R_{rob} = 8m$ and $R_{civ} = 4m$, (ii)$R_{rob} = 14m$ and $R_{civ} = 4m$, (iii)$R_{rob} = 8m$ and $R_{civ} = 14m$: When the civilian range is small one robot must be dedicated to each civilian, whereas when it is large one robot may suffice to connect multiple civilians. In addition, when the robot range is not large enough, a significant number of the robots must be used for maintaining connectivity; hence, they move to locations of high density of civilians for optimal performance. 163
6.3. Maximum number of connected civilians for varying R_{civ} and $R_{rob} = 8m, 10m, 12m$ 164
6.4. Maximum number of connected civilians for varying R_{rob} and $R_{civ} = 2m, 6m, 10m$ 165
6.5. Maximum number of connected civilians for varying number of robots and different combinations of ranges 166
6.6. Average percentage of connected civilians for varying R_{rob} and $R_{civ} = 2m, 6m, 10m$, for uncertain civilian locations 167
6.7. Average percentage of connected civilians for varying number of robots, for uncertain civilian locations 168
6.8. Minimum number of robots required to connect all civilians when objectives (6.2a), (6.5) and (6.6) are utilised. 168

6.9. Average locomotion energy consumed per robot when objectives (6.2a), (6.5) and (6.6) are utilised. 169
6.10. General flow diagram of the distributed heuristic algorithm 169
6.11. Solution of the distributed algorithm for the given clustering of civilians 171
6.12. Comparison between the distributed and centralised approach in terms of number of connected civilians against the number of robots 172
6.13. Comparison between the distributed and centralised approach in terms of number of connected civilians against the wireless range of the robots 172
6.14. Connectivity is guaranteed within a cluster if its radius is smaller than $R_{rob} + R_{civ}$ 173
6.15. Evaluation of the distributed heuristic for different numbers of robots N_{rob} and varying risk parameter q 174
6.16. Illustration of the Minimum Spanning Tree formed by the clusters and the movement options of the robots 175
6.17. Percentage of trapped civilians connected against time for 10, 15 and 20 robots. 176

List of Tables

2.1. Summary of RNN applications 55

4.1. Optimal regularisation weights for the "collective" RNN architecture 121

5.1. Average relative percentage deviation from the optimal solutions of data family 1 147
5.2. Average relative percentage deviation from the optimal solutions of data family 2 148
5.3. Average relative percentage deviation from the lower bound of data family 1 150
5.4. Average relative percentage deviation from the lower bound of data family 2 151

Abbreviations and Acronyms

AEUI	Assignment of Emergency Units to Incidents
ANN	Artificial Neural Networks
APA	Armijo along the Projection Arc
ATAU	Asset-Task Assignment under execution Uncertainty
BFGS	Broyden - Fletcher - Goldfarb - Shanno
BRNN	Bipolar Random Neural Network
CPN	Cognitive Packet Network
CRNN	Clamped Random Neural Network
DFP	Davidon - Fletcher - Powell
DRNCTC	Deployment of a Robotic Network for Communication with Trapped Civilians
DRNN	Dynamic Random Neural Network
ES	Expected Shortfall
FNNLS	Fast NonNegative Least Squares
KKT	Karush-Kuhn-Tucker
LBA	Lower Bounding Algorithm
MCF	Minimum Cost Flow
MCRNN	Multiple Class Random Neural Network
MMR	Maximum Marginal Return
MSE	Mean Square Error
MST	Minimum Spanning Tree
MSTP	Minimum Steiner Tree Problem
MVCP	Minimum Vertex Covering Problem
NN	Neural Network
NNLS	NonNegative Least Squares
NP-hard	Non-determenistic Polynomial-time hard
PCMOS	Probabilistic Complementary Metal-Oxide-Semiconductor

PGNNLS	Projected Gradient NonNegative Least Squares
PQN	Projected Quasi Newton
QoS	Quality of Service
QSAP	Quadratic Semi-Assignment Problem
RL	Reinforcement Learning
RNN	Random Neural Network
RNNSDF	Random Neural Network with State-Dependent Firing
RNNSI	Random Neural Network with Synchronised Interactions
RPROP	Resilient Propagation
SATP	Satisfiability Problem
SF	Synchronised Firing
SPG	Spectral Projected Gradient
WTA	Weapon Target Assignment

List of main symbols

RNN and RNNSI symbols

$k_i(t)$	Potential of neuron i at time t
$q_i(t)$	Probability neuron i is excited at time t
Λ_i	External arrival rate of positive signals to neuron i
λ_i	External arrival rate of negative signals to neuron i
$\lambda^+(i)$	Total arrival rate of positive signals to neuron i
$\lambda^-(i)$	Total arrival rate of negative signals to neuron i
$p^+(i,j)$	Probability neuron j receives a positive signal from firing neuron i
$p^-(i,j)$	Probability neuron j receives a negative signal from firing neuron i
$Q(i,j,l)$	Probability of firing neuron i to create a synchronous interaction together with neuron j to affect neuron l
$w^+(i,j)$	Rate of positive signals or positive weight to neuron j from firing neuron i
$w^-(i,j)$	Rate of negative signals or negative weight to neuron j from firing neuron i
$w(i,j,l)$	Rate or weight of synchronous interactions to neuron l generated from firing neuron i together with neuron j
$a(j,l)$	Ratio of $w(i,j,l)$ over $w^-(i,j)$
$\mathbf{W}^+$	Matrix of positive weights
$\mathbf{W}^-$	Matrix of negative weights
$\mathbf{A}$	Matrix of parameters $a(j,l)$
$d(i)$	Probability a signal from firing neuron i departs from the network
r_i	Firing rate of neuron i
N	Number of neurons in the network
K	Number of input-output training pairs

$N(i)$	Nominator of the equation describing the excitation probability of neuron i
$D(i)$	Denominator of the equation describing the excitation probability of neuron i
x_{ik}	The ith input value of the kth training pair
y_{ik}	The ith output value of the kth training pair
I_{out}	Set of output neurons
$I_{\overline{out}}$	Set of non-output neurons

NNLS-related symbols

$\mathbf{B}$	Matrix of the NNLS formulation
$\mathbf{b}$	Vector of the NNLS formulation
$\mathbf{w}$	Decision variables of the NNLS formulation
τ	Represents the τth iteration of the algorithm
$\mathcal{F}$	Free-set of variables
$\mathcal{B}$	Binding-set of variables
$\mathbf{S}$	Gradient scaling matrix
$\mathbf{d}$	Search-direction
θ_1	Weight for $l1$-norm regularisation
θ_2	Weight for $l2$-norm regularisation
$\Delta\mathbf{g}$	Change of gradient vector between successive iterations
$\Delta\mathbf{w}$	Change of decision vector between successive iterations
M	Number of correction vectors for limited-memory BFGS update

Symbols of AEUI problem

N_U	Number of emergency units
N_L	Number of incident locations
T_{ij}	Response time of emergency unit i to incident j
I_j	Number of people injured at incident j
c_i	Capacity of emergency unit i

Symbols of ATAU problem

$\mathcal{A}$	Set of assets
$\mathcal{T}$	Set of tasks
$U(t)$	Cost of task t
$C_a(a,t)$	Cost of assigning asset a to task t
$p_s(a,t)$	Probability that asset a will succeed in executing task t
$p_f(a,t)$	Probability that asset a will fail in executing task t
$\max\{0, b(a,t)\}$	Net expected reduction in the objective function for (a,t) assignment pair
$a_{t(m)}$	Asset a is the mth assignment to task t

Symbols of DRNCTC problem

$\mathcal{R}$	Set of robots
$\mathcal{C}$	Set of civilians
R_{rob}	Connectivity range of robots
R_{civ}	Connectivity range of civilians
y_c	Decision variable indicating whether civilian c is connected
x_i	Decision variable showing whether there is a robot at vertex i
$C^{RR}_{i,j}$	Element of matrix $\mathbf{C}^{RR}$ representing whether a robot on vertex j would be connected with a robot on vertex i
$C^{RC}_{c,j}$	Element of matrix $\mathbf{C}^{RC}$ representing whether a robot on vertex j would be connected to civilian c
$C^{RL}_{i,j}$	Element of matrix $\mathbf{C}^{RL}$ representing whether a robot at location i can connect civilians at location j
$E_{i,j}$	Energy consumed by a robot to move from node i to j
$E[Z_i]$	Expected number of civilians on vertex i
$ES_q(i)$	Expected number of civilians on location i in the worst q% cases

Symbols of MCF problem

$\mathcal{N}$	Set of nodes or vertices
$\mathcal{E}$	Set of arcs or edges
$X_f(i,j)$	Decision variable representing the flow on arc (i,j)
$C_f(i,j)$	Cost per unit of flow on arc (i,j)
$u(i,j)$	Flow capacity of arc (i,j)
d	Demand or sink node
$s(i)$	Supply of node i

List of Author Publications

Journal Papers

J1. Timotheou, S. (2011) Asset-task assignment algorithms in the presence of execution uncertainty, *The Computer Journal*, accepted for publication: Oxford University Press. [Special issue of ISCIS'10]

J2. Gelenbe, E., Timotheou, S. and Nicholson, D. (2010) Fast Distributed Near-optimum Assignment of Assets to Tasks, *The Computer Journal*, Advance Access published on February 19, 2010, 10.1093/comjnl/bxq010.

J3. Timotheou, S. (2010) The Random Neural Network: A Survey, *The Computer Journal*, 53(3), 251-267: Oxford University Press.

J4. Timotheou, S. (2009) A Novel Weight Initialization Method for the Random Neural Network, *Neurocomputing*, 73(2), 160-168: Elsevier. [Special issue of ISNN'08]

J5. Gelenbe, E. and Timotheou, S. (2008) Random Neural Networks with Synchronized Interactions, *Neural Computation*, 20(9), 2308-2324: MIT Press.

J6. Gelenbe, E. and Timotheou, S. (2008) Synchronized Interactions in Spiked Neuronal Networks, *The Computer Journal*, 51(6), 723-730: Oxford University Press.

Conference Papers

C1. Timotheou, S. (2010) Network flow approaches for an asset-task assignment problem with execution uncertainty, To appear in *Proceedings of the 25th International Symposium on Computer and Information Sciences (ISCIS'10)*,

London, United Kingdom, 22-24 September, Lecture Notes in Electrical Engineering: Springer, Berlin / Heidelberg. [Extended version to appear in special issue of The Computer Journal]

C2. Gelenbe, E., Timotheou, S. and Nicholson, D. (2010) A Random Neural Network Approach to an Assets to Tasks Assignment Problem, In *Proceedings of SPIE Symposium on Defense, Security and Sensing*, 7697(76970Q), Orlando, Florida, 5-9 April. SPIE, Bellingham, WA.

C3. Gelenbe, E., Timotheou, S. and Nicholson, D. (2010) Random Neural Network for Emergency Management, n *Proceedings of the Workshop on Grand Challenges in Modeling, Simulation, and Analysis for Homeland Security (MSAHS'10)*, Washington, D.C., 17-18 March. Directorate of Science & Technology, US Department of Homeland Security, Washington, D.C.

C4. Filippoupolitis, A., Loukas, G., Timotheou, S., Dimakis, N. and Gelenbe, E. (2009) Emergency Response Systems for Disaster Management in Buildings, In *Proceedings of NATO IST-086 Symposium on C3I for Crisis, Emergency and Consequence Management*, Bucharest, Romania, 11-12 May, pp. 15-1 - 15-14. NATO-RTO [Best Paper Award].

C5. Timotheou, S. and Loukas, G. (2009), Autonomous Networked Robots for the Establishment of Wireless Communication in Uncertain Emergency Response Scenarios, 24th ACM Symposium on Applied Computing (SAC'09), Honolulu, Hawaii, 8-12 March, pp. 1171-1175. ACM, NY.

C6. Loukas, G. and Timotheou, S. (2008) Connecting Trapped Civilians to a Wireless Ad Hoc Network of Emergency Response Robots, In *Proceedings of the 11th IEEE International Conference on Communication Systems (ICCS'08)*, Guangzhou, China, 19-21 November, pp. 599-603. IEEE, Piscataway, NJ.

C7. Loukas, G., Timotheou, S. and Gelenbe, E. (2008) Robotic Wireless Network Connection of Civilians for Emergency Response Operations, In *Proceedings of the 23rd International Symposium on Computer and Information Sciences (ISCIS'08)*, Istanbul, Turkey, 27-29 October, pp. 1-6. IEEE, Piscataway, NJ.

C8. Timotheou, S. (2008) A Novel Weight Initialization Method for the Random Neural Network, *5th International Symposium on Neural Networks (ISNN'08)*,

Beijing, China, 24-28 September. [Appears in special issue of Neurocomputing, 73(2), 160-168: Elsevier].

C9. Timotheou, S. (2008) Nonnegative Least Squares Learning for the Random Neural Network, In *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN'08)*, Prague, Czech Republic, 3-6 September 2008, Lecture Notes Computer Science, Part I, 5163, pp. 195-204: Springer, Berlin / Heidelberg.

C10. Filippoupolitis, A., Gelenbe, E., Gianni, D., Hey, L., Loukas, G. and Timotheou, S. (2008) Distributed Agent-oriented Building Evacuation Simulator, In *Proceedings of the 2008 Summer Computer Simulation Conference (SCSC'08)*, Edinburgh, Scotland, 16-19 June. SCS, Vista, CA.

C11. Filippoupolitis, A. Hey, L., Loukas, G., Gelenbe, E. and Timotheou, S. (2008) Wireless Sensor Networks in Augmented Reality: Emergency Response Simulation, In *Proceedings of the 1st International Conference on Ambient Media and Systems (Ambi-Sys'08)*, Quebec City, Canada, 11-14 February, pp. 1-7. ICST, Brussels, Belgium.

Poster Presentations

P1. Timotheou, S. and Gelenbe, E. (2009) Random Neural Networks with Synchronized Interactions, *2nd Symposium of the Neuroscience Technology Network*, London, United Kingdom, 7 October 2009.

P2. Gelenbe, E. and Timotheou, S. (2008) Spiked Random Neural Networks with Synchronized Firing, *Symposium on Perspectives in Modeling and Performance Analysis of Computer Systems & Networks (Model35)*, Paris-Rocquencourt, France, 2-3 April 2008.

P3. Filippoupolitis, A., Gelenbe, E., Gianni, D., Hey, L., Loukas, G. and Timotheou, S. (2008) A Distributed Multi-Agent Simulator for Building Evacuation, *Symposium on Perspectives in Modeling and Performance Analysis of Computer Systems & Networks (Model35)*, Paris-Rocquencourt, France, 2-3 April 2008.

1. Introduction

The main aim of this thesis is the investigation of different aspects of the Random Neural Network (RNN) and the development of corresponding algorithms for the solution of combinatorial optimisation problems. Specifically, we consider assignment problems which involve a number of agents, the decision makers, that need to act in order to optimise a common objective function subject to a number of global constraints. The examined problems have a number of challenging characteristics that should be addressed by any developed algorithms:

- **Real-time solution**: By "real-time" we mean that the time required by the optimisation algorithm to solve the problem is negligible compared to the time needed to execute any action imposed by the solution. For example, if the problem considered is the dispatching of ambulances to locations of accidents, then it is sufficient for the algorithm to provide a solution in milliseconds or a few seconds, since this time is negligible compared to the time that will be consumed by any of the ambulances to reach an accident. Hence, any developed algorithm should be fast and desirably of polynomial computational complexity to ensure that it will be executed in real-time for a given problem.

- **Hard Problems**: The considered problems are complex and of combinatorial nature, resulting in NP-hard optimisation problems with exponentially increasing search spaces, which almost surely cannot be optimally solved with polynomial algorithms. As a result, the developed algorithms should provide close to optimal solutions despite being polynomial.

- **Imperfect Information**: In many cases, complete information about the problem dealt with cannot be collected, so that decision making has to rely on limited information. Limitations can occur in various forms, such as missing or imprecise data, ambiguity, or even uncertainty in the sense that we may only know the probability distribution rather than the actual value of particular

data. Hence, any developed algorithm should be able to appropriately handle these limitations by incorporating uncertainty into the model and utilising the available information in the best possible way.

- **No central control**: Having no central control is highly desirable for several reasons: (a) there is no central point of failure, (b) there is no communication bottleneck as there is no need to send all information to a central control unit, (c) local information can be incorporated into the decisions of individual agents.

1.1. Application context: disaster management

Problems with the aforementioned characteristics naturally arise in disaster or emergency management, which deals with physical and man-made incidents that threaten life, property, operations, or the environment. The process of disaster management involves four phases: (1) planning to reduce the effect or the risk of disasters (*mitigation*), (2) developing plans of actions to be used once a disaster occurs (*preparedness*), (3) responding to such situations (*response*) and (4) restoring the affected environments to their original state (*recovery*). The main goal of disaster management is to minimise the human casualties as well as the property and environmental damages in an emergency event [175, 142].

Perhaps the most challenging of the four phases is the response phase, when the emergency services have to deal with the effects of the disaster in real-time, under extremely difficult conditions with imperfect information and usually disrupted communications. The following large-scale disaster scenario demonstrates a situation where optimisation problems with such characteristics arise:

A major earthquake has struck a large city during the morning hours of a weekday. As a result, several buildings have partially or fully collapsed and there are many injured civilians spatially distributed around the city. These civilians have to be found and collected by emergency units in the least possible time, taking into account the severity of their injuries, the limited number and capacity of the emergency units, as well as the fact that a number of roads have been blocked. Also a number of civilians have been trapped inside the ruins of the buildings so that search and rescue personnel need to identify their locations, assess their condition, and launch a rescue operation trying to maximise the number of collected civilians,

given that each one of them can only survive for a limited amount of time. In addition, multiple fires have broken out around the city and need to be quickly dealt with by the fire-brigade taking into consideration the potential effect of each one of the fires, the weather conditions, and the scarcity of resources. To facilitate the rescue operations, the roads need to be unblocked starting from roads that accommodate more traffic or that significantly increase the connectivity of the city. The work of the emergency services is further impaired by the fact that the communication network has been disrupted, so that affected people cannot easily report to them incidents that require their attention. As the amount of information collected by the emergency services is limited, their actions need to also rely on *a priori* known information. For example, if a number of buildings have collapsed, then the operations centre can take into consideration probability distributions associated with the number of people expected to be in each building, so as to prioritise their search and rescue operations.

The above scenario illustrates that in the event of a disaster, several complex optimisation problems with imperfect information may arise, requiring real-time and distributed decision making. In this thesis we will particularly look into three specific combinatorial optimisation problems of this nature, and develop algorithms for their solution mainly associated with the RNN. These problems are discussed in more detail in the next section.

1.2. Review of examined problems

In this thesis we will examine three different combinatorial problems that arise in emergency management:

1. *Assignment of emergency units to incidents (AEUI)*: In this problem, a number of incidents have taken place simultaneously and there are a number of injured civilians at each location. At the time of the incidents, a number of emergency units are spatially distributed around the area, each having a different capacity to collect a number of those civilians, as well as a different response time to each of the incidents. The objective is to collect as many of the injured as possible and also minimise their total response time.

2. *Asset-task assignment under execution uncertainty (ATAU)*: We investigate a

general problem associated with the assignment of assets to tasks when each asset can potentially execute any of the tasks, but assets execute tasks with a probabilistic outcome of success. There is a cost associated with each possible assignment of an asset to a task, and if a task is not executed there is also a cost associated with the non-execution of the task. Thus any assignment of assets to tasks will result in an expected overall cost which we wish to minimise. Assets can represent rescuers whose task is to collect a number of spatially distributed injured civilians. Each rescuer can collect at most one injured but it is uncertain whether s/he will be able to accomplish his/her task either because of difficulty in accessing the location of the injured or because s/he cannot handle the injured alone.

3. *Deployment of a wireless ad hoc robotic network for the connection of trapped civilians (DRNCTC)*: During a disaster, emergency response operations can benefit from the establishment of a wireless ad hoc network. We investigate the use of autonomous robots that move inside a disaster area and establish a network for two-way communication between trapped civilians with *a priori* known or uncertain locations and an operations centre. The civilians may have uncertain locations, in the sense that we only known a probability distribution describing the number of civilians at any possible position. The specific problem considered is to find optimal locations for the robots so that we maximise the number of civilians connected to the network, assuming that each civilian carries a short-range communication device. This problem is in close connection to the other two, as its solution can provide the means for locating and assessing the health condition of the injured civilians.

1.3. Summary of contributions

The contributions of this thesis can be divided into two main categories: (a) theoretical developments for the RNN and (b) mathematical formulation of the emergency management problems posed above and development of algorithms for their solution. The proposed algorithms are primarily based on RNN, but we have also developed network flow and greedy heuristic approaches. Specifically, the contributions of this thesis are:

a. Theoretical developments for RNN

i. We have introduced RNNSI model, an extension of the RNN that incorporates synchronous interactions and developed a gradient descent learning algorithm of the same computational complexity as the corresponding RNN algorithm.

ii. We have developed a new supervised learning algorithm for the RNN, called RNN-NNLS, that can be used both for learning and weight initialisation. The core of the algorithm is the solution of a nonnegative least squares (NNLS) problem formulated by approximating the RNN equations. Solution to the NNLS problem is accomplished by a limited-memory quasi-Newton algorithm. We have also derived efficient analytical expressions for the computation of the objective and gradient NNLS functions, which speed up the procedure by up to fifty times.

iii. We have conducted the first extended survey on RNN, since its discovery two decades ago.

b. Investigation of emergency management optimisation problems

i. We have proposed a supervised learning methodology for the real-time solution of hard combinatorial optimisation problems when distributed and consistent decision making is necessary. In relation to that we have examined the AEUI problem using the developed RNNSI and RNN-NNLS learning algorithms.

ii. For the solution of ATAU problem, we have developed an RNN parameter association approach, in which the parameters of the optimisation problem are associated with parameters of the RNN model. In addition, we have proposed the use of network flow algorithms that are based on solving a sequence of minimum cost flow problems on appropriately constructed networks with estimated arc costs and introduced three different estimation schemes. We have also designed an approach for obtaining tight lower bounds to the optimal solution based on a piecewise linear approximation of the considered problem.

iii. We have introduced the problem of maximising the number of connected trapped civilians to a wireless ad-hoc robotic network when the locations

of the civilians are either *a priori* known or uncertain. For its optimal solution, we have derived a mixed-integer linear programming formulation based on network flows. We have also designed and implemented distributed heuristic algorithms based on clustering possible locations of civilians both for certain and uncertain civilian locations.

1.4. Thesis outline

The remainder of this thesis is organised as follows. In Chapter 2 we survey the research work on the RNN, including the description and mathematical properties of the original model, other extensions that incorporate additional signal capabilities, RNN-related learning algorithms, as well as applications of the model with emphasis on those related to the solutions of combinatorial optimisation problems.

In Chapter 3, we present the developed RNNSI and RNN-NNLS learning algorithms. First, we describe the motivation for this work and discuss associated research. In the next section, we present the RNNSI learning algorithm. We start with a discussion of the model's biological relevance and a description of its mathematical properties. Then, we derive the main steps of the algorithm; the details of the derivation are included in Appendix A. The section finishes with an extensive analysis of its computational complexity that results in efficient modifications of the algorithm. In the subsequent section, we discuss the details of the RNN-NNLS algorithm. Firstly, we illustrate how to obtain the NNLS formulation from the RNN supervised learning problem when all neurons have desired output values. Next, we develop a limited-memory quasi-Newton algorithm for the solution of the NNLS problem, and present the RNN-NNLS algorithm that can be employed for the solution of problems involving both output and non-output neurons. Before discussing the computational complexity of RNN-NNLS, we outline two approaches for the efficient evaluation of the objective and gradient NNLS functions, by manipulating the special structure of the examined problem; the analysis of the corresponding expressions is given in Appendix B. The final section is a summary of the chapter's outcomes.

The evaluation of the developed supervised learning algorithms is undertaken in Chapter 4, for the solution of the optimisation problem associated with the assignment of emergency units to incidents. We start with the description and mathemat-

ical formulation of the problem followed by the proposed solution approach, which is based on training a neural network tool to act as an "oracle" for decision making. For this purpose, the RNN and RNNSI models are employed and trained using the learning algorithms developed in Chapter 3. In the remainder of the chapter an extensive evaluation is carried out to show the learning ability of the proposed algorithms as well as their efficiency in solving the investigated problem. In addition to the performance evaluation of RNNSI and RNN-NNLS learning algorithms, we examine the efficiency of RNN-NNLS as a weight initialisation method and finish with the main conclusions of the chapter.

In Chapter 5, we examine the asset-task assignment problem under execution uncertainty. We start with the description and mathematical formulation of the problem, followed by a discussion of other related problems. Then, we describe two polynomial deterministic approaches for its solution: (a) an RNN algorithm based on associating parameters of the optimisation problem with parameters of RNN, and (b) a minimum cost flow algorithm that is based on estimating the cost values of specific arcs in the flow network. We also develop a piecewise linear approach for obtaining tight lower bounds to the studied problem, before examining the performance of the proposed approaches and concluding.

Chapter 6 studies the problem of deploying a wireless ad-hoc robotic network for the connection of trapped civilians. First, we discuss the motivation for the solution of this problem followed by a description of related research topics. Then, we give a formal description of the problem with the assumptions made and formulate it as a mixed-integer mathematical program that can be solved by a central processing unit. Apart from the centralised approach we also describe three versions of a distributed heuristic algorithm for its solution. The first deals with the problem when the locations of the civilians are *a priori* known. The second is a modified version of the first one, which tackles the problem with uncertain civilian locations using a risk measure for economic theory, called *expected shortfall*. The third version is a modification of the second one, with which the deployment of robots relies on an appropriately constructed *minimum spanning tree*, aiming to reduce the connection time of the civilians and the total energy spent by the robots. Performance evaluation of the developed algorithms in this section is undertaken throughout the chapter with respect to the centralised algorithm and between the different versions of the distributed heuristic.

Finally, in Chapter 7 we summarise the main contributions of this thesis and discuss possibilities for exploitation. The thesis finishes by providing directions for future work for the core research chapters.

2. The random neural network

This chapter attempts to briefly and comprehensively present the large amount of research published on the RNN since its introduction two decades ago. Our intention is to review the theory and present different RNN tools that can be utilised for the solution of practical problems.

The chapter is organised as follows: In section 2.1, an introduction on RNN is provided along with its main attractive features. The mathematical model and its steady-state properties are described in Section 2.2, while extension models are discussed in Section 2.3. Following is a presentation of the RNN learning algorithms, as well as algorithms proposed for RNN extension models. The RNN applications are summarised in Section 2.5, with particular emphasis on the approaches used for the solution of optimisation problems. The chapter concludes in section 2.6.

2.1. Introduction

The Random Neural Network (RNN) is a neural network model inspired by the spiking behaviour of biophysical neurons [56]. When a biophysical neuron is excited, it transmits a train of signals, called *action potentials* or *spikes*, along its axon to either excite or inhibit the receiving neurons. The combined effect of excitatory and inhibitory inputs changes the potential level of the receiving neuron and determines whether it will become excited. In RNN these signals are represented as excitatory and inhibitory spikes of amplitude +1 and -1 respectively, that are transmitted either from other neurons or from the outside world. Each neuron can fire only when its potential is strictly positive. The potential is equal to the number of positive spikes received that have not yet been fired or cancelled by inhibitory spikes.

RNN has attracted a lot of attention in the scientific community. Various aspects of it have been explored, while several extension models and learning algorithms

have been developed. In addition, RNN has found widespread application in diverse areas of engineering and physical sciences. The success of the model can be attributed to its unique features which include the following [82]:

- Although it is a recurrent neural network, its steady-state probability distribution is described by an analytical equation that can be easily and efficiently computed without the use of Monte Carlo methods
- Its standard learning algorithm has low complexity and strong generalisation capacity even for a relatively small training data set
- It represents in a closer manner the signals transmitted in a biological neuronal network than other Artificial Neural Networks (ANN)
- It can be easily implemented in both software and hardware since its neurons can be represented by simple counters
- There is a direct analogy between the RNN and the connectionist ANN
- The neuron potential is represented as an integer rather than a binary variable resulting in a more detailed system-state description
- It is a universal approximator for bounded continuous functions
- The stochastic excitatory and inhibitory interactions in the network make it an excellent modelling tool for various interacting entities

2.2. The random neural network model

In this section, a mathematical description and the main results of the standard random neural network model are given. We also discuss the stability of the network as well as its analogy to connectionist ANN.

2.2.1. Mathematical model

RNN is a recurrent network of N fully connected neurons which exchange positive and negative signals in the form of unit amplitude spikes. At any time t, the state of neuron i is described by its *signal potential* $k_i(t)$ which is a nonnegative integer

associated with the accumulation of positive signals at the neuron. We say that neuron i is *excited* when $k_i(t) > 0$, else if $k_i(t) = 0$ then it is *idle* or *quiescent*. A closely related parameter is $q_i(t) = Pr[k_i(t) > 0] \leq 1$, which is the *neuron excitation probability*.

When neuron i is excited, it can randomly fire according to the exponential distribution with rate r_i resulting in the reduction of its potential by 1. The fired spike either reaches neuron j as a positive signal with probability $p^+(i,j)$ or as a negative signal with probability $p^-(i,j)$, or it departs from the network with probability $d(i)$. These probabilities must sum up to one yielding

$$\sum_{j=1}^{N} \left[p^+(i,j) + p^-(i,j)\right] + d(i) = 1, \quad \forall i \tag{2.1}$$

Hence, when neuron i is excited, it fires positive and negative signals to neuron j with rates:

$$w^+(i,j) = r_i p^+(i,j) \ \geq \ 0 \tag{2.2}$$

$$w^-(i,j) = r_i p^-(i,j) \ \geq \ 0 \tag{2.3}$$

Combining Eqs. (2.1), (2.2) and (2.3) an expression which associates r_i with $w^+(i,j)$ and $w^-(i,j)$ is derived:

$$r_i = (1 - d(i))^{-1} \sum_{j=1}^{N} [w^+(i,j) + w^-(i,j)] \tag{2.4}$$

Positive and negative signals can also arrive from the outside world according to Poisson processes of rates Λ_i and λ_i respectively. Positive signals have an excitatory effect in the sense that they increase the signal potential of neuron j by 1. Contrary, negative signals have an inhibitory effect and cancel a positive spike if $k_j(t) > 0$, while if $k_j(t) = 0$ the negative signal has no effect.

2.2.2. Network behaviour in steady-state

The state of the network is described by the vector of signal potentials at time t, $\mathbf{k}(t) = [k_1(t), ..., k_N(t)]$. Due to the stochastic nature of the network we are interested in determining the stationary probability distribution $\pi(\mathbf{k}) = \lim_{t\to\infty} \pi(\mathbf{k}, t)$=

$\lim_{t\to\infty} Pr[\mathbf{k}(t) = \mathbf{k}]$ which can be described by the steady-state Chapman-Kolmogorov equations for continuous time Markov chain systems [56]:

$$\pi(\mathbf{k})\sum_{i=1}^{N}\left[\Lambda_i + (\lambda_i + r_i)\mathbf{1}_{\{k_i>0\}}\right] = \sum_{i=1}^{N}\Big\{\pi\left(\mathbf{k}_i^{+}\right) r_i d(i) + \pi\left(\mathbf{k}_i^{-}\right)\Lambda_i \mathbf{1}_{\{k_i>0\}} + \pi\left(\mathbf{k}_i^{+}\right)\lambda_i + \sum_{j=1}^{N}\Big[\pi\left(\mathbf{k}_{ij}^{+-}\right) r_i p^{+}(i,j)\mathbf{1}_{\{k_j>0\}} + \pi\left(\mathbf{k}_{ij}^{++}\right) r_i p^{-}(i,j) + \pi\left(\mathbf{k}_i^{+}\right) r_i p^{-}(i,j)\mathbf{1}_{\{k_j=0\}}\Big]\Big\} \quad (2.5)$$

The values of the stationary parameters of the network, the stationary excitation probabilities $q_i = \lim_{t\to\infty} q_i(t) \quad i = 1, ..., N$ and the stationary probability distribution $\pi(\mathbf{k})$ are derived from Theorem 1.

Theorem 1 [56]: *Let the total arrival rates of positive and negative signals $\lambda^{+}(i)$ and $\lambda^{-}(i)$, $i = 1, ...N$ be given by the following system of equations*

$$\lambda^{+}(i) = \Lambda_i + \sum_{j=1}^{N} r_j q_j p^{+}(j,i) \quad (2.6)$$

$$\lambda^{-}(i) = \lambda_i + \sum_{j=1}^{N} r_j q_j p^{-}(j,i) \quad (2.7)$$

where

$$q_i = \min\left\{1, \frac{\lambda^{+}(i)}{r_i + \lambda^{-}(i)}\right\} \quad (2.8)$$

If a unique non-negative solution $\{\lambda^{-}(i),\ \lambda^{+}(i)\}$ exists for the non-linear system of Eqs. (2.6)-(2.8) such that $q_i < 1\ \forall i$, then:

$$\pi\left(\mathbf{k}\right) = \prod_{i=1}^{N}\pi_i(k_i) = \prod_{i=1}^{N}\left(1 - q_i\right) q_i^{k_i} \quad (2.9)$$

The theorem states that whenever a solution to the signal flow Eqs. (2.6)-(2.8) can be found such that $q_i < 1, \forall i$, then the stationary joint probability distribution of the network has the simple product form (2.9) associated with the marginal prob-

abilities of each neuron, $\pi_i(k_i)$. The condition $q_i < 1$ can be viewed as a "stability condition" that guarantees that the excitation level of each neuron remains finite with probability one. Product form implies independence of the neurons despite the fact that the neurons are coupled through the exchanged signals. A result of their independence is that we can easily compute parameters that are associated with a single neuron such as the average steady-state excitation level of neuron i which is equal to $q_i/(1-q_i)$.

In [56], the case where a number of neurons are saturated is also discussed. Neuron i is *saturated* if $\lambda^+(i) \geq r_i + \lambda^-(i)$ so that it continuously fires in steady-state and its excitation probability is equal to one. It is shown that the product form solution given by Eq. (2.9) is still valid for the set of non-saturated neurons.

2.2.3. Network Stability

The network is stable if the signal potential of each neuron does not tend to increase without bounds. Due to the product form stationary probability distribution of the system, stability is guaranteed if a unique solution exists to the nonlinear system of Eqs. (2.6)-(2.8) and $q_i < 1, \ \forall i$. In addition, it can be easily shown that if a solution to Eqs. (2.6)-(2.8) exists with $q_i < 1, \ \forall i$ then it is unique [57]. The result stems from the fact that $\pi(\mathbf{k})$ is unique when $0 < q_i < 1, \ \forall i$ because the process $\{\mathbf{k}(t), t \geq 0\}$ is an irreducible continuous time Markov chain and $\pi(\mathbf{k})$ is positive with unit norm which follows from Theorem 1. Furthermore, for any i it is impossible to have two different values q_i and q_i' satisfying the unique $\pi(\mathbf{k})$ when $k_i = 0$; hence existence of the solution implies its uniqueness.

As a result, the key to proving stability is to show the existence of the solution, a result that is non-trivial due to the non-linearity of the signal-flow equations. Early studies examined the solution existence in special RNN architectures. In [56], it is proven that a solution always exists in the feed-forward RNN architecture since the computation of q_i in one layer depends only upon the values of neurons in the preceding layer which have already been computed. In [57], solution existence is presented for balanced networks which have identical $q_i, \forall i$ and damped networks which are governed by the hyper-stability condition:

$$r_i + \lambda_i > \Lambda_i + \sum\nolimits_{j=1}^{N} r_j p^+(j,i) \tag{2.10}$$

Although the hyper-stability condition appears to be strong, it can be used to appropriately select parameters of the network to guarantee stability [87].

Solution existence to the general case has been established in [60]. The approach followed is general and has also been used to examine solution existence in extensions of RNN. Next, the proof to the existence of a solution $\{\lambda^+(i), \lambda^-(i)\}$ to Eqs. (2.6) and (2.7) is outlined.

Initially, the q_i terms are eliminated from Eqs. (2.6)-(2.7) and the latter are combined to obtain:

$$\boldsymbol{\lambda}^- - \boldsymbol{\lambda} = \boldsymbol{\lambda}^+\mathbf{H}\mathbf{P}^- = \boldsymbol{\Lambda}(\mathbf{I} - \mathbf{H}\mathbf{P}^+)^{-1}\mathbf{H}\mathbf{P}^- \tag{2.11}$$
$$\boldsymbol{\lambda}^-,\ \boldsymbol{\lambda}^+,\ \boldsymbol{\lambda},\ \boldsymbol{\Lambda} \in \mathbb{R}^{1\times N} \text{ and } \mathbf{I},\ \mathbf{H},\ \mathbf{P}^{\pm} \in [0,1]^{N\times N}$$

where $\boldsymbol{\lambda}^{\pm}$, $\boldsymbol{\Lambda}$ and $\boldsymbol{\lambda}$ are vectors representing the total and exogenous arrival rates of excitatory-inhibitory signals, $\mathbf{P}^+$ and $\mathbf{P}^-$ are square matrices with elements the transition probabilities $p^{\pm}(i,j)$, $\mathbf{I}$ is the identity matrix and $\mathbf{H}$ is a diagonal matrix with elements $h_{ii} = r_i/(r_i + \lambda^-(i)) \leq 1$.

Because $\mathbf{P}^+$ is sub-stochastic and all elements of $\mathbf{H}$ are smaller than 1, the series $\sum_{m=0}^{\infty}(\mathbf{H}\mathbf{P}^+)^m$ is geometrically convergent so that we can write:

$$(\mathbf{I} - \mathbf{H}\mathbf{P}^+)^{-1} = \sum\nolimits_{m=0}^{\infty}(\mathbf{H}\mathbf{P}^+)^m \tag{2.12}$$

Defining $\mathbf{y} = \boldsymbol{\lambda}^- - \boldsymbol{\lambda}$ the system can be written in the fixed point form:

$$\mathbf{y} = \mathbf{g}(\mathbf{y}) = \sum\nolimits_{m=0}^{\infty}(\mathbf{H}\mathbf{P}^+)^m\mathbf{H}\mathbf{P}^- \tag{2.13}$$

where the dependence of $\mathbf{g}$ on $\mathbf{y}$ comes from $\mathbf{H}$, while $\mathbf{g}(\mathbf{y})$ is continuous and always nonnegative. According to Brouwer's fixed point theorem, Eq. (2.13) has at least one fixed point solution. In this case, exactly one fixed point must exist $\mathbf{y}^*$ since solution uniqueness has already been established. As a result, a solution to Eq. (2.6) - (2.8) always exists and it is unique.

2.2.4. Analogy with the formal neural networks

In [56], the analogy between formal neurons and RNN neurons is discussed. In formal neural networks, the input to neuron i, v_i, is a combination of the weighted sum of other neuron outputs, y_j, and a threshold value θ_i such that $v_i = \sum_j w_{ji}^A y_j -$

θ_i. Whether neuron i will be excited or not is determined by an activation function according to $y_i = g(v_i)$. The analogy of RNN with this model is established for the unit-step activation function.

Because the RNN weights are non-negative, each weight $w_{ij}^A \in \mathbb{R}$ is represented by a pair of weights such that:
$w^+(i,j) = \max\{0, w_{ij}^A\}$, $w^-(i,j) = \max\{0, -w_{ij}^A\}$
Moreover, non-output RNN neurons do not dissipate, $d(i) = 0$, and their firing rate r_i is given by Eq. (2.4), while for output neurons, $d(i) = 1$. Parameters θ_i and y_i are associated with λ_i and q_i respectively. When y_i is binary, a threshold value, α, can separate 0 and 1 according to:
$[y_i = 0] \Leftrightarrow q_i < 1 - \alpha$ and $[y_i = 1] \Leftrightarrow q_i \geq 1 - \alpha, \forall i$.
Note that all RNN parameters are mapped to formal neurons' parameters except from the firing rates of the output neurons, the rate of external positive signal Λ_i and α. These parameters are set to appropriate values so that output neurons have the desired behaviour.

2.2.5. Function approximation

One important feature of a neural network model is its ability to approximate functions with an arbitrary degree of accuracy. The authors of [76] have proven that the feed-forward Bipolar-RNN (BRNN), discussed in section 2.3.1, and Clamped-RNN (CRNN), same as the RNN with the addition of a constant value to the average potential of output neurons, have the universal approximation ability for any continuous function on a bounded set [0,1], i.e. functions of the form $f : [0,1]^s \rightarrow \mathbb{R}^w$. Such functions can be separated into one dimensional functions of the form $f_w : [0,1]^s \rightarrow \mathbb{R}$. To prove universal approximation for the latter case, the authors first established the result for a 1-input 1-output function and then generalised it to the s-input 1-output case. Their method is based on constructing an RNN that reproduces a polynomial that is an estimator of the bounded continuous function under consideration.

In [77], the approximation capabilities of RNN were further explored to limit the number of total layers required. The authors proved that for both the BRNN and CRNN models, functions of the form $f : [0,1] \rightarrow \mathbb{R}$ can be arbitrarily approximated with an architecture of one input, one hidden and one output layers. Extending the result for the approximation of functions $f_w : [0,1]^s \rightarrow \mathbb{R}$, it was derived that for

both the BRNN and CRNN models arbitrary approximation can be accomplished by considering an architecture with s-hidden layers.

2.2.6. Hardware implementations

The processing capabilities of brain neuronal networks rely on their massively parallel architecture. Artificial neural networks are parallel as well, but software implementations result in sequential execution. The power of neural processing is unveiled with their implementation in hardware. In [1], an analog implementation of RNN that captures the performed addition and multiplication operations has been proposed. An implementation of RNN using discrete logic integrated circuits has also been proposed in [37]. The realisation of the network is achieved using four modules. An input module is needed for the input signals, a second module for the signal aggregation at each neuron, a random number generator for the generation of the exponential distributed signals fired by neurons and a routing module for the propagation of signals between neurons.

The stochastic nature of the RNN has also been manipulated for its efficient realisation on probabilistic CMOS (PCMOS). PCMOS harness the probabilistic behaviour of the circuits exhibited in the nanoscale regime, because of process variations and noise, yielding significant improvements in terms of energy consumption and performance [14]. The authors of [38], realised the RNN on a PCMOS co-processor for the solution of the minimum vertex covering problem. They implemented the core probabilistic module of RNN associated with the random firing of neurons on PCMOS, instead of a pseudorandom number generator, and the rest of the network on a conventional microprocessor. Experimental evaluation showed that PCMOS RNN co-processor exhibited orders of magnitude less energy consumption and execution speed-up compared to an implementation on a conventional microprocessor.

2.3. RNN extension models

Apart from the original RNN, models of RNN with additional capabilities have been developed. Similar to the original RNN, all models maintain a product-form solution which may differ according to the model considered. In this section we

describe the Bipolar RNN (BRNN), a model of RNN with State-Dependent Firing (RNNSDF) and the Multiple Class RNN (MCRNN).

2.3.1. Bipolar random neural network

The bipolar RNN has been introduced in [81] to represent bipolar patterns and facilitate associative memory capabilities. Contrary to the original RNN there are two different types of neurons: (a) the positive neurons which have the same behaviour with the neurons of the original RNN and (b) the negative neurons which have opposite behaviour to the positive ones. In other words, negative neurons accumulate negative signals so that the reception of positive signals has the suppressive role. Hence, signals emitted from a negative neuron i arrive to neuron j as positive (resp. negative) signals with probability $p^-(i,j)$ (resp. $p^+(i,j)$). The model is governed by similar signal-flow equations to the original RNN, taking into consideration the effect of both positive and negative neurons, while it retains a geometric product form stationary probability distribution for the neuron potentials.

The BRNN has been applied in associative memory to obtain better separation between bipolar patterns. Moreover, the feed-forward BRNN has been utilised to prove the universal approximation properties of RNN.

2.3.2. RNN with state-dependent firing

Although in the original RNN the firing rate of neurons is constant, it is more biologically plausible to assume that the firing rate of neurons depends on the signal potential. In [170], a model with state-dependent firing has been proposed and its properties have been investigated. The RNNSDF differs from the original RNN in two aspects:

1. The firing rate is exponentially distributed but it is potential-dependent instead of constant. Dependence is added as a multiplication factor so that the firing rate is $r_i\psi_i(k_i)$, where $\psi_i(k_i) > 0$ for $k_i > 0$ and bounded above by B_i.

2. When a negative signal arrives at an excited neuron j it reduces the potential of the neuron by 1 with a state-dependent probability $\psi_j(k_j)/B_j$, otherwise it has no effect.

The authors proved that under certain conditions the model has a simple product-form solution which is dependent on $\psi_i(k_i), \forall i$; this implies that the RNNSDF can exhibit a variety of stationary probability distribution structures by altering $\psi_i(k_i)$, contrary to the RNN whose distribution is geometrical and decreasing with respect to k_i.

2.3.3. Multiple class random neural network

In the Multiple Class Random Neural Network (MCRNN) [66], there are C different classes of positive signals and a class of negative signals. As a result, the potential of neuron i is described by a vector of signal potentials, each associated with a different class of signals, $\mathbf{k}_i = [k_{i1}, ..., k_{iC}]$ so that $k_i = \sum_c k_{ic}$. Positive exogenous signals arrive to neuron i according to a Poisson distribution of rate Λ_{ic} and increase the potential k_{ic} by 1. Negative exogenous signals also arrive according to a Poisson distribution with rate λ_i. If at time t a negative signal is received by neuron i, then if it is excited, $k_i(t) > 0$, the potential of class c signals will become $k_{ic}(t^+) = k_{ic}(t) - 1$ with probability k_{ic}/k_i. When a neuron is excited it fires a class c signal with probability k_{ic}/k_i at rate r_{ic} and the potential k_{ic} is reduced by 1. If such an event occurs the following can happen: (a) with probability $p^+(i, c; j, \xi)$ it goes to neuron j as a positive class ξ signal, (b) with probability $p^-(i, c; j)$ it goes to neuron j as a negative signal, or (c) it leaves the network with probability $d(i, c)$. As the other models, MCRNN also obeys to a product-form solution for each neuron and each class of signals.

The MCRNN can be used in applications associated with the concurrent processing of different streams of information such as colours in image processing or attributes in a data network.

2.4. Learning algorithms

One of the most important features of a neural network model is its ability to learn from examples. In this section we describe the standard gradient descent supervised learning algorithm for the RNN [60] and other supervised learning algorithms proposed for the model and its extensions. Initialisation algorithms that can be exploited by the supervised learning algorithms are also discussed. The section is completed with a discussion on RNN reinforcement learning algorithms.

2.4.1. Gradient descent supervised learning algorithm for the RNN model

A gradient descent supervised learning algorithm for the recurrent RNN has been developed by Gelenbe in [60]. In RNN, the kth input training pattern $\mathbf{x}_k$ is represented by the vectors $\boldsymbol{\Lambda_k} = [\Lambda_{1k},\ \ldots\ , \Lambda_{Nk}]$ and $\boldsymbol{\lambda_k} = [\lambda_{1k},\ \ldots\ , \lambda_{Nk}]$. Usually the approach taken is to assign the input training values, x_{ik} to the exogenous arrival rates such that:

- If $x_{ik} > 0$ then $\Lambda_{ik} > 0$ and $\lambda_{ik} = 0$
- If $x_{ik} \leq 0$ then $\Lambda_{ik} = 0$ and $\lambda_{ik} > 0$

The values of the non-zero elements produced from the above expressions can be taken equal to $|x_{ik}|$, or some constant value Λ and λ respectively to ensure network stability.

The desired values of the kth pattern, $\mathbf{y}_k$, are represented by the steady-state excitation probabilities of the neurons $\mathbf{q}_k = [q_{1k},\ \ldots\ , q_{Nk}]$ emanating from applying input training pattern k to the network. The RNN weights updated during the learning process are $w^+(i,j)$ and $w^-(i,j)$.

Without loss of generality we assume that the error function to be minimised is a general quadratic function of the form:

$$E = \sum_{k=1}^{K} E_k = \frac{1}{2} \sum_{k=1}^{K} \sum_{i=1}^{N} \bar{c}_i (g_i(q_{ik}) - y_{ik})^2 \tag{2.14}$$

where E_k is the error function of the kth input-output pair, $\bar{c}_i \in \{0, 1\}$ shows whether neuron i is an output neuron and $g_i(q_{ik})$ is a differentiable function of neuron i.

In the proposed approach by Gelenbe, the training examples are sequentially processed and the weights of the network are updated according to the gradient descent rule until a minimum of the error function is reached. If we denote by the generic term $w(u,v)$ either $w^+(u,v)$ or $w^-(u,v)$, the rule for updating the weights using the $k-th$ input-output pair at step $(\tau+1)$ is:

$$w_{\tau+1}(u,v) = w_\tau(u,v) - \eta \left[\frac{\partial E_k}{\partial w(u,v)} \right]_\tau \tag{2.15}$$

The partial derivative of the error function with respect to $w(u,v)$ can be calculated based on (2.14) and yields:

$$\left[\frac{\partial E_k}{\partial w(u,v)}\right]_\tau = \sum_{i=1}^{N} \bar{c}_i \left(g_i(q_{ik}) - y_{ik}\right)) \times \left[\frac{\partial g_i(q_i)}{\partial q_i}\right]_\tau \left[\frac{\partial q_i}{\partial w(u,v)}\right]_\tau \tag{2.16}$$

where the operator $[\cdot]_\tau$ denotes that all calculations are performed using the weight values of step τ and the q_{ik} values derived from solving Eqs. (2.6)-(2.8) when the current weights $w_\tau(u,v)$ are used. The challenging step in the evaluation of Eqs. (2.15) - (2.16) is the derivation of a closed expression for the term $[\partial q_i/\partial w(u,v)]_\tau$ which depends on the nonlinear system of Eqs. (2.6)-(2.8).

Gelenbe [60] proved that the above term can be expressed in the following form:

$$\frac{\partial \mathbf{q}}{\partial w^+(u,v)} = \boldsymbol{\gamma}^+(u,v)\left(\mathbf{I} - \mathbf{W}\right)^{-1} \tag{2.17}$$

$$\frac{\partial \mathbf{q}}{\partial w^-(u,v)} = \boldsymbol{\gamma}^-(u,v)\left(\mathbf{I} - \mathbf{W}\right)^{-1} \tag{2.18}$$

where $\mathbf{I}$ is the identity matrix, while $\mathbf{W}$, $\boldsymbol{\gamma}^+(u,v)$ and $\boldsymbol{\gamma}^-(u,v)$ are given by equations (2.19), (2.20) and (2.21) respectively, when $d(i) = 0,\ \forall i$.

$$\mathbf{W}(i,j) = \frac{w^+(i,j) - w^-(i,j)q_j}{r_j + \lambda^-(j)}, \forall i,j \tag{2.19}$$

$$\gamma_i^+(u,v) = \begin{cases} -q_u/D(i) & u = i,\ v \neq i \\ q_u/D(i) & u \neq i,\ v = i \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

$$\gamma_i^-(u,v) = \begin{cases} -q_u/D(i) & u = i,\ v \neq i \\ -q_u q_i/D(i) & u \neq i,\ v = i \\ -q_u(1+q_i)/D(i) & u = i,\ v = i \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

The term $D(i) = r_i + \lambda^-(i)$ is the denominator of q_i.

The steps of the gradient descent RNN learning algorithm are the following:

(1) Initialise the weights $w^+(u,v)$ and $w^-(u,v)\ \forall u,\ v$ and appropriately choose the learning rate η.

(2) For each input-output pattern k do:

- **(a)** Initialise Λ_{ik} and λ_{ik} according to x_{ik} and define the desired output values y_{ik} for all i, k.
- **(b)** Solve the system of Eqs. (2.6)-(2.8) using the current weight values.
- **(c)** Based on the values attained, calculate $\mathbf{W}$, $\boldsymbol{\gamma}^+(u,v)$ and $\boldsymbol{\gamma}^-(u,v)$, $\forall u, v$.
- **(d)** Calculate $[\partial \mathbf{q}/\partial w^+(u,v)]_\tau$ and $[\partial \mathbf{q}/\partial w^-(u,v)]_\tau$ according to Eqs. (2.17) and (2.18).
- **(e)** Update the weights from Eqs. (2.15) and (2.16). To satisfy the weight nonnegativity constraint either the negative values can be set to zero or the iteration can be repeated with a smaller value of η.

(3) Repeat the procedure of step (2) until a stopping criterion is met.

The complexity of the algorithm for updating one weight $w(u,v)$ is $O(N^3)$ because of the inversion operation in Eqs. (2.17) and (2.18), which is the most demanding step of the algorithm [60]. Note that according to an iterative algorithm presented in [55], the complexity of solving the system of Eqs. (2.6)-(2.8) is $O(N^2)$ per iteration and the system converges at a rate better than a geometric sequence; hence it is less computationally demanding than the inversion operation.

The complexity of the algorithm can be further reduced if $(\mathbf{I}-\mathbf{W})^{-1}$ is approximated by the linear term $(\mathbf{I}+\mathbf{W})$; the approximation holds when $\|\mathbf{W}\| < 1$ [98].

Weight initialisation

Supervised learning can be considered as a nonlinear and non-convex optimisation problem where our goal is to minimise the error function subject to the satisfaction of the signal-flow equations and the non-negativity constraints for our decision vector, the weights $w^+(i,j)$ and $w^-(i,j)$, $\forall i,j$; hence, convergence to a global optimum cannot be guaranteed. For this reason, developing efficient weight initialisation algorithms can help us obtain good solutions.

In the context of RNN, two weight initialisation methods have been mainly used: random initialisation and initialisation based on Hebbian rule.

In random initialisation small random values are used for the weights which are drawn from the uniform distribution in the range $[0, w_{max}]$. In practice, a good choice for w_{max} is 0.2.

Hebbian rule is a simple learning algorithm which has been used in perceptron neural networks. In the RNN, the weights can be initialised according to Eqs. (2.22) and (2.23) [106]:

$$w_{ij}^{H} = \sum_{k=1}^{K} (2x_{ik} - 1)(2x_{jk} - 1), \quad x_{ik} \in \{0, 1\} \tag{2.22}$$

$$w^{+}(i,j) = \max\{0, w_{ij}^{H}\}, \; w^{-}(i,j) = \max\{0, -w_{ij}^{H}\} \tag{2.23}$$

2.4.2. Alternative RNN supervised learning algorithms

Apart from the standard gradient descent algorithm described in section 2.4.1, other authors have also examined supervised learning in the context of RNN.

The author of [105] has modified the Resilient Propagation (RPROP) algorithm and utilised it for RNN supervised learning. In RPROP, the weights are updated based only on the temporal behaviour of the sign of the error function derivative. It is considered to be a resilient and transparent method because it is not influenced by any unexpected behaviour of the value of the error function derivative. Nevertheless, the RPROP-RNN method has the same complexity with the standard learning algorithm, while the two methods produce comparable results in terms of recognition of noisy patterns.

In [7], the use of genetic algorithms in conjunction with the gradient descent RNN learning algorithm is proposed to address the problem of converging to a local solution. M RNNs are trained in parallel according to an iterative process that involves RNN gradient descent learning and genetic operations on the network topologies, such as mutation and crossover, until convergence to a good solution. The genetic representation of each network is performed through an extended direct coding scheme where both the presence of links and the values of the weights are included accomplishing both parametric and structural modifications. Apart from mutation and crossover operations, local search and optimisation are also performed. Although this algorithm performs better than the gradient descent RNN algorithm it is significantly slower.

In [123], a quasi-Newton algorithm is developed for RNN supervised learning. Quasi-Newton algorithms are a well established class of iterative nonlinear optimisation techniques which rely on second order gradient information; however, instead of computing the Hessian matrix, which holds the second-order partial derivatives

of the error function, an approximation matrix with desirable properties is updated in each iteration of the algorithm. To update this matrix, the authors employ the well-known Broyden - Fletcher - Goldfarb - Shanno (BFGS) and Davidon - Fletcher - Powell (DFP) formulas. Contrary to the standard RNN learning algorithm, their approach is a batch learning algorithm where the weights are updated after processing all the training examples. The developed quasi-Newton algorithm is evaluated on the parity problem and comparison with the gradient descent algorithm demonstrates better performance and convergence to less than half steps compared to the standard RNN learning algorithm. Nonetheless, the algorithm cannot be used for online learning because it operates in a batch mode.

A gradient descent algorithm has also been developed for the MCRNN model [68]. Similar to the standard RNN learning algorithm, the gradient descent rule is employed to update the weights $w^+(i,c;j,\xi) = r_{ic}p^+(i,c;j,\xi)$ and $w^-(i,c;j) = r_{ic}p^-(i,c;j)$. This is achieved by obtaining the partial derivatives of $q_{ic}, \forall i,c$ with respect to these weights; the equations obtained have the same form with Eq. (2.17). However, in the multiple class case, the computational complexity of updating a weight is $O((NC)^3)$ because although we have N neurons, they must accumulate C different classes of positive signals.

2.4.3. Reinforcement learning in RNN

Reinforcement learning (RL) methodologies have also been developed in the context of RNN. In reinforcement learning a system takes a sequence of cascaded decisions related to the perceived state of the environment and accordingly receives external reinforcement either positive (reward) or negative (punishment). The goal is to find an optimal policy to obtain maximum reward for each perceived state.

Halici proposed a reinforcement learning scheme for a tree RNN-architecture for single and cascaded decisions [97, 99, 100]. In the general case, the system is composed of an input, a number of intermediate and an output layer of neurons. The input neurons perceive the state of the environment, while the neurons of intermediate layer m represent the possible states that can be reached after the mth decision step. Each connection (i_{m-1},j_m) represents the transition from state i_{m-1} to state j_m when decision a_m^τ of the τth trial is taken.

In each trial a signal is propagated from the input neurons to the output neurons which dissipate and excite the environment that returns the external reinforcement.

Decisions in the network are taken probabilistically. When neuron i is excited at trial τ, which means that it is an activated state, it fires a signal that reaches neuron j with probability $p_\tau^+(i,j)$ and activates it. The sequence of activated neurons from the input to the output layer is the decision cascade of trial τ, $\mathbf{a}^\tau$.

Reinforcement learning in RNN works by updating the weights of the connections at the end of each trial. The update is related to the attained environmental reinforcement so that "'good" decisions are rewarded. The reinforcement $R_\tau(i, \mathbf{a}_\tau)$ that neuron i receives in this cascaded decision environment is a function of several parameters such as the trial, the external reinforcement associated with the output neuron and the cost of the activated decision path.

Halici proposed three different weight update rules for single and cascaded decision rules: the reward rule (R-rule)[97], the reward and punishment rule (L-rule)[99], and the update rule with internal expectation of the reward (E-rule)[99, 100]. Experimental analysis showed that the E-rule is superior to the other rules both in terms of learning and adaptivity to environmental changes.

The success of the E-rule relies on an adaptive internal expectation of the reward. In this rule, the weights are updated according to the difference between the actual reinforcement and the internal expectation. For example, if the difference is positive, the weights of all selected neurons in the decision path are reinforced proportionally to the difference, while their neighbour neurons are punished. In this way, the algorithm is adaptive to changes in the environment and results in obtaining time-varying reinforcement.

In the following subsection we describe a variation of the E-rule that is extensively used for routing packets in the Cognitive Packet Network (CPN) [155].

RNN reinforcement learning in CPN

The RNN reinforcement learning (RNN-RL) algorithm used in the CPN is a simple cascaded decision algorithm that employs the idea of the internal expectation of the reward as the E-rule. However, contrary to the tree-RNN architecture a recurrent architecture is employed, while decisions are based on the excitation probabilities of the neurons rather than the connection probabilities.

In this algorithm the reinforcement value is a quality of service (QoS) metric of the communication system such as path delay or packet loss. The reinforcement $R_\tau^+(i, \mathbf{a}_\tau)$ is also a function of the network node. For instance if the reinforcement is

associated with the delay experienced in a source-destination pair in the communication network, then the reward is the inverse value of the delay from the particular node to the destination.

For each destination and QoS class, a node maintains a recurrent RNN, with non-dissipating neurons. Each neuron in an RNN, corresponds to a neighbour of the particular node. The weights and the internal expectation of the reward, $R^+_{\tau,\beta}$, are updated whenever a new reward $R^+_{\tau+1}(i, \mathbf{a}_{\tau+1})$ reaches node i according to the following rule for all $i \neq j$:

- If $R^+_{\tau,\beta} \leq R^+_{\tau+1}(i, \mathbf{a}_{\tau+1})$
  $w^+_{\tau+1}(i,j) = w^+_\tau(i,j) + R^+_{\tau+1}(i, \mathbf{a}_{\tau+1})$,
  $w^-_{\tau+1}(i,k) = w^-_\tau(i,j) + \frac{R^+_{\tau+1}(i,\mathbf{a}_{\tau+1})}{N-2}$, $\forall k \neq j$

- Else
  $w^+_{\tau+1}(i,j) = w^+_\tau(i,j) + \frac{R^+_{\tau+1}(i,\mathbf{a}_{\tau+1})}{N-2}$, $\forall k \neq j$,
  $w^-_{\tau+1}(i,k) = w^-_\tau(i,j) + R^+_{\tau+1}(i, \mathbf{a}_{\tau+1})$

In the above equation, index j corresponds to the selected neuron of the network that resulted in the particular reward value. Numerical instability is avoided by re-normalising the weights after each update by performing two operations. First, for each neuron i its new r^*_i is computed:

$$r^*_i = \sum_{m=1}^{n} \left[w^+_{\tau+1}(i,m) + w^-_{\tau+1}(i,m) \right].$$

Second, the weights are re-normalised such that:

$$w^+_{\tau+1}(i,j) \leftarrow w^+_{\tau+1}(i,j) \frac{r_i}{r^*_i} \tag{2.24}$$

$$w^-_{\tau+1}(i,j) \leftarrow w^-_{\tau+1}(i,j) \frac{r_i}{r^*_i} \tag{2.25}$$

Whenever an action is to be taken, the system of the signal flow Eqs. (2.6)-(2.8) is solved for the most recent weights to obtain values for the excitation probabilities of the neurons. The most excited neuron corresponds to the action to be taken and in the context of CPN represents the next node in the route of a packet.

2.5. Applications

In this section we review different applications of RNN with special emphasis on the solution of optimisation problems. We also review modelling and learning applications of RNN.

2.5.1. Solution of optimisation problems

Combinatorial optimisation problems routinely arise in many applications. However, they are usually NP-hard and cannot be optimally solved in a timely manner. Hopfield and Tank [103] proposed the use of artificial neural networks, such as the discrete Hopfield Network, for the solution of such problems attracting a lot of attention. Since their seminal work, different types of artificial neural networks have been employed for the solution of various combinatorial optimisation problems [165] and several neural techniques have been developed [166]. In this section we investigate the use of RNN for the solution of discrete optimisation problems. Emphasis is given to explain different solution approaches which include:

1. The parameter association approach
2. The dynamical RNN approach
3. The energy function approach

Parameter association approach

In the parameter association approach, different parameters of the RNN are associated with parameters of the optimisation problem under consideration. Binary decision variables of the problem are represented by the q_i parameters of RNN, while input parameters are usually associated with positive external arrivals Λ_i. The positive and negative probabilities of RNN, $p^+(i,j)$ and $p^-(i,j)$ are employed to capture the interactions between decision variables, the neurons, that stem from the constraints of the problem. For example, if in a row of neurons only one must be selected, then each of these neurons attempts to inhibit the others to be the one selected. Excitation interactions are also possible when two neurons benefit from the mutual selection. Some parameters such as the firing rate r_i can be selected to guarantee hyper-stability of the network so that there is a balance between excited and not excited neurons.

The procedure for the solution of problems using this approach is very simple. After the RNN parameters have been given appropriate values, the q_i values are computed and the most excited neuron is selected. Then the problem is reduced and the procedure is repeated until the final solution is reached.

The approach has been applied for the solution of several NP-hard optimisation problems including the minimum vertex covering problem (MVCP) [87], a task assignment problem with communication and imbalance costs emerging in distributed systems [8] and the satisfiability problem (SATP) [58].

In the above problems, the RNN approach has been used to construct a solution from scratch. An alternative approach is to consider a good heuristic for the examined problem to find an initial solution and then apply the parameter association method to indicate variables that could be included in the solution to improve its quality. This method has been proposed in [67] for the solution of the minimum Steiner tree problem (MSTP). The authors used a known heuristic for the MSTP problem to obtain a solution and then employed an RNN to select a vertex-neuron that could be a good "candidate" to be included in the solution. A reoptimisation algorithm was then adopted to find a new solution based on the current MSTP and the selected vertex. The new solution is definitely not worse than the current one. By repeating this procedure until all vertices have been considered, the final solution is at least as good as the initial. Experimentation illustrated that the particular method can improve the solution quality by $10-20\%$ without increasing the computational complexity.

The RNN approach for the solution of the MSTP problem has also been exploited for the solution of the Dynamic Multicast problem [13] and the Access Network Design problem [34], which are closely related to the MSTP problem.

In general, the parameter association approach provides a tradeoff between solution quality and computational complexity. It has consistently outperformed greedy heuristics for the different problems, but exhibited worse performance than metaheuristic techniques such as simulated annealing and genetic algorithms which are very computationally demanding. The performance of the parameter association approach has also been compared against a Hopfield neural network algorithm in the MVCP problem [91]. It was shown that the RNN is substantially better in solving the particular problem especially for large problems. Despite its success in many optimisation problems, one of the limitations of this approach is that it

cannot be easily generalised. Hence, it cannot be directly adopted for new problems but different parameter mappings must be tested to discover a good association.

Dynamical RNN approach

The idea behind the use of the Hopfield recurrent neural network for the solution of optimisation problems relies on mapping the parameters of the problem to the energy function of this dynamic network. The energy function of the Hopfield network is guaranteed to converge to an attractor of the energy landscape, which is hopefully a good solution to the problem. Although the RNN is a recurrent neural network as well, it cannot perform dynamic behaviour in terms of its output parameters in the form $\frac{d\mathbf{q}}{dt} = f(\mathbf{q})$, because of its unique equilibrium point. Therefore, the Dynamical Random Neural Network (DRNN) has been developed to exhibit dynamic evolution with regard to its input [148, 72].

DRNN uses only positive inputs and negative weights with no signals leaving the network ($d(i) = 0$) whereas its dynamical equation is similar to the Cohen-Grossberg equation with time delayed feedbacks:

$$\frac{d\Lambda_i}{dt} = a(q_i)\left(B(q_i) - \frac{\partial F(\mathbf{q})}{\partial q_i}\right), \ \forall i \tag{2.26}$$

where function $a(q_i)$ regulates the convergence rate while $B(q_i)$ is appropriately selected to place the attractors of the dynamical system in the best possible positions. $F(\mathbf{q})$ is the penalty function associated with the optimisation problem we are dealing with. Any constraints associated with the problem can be incorporated into $F(\mathbf{q})$ using the Lagrange multipliers method. The negative weights are assigned appropriate values so that a neuron inhibits the neurons that cannot be simultaneously excited with it.

After assigning appropriate values to the parameters of the DRNN model, the procedure progresses by iteratively computing the q_i values based on Eqs. (2.6)-(2.8) and updating the Λ_i values according to Eq. (2.26) until the Λ_i values have stabilised.

The DRNN has been exploited for the solution of several optimisation problems including the independent set [148] and travelling salesman problems [72], as well as a problem of optimal resource allocation with minimum and maximum activation levels for each resource and fixed costs [184]. Interestingly, the latter problem

includes both binary and continuous variables. For modelling each of the continuous variables, the authors used n neurons. Neuron i_r, $1 \leq i_r \leq n$ is associated with variable a_r and represents value 2^{-i_r} so that: $a_r = \sum_{i_r=1}^{n} q_{i_r} 2^{-i_r}$.

Performance comparison in the various problems has demonstrated the superiority of the DRNN over heuristic and Hopfield neural network approaches. An advantage of the DRNN compared to the parameter association approach is that it incorporates into the formulation the penalty function $F(\mathbf{q})$; nevertheless, some parameters such as the $B(q_i)$ and $w^-(i,j)$ are assigned values in an ad-hoc manner.

Energy function approach

In the DRNN the dynamic evolution of the input parameters Λ_i is used for the solution of optimisation problems. Another approach is to evolve Λ_i according to the gradient of the cost function $F(\mathbf{q})$ [117]:

$$\Lambda_i^{\tau+1} = \Lambda_i^{\tau} - \eta \left[\frac{\partial F(\mathbf{q})}{\partial \Lambda_i} \right]_{\tau} = \Lambda_i^{\tau} - \eta \left[\frac{\partial F(\mathbf{q})}{\partial q_i} \frac{\partial q_i}{\partial \Lambda_i} \right]_{\tau}, \forall i \tag{2.27}$$

In the above equation, the term $\frac{\partial q_i}{\partial \Lambda_i}$ can be computed based on Eqs. (2.6) - (2.8), similar to the derivation of $\frac{\partial q_i}{\partial w(u,v)}$ performed for the RNNSI model in Appendix A, yielding a linear system of equations [9]. Furthermore, to simplify the computation of the term $\frac{\partial F(\mathbf{q})}{\partial q_i}$ the general quadratic energy function $E(\mathbf{q})$ has been proposed:

$$E(\mathbf{q}) = \sum_{i=1}^{N} \sum_{j<i} a_{ij} q_i q_j + \sum_{i=1}^{N} a_{ii} q_i^2 + \sum_{i=1}^{N} b_i q_i + c \tag{2.28}$$

A result of the above formulation is that if the optimisation problem at hand is of similar form, then by mapping the parameters of the problem to the energy function, the term $\frac{\partial E(\mathbf{q})}{\partial q_i}$ can be easily computed without any additional effort. The other parameters of the RNN are initialised so that the constraints associated with the problem are strengthened through the neuron interactions, as discussed in previous approaches.

The solution procedure is similar to that of the DRNN approach, with the difference that the Λ_i values are updated not based on Eq. (2.26), but based on Eq. (2.27). Additionally, both the RNN and the energy function parameters must be assigned appropriate values.

The approach has been evaluated with respect to two optimisation problems: the

graph partitioning problem and the minimum vertex covering problem [9]. The method exhibits comparable or better performance compared to greedy heuristics, simulated annealing and genetic algorithms.

2.5.2. Modelling applications

The RNN is a prominent modelling tool that can capture the behaviour of interacting entities in complex systems such as biological and queueing networks.

To begin with, due to the direct analogy between the RNN and queueing networks [83], the notion of inhibitory signals in RNN inspired the use of negative customers in queueing networks for work removal [86]. This model attracted a lot of attention, resulting in a generalised class of networks called *G-networks* [85].

In G-networks positive and negative customers (excitatory and inhibitory signals) circulate in the network. When a positive customer arrives at a server (neuron), it increases the size of the queue (neuron potential) by 1, while the arrival of a negative customer cancels a positive one, if at least one is present in the queue. Service completion (neuron firing) decreases the queue size by one and causes the movement of a customer that will reach another node as a positive or negative customer or depart from the network in a probabilistic manner.

Negative customers may be replaced by signals that have a more general role. They can be used to trigger the instantaneous movement of a customer from one queue to another [59], or reset the queue to a new value when it is empty [88]. It is also possible to have multiple classes of positive and negative customers [54], as well as multiple classes of signals with triggering effect [89]. As in RNN, the stationary probability distribution of the queue lengths have product form under certain conditions. Furthermore, it has been proven that two G-networks of the same size that have the same stationary rates of positive customers and signals for all queues (*flow equivalence*) are also governed by the same joint queue length probability distribution (*stationary equivalence*) [53]. An extended survey of G-networks can be found in [15].

In [84], an analytical solution for a class of stochastic genetic algorithms is derived according to the behaviour of a population of different chromosome types. Different genetic operations are modelled using first and second order probabilistic interactions including the birth and death of a type i chromosome, the mutation of a type i chromosome to a type j chromosome, as well as the crossover operation

where a type i chromosome combines with a type j chromosome to produce a type k chromosome.

In [62], the modelling of a gene regulatory network with interacting genes or other biochemical substances is studied. As in the genetic algorithms model, interactions between agents include not only first order interactions but also second order interactions where more than two agents have a combined effect on a third one. Logical dependencies among agents are also modelled and an analytic steady-state solution of the overall system is derived.

Moreover, RNN has been used to model interactions between different areas of the human brain to predict the oscillatory behaviour of those areas with response to somatosensory inputs [63]. In particular, a recurrent RNN is constructed with three neurons, each representing a neuronal layer associated with the thalamus, the cortex and the reticular layer. The strength of excitatory and inhibitory interactions betweens these areas, the measured firing rates and the delays in the signal propagation from one area to the other are considered in the model. The parameters of the constructed model are estimated using experimental data and exploited to predict the oscillatory behaviour of the system relating to different interaction strengths and delays.

2.5.3. Learning applications

RNN has not only been successful in optimisation and modelling, but also in applications that exploit its learning capabilities such as image processing, communication systems, simulation, pattern recognition and classification.

A number of applications utilise the RNN supervised learning algorithm to group items into two or more different classes according to related inputs. For the solution of a problem involving M different classes, one trains either an RNN with M output neurons or M different RNNs, each having one output neuron. The excitation value of an output neuron is close to one for training instances of its associated class, otherwise its value is close to zero. For each testing instance, the class corresponding to the most excited neuron is selected. This technique have been exploited for texture classification [173] and image segmentation [65, 127], for the identification of mines [2, 3], the detection of denial of service attacks in communication networks [144], as well as for target [20], vehicle [107] and noisy pattern recognition [167, 122].

In other applications, supervised learning is used to construct an RNN that rep-

resents the input-output mapping of an unknown continuous function. In this case, the desired excitation values of the output neurons correspond to the function values for the specific training instance. Several applications are based on this approach including the modelling and generation of gray [18, 69] and colour textures [68, 17], image fusion [21], compression of still and moving images [41, 64, 43, 42], quantification of the quality of service in multimedia services [137, 138, 44] and the prediction of the profile of cross-section wafer surface images in semiconductor fabrication [71].

Despite the widespread applicability of the RNN supervised learning algorithm, it cannot be utilised for the solution of problems with unknown desired output. Such problems are solved using reinforcement learning, where the desired outputs are discovered by trying different actions and observing the received reward. For this reason, the RNN-RL algorithm has been mostly exploited in dynamic and unknown environments, where information needs to be collected before finding the best course of action. Therefore, RNN-RL has been utilised to control the routing of packet in the Cognitive Packet Network [75, 74, 48, 78], a connectionless packet switching network with "intelligent" packets, and to design learning agents with realistic behaviour in augmented reality simulation environments [70, 79].

Table 2.1, summarises the different problems addressed in various application areas of RNN.

2.6. Conclusions

In this chapter we have surveyed the research work undertaken in the Random Neural Network. RNN is a biologically inspired, open, recurrent neural network with closed form expression for the probability steady-state and analytically solvable signal-flow equations. The properties of the model as well as different learning algorithms and extension models have been described. Furthermore, numerous applications of the model have been reviewed with emphasis on optimisation and modelling applications. The plethora and diversity of applications reflect the prominence of the RNN either as a modelling tool or as a learning tool that can be trained fast and exhibit strong generalisation capabilities.

Table 2.1.: Summary of RNN applications

Application Area	**Investigated problems and related references**
Modelling	G-networks [15, 53, 54, 59, 83, 85, 86, 88, 89], genetic chromosome population [84], gene regulatory networks [62], corticothalamic oscillatory behaviour [63]
Optimisation	minimum vertex covering [87], task assignment in distributed systems [8], satisfiability problem [58], Minimum Steiner Tree [67], Dynamic Multicast [13], Access Network Design [34], independent set [148], travelling salesman [72], optimal resource allocation [184], graph partitioning [9]
Image Processing	texture generation of gray [18, 69] and colour [17, 68] images, texture classification and retrieval [173], biomedical image segmentation [65, 127], image fusion [21], image enlargement [21, 22], image and video compression [41, 42, 43, 64]
Communication Systems	cognitive packet network [48, 61, 73, 74, 75, 78, 101], DoS attack detection [144], automatic quantification of the PSQA metric for multimedia applications [44, 137, 138, 153], call admission control in ATM networks [19], multimedia server modelling [80]
Simulation	learning agents[79], injection of autonomous agents in augmented reality[70]
Pattern Recognition & Classification	associative memory [10, 81, 105, 106, 122, 167] target recognition [20], laser intensity vehicle classification system [107], wafer surface reconstruction [71], mine detection [2, 3]

3. Learning extensions of the random neural network model

In this chapter we propose two novel supervised learning algorithms related to the RNN. The first, is a gradient descent learning algorithm for the RNN with synchronised interactions (RNNSI). In the RNNSI apart from the excitatory and inhibitory interactions, neurons can also exhibit synchronised interactions where two neurons can jointly act to excite a third one. We derive the steps of the algorithm and show that it retains the form and complexity of the standard RNN learning algorithm [60].

The second is a learning algorithm for the RNN that is mostly suitable for problems where the ratio of the number of output to the total number of neurons is large. This approach is based on modelling the signal-flow equations of the network as a nonnegative least squares (NNLS) problem; this can be accomplished when all the neurons have desired values. We then solve the NNLS problem by developing a large-scale projected gradient descent algorithm. To deal with the case that the network is also composed of non-output neurons, we develop the RNN-NNLS algorithm, a procedure in which we iteratively solve an NNLS problem and the RNN signal-flow equations with respect to the NNLS solution. The obtained weights can serve as the final trained weights and as a good initialisation point of the standard RNN learning algorithm.

Chapter 3 is structured as follows: section 3.1 describes the motivation of this work and discusses associated research approaches. In section 3.2, we present the RNNSI learning algorithm. We start with a discussion of the model's biological relevance and a description of its mathematical properties. Then, we derive the main steps of the algorithm and finish with an extensive computational complexity analysis that results in efficient modifications. In Section 3.3, we develop the RNN-NNLS algorithm. Firstly, we formulate RNN supervised learning when all neurons

have desired output values as an NNLS problem. For its solution we develop a limited memory quasi-Newton algorithm, and present the RNN-NNLS algorithm that can be employed for the solution of problems involving both output and non-output neurons. We also outline two approaches for the efficient evaluation of the objective and gradient NNLS functions by manipulating the special structure of the examined problem and discuss the computational complexity of the approach. Section 3.4 summarises the main outcomes of the chapter.

3.1. Introduction

One of the salient features of an artificial neural network is its ability to learn from the environment to improve its performance on a given task. There are various learning paradigms including unsupervised and reinforcement learning but the most important is supervised learning in which the neural network is presented with a set of input-output pairs, the training data, and its task is to learn how to map specific inputs to outputs.

Although the majority of the literature in supervised learning is concerned with feedforward neural networks [108], recurrent neural networks are much more powerful because they possess many degrees of freedom and they exhibit nonlinear dynamic behaviour [150]. Also their recurrent structure, that allows every neuron to interact with all other neurons, offers a natural approach to problems where each neuron represents an element of the problem under investigation. For example, in image processing if each image element is associated with a neuron then the relationship between neighbouring elements can be captured by local neighbourhood neuron interactions. Also, in combinatorial optimization the decision variables can be represented by different neurons so that the relationship between these variables can be captured through recurrent interactions between the neurons. As a result, designing efficient learning algorithms for recurrent neural networks is an active area of research for the past twenty years [149, 102, 109, 156].

In addition, to further enhance the ability of recurrent neural networks of dealing with hard learning tasks and solving combinatorial optimisation problems, investigators have considered networks with high-order connections, where more than one state variables may appear in product terms in the model equations. It has been shown that high-order networks can converge faster while they have better

generalisation and storage capacity [95, 172]. Applications of high-order neural networks include the representation and identification of finite state automata [93] and dynamical systems [116], as well as the solution of combinatorial optimisation problems [4, 5, 40].

In the context of RNN, general supervised learning algorithms have been designed for the original RNN and MCRNN (see chapter 2). However, no algorithms have been developed for RNN related models with higher-order connections. In section 3.2, we develop a learning algorithm for an extension of RNN with second-order interactions, the RNNSI. We also examine the complexity of the developed RNNSI algorithm and show that it matches the complexity of the gradient descent algorithm of the original RNN [60].

Apart from the introduction of the RNNSI gradient descent algorithm, we also develop a supervised learning algorithm for the RNN based on NNLS. The idea associated with this algorithm is that we can approximate the equations governing the RNN as a linear system, when each neuron in the network is an output neuron and hence has a desirable value for each input pattern. This approximation yields a linear least squares problem with nonnegativity constraints which is a convex optimization problem and can be solved to optimality. We also extend the approach to deal with cases involving non-output neurons. The trained weights produced by the algorithm can serve both as the final weights used in testing or as "good" initial weights of the standard learning algorithm which can potentially lead to a better minimum of the supervised learning problem and hence improve the learning capacity of the network; or they can reduce the starting error of the gradient descent algorithm leading to a substantial reduction in the execution time of the overall procedure.

Linear least squares techniques for learning and weight initialisation have been utilised in feedforward connectionist neural networks. These methods are based on the observation that the inputs to the neurons of a given layer is a linear function of the outputs of the preceding layer. The nonlinearity arises from the application of the activation function (a sigmoidal monotonically increasing function) to the input of each neuron in order to obtain its output. Hence, if the outputs of two consecutive layers are known then the optimal weights connecting the two layers can be derived by minimising the Mean Square Error (MSE) between the actual and the desired input to the second layer. The actual inputs to the second layer

are a linear function of the outputs of the first layer, while the desired inputs of the second layer are the values obtained by applying the inverse activation function to its outputs. Furthermore, if the weights of a given layer and the desired outputs of the succeeding layer are known then the "best" in the least squares sense neuron outputs of the preceding layer can be computed. Konig and Barmann [27] where the first to suggest this approach for training multilayer perceptron neural networks. In their approach the weights were randomly initialised and the outputs for each layer were computed. Then, starting from the output layer and considering preceding layers, the "best" weights and required outputs of the preceding layers were obtained using least squares.

One problem with this approach is that it does not take into consideration the scaling effect of the nonlinear activation function. This means that if the MSE between the actual and the desired input is small, the MSE between the actual and the desired output may not be necessarily small. Attempts to improve this deficiency include approximating the activation function with a linear combination of convex functions [35], considering the slope of the activation function at the desired output values to achieve better scaling [49] and restricting the output neuron values in the non-saturation region of the activation function either directly [181] or indirectly by obtaining adaptive threshold values for the network weights [180].

Least squares have also been considered in hybrid algorithms. One approach is to obtain the weights of all layers by standard backpropagation algorithms, apart from the output layer where least squares are used to exploit the desired output values from the training data [52, 96]. Other hybrid algorithms have sophisticated iterative methods for choosing the desired weights or output values of the non-output neurons such as penalised functions [164] and sensitivity analysis [36], but employ least squares to optimise the performance of the network for given values of those parameters.

In terms of performance, the aforementioned studies have illustrated that least squares supervised learning approached are very efficient techniques that can obtain smaller training errors and better results much faster than backpropagation techniques; also they can boost the convergence of backpropagation techniques when used as initialisation methods.

In the context of RNN, least squares have been utilised for the task of texture reconstruction where all neurons have desired values, while the neural network is

not fully recurrent as each neuron can interact only with its local neighbours and is partially symmetric with respect to its weights [16]. The author proposed an active-set algorithm based on the Kuhn-Tucker optimality conditions. However, this approach is only suitable for the particular problem; the general problem of NNLS learning in RNN has not been considered, while the applicability of the approach is limited to small-size problems.

To confront the general case that the network is fully recurrent, and comprised of not only output neurons there are two main difficulties: a) the formulated problem is of very large dimensionality (NK equations and $2N^2$ unknowns) and b) we must deal efficiently with the non-output neurons. In section 3.3, we propose a projected gradient algorithm for the NNLS problem suitable for large-scale problems as well as an approach to deal efficiently with networks involving non-output neurons. Our approach also differs from existing least squares techniques for connectionist neural networks because it is developed for a different neural network model and hence requires a different approximation approach; moreover, it is applied to a fully recurrent network with the least squares method applied to the whole network rather than on a layer-by-layer basis.

3.2. Gradient descent learning in the RNN with synchronised interactions

In this section we introduce the RNNSI model and derive the steps of its supervised gradient descent learning algorithm which is of computational complexity $O(N^3)$ for an N-neuron network.

3.2.1. Synchronised interactions in biological neural networks

Synchronised firing (SF), where several cells fire simultaneously, and neurons jointly act upon other cells, provide a richer form of inter-cellular interaction than the binary (excitatory-inhibitory) action between pairs of cells. SF has been observed among cultured cortical neurons [140, 152] and it is believed that it serves a prominent role in information processing functions of both sensory and motor systems [115]. Temporal firing synchrony may be a result of functional coupling which dy-

namically varies according to the internal state of the neural system and the stimuli, and it appears both in homogeneous and clustered neuronal networks [159]; it has been observed that under special population density conditions a neuronal culture can self-organize into linked clusters [158], to generate synchronous bursts of spikes similar to the one observed in homogeneous networks [133, 158]. This behaviour may also be related to the correlation in connectivity, which is usually measured in neuron cultures because it is difficult to identify the synaptic strength among neurons and hence determine the characteristic node connectivity [110]. Furthermore, studies on synchronised firing in the retina have indicated that pairwise interactions between spatially neighbouring neurons are sufficient to explain the spatial scale and structure of synchronised firing [161, 162, 160].

The random neural network with synchronised interactions can exhibit synchronised firing between cells, where one cell may trigger firing in another one. In fact, cascades of such triggered firings can occur in the model that we study. It appears that some experimental observations of synchronised firing in cultured or sliced neuron cell ensembles are in fact bursts of firing resulting from the nonlinear dynamics of the neuronal interactions. Our model describes the triggering of firing between two cells and also allows triggered firing by cascades of cells, and these cascades can also include feedback loops so that lengthy bursts of firing can also be modelled. Thus the present model can to a certain extent be used to mimic the spike bursts which have been experimentally observed.

3.2.2. RNNSI mathematical model

The RNNSI model exhibits not only the ordinary excitatory and inhibitory interactions of RNN but also synchronised interactions. When neuron i fires the resulting spike can travel to neuron j as an excitatory spike with probability $p^+(i,j)$ or as an inhibitory spike with probability $p^-(i,j)$, or the spike departs the network going to the outside world with probability $d(i)$, or it creates a *synchronous interaction* together with neuron j to affect some third neuron m, with probability $Q(i,j,m)$. When a synchronous interaction from neurons i and j on neuron m takes place at time t then the following happen: of course $k_i(t^+) = k_i(t) - 1$, but also $k_j(t^+) = k_j(t) - 1$ and $k_m(t^+) = k_m(t) + 1$ if $k_j(t) > 0$. However if $k_j(t) = 0$ then the only thing that will occur is that $k_i(t^+) = k_i(t) - 1$, and the firing of i will have no other effect. Thus, synchronous interactions take the form of a *joint second order*

excitation by cells i, j on m only if both neurons i and j are excited. Note also that:

$$\sum_{j=1}^{N}\left[p^{+}(i,j)+p^{-}(i,j)+\sum_{m=1}^{N}Q(i,j,m)\right]+d(i)=1 \tag{3.1}$$

An important feature of this network is that synchronised interactions can generalise to an arbitrary number of neurons so that the model can capture some quite general forms of synchronised firing. Indeed, if we have a sequence of neurons $j_1,\ \ldots\ , j_{n+1}, j_{n+2}$ such that $Q(j_l, j_{l+1}, j_{l+2}) = 1$ for $1 \leq l \leq n$, then if neurons j_1 and j_2 are excited, then eventually all the neurons $j_1,\ \ldots\ , j_{n+1}, j_{n+2}$ will fire.

3.2.3. Steady-state solution

Let the state of the network be $\mathbf{k}(t) = [k_1(t),\ k_2(t),\ ...,\ k_N(t)]$. With the previous assumptions, the system state is a continuous time Markov chain, and the probability distribution of the system state $\{\mathbf{k}(t) : t \geq 0\}$ satisfies a set of Chapman-Kolmogorov equations. Let us use the following vectors to denote specific values of the network state, where all of these vectors' values must be non-negative:

$$\begin{aligned}
\mathbf{k} &= [k_1,\ ...,\ k_N] \\
\mathbf{k}_i^{+} &= [k_1, ...,\ k_i + 1,\ ...,\ k_N] \\
\mathbf{k}_i^{-} &= [k_1, ...,\ k_i - 1,\ ...,\ k_N] \\
\mathbf{k}_{ij}^{+-} &= [k_1, ...,\ k_i + 1, ...,\ k_j - 1,\ ...,\ k_N] \\
\mathbf{k}_{ij}^{++} &= [k_1, ...,\ k_i + 1, ...,\ k_j + 1,\ ...,\ k_N] \\
\mathbf{k}_{ijm}^{++-} &= [k_1, ...,\ k_i + 1, ...,\ k_j + 1, ...,\ k_m - 1,\ ...,\ k_N]
\end{aligned}$$

If the steady-state distribution $\pi(\mathbf{k}) = \lim_{t\to\infty} P\left[\mathbf{k}(t) = \mathbf{k}\right]$ exists, it satisfy the Chapman-Kolmogorov equations given in steady-state:

$$\pi\left(\mathbf{k}\right)\sum_{i=1}^{N}\left[\Lambda(i)+(\lambda(i)+r_i)\mathbf{1}_{\{k_i>0\}}\right]=$$

$$\sum_{i=1}^{N}\Bigg\{\pi\left(\mathbf{k}_i^{+}\right)r_i d(i)+\pi\left(\mathbf{k}_i^{-}\right)\Lambda(i)\mathbf{1}_{\{k_i>0\}}+\pi\left(\mathbf{k}_i^{+}\right)\lambda(i)$$

$$+\sum_{j=1}^{N}\Bigg[\pi\left(\mathbf{k}_{ij}^{+-}\right)r_i p^{+}(i,j)\mathbf{1}_{\{k_j>0\}}+\sum_{m=1}^{N}\pi\left(\mathbf{k}_i^{+}\right)r_i Q(i,j,m)\mathbf{1}_{\{k_j=0\}}$$

$$+\pi\left(\mathbf{k}_{ij}^{++}\right)r_i p^{-}(i,j)+\pi\left(\mathbf{k}_i^{+}\right)r_i p^{-}(i,j)\mathbf{1}_{\{k_j=0\}}$$

$$+\sum_{m=1}^{N}\pi\left(\mathbf{k}_{ijm}^{++-}\right)r_i Q(i,j,m)\mathbf{1}_{\{k_m>0\}}\Bigg]\Bigg\} \quad (3.2)$$

where $\mathbf{1}_{\{Y\}}$ is equal to 1 if Y is true and 0 otherwise.
The following theorem can be proven by following a similar procedure to that used for the proof of Theorem 1 in [59].

Theorem: *Let $\lambda^{-}(i)$ and $\lambda^{+}(i)$, $i=1,...N$ be given by the following system of equations*

$$\lambda^{+}(i)=\sum_{j=1}^{N}r_j q_j p^{+}(j,i)+\sum_{j=1}^{N}\sum_{m=1}^{N}q_j q_m r_j Q(j,m,i)+\Lambda(i) \quad (3.3)$$

$$\lambda^{-}(i)=\lambda(i)+\sum_{j=1}^{N}r_j q_j[p^{-}(j,i)+\sum_{m=1}^{N}Q(j,i,m)] \quad (3.4)$$

where

$$q_i=\lambda^{+}(i)/\left(r_i+\lambda^{-}(i)\right) \quad (3.5)$$

If a unique non-negative solution $\{\lambda^{-}(i),\ \lambda^{+}(i)\}$ exists for the non-linear system of equations (3.3), (3.4), (3.5) such that $q_i<1\ \forall i$,then:

$$\pi\left(\mathbf{k}\right)=\prod_{i=1}^{N}\left(1-q_i\right)q_i^{k_i} \quad (3.6)$$

Thus whenever a solution can be found to equations (3.3), (3.4), (3.5) such that all the $q_i<1$, then the network's steady-state has the simple product form (3.6).

The condition $q_i < 1$ can be viewed as a "stability condition" which guarantees that the excitation level of each neuron remains finite with probability one. Note also, that the average excitation level of neuron i in steady-state is $q_i/(1-q_i)$.

We will now introduce a notation which is similar to the one used in [60], where we replace the firing rates r_i and the probabilities $p^+(i,j),\ p^-(i,j)$ and $Q(i,j,l)$ by "weights", which in this model represent the *rates at which the neurons interact*. Let:

$$w^+(i,j) = r_i p^+(i,j), \tag{3.7}$$

$$w^-(i,j) = r_i p^-(i,j), \tag{3.8}$$

and

$$w(i,j,l) = r_i Q(i,j,l) \tag{3.9}$$

As a result we can write:

$$r_i = \frac{\sum_{j=1}^{N} \left[w^+(i,j) + w^-(i,j) + \sum_{m=1}^{N} w(i,j,m) \right]}{1 - d(i)} \tag{3.10}$$

The denominator of q_i can be written as:

$$D(i) = r_i + \sum_{j=1}^{N} q_j [w^-(j,i) + \sum_{m=1}^{N} w(j,i,m)] + \lambda(i) \tag{3.11}$$

while its numerator becomes:

$$N(i) = \sum_{j=1}^{N} q_j w^+(j,i) + \sum_{j=1}^{N} \sum_{m=1}^{N} q_j q_m w(j,m,i) + \Lambda(i) \tag{3.12}$$

so that $q_i = N(i)/D(i)$. The results summarised in this section will now be used to design an efficient learning algorithm for this network with second order effects.

3.2.4. RNNSI gradient descent supervised learning

In order to perform gradient descent learning with the RNNSI we need to update the weights of the network $w^+(i,j)$, $w^-(i,j)$ and $w(i,j,l)$ according to the gradient descent rule and the procedure introduced in section 2.4.1.

In general we can select the $w(i,j,l)$ in an arbitrary manner as long as $w(i,j,l) \geq$

0, and $w^+(i,j),\ w^-(i,j)\ \geq 0$. However, we see that $w(i,j,l)$ in fact acts as an inhibitory term from i to j, followed by an excitatory term from j to l. Thus we propose to simplify the computation involved in seeking a minimum of the error function by writing:

$$w(i,j,l) = w^-(i,j)a(j,l), \forall i,j,l \tag{3.13}$$

where $a(j,l) \geq 0$.

We will therefore design a gradient descent algorithm to obtain the unknown parameters of the network i.e. the matrices $\mathbf{W}^+ = \{w^+(i,j)\}, \mathbf{W}^- = \{w^-(i,j)\}$ and $\mathbf{A} = \{a(i,j)\}$ for $i,j = 1,...,N$ in order to minimise the cost function. In the sequel we will use the generic term $w(u,v)$ to represent either $w(u,v) \equiv w^+(u,v)$ or $w(u,v) \equiv w^-(u,v)$ or $w(u,v) \equiv a(u,v)$.

The weights are updated based on the gradient descent rule using Eqs. (2.15) and (2.16). As noted in section 2.4.1, the difficult step is the derivation of $[\partial q_i/\partial w(u,v)]_\tau$, which in the RNNSI model is even more challenging as the expressions involve second-order terms with respect to q_i.

By taking the derivatives of $q_i = N(i)/D(i)$ with respect to the generic variable $w(u,v)$ one obtains after some calculations each of the terms of interest. The detailed derivation of the expressions given below can be found in Appendix A. Writing the vector $\mathbf{q} = [q_1, q_2, \ ... \ , \ q_N]$ and using matrix $\mathbf{W} \in \mathbb{R}^{N\times N}$ with elements:

$$\begin{aligned} W(i,j) \ = \ & \frac{1}{D(j)} \cdot \Bigg\{ w^+(i,j) - q_j w^-(i,j) + \sum_{m=1}^{N} q_m w^-(i,m) a(m,j) \\ & + a(i,j) \sum_{m=1}^{N} q_m w^-(m,i) - q_j w^-(i,j) \sum_{m=1}^{N} a(j,m) \Bigg\} \ \forall i,j \end{aligned} \tag{3.14}$$

we obtain:

$$\frac{\partial \mathbf{q}}{\partial w^+(u,v)} = \frac{\partial \mathbf{q}}{\partial w^+(u,v)} \mathbf{W} + \boldsymbol{\gamma}^+(u,v) \tag{3.15}$$

$$\frac{\partial \mathbf{q}}{\partial w^-(u,v)} = \frac{\partial \mathbf{q}}{\partial w^-(u,v)} \mathbf{W} + \boldsymbol{\gamma}^-(u,v) \tag{3.16}$$

$$\frac{\partial \mathbf{q}}{\partial a(u,v)} = \frac{\partial \mathbf{q}}{\partial a(u,v)} \mathbf{W} + \boldsymbol{\gamma}^a(u,v) \tag{3.17}$$

where we have used:

$$\boldsymbol{\gamma}^+(u,v) = [\gamma_1^+(u,v), \gamma_2^+(u,v), ..., \gamma_N^+(u,v)], \tag{3.18}$$

$$\boldsymbol{\gamma}^{-}(u,v) = [\gamma_1^{-}(u,v), \gamma_2^{-}(u,v), ..., \gamma_N^{-}(u,v)], \tag{3.19}$$

and

$$\boldsymbol{\gamma}^{a}(u,v) = [\gamma_1^{a}(u,v), \gamma_2^{a}(u,v), ..., \gamma_N^{a}(u,v)], \tag{3.20}$$

The above parameters are given by the following equations:

$$\gamma_i^{+}(u,v) = \frac{1}{D(i)} \cdot \begin{cases} q_u - q_u/(1-d(i)) & u = i,\ v = i \\ -q_u/(1-d(i)) & u = i,\ v \neq i \\ q_u & u \neq i,\ v = i \\ 0 & u \neq i,\ v \neq i \end{cases} \tag{3.21}$$

$$\gamma_i^{-}(u,v) = \frac{1}{D(i)} \cdot \begin{cases} q_u q_v [a(v,i) - 1 - \sum_{m=1}^{N} a(v,m)] & \\ \quad - q_u [1 + \sum_{m=1}^{N} a(v,m)](1-d(i))^{-1} & v = i,\ u = i \\ q_u q_v [a(v,i) - 1 - \sum_{m=1}^{N} a(v,m)] & v = i,\ u \neq i \\ q_u q_v a(v,i) - q_u [1 + \sum_{m=1}^{N} a(v,m)](1-d(i))^{-1} & v \neq i,\ u = i \\ q_u q_v a(v,i) & v \neq i,\ u \neq i \end{cases} \tag{3.22}$$

$$\gamma_i^{a}(u,v) = \frac{1}{D(i)} \cdot \begin{cases} -q_i w^{-}(i,u)(1-d(i))^{-1} & v = i,\ u = i \\ -q_i w^{-}(i,u)(1-d(i))^{-1} + q_u \sum_{j=1}^{N} q_j w^{-}(j,u) & v = i,\ u \neq i \\ -q_i w^{-}(i,u)(1-d(i))^{-1} - q_u \sum_{j=1}^{N} q_j w^{-}(j,u) & v \neq i,\ u = i \\ -q_i w^{-}(i,u)(1-d(i))^{-1} & v \neq i,\ u \neq i \end{cases} \tag{3.23}$$

Notice that (3.15) - (3.17) can also be written as:

$$\frac{\partial \mathbf{q}}{\partial w^{+}(u,v)} = \boldsymbol{\gamma}^{+}(u,v)\,(\mathbf{I} - \mathbf{W})^{-1} \tag{3.24}$$

$$\frac{\partial \mathbf{q}}{\partial w^{-}(u,v)} = \boldsymbol{\gamma}^{-}(u,v)\,(\mathbf{I} - \mathbf{W})^{-1} \tag{3.25}$$

$$\frac{\partial \mathbf{q}}{\partial a(u,v)} = \boldsymbol{\gamma}^{a}(u,v)\,(\mathbf{I} - \mathbf{W})^{-1} \tag{3.26}$$

where $\mathbf{I}$ is the $N \times N$ identity matrix. We now summarise the steps of the learning algorithm:

(1) Initialise the matrices $\mathbf{W}^+$, $\mathbf{W}^-$ and $\mathbf{A}$ and choose a value for η.

(2) For each input-output pattern k do:

(a) Set appropriate values for the inputs $[\mathbf{\Lambda}_k, \boldsymbol{\lambda}_k]^T = \mathbf{x}_k$ and desired outputs $\mathbf{y}_k$ for the particular pattern.

(b) Solve the system of the N non-linear equations (3.3)-(3.5) based on the above values.

(c) Based on the values attained, calculate $\mathbf{W}$, $\boldsymbol{\gamma}^+(u,v)$, $\boldsymbol{\gamma}^-(u,v)$ and $\boldsymbol{\gamma}^a(u,v)$, $\forall u, v$.

(d) Using the values obtained from Steps (2a)-(2c), solve the three systems of the N linear equations (3.24), (3.25) and (3.26) for all values of u and v.

(e) Using the results from the previous steps, update the matrices $\mathbf{W}^+ = \{w^+(i,j)\}$, $\mathbf{W}^- = \{w^-(i,j)\}$ and $\mathbf{A} = \{a(i,j)\}$ using (2.15) and (2.16). Set any negative weights to zero.

(3) Repeat the procedure of step (2) until a stopping criterion is met.

3.2.5. Computational complexity

To examine the computational complexity of the RNNSI gradient descent algorithm we need to consider the costly steps of the algorithm. In this way, we can establish the complexity of computing one or all weights in one iteration of the algorithm. Specifically, we will investigate the computational complexity of the following steps:

1. Solution of the RNNSI signal-flow equations (3.3)-(3.5).
2. Derivation of $\mathbf{W}$ based on Eq. (3.14).
3. Computation of the terms $\gamma^+(u,v)$, $\gamma^-(u,v)$ and $\gamma^a(u,v)$.
4. Evaluation of the terms $\partial\mathbf{q}/\partial w(u,v)$.

Let us start our examination with the RNNSI signal-flow equations. Similar to the RNN case, the number of iterations required to compute $q_i, \forall i$ is small so that the total complexity of this calculation depends on the complexity of one iteration. As $w(i,j,l) = w^-(i,j)a(j,l)$ Eqs. (3.3) and (3.4) become:

$$\lambda^+(i) = \Lambda(i) + \sum_{j=1}^{N} q_j w^+(j,i) + \sum_{j=1}^{N}\sum_{m=1}^{N} q_j q_m w^-(j,m) a(m,i)$$

$$\lambda^-(i) = \lambda(i) + \sum_{j=1}^{N} q_j w^-(j,i) + \sum_{j=1}^{N}\sum_{m=1}^{N} q_j w^-(j,i) a(i,m)$$

If we follow the direct way of updating $\lambda^+(i)$ and $\lambda^-(i)$ then each of these terms requires $O(N^2)$ operations due to the presence of $\sum_j \sum_m[\cdot]$ terms. Hence, as we have N such terms the computational complexity of one iteration is $O(N^3)$. However, we can reduce the complexity of these calculations by an order of magnitude, by rewriting the terms $\lambda^+(i)$ and $\lambda^-(i)$ as:

$$\begin{aligned}\lambda^+(i) &= \Lambda(i) + \sum_{j=1}^{N} q_j w^+(j,i) + \sum_{m=1}^{N} q_m a(m,i) \sum_{j=1}^{N} q_j w^-(j,m) \\ &= \Lambda(i) + \sum_{j=1}^{N} q_j w^+(j,i) + \sum_{m=1}^{N} q_m a(m,i)\sigma_1(m) \end{aligned} \tag{3.27}$$

$$\begin{aligned}\lambda^-(i) &= \lambda(i) + \sum_{j=1}^{N} q_j w^-(j,i) + \sum_{j=1}^{N} q_j w^-(j,i) \sum_{m=1}^{N} a(i,m) \\ &= \lambda(i) + \sigma_1(i)(1+\sigma_2(i)) \end{aligned} \tag{3.28}$$

where $\sigma_1(u)$ and $\sigma_2(u)$ are given by:

$$\sigma_1(u) = \sum_{j=1}^{N} q_j w^-(j,u),\ u = 1, ..., N$$

$$\sigma_2(u) = \sum_{j=1}^{N} a(u,j),\ u = 1, ..., N$$

Note that the terms $\sigma_2(u)$ need to be computed once at the beginning of the algorithm execution, while the terms $\sigma_1(u)$, $u = 1, ..., N$ need to be computed at the start of each iteration and require $O(N^2)$ operations in total. Having derived $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ the evaluation of $\boldsymbol{\lambda}^+$ and $\boldsymbol{\lambda}^-$ requires $O(N^2)$ operations per iteration as well. Hence, the total complexity of computing $q_i, \forall i$ is $O(N^2)$ similar to the RNN

case.

The derivation of matrix $\mathbf{W}$ is of computational complexity $O(N^3)$ as it requires the evaluation of N^2 terms of the form $\sum_{m=1}^{N} q_m w^-(i,m)a(m,j)$, $\forall i,j$ each needing $O(N)$ operations. Furthermore, the most computational demanding terms amongst $\gamma_i(u,v)$ are the $\gamma_i^-(u,v)$. By setting $\sigma_2(v) = \sum_{m=1}^{N} a(v,m)$ in $\gamma_i^-(u,v)$, we can obtain these terms in $O(N^3)$ time, as each term requires $O(1)$ operations and there are N^3 such terms.

Finally, the evaluation of $\partial\mathbf{q}/\partial w(u,v)$ requires the derivation of $\mathbf{V} = (\mathbf{I} - \mathbf{W})^{-1}$ which is of computational complexity $O(N^3)$ or $O(mN^2)$ if a relaxation method with m iterations is followed. Note that matrix $\mathbf{V}$ needs to be computed only once. Additionally, to derive a $\partial\mathbf{q}/\partial w(u,v)$ term the matrix-vector product $\boldsymbol{\gamma}(u,v)\mathbf{V}$ needs to be computed that takes $O(N^2)$ operations. Consequently, the complexity of updating one weight $w(u,v)$ is equal to $O(N^3)$, due to the derivation of the terms $\mathbf{W}$, $\mathbf{V}$ and $\boldsymbol{\gamma}^-(u,v)$, while the total complexity of one iteration of the algorithm is $O(N^4)$ as there are $3N^2$ terms $\partial\mathbf{q}/\partial w(u,v)$ that need to be computed. However, the total time complexity of one iteration can be reduced to $O(N^3)$ as explained below.

Let us assume that $a_i = \{\bar{c}_i(g_i(q_{ik}) - y_{ik})\frac{\partial g_i(q_i)}{\partial q_i}\}$, $\forall i$, where a_i is the i-th element of vector $\mathbf{a}$ at the kth iteration. Substituting Eq. (2.17) or Eq. (2.18) into Eq. (2.16) yields:

$$\begin{aligned}
\frac{\partial E_k}{\partial w(u,v)} &= \mathbf{a}\left[\frac{\partial \mathbf{q}}{\partial w(u,v)}\right]^T \\
&= \mathbf{a}\left[\boldsymbol{\gamma}(u,v)(\mathbf{I}-\mathbf{W})^{-1}\right]^T \\
&= \mathbf{a}\mathbf{V}^T\boldsymbol{\gamma}^T(u,v) \\
&= \mathbf{a}_V\boldsymbol{\gamma}^T(u,v),\ \forall\ u,v
\end{aligned} \tag{3.29}$$

where $\mathbf{a}_V = \mathbf{a}\mathbf{V}^T$ and $\boldsymbol{\gamma}(u,v)$ is used to represent $\boldsymbol{\gamma}^+(u,v)$ or $\boldsymbol{\gamma}^-(u,v)$ or $\boldsymbol{\gamma}^a(u,v)$. In order to update $w(u,v)$, $\forall u,v$ it is required to compute once matrix $\mathbf{V}$ and vectors $\mathbf{a}$ and $\mathbf{a}_V$. According to Eq. (3.29) the complexity of computing $\frac{\partial E_k}{\partial w(u,v)}$ is $O(N)$ so that the complexity of computing all N^2 weights is $O(N^3)$. We can further reduce the complexity of each $\frac{\partial E_k}{\partial w(u,v)}$ term to $O(1)$, by substituting Eqs. (3.21)-(3.23) into

the generic Eq. (3.29) to obtain:

$$\frac{\partial E_k}{\partial w^+(u,v)} = \sum_{i=1}^{N} a_V(i)\gamma_i^+(u,v) = q_u \frac{a_V(v)}{D(v)} - q_u \frac{a_V(u)}{D(u)}(1-d(u))^{-1} \tag{3.30}$$

$$\begin{aligned} \frac{\partial E_k}{\partial w^-(u,v)} &= \sum\nolimits_{i=1}^{N} a_V(i)\gamma_i^-(u,v) \\ &= q_u\sigma_3(v) - \frac{a_V(u)}{D(u)} q_u(1+\sigma_2(v))(1-d(u))^{-1} \\ &\quad -q_u \frac{a_V(v)}{D(v)} q_v(1+\sigma_2(v))(1-d(v))^{-1} \end{aligned} \tag{3.31}$$

$$\frac{\partial E_k}{\partial a(u,v)} = \sum_{i=1}^{N} a_V(i)\gamma^a(u,v) = -\sigma_4(u) - q_u \frac{a_V(u)}{D(u)}\sigma_1(u) + q_u \frac{a_V(v)}{D(v)}\sigma_1(u) \tag{3.32}$$

where the sum terms $\sigma_1(u)$ and $\sigma_2(u)$ have already been defined, while $\sigma_3(u)$ and $\sigma_4(u)$, $u = 1, ..., N$ are given by:

$$\sigma_3(u) = q_u \sum_{j=1}^{N} \frac{a_V(j)}{D(j)} a(u,j),\ u = 1, ..., N$$

$$\sigma_4(u) = \sum_{j=1}^{N} \frac{a_V(j)}{D(j)} q_j w^-(j,u)(1-d(j))^{-1},\ u = 1, ..., N$$

We can now outline an improved version of the RNNSI algorithm:

(1) Initialise the matrices $\mathbf{W}^+$, $\mathbf{W}^-$ and $\mathbf{A}$ and choose a value for η.

(2) For each input-output pattern k do:

- **(a)** Set appropriate values for the inputs $[\mathbf{\Lambda}_k, \boldsymbol{\lambda}_k]^T = \mathbf{X}_k$ and desired outputs $\mathbf{y}_k$ for the particular pattern.
- **(b)** Solve the system of the N non-linear equations (3.3)-(3.5) based on the above values.
- **(c)** Compute matrix $\mathbf{W}$ from Eq. (3.14) and derive $\mathbf{a}_V = \mathbf{aV} = \mathbf{a}(\mathbf{I} - \mathbf{W})^{-1}$.
- **(d)** Based on the values obtained, calculate the terms $\sigma_i(u)$ for $i = 1, 2, 3, 4$ and $u = 1, ..., N$.

(e) Derive the terms $\frac{\partial E_k}{\partial w^+(u,v)}$, $\frac{\partial E_k}{\partial w^-(u,v)}$ and $\frac{\partial E_k}{\partial a(u,v)}$, $\forall u, v$ according to expressions (3.30),(3.31) and (3.32) respectively.

(f) Using the results from the previous step, update the matrices $\mathbf{W}^+ = \{w^+(i,j)\}$, $\mathbf{W}^- = \{w^-(i,j)\}$ and $\mathbf{A} = \{a(i,j)\}$ using Eq. (2.15). Set all negative weights to zero.

(3) Repeat the procedure of step (2) until a stopping criterion is met.

In terms of computational complexity the computation of the terms $\sigma_i(u)$ for $i = 1, 2, 3, 4$ and $u = 1, ..., N$ are of $O(N)$ complexity, so that all these terms can be computed with $O(N^2)$ operations. With respect to the derivation of Eqs. (3.30)-(3.32), we have not only achieved to reduce the complexity of evaluating $\frac{\partial E_k}{\partial w(u,v)}$, $\forall u, v$ by one order of magnitude, but also made the evaluation of $\gamma_i(u,v)$ redundant. The reduction in the computational complexity by $O(N)$ has been achieved by the replacement of steps (2c)-(2d) of the original RNNSI learning algorithm with steps (2c)-(2e) of the improved one.

Taking everything into account, the complexity of one step of the algorithm is $O(N^3)$, due to the derivation of $\mathbf{W}$ and $\mathbf{V}$. The overall complexity of the algorithm is $N_{GD-RNNSI} \times O(KN^3)$, where $N_{GD-RNNSI}$ is the number of times that all patterns are processed until a stopping criterion is met. The procedure described above can also be extended to the gradient descent algorithm of RNN to reduce its computational complexity as well.

3.3. RNN supervised learning using nonnegative least squares

In this section, we propose a novel supervised learning algorithm and weight initialisation method for the RNN. Firstly, we show how to approximate the supervised learning problem in RNN to produce a *Non-Negative Least Squares* (NNLS) formulation, when all the neurons of the network have known desired q_{ik} values. The weights obtained from the solution of the NNLS problem can either be considered as the learnt weights or as good initial values for the standard supervised learning RNN algorithm. Then, we describe an improved version of a projected gradient algorithm for the solution of the NNLS problem as well as an iterative algorithm

that has been developed to deal with the case that not all neurons have desired values. We also derive efficient expressions for the function and gradient evaluation of the NNLS formulation, which are a result of the special structure of the NNLS problem in the RNN case and discuss the computational complexity of the proposed approach.

3.3.1. Problem formulation

In supervised learning an optimal set of weights must be found, such that the error associated with the observed and desired output is minimized. As already mentioned the observed output in RNN is associated with parameters q_{ik}, $i \in I_{out}$ and $k = 1, ..., K$. Ideally we would like to observe the desired output $\mathbf{y}_k$ for all patterns. This means, that for all patterns we should have: $q_{ik} = g_i^{-1}(y_{ik})$, $i \in I_{out}$, where $g_i^{-1}(\cdot)$ is the inverse function of $g_i(\cdot)$ and I_{out} is the set of output neurons. This is achieved, if the q_{ik} values for all the patterns and for i belonging to the set of indices of non-output neurons $I_{\overline{out}}$, as well as the weights are appropriately selected. Without loss of generality, in the sequel we assume that $g_i(q_{ik}) = q_{ik}$, so that $g_i^{-1}(y_{ik}) = y_{ik}$.

Combining Eqs. (2.6)-(2.8) we obtain:

$$q_{ik} = \min\left\{1, \frac{\lambda^+(i,k)}{r_i + \lambda^-(i,k)}\right\} = \min\left\{1, \frac{\Lambda_{ik} + \sum_{j=1}^{N} q_{jk} w^+(j,i)}{r_i + \lambda_{ik} + \sum_{j=1}^{N} q_{jk} w^-(j,i)}\right\} \ \forall i,k \quad (3.33)$$

If we further assume that $\lambda^+(i,k) < r_i + \lambda^-(i,k)\ \forall i,k$ and also substitute Eq. (2.4) into Eq. (3.33) we obtain:

$$\begin{aligned} & q_{ik}(1-d(i))^{-1}\sum_{j=1}^{N}(w^-(i,j)+w^+(i,j)) \\ & +q_{ik}\sum_{j=1}^{N} q_{jk} w^-(j,i) - \sum_{j=1}^{N} q_{jk} w^+(j,i) = \Lambda_{ik} - q_{ik}\lambda_{ik},\ \forall i,\ k \end{aligned} \quad (3.34)$$

If the network is only composed of output neurons, and if we assume that $q_{ik} = y_{ik},\ \forall i,k$, then Eq. (3.34) becomes a linear system of NK equations with $2N^2$ nonnegative unknowns, the weights $w^+(i,j)$ and $w^-(i,j)$. If there are both output and non-output neurons then by selecting appropriate values for the excitation

probabilities of the latter we can still obtain a linear system, as discussed in Section 3.3.2.

However, an accurate solution to Eq. (3.34) may not be available for two reasons. First, the number of equations may be larger that the number of unknowns; this is true when $K > 2N$. Second, the nonnegativity constraints restrict the values of the variables and a solution may not exist even if $K < 2N$. As a result we formulate Eq. (3.34) as an NNLS problem in order to approach equality as much as possible in the least square sense.

$$\begin{aligned} \min_{\mathbf{w} \geq 0} f(\mathbf{w}) \;&=\; \frac{1}{2}\|\mathbf{B}\mathbf{w} - \mathbf{b}\|_2^2, \\ \mathbf{B} \in \mathbb{R}^{NK \times 2N^2},\; \mathbf{b} \in \mathbb{R}^{NK \times 1},\; \mathbf{w} &\in \mathbb{R}^{2N^2 \times 1} \end{aligned} \tag{3.35}$$

NNLS is a convex quadratic optimisation problem [119] that can be solved to optimality using various methods as will be explained in the next section. The gradient of the objective function is given by:

$$\nabla f(\mathbf{w}) \;=\; \mathbf{B}^T\mathbf{B}\mathbf{w} - \mathbf{B}^T\mathbf{b} \tag{3.36}$$

In order to improve the generalisation ability of the model we can also introduce regularisation. Specifically we consider two regularisation terms: (a) the squared $l2$-norm of the weights, $|\mathbf{w}|_2^2 = \mathbf{w}^T\mathbf{w} = \sum_i w_i^2$, and (b) the $l1$-norm of the weights, $|\mathbf{w}|_1 = \sum_i |w_i|$. The former, improves the numerical stability of $\mathbf{B}^T\mathbf{B}$ and restricts the weights in taking large values that improves the generalisation ability of the network by avoiding overfitting. The latter, reduces the number of nonzero weights in the solution, but can also perform better compared to $l2$-norm regularisation in some cases [185]. Because in our case the weights are nonnegative, the regularisation term $|\mathbf{w}|_1$ is equal to $\mathbf{1}^T\mathbf{w} = \sum_i w_i$. Adding the two regularisation terms with appropriate regularisation weights to the NNLS problem yields:

$$\begin{aligned} \min_{\mathbf{w} \geq \mathbf{0}} f_{reg}(\mathbf{w}) \quad &= \tfrac{1}{2}(\mathbf{B}\mathbf{w} - \mathbf{b})^T(\mathbf{B}\mathbf{w} - \mathbf{b}) + \theta_1|\mathbf{w}|_1 + \theta_2|\mathbf{w}|_2^2 \\ &= \tfrac{1}{2}(\mathbf{B}\mathbf{w} - \mathbf{b})^T(\mathbf{B}\mathbf{w} - \mathbf{b}) + \theta_1\mathbf{1}^T\mathbf{w} + \tfrac{1}{2}\theta_2\mathbf{w}^T\mathbf{w} \end{aligned} \tag{3.37}$$

The gradient of the regularised NNLS problem $\nabla f_{reg}(\mathbf{w})$ is given by:

$$\nabla f_{reg}(\mathbf{w}) = \mathbf{B}^T(\mathbf{B}\mathbf{w} - \mathbf{b}) + \theta_1\mathbf{1} + \theta_2\mathbf{w} \tag{3.38}$$

Notice that the computational complexity of the objective and gradient NNLS functions are not affected by the introduction of the regularisation terms.

The ik row of matrix $\mathbf{B}$ and vector $\mathbf{b}$ in Eqs. (3.35) and (3.37), which correspond to the ith signal flow equation of the kth pattern, are given by the following expressions:

$$\begin{aligned}
B(ik, ij^+) &= q_{ik}(1-d(i))^{-1}, && \forall\ j \neq i \\
B(ik, ij^-) &= q_{ik}(1-d(i))^{-1}, && \forall\ j \neq i \\
B(ik, ji^-) &= q_{ik}q_{jk}, && \forall\ j \neq i \\
B(ik, ji^+) &= -q_{jk}, && \forall\ j \neq i \\
B(ik, ii^+) &= q_{ik}(1-d(i))^{-1} - q_{ik}, && j = i \\
B(ik, ii^-) &= q_{ik}(1-d(i))^{-1} + q_{ik}^2, && j = i \\
B(ik, otherwise) &= 0, &&
\end{aligned} \tag{3.39}$$

$$b(ik) = \Lambda_{ik} - q_{ik}\lambda_{ik}, \quad \forall\ i, k \tag{3.40}$$

The column indices of $\mathbf{B}$, ij^+ and ij^-, indicate the position of the variables $w^+(i,j)$ and $w^-(i,j)$ in $\mathbf{w}$ respectively. Notice that every value of $\mathbf{B}$ can be found by only using matrix $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_k, ..., \mathbf{q}_K]$, $\mathbf{Q} \in \mathbb{R}^{N \times K}$, which holds the q_{ik} values of both output and non-output neurons; the $d(i)$ values are usually constant and for simplicity we assume that $d(i) = 0, \forall i$. Also, despite the fact that every row of matrix $\mathbf{B}$ has $2N^2$ elements, only $4N$ of them are nonzero and hence the density of nonzero elements in $\mathbf{B}$ is $2/N$.

One difficulty associated with the above formulations is the large dimensionality of $\mathbf{B}$ which implies that it may not be possible to be stored in memory. For example, in Chapter 4 we consider supervised RNN problems with dimensions up to $N = 300$ and $K = 1000$, in which case matrix $\mathbf{B}$ has dimensions 300000×180000 so that it is impossible to be stored in memory. Moreover, initial experimentation showed that $\mathbf{B}$ is ill-conditioned. Therefore, we want to develop an approach for the solution of the NNLS problem that does not require either storing large matrices or performing matrix inversion operations. It is important that only simple operations are performed, such as matrix-vector products, avoiding inefficient matrix-matrix multiplications or matrix inversion operations. To achieve the requirements of the solution approach, it is also important to consider the sparseness of $\mathbf{B}$.

In the next section, we discuss the proposed solution approach both for the case

that the network is only composed of output neurons as well as for the case that the network is composed of output and non-output neurons.

3.3.2. Solution approach

Algorithms for the solution of the NNLS problem

The algorithms proposed in the literature for the solution of the NNLS problem can generally be classified into *active set algorithms* and *iterative approaches*.

In active set algorithms variables are divided into two sets: the *active set* and the *passive set*. A variable belongs to the active set if it is negative or zero at a particular iteration, otherwise it belongs to the passive set. When the unconstrained least squares problem is solved, negative or zero variables do not contribute to the constrained problem; therefore, if the active set corresponding to the optimal solution is known then the solution can be found by solving the unconstrained problem for the passive set of variables and setting the active variables equal to zero.

The most widely known active set algorithm is the one proposed by Lawson and Hanson [119]. In this approach, initially all the variables are inserted into the active set. Then an iterative procedure is followed where in each iteration variables that result in a strictly better evaluation of the cost function are identified and removed from the active set. The procedure continues until no more active variables can be freed to reduce further the cost function. Although it is possible to free many variables at a single iteration, general practice has shown that it is better to free from the active set only one variable at a time [45].

A modified version of this algorithm identifies calculations that can be computed beforehand to reduce the computational cost. The algorithm called FNNLS (Fast Nonnegative Least Squares) [29] speeds-up the procedure, but requires the storage of the square matrix $\mathbf{B}^T\mathbf{B}$ as well as matrix inversion operations. Active set methods are in general not appropriate in our case because they involve matrix inversion operations, which are undesirable due to memory limitations and high computational cost.

Iterative approaches adhere to nonlinear optimisation methods to update the decision vector at iteration τ, $\mathbf{w}^\tau$. Usually the update of the current solution is based on *projected gradient methods* which can identify several active set constraints in one

iteration. They generally require simple matrix vector operations and can perform well in ill-conditioned systems. In iteration τ of a projected gradient method, a search direction $\mathbf{d}^\tau$ and a step-size s^τ are appropriately selected; then to obtain $\mathbf{w}^{\tau+1}$ the projection operation is applied to ensure that the new point is within the feasible region as shown in Eqs. (3.41) and (3.42).

$$\mathbf{w}^{\tau+1} = P[\mathbf{w}^\tau - s^\tau \mathbf{d}^\tau] = P[\mathbf{w}^\tau - s^\tau \mathbf{S}^\tau \nabla f(\mathbf{w}^\tau)], \quad s^\tau \geq 0,\ \mathbf{S}^\tau \in \mathbb{R}^{2N^2 \times 2N^2} \tag{3.41}$$

$$P[w_i] = \begin{cases} w_i, & w_i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.42}$$

Projected gradient methods usually differ in the procedure used for the selection of the step-size s^τ and the update of the gradient scaling matrix $\mathbf{S}^\tau$, which must be symmetric and positive definite. With respect to the step-size selection, one of the most successful methods is the "Armijo rule along the projection arc" (APA) [25] which will be discussed later in this section. The selection of $\mathbf{S}^\tau$ is also important because it utilises second order gradient information and results in fast convergence to the solution. However, this selection involves a tradeoff between computational time, memory and convergence speed.

For the solution of the NNLS problem (or the closely related convex bound or box constrained quadratic programming problem) several schemes for updating the scaling matrix have been proposed. In [124] and [139], first order gradient projection methods that do not utilise the scaling matrix ($\mathbf{S}^\tau = \mathbf{I}$) were considered, resulting in short execution time per iteration but slow convergence. A gradient scaling matrix with only diagonal entries was proposed in [23], achieving convergence in fewer iterations compared to [139]. Despite the fact that diagonal scaling is better than no scaling, the use of non-diagonal $\mathbf{S}^\tau$ matrices results in faster convergence, but can be prohibitive in terms of memory and computation.

Newton methods consider the inverse of the Hessian matrix ($(\nabla^2 f(\mathbf{w}))^{-1}$) for updating the scaling matrix at each iteration, which is prohibitive both in terms of computation and storage. Quasi-Newton methods provide an attractive alternative by constructing an approximation of the Hessian or its inverse based on information provided by the change in the decision and gradient vectors $\Delta\mathbf{w}^{\tau-1} = \mathbf{w}^\tau - \mathbf{w}^{\tau-1}$ and $\Delta\mathbf{g}^{\tau-1} = \nabla f(\mathbf{w}^\tau) - \nabla f(\mathbf{w}^{\tau-1})$ at successive iterations. Nevertheless, quasi-Newton methods also require full storage of $\mathbf{S}^\tau$. Note that in order to guarantee

convergence for the solution of the NNLS problem, Newton [24] and quasi-Newton methods [111] construct a reduced gradient scaling matrix $\widetilde{\mathbf{S}}^{\tau}$ at each iteration which is the principal submatrix of $\mathbf{S}^{\tau}$ corresponding to the set of free variables (see Eq. (3.43)).

Limited-memory quasi-Newton methods offer a good trade-off between having a diagonal or a full scaling matrix by updating the search direction based on a positive definite approximation of $\mathbf{S}^{\tau}$ obtained by the difference vectors $\Delta\mathbf{w}^{\tau}$ and $\Delta\mathbf{g}^{\tau}$ of the last M iterations. Two quasi-Newton methods that can be applied for the solution of the NNLS problem are the L-BFGS-B [31] and PQN-SPG [157]. Each iteration of the L-BFGS-B involves three important steps: (a) a projected gradient line-search along $\mathbf{d} = \nabla f(\mathbf{w}^{\tau})$ on the quadratic model constructed by the limited-memory approximate Hessian matrix to obtain $\mathbf{w}_c^{\tau}$, (b) the approximate solution of the quadratic model over the set of free variables at $\mathbf{w}_c^{\tau}$ followed by projection of the free variables in the feasible region to obtain $\bar{\mathbf{w}}^{\tau+1}$, and (c) line-search along the feasible direction $\mathbf{d}_f = \bar{\mathbf{w}}^{\tau+1} - \mathbf{w}^{\tau}$ applied to the original objective function. The PQN-SPG method also constructs a quadratic model based on the limited-memory approximate Hessian matrix, but directly attempts to solve the constrained quadratic model using a Spectral Projected Gradient (SPG) method. Then it employs a line-search along the feasible direction for the original objective function. Notice that both methods do not consider projected gradient search along the direction $\mathbf{d} = \mathbf{S}^{\tau}\nabla f(\mathbf{w}^{\tau})$; they employ a line-search along the feasible direction for the original objective function. However, the APA line-search can lead to faster convergence than a line-search along the feasible direction because the iterates produced by the former are more likely to be at the boundary of the constraint set, resulting in the identification of the final active set in fewer iterations (p.228 in [25]).

The PGNNLS algorithm

We have developed a Projected Gradient NNLS (PGNNLS) algorithm based on updating the search-direction using a limited-memory BFGS formula and performing an APA line-search, which is outline in Algorithm 3.1. Our approach is a modified version of the quasi-Newton NNLS algorithm proposed in [111]; nevertheless it is different both in terms of the employed line-search (hyper-exponential instead of standard APA) and the procedure for updating the search direction (limited

memory instead of full BFGS).

The key aspect of Algorithm 3.1 is that at iteration τ we only perform a line-search for the variables that are in the *free-set* $\mathcal{F}^\tau$ defined as:

$$\mathcal{F}^\tau = \{i | w_i^\tau > 0 \text{ or } (w_i^\tau = 0 \text{ and } [\nabla f(\mathbf{w}^\tau)]_i \leq 0)\} \tag{3.43}$$

To understand the reason behind this, let us define the complement of $\mathcal{F}^\tau$, called the *binding set* $\mathcal{B}^\tau$:

$$\mathcal{B}^\tau = \{i | w_i^\tau = 0 \text{ and } [\nabla f(\mathbf{w}^\tau)]_i > 0\} \tag{3.44}$$

For the variables belonging to the binding set there are two possibilities about the search direction: (a) $d_i^\tau \geq 0$, and (b) $d_i^\tau < 0$. In the first case, we have that $x_i^{\tau+1}$ $= P[x_i^\tau - s^\tau d_i^\tau] = P[-s^\tau d_i^\tau] = 0$ so that this variable remains constant and does not affect the cost function. In the second case, we have that $x_i^{\tau+1} = P[x_i^\tau - s^\tau d_i^\tau] = P[-s^\tau d_i^\tau] = -s^\tau d_i^\tau > 0$; however, the fact that $-d_i^\tau [\nabla f(\mathbf{w}^\tau)]_i > 0$ is undesirable, as it contributes negatively towards the condition that guarantees function reduction at the particular direction $(-(\mathbf{d}^\tau)^T \nabla f(\mathbf{w}^\tau) < 0)$.

As a result, variables belonging to $\mathcal{B}^\tau$ should not affect the line-search procedure of iteration τ. This is achieved by considering a modified direction $\widetilde{\mathbf{d}}$ defined as:

$$\widetilde{d}_i^\tau = \begin{cases} \bar{d}_i^\tau, & i \in \mathcal{F}^\tau \\ 0, & i \in \mathcal{B}^\tau \end{cases}, \forall i \tag{3.45}$$

where $\bar{\mathbf{d}}^\tau = \mathbf{S}^\tau \nabla_P f(\mathbf{w}^\tau)$ and $\nabla_P f(\mathbf{w}^\tau)$ is the *projected gradient* given by Eq. (3.46).

$$[\nabla_P f(\mathbf{w}^\tau)]_i = \begin{cases} [\nabla f(\mathbf{w}^\tau)]_i, & i \in \mathcal{F}^\tau \\ 0, & i \in \mathcal{B}^\tau \end{cases}, \forall i \tag{3.46}$$

In this way, Eq. (3.41) becomes:

$$\mathbf{w}^{\tau+1} = P[\mathbf{w}^\tau - s^\tau \widetilde{\mathbf{d}}^\tau]$$

An equivalent expression can be obtained by updating only the variables belonging to the free-set: $\mathbf{w}_\mathcal{F}^{\tau+1} = P[\mathbf{w}_\mathcal{F}^\tau - s^\tau \widetilde{\mathbf{S}}^\tau \mathbf{g}_\mathcal{F}^\tau]$, where $\widetilde{\mathbf{S}}^\tau \in \mathbb{R}^{|\mathcal{F}| \times |\mathcal{F}|}$ is the principal sub-matrix of $\mathbf{S}^\tau$ corresponding to the free variables and similarly, $g_{i,\mathcal{F}}^\tau = [\nabla f(\mathbf{w}^\tau)]_{\mathcal{F}(i)}$, $i = 1, ..., |\mathcal{F}|$.

Algorithm 3.1 - **PGNNLS**: Projected Gradient Algorithm for the NNLS problem

Input: $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_K]$, $\mathbf{b}$, M
Output: $\mathbf{w}^\tau$
Initialise: $\tau \leftarrow 0$; $\mathbf{w}^\tau \leftarrow \mathbf{0}$; $s^{\tau-1} \leftarrow 1$;
Set $f^\tau \leftarrow f(\mathbf{w}^\tau)$ and $\mathbf{g}^\tau \leftarrow \nabla f(\mathbf{w}^\tau)$ using Eqs. (3.35) and (3.36);
Find the binding set $\mathcal{B}^\tau$ according to expression (3.44)
Set $\widetilde{\mathbf{d}}^\tau = \nabla_P f(\mathbf{w}^\tau)$ using Eq. (3.46);
repeat
  %Perform an APA line-search
  if $(\tau < NI_{HE})$ **then**
    $[\mathbf{w}_{temp},\ s_{temp}, \text{flag}] \leftarrow \text{lineSearchHE}(f^\tau, \nabla f(\mathbf{w}^\tau), \widetilde{\mathbf{d}}^\tau, \mathbf{w}^\tau, s^{\tau-1}, \mathbf{Q}, \mathbf{b})$;
    if (flag = FALSE) **then**
      $[\mathbf{w}_{temp},\ s_{temp}] \leftarrow \text{lineSearchLin}(f^\tau, \nabla f(\mathbf{w}^\tau), \widetilde{\mathbf{d}}^\tau, \mathbf{w}^\tau, s^{\tau-1}, \mathbf{Q}, \mathbf{b})$;
    end if
  else
    $[\mathbf{w}_{temp},\ s_{temp}] \leftarrow \text{lineSearchLin}(f^\tau, \nabla f(\mathbf{w}^\tau), \widetilde{\mathbf{d}}^\tau, \mathbf{w}^\tau, s^{\tau-1}, \mathbf{Q}, \mathbf{b})$;
  end if
  $s^\tau \leftarrow s_{temp}$; $\tau \leftarrow \tau + 1$; $\mathbf{w}^\tau \leftarrow \mathbf{w}_{temp}$;
  Set $f^\tau \leftarrow f(\mathbf{w}^\tau)$, $\mathbf{g}^\tau \leftarrow \nabla f(\mathbf{w}^\tau)$ using Eqs. (3.35) and (3.36);
  Find the binding set $\mathcal{B}^\tau$ according to expression (3.44);
  %Update the search direction
  if $(s^\tau \geq s_{min})$ **then**
    if $(\tau > M)$ **then**
      Discard the vector pair $\{\Delta\mathbf{w}^{\tau-1-M}, \Delta\mathbf{g}^{\tau-1-M}\}$ from storage;
    end if
    Store $\Delta\mathbf{w}^{\tau-1} = \mathbf{w}^\tau - \mathbf{w}^{\tau-1}$; $\Delta\mathbf{g}^{\tau-1} = \mathbf{g}^\tau - \mathbf{g}^{\tau-1}$;
    $\mathbf{d}^\tau \leftarrow \text{updateLBFGS}(\nabla_P f(\mathbf{w}^\tau), \Delta\mathbf{w}^k, \Delta\mathbf{g}^k, k = \max\{0, \tau - M\}, ..., \tau - 1)$;
    Define $\widetilde{\mathbf{d}}^\tau$ according to Eq. (3.45)
  else
    Discard all stored vector pairs $\{\Delta\mathbf{w}^k, \Delta\mathbf{g}^k\}$;
    Set $\widetilde{\mathbf{d}}^\tau = \nabla_P f(\mathbf{w}^\tau)$ using Eq. (3.46);
  end if
until a stopping criterion is met

As mentioned above, Algorithm 3.1 relies on a limited-memory BFGS update of the scaling matrix. In each iteration, the BFGS formula is updated so that the new matrix is symmetric, satisfies the secant equation and also is the closest to the current approximation matrix in the least squares sense. In addition, if the associated problem is strictly convex and an appropriate line-search is considered, then the updated matrices are also positive definite [157]. The BFGS formula for updating $\mathbf{S}^\tau$ is given by:

$$\mathbf{S}^\tau = (\mathbf{V}^{\tau-1})^T \mathbf{S}^{\tau-1} \mathbf{V}^{\tau-1} + \rho^{\tau-1} \Delta\mathbf{w}^{\tau-1} (\Delta\mathbf{w}^{\tau-1})^T \tag{3.47}$$

where

$$\rho^k = \frac{1}{(\Delta\mathbf{g}^k)^T \Delta\mathbf{w}^k}, \;\; \mathbf{V}^k = \mathbf{I} - \rho^k \Delta\mathbf{g}^k (\Delta\mathbf{w}^k)^T, \text{ and } \mathbf{S}^0 = \sigma_{BFGS}\mathbf{I}, \;\; \sigma_{BFGS} > 0$$

Notice that $\mathbf{S}^\tau$ is a rank-two modification of $\mathbf{S}^{\tau-1}$ which can be obtained using $\Delta\mathbf{w}^{\tau-1}$ and $\Delta\mathbf{g}^{\tau-1}$. Hence, if we store all vectors $\Delta\mathbf{w}^k$ and $\Delta\mathbf{g}^k$ from the start of the algorithm, we can obtain $\mathbf{S}^\tau$ without storing any matrix.

In the limited-memory variant of BFGS, instead of storing all vectors, we update $\mathbf{S}^\tau$ based on the M most recent $\Delta\mathbf{w}^k$ and $\Delta\mathbf{g}^k$ vector pairs. This is achieved with the use of the following recursive formula which is directly derived from (3.47) [32].

$$\begin{aligned} \mathbf{S}^\tau \;=\; & (\mathbf{V}^{\tau-M} \cdots \mathbf{V}^{\tau-1})^T \mathbf{S}_0^\tau (\mathbf{V}^{\tau-M} \cdots \mathbf{V}^{\tau-1}) \\ & + \rho^{\tau-M} (\mathbf{V}^{\tau-M+1} \cdots \mathbf{V}^{\tau-1})^T \Delta\mathbf{w}^{\tau-M} (\Delta\mathbf{w}^{\tau-M})^T (\mathbf{V}^{\tau-M+1} \cdots \mathbf{V}^{\tau-1}) \\ & \rho^{\tau-M+1} (\mathbf{V}^{\tau-M+2} \cdots \mathbf{V}^{\tau-1})^T \Delta\mathbf{w}^{\tau-M+1} (\Delta\mathbf{w}^{\tau-M+1})^T (\mathbf{V}^{\tau-M+2} \cdots \mathbf{V}^{\tau-1}) \\ & + \cdots \\ & + \rho^{\tau-1} \Delta\mathbf{w}^{\tau-1} (\Delta\mathbf{w}^{\tau-1})^T \end{aligned} \tag{3.48}$$

Using Eq. (3.48) we can efficiently update the search direction $\mathbf{d}^\tau = \mathbf{S}^\tau \nabla f(\mathbf{w}^\tau)$, without storing $\mathbf{S}^\tau$ at any iteration. As a result, the required memory for the quasi-Newton update is reduced from $2N^2 \times 2N^2$ to $2M \times 2N^2$. This is a substantial memory saving, as it has been observed in practice that even small values of M (say $M \in [3, 7]$) provide satisfactory results [32]. Nocedal and Wright [143] describe in detail the limited memory BFGS method and outline an iterative procedure for updating the search direction based on (3.48); we outline this procedure in Algorithm 3.2.

Algorithm 3.2 - updateLBFGS: Compute the product of the limited memory scaling matrix and the gradient

Input: $\mathbf{g}, \Delta\mathbf{w}^k, \Delta\mathbf{g}^k, k = \max\{0, \tau - M\}, ..., \tau - 1$
Output: $\mathbf{d}$
for $(k = \tau - 1, ..., \max\{0, \tau - M\})$ **do**
  $\rho^k \leftarrow 1/((\Delta\mathbf{g}^k)^T \Delta\mathbf{w}^k)$;
  $\alpha_1^k \leftarrow \rho^k (\Delta\mathbf{w}^k)^T \mathbf{g}$;
  $\mathbf{g} \leftarrow \mathbf{g} - \alpha_1^k \Delta\mathbf{g}^k$;
end for
$\mathbf{d} \leftarrow \mathbf{S}_0^\tau \mathbf{g}$;
for $(k = \max\{0, \tau - M\}, ..., \tau - 1)$ **do**
  $\alpha_2 \leftarrow \rho^k (\Delta\mathbf{g}^k)^T \mathbf{d}$;
  $\mathbf{d} \leftarrow \mathbf{d} + (\alpha_1^k - \alpha_2)\Delta\mathbf{w}^k$;
end for

The use of the limited-memory BFGS scheme also provides computational benefits. Note that updating the scaling matrix using the BFGS method requires several matrix-vector operations whose computational complexity is $O((2N^2)^2)$. On the hand, the use of Algorithm 3.2 requires $5M$ vector-vector products so that its computational complexity is $O(5M(2N^2))$ which is significantly less than the complexity of a single matrix-vector product.

Let us now turn our attention to the discussion of the line-search procedure. As mentioned above, the step-size s^τ is found by employing the "Armijo rule along the projection arc" (APA) [25]. In APA rule the step-size is chosen to be equal to $s^\tau = \beta^m$, where m is the smallest nonnegative integer satisfying the APA condition:

$$f(\mathbf{w}_{cand}^{\tau+1}(\beta^m)) - f(\mathbf{w}^\tau) \leq \sigma_{APA} \nabla f(\mathbf{w}^\tau)^T (\mathbf{w}_{cand}^{\tau+1}(\beta^m) - \mathbf{w}^\tau) \tag{3.49}$$

where $\mathbf{w}_{cand}^{\tau+1}(\beta^m) = P[\mathbf{w}^\tau - \beta^m \widetilde{\mathbf{d}}^\tau]$, $0 < \sigma_{APA} < 1/2$ and $0 < \beta < 1$. An important advantage of the APA over other step-size rules is that it identifies many active constraints in one iteration. In addition, it is proven that the sequence $\{\mathbf{w}^\tau\}$ produced when applying the APA rule, converges to a stationary point $\{\mathbf{w}^*\}$[25], which in our case is a global minimum. In [33], a more detailed analysis of projected gradient algorithms further relaxed the convergence conditions. The authors showed that convergence to a stationary point can be achieved by choosing any step-size satisfying condition (3.49), under the assumptions that s^τ is not too small, the cost function is bounded below and the gradient is uniformly continuous (Theorem 2.3

in [33]), which are true in the NNLS case. Hence, convergence is guaranteed even if we choose any value of m satisfying (3.49) rather than the smallest integer, as long as the selected β^m are not too small.

Nevertheless, identifying the appropriate m value may require a large number of function evaluations and projections. To alleviate this problem, Lin proposed a different strategy for the identification of m [124]. Lin's approach manipulates the fact that the value of s^τ is similar to $s^{\tau-1}$ to start the line-search from $s^{\tau-1}$ (Algorithm 3.3). If condition (3.49) is satisfied then this value is stored and the step-size is increased (division of the current step-size by β), until we find a value that violates the APA condition. Otherwise, if the APA condition is not satisfied the step-size is decreased (multiplication of the current step-size by β) until we find a value that satisfies (3.49).

Algorithm 3.3 - lineSearchLin: Lin's APA line-search procedure

Input: $f(\mathbf{w}^\tau)$, $\nabla f(\mathbf{w}^\tau)$, $\mathbf{d}^\tau$, $\mathbf{w}^\tau$,$s^{\tau-1}$, $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_K]$, $\mathbf{b}$
Output: $\mathbf{w}^{\tau+1}$, s^τ
Initialise: $\sigma \leftarrow 0.01$; $\beta \leftarrow 0.9$; $s^\tau_{cand} \leftarrow s^{\tau-1}$;
$\mathbf{w}^{\tau+1}_{cand} \leftarrow P[\mathbf{w}^\tau - s^\tau_{cand}\mathbf{d}^\tau]$;
if ((3.49) is satisfied) **then**
  repeat
    $s^\tau \leftarrow s^\tau_{cand}$;
    $s^\tau_{cand} \leftarrow s^\tau_{cand}/\beta$;
    $\mathbf{w}^{\tau+1}_{cand} \leftarrow P[\mathbf{w}^\tau - s^\tau_{cand}\mathbf{d}^\tau]$;
  until (((3.49) is not satisfied) or ($\mathbf{w}(s^\tau_{cand}/\beta) = \mathbf{w}(s^\tau_{cand})$))
else
  repeat
    $s^\tau_{cand} \leftarrow s^\tau_{cand}\beta$;
    Set $\mathbf{w}^{\tau+1}_{cand} \leftarrow P[\mathbf{w}^\tau - s^\tau_{cand}\mathbf{d}^\tau]$;
  until ((3.49) is satisfied)
  $s^\tau \leftarrow s^\tau_{cand}$;
end if
$\mathbf{w}^{\tau+1} \leftarrow P[\mathbf{w}^\tau - s^\tau\mathbf{d}^\tau]$;

Although the particular line search is quite efficient after the first iteration, no suggestion has been made in efficiently obtaining s^1. In a typical execution of this algorithm, if 100 iterations are undertaken, the first may require 90 trials while the rest 250 trials in total. Hence, the first iteration requires a significant amount of the execution time.

As described in Algorithm 3.4, we propose to hyper-exponentially alternate s^τ

Algorithm 3.4 - lineSearchHE: Hyper-exponential APA line-search procedure

Input: $f(\mathbf{w}^\tau)$, $\nabla f(\mathbf{w}^\tau)$, $\mathbf{d}^\tau$, $\mathbf{w}^\tau$, $s^{\tau-1}$, $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_K]$, $\mathbf{b}$
Output: $\mathbf{w}^{\tau+1}$, s^τ, flag
Initialise: $\sigma \leftarrow 0.01$; $\beta \leftarrow 0.9$; $s^\tau_{cand} \leftarrow s^{\tau-1}$; $s^\tau_{init} \leftarrow s^{\tau-1}$; $k \leftarrow -1$; flag $\leftarrow$ TRUE;
$\mathbf{w}^{\tau+1}_{cand} \leftarrow P[\mathbf{w}^\tau - s^\tau_{cand}\mathbf{d}^\tau]$;
if ((3.49) is satisfied) **then**
  repeat
    $s^\tau \leftarrow s^\tau_{cand}$; $k \leftarrow k+1$; $s^\tau_{cand} \leftarrow s^\tau_{init}/\beta^{2^k}$;
    $\mathbf{w}^{\tau+1}_{cand} \leftarrow P[\mathbf{w}^\tau - s^\tau_{cand}\mathbf{d}^\tau]$;
  until ((3.49) is not satisfied)
  $low \leftarrow \lfloor 2^{k-1}+1 \rfloor$; $high \leftarrow 2^k$;
  while $(low < high)$ **do**
    $mid \leftarrow \lfloor (low+high)/2 \rfloor$; $s^\tau_{cand} \leftarrow s^\tau_{init}/\beta^{mid}$;
    $\mathbf{w}^{\tau+1}_{cand} \leftarrow P[\mathbf{w}^\tau - s^\tau_{cand}\mathbf{d}^\tau]$;
    if ((3.49) is not satisfied) **then**
      $high \leftarrow mid$;
    else
      $s^\tau \leftarrow s^\tau_{cand}$; $low \leftarrow mid+1$;
    end if
  end while
else
  repeat
    if $(s^\tau_{cand} < s_{min})$ **then**
      flag $\leftarrow$ FALSE; **break**;
    else
      $k \leftarrow k+1$; $s^\tau_{cand} \leftarrow s^\tau_{init} \cdot \beta^{2^k}$;
      $\mathbf{w}^{\tau+1}_{cand} \leftarrow P[\mathbf{w}^\tau - s^\tau_{cand}\mathbf{d}^\tau]$;
    end if
  until ((3.49) is satisfied)
  $low \leftarrow \lfloor 2^{k-1}+1 \rfloor$; $high \leftarrow 2^k$; $s^\tau \leftarrow s^\tau_{cand}$;
  while $(low < high)$AND(flag) **do**
    $mid \leftarrow \lfloor (low+high)/2 \rfloor$; $s^\tau_{cand} \leftarrow s^\tau_{init} \cdot \beta^{mid}$;
    $\mathbf{w}^{\tau+1}_{cand} \leftarrow P[\mathbf{w}^\tau - s^\tau_{cand}\mathbf{d}^\tau]$;
    if ((3.49) is satisfied) **then**
      $high \leftarrow mid$; $s^\tau \leftarrow s^\tau_{cand}$;
    else
      $low \leftarrow mid+1$;
    end if
  end while
end if
$\mathbf{w}^{\tau+1} \leftarrow P[\mathbf{w}^\tau - s^\tau \mathbf{d}^\tau]$;

for the identification of an appropriate step-size value. In the hyper-exponential line-search (lineSearchHE), the first trial also starts from $s^{\tau-1}$. If the APA condition is satisfied, which means that $s^{\tau} \geq s^{\tau-1}$, we hyper-exponentially increase the step-size (division of the initial step-size by β^{2^k}, $k = 0, 1, 2...$) until a step-size $s^{\tau}_{init}/\beta^{2^{k_v}}$ violating condition (3.49) is found; hence, the step-size value is in the region $[s^{\tau}_{init}/\beta^{2^{k_v-1}},\ s^{\tau}_{init}/\beta^{2^{k_v}})$. Then, a divide-and-conquer procedure is followed, until the largest value β^m satisfying the APA rule is identified. In the case that the APA condition is initially not satisfied ($0 < s^{\tau} < s^{\tau-1}$), we hyper-exponentially decrease the step-size (multiplication of the initial step-size by β^{2^k}, $k = 0, 1, 2...$) until a step-size $s^{\tau}_{init}\beta^{2^{k_s}}$ satisfying condition (3.49) is found, so that $s^{\tau} \in [s^{\tau}_{init}\beta^{2^{k_s}}, s^{\tau}_{init}\beta^{2^{k_s-1}})$. Then, we follow a divide and conquer procedure to find the largest value β^m in the identified region that satisfies Eq. (3.49).

At this point, it is important to mention that function $f(\mathbf{w}^{\tau+1}_{cand}(s)), s \geq 0$ is generally non-convex. As a result, if the initial trial does not satisfy the APA condition, it is possible that no trial with sufficiently large step-size will satisfy it, in which case the step-size will go to zero. If this situation occurs, we stop the hyper-exponential line-search and restart with procedure **lineSearchLin**, which is guaranteed to be successful. It is also possible due to the hyper-exponential reduction of the step-size, to overleap a region of s^{τ} values where the APA condition is satisfied. In this case the algorithm's convergence is still not affected, as either another satisfactory value will be found, or the restarted **lineSearchLin** procedure will find the overleapt value.

The proposed line search requires approximately $2log_2(NT^{\tau})$ trials to compute s^{τ} if the procedure is successful, where NT^{τ} is the number of trials required by Lin's approach. Otherwise, the additional number of trials performed is $\lceil \log_2(\ln(s_{min}/s^{\tau}_{init})/\ln(\beta)) \rceil$. For example, if $s_{min} = 10^{-8}$, $s^{\tau}_{init} = 0.3$ and $\beta = 0.9$ then an additional number of 8 trials will be performed.

After the first few iterations the step-sizes of subsequent iterations are similar and the benefit of using the hyper-exponential line search is small. Hence, we can employ **lineSearchLE** only for the first NI_{HE} iterations as indicated in the PGNNLS algorithm.

Formally the stopping criterion that should be met for the termination of the PGNNLS algorithm is related to the Karush-Kuhn-Tucker (KKT) optimality conditions. As the NNLS problem is convex, the KKT conditions are both necessary

and sufficient for optimality (see Section 5.5.3 in [28]). For the NNLS problem, these conditions are satisfied when:

$$[\nabla f(\mathbf{w})]_i = 0, \quad \text{if } w_i > 0$$
$$[\nabla f(\mathbf{w})]_i \geq 0, \quad \text{if } w_i = 0$$

In the first case, the violation of the condition is equal to the value of the gradient $[\nabla f(\mathbf{w})]_i$, while in the second case it is equal to the value of the gradient if it is negative, $\min(0, [\nabla f(\mathbf{w})]_i)$. These violations are exactly represented by the projected gradient $\nabla_P f(\mathbf{w})$ defined in Eq. (3.46), so that the KKT optimally criterion for the NNLS problem can be expressed as:

$$\|[\nabla_P f(\mathbf{w})]\| \leq \epsilon$$

However, we do not require the accurate solution of the NNLS problem as it is only used to approximately train the RNN. Hence, we may use other stopping criteria such as the maximum number of iterations and the relative change in the cost function or the decision variables.

The most costly operations that need to be performed at each iteration of Algorithm 3.1 involve the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$, which require matrix-vector product operations. In particular, at the start of each iteration, it is needed to evaluate $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ once. Additionally, each trial of the line-search procedure requires the evaluation of $f(\mathbf{w}_{cand}^{\tau+1})$. In Section 3.3.3, we discuss two different approaches for the efficient evaluation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ and derive efficient analytical expressions.

The RNN-NNLS algorithm

The PGNNLS algorithm can be used to deal with the RNN supervised learning problem when our recurrent network is only composed of output neurons. Next, we extend this approach to the case that the network is composed of both output and non-output neurons.

The approach that we take is the following: if neuron $i \in I_{out}$ then we set $q_{ik} = y_{ik}, \ \forall k$, while if neuron $i \in I_{\overline{out}}$ then we set $q_{ik} = U(a, b), \ \forall k$, where $0 \leq a \leq b \leq 1$ and $U(a, b)$ denotes a sample from the uniform distribution in the interval $[a, b]$. Following this approach, we obtain q_{ik} values for all neurons and patterns; thus,

an NNLS problem is derived (Eqs. (3.35),(3.39),(3.40)), which can be solved using the PGNNLS algorithm. As a result, the larger the portion of output neurons the better the obtained weights will be.

As already mentioned, due to the nonnegativity constraints and depending on the dimensions N,K the system of Eqs. (3.34) may not have a solution; therefore, the obtained weights from the solution of the NNLS optimisation problem will not accurately satisfy Eqs. (2.6)-(2.8), and the obtained weights will not result in good performance. To deal with this issue, we use the weights acquired from the execution of Algorithm 3.1, to compute q_{ik} from Eqs. (2.6)-(2.8). Then, a weighted version of the desired values q_{ik}^d and the exact values q_{ik} is computed and used as the new q_{ik}^d values in PGNNLS. Using this iterative procedure, we progressively move towards weights that satisfy $q_{ik}^d \approx q_{ik}$.

To retain our original goal of achieving the desired output values $y_{ik},\ i \in I_{out}$ we update the output q_{ik}^d values in two different ways: (a) there is a different weighting parameter for these neurons, $0 \leq \alpha_o \leq 1$, typically close to one so that their desired values slowly vary, and (b) we restrict the neuron values within a desired region so that neurons corresponding to "0" decisions must have $q_{ik} \leq 0.4$ and neurons corresponding to "1" decisions must have $q_{ik} \geq 0.6$. By selecting the specific boundary values, we achieve to constrain each q_{ik} in the desired region and to have a large variation range for the parameters.

The overall procedure is outlined in Algorithm 3.5, called RNN-NNLS. It is important to note that the NNLS algorithm does not require matrix $\mathbf{B}$ as input, which would be prohibitive for a large network. Due to Eq. (3.39), we can perform all the computations involving $\mathbf{B}$ using matrix $\mathbf{Q}$ which holds all the q_{ik} values. Thereby, the order of memory required is the same with the standard RNN learning algorithm. The iterative procedure RNN-NNLS needs only a small number of iterations, $NI_{RNN-NNLS}$, before the error stabilises.

Finally, we should highlight that the RNN-NNLS algorithm can be used both for supervised learning and weight initialisation. When used as a supervised learning algorithm a more accurate solution of problem (3.35) may be found, while when used as a weight initialization method it is sufficient to find a close to optimal solution.

Before we discuss the computational complexity of the PGNNLS and RNN-NNLS algorithms, we propose two different approaches to efficiently evaluate functions

Algorithm 3.5 -RNN-NNLS: RNN supervised learning algorithm based on NNLS formulation, for the solution of problems involving both output and non-output neurons

Input: $\mathbf{x}_k,\ \mathbf{y}_k, \forall k$
Output: $\mathbf{w}$
Initialise Λ_{ik} and $\lambda_{ik}\ \forall\ i,\ k$ based on x_{ik};
Set $q_{ik}^d = y_{ik},\ i \in I_{out}$;
Set $q_{ik}^d = U(a,b),\ i \in I_{\overline{out}}$;
Form matrix $\mathbf{Q}^d = [\mathbf{q}_1^d, ..., \mathbf{q}_K^d]$, where $\mathbf{q}_k^d(i) = q_{ik}^d,\ \forall i, k$;
for $l = 1$ to $NI_{RNN-NNLS}$ **do**
  Update $\mathbf{b}$ according to Eq. (3.40);
  $\mathbf{w} \leftarrow$ PGNNLS($\mathbf{Q}^d, \mathbf{b}$);
  for all k **do**
    Obtain $q_{ik},\ i \in I_{\overline{out}}$ by solving Eqs. (2.6)-(2.8);
  end for
  Set $q_{ik}^d \leftarrow \alpha_{no} q_{ik}^d + (1 - \alpha_{no}) q_{ik}, i \in I_{\overline{out}}, \forall k$;
  Set $q_{ik}^d \leftarrow \alpha_o q_{ik}^d + (1 - \alpha_o) q_{ik}, i \in I_{out}, \forall k$;
  for $i \in I_{\overline{out}},\ k = 1, ..., K$ **do**
    if $((y_{ik} = 1)$ AND $(q_{ik}^d < 0.6))$ **then**
      $q_{ik}^d = 0.6$;
    end if
    if $((y_{ik} = 0)$ AND $(q_{ik}^d > 0.4))$ **then**
      $q_{ik}^d = 0.4$;
    end if
  end for
end for

$f(\mathbf{w})$ and $\nabla f(\mathbf{w})$.

3.3.3. Efficient computation of NNLS costly functions

As mentioned earlier, the most computationally expensive functions in Algorithm 3.1 are $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$. However, computing these functions directly is very inefficient, so that the structure and sparsity of matrix $\mathbf{B}$ should be exploited to find efficient ways to compute $\nabla f(\mathbf{w})$. In this section we develop two such approaches. The first is based on the efficient computation of $\mathbf{B}^T\mathbf{z}_1$ and $\mathbf{B}\mathbf{z}_2$ where $\mathbf{z}_1$ and $\mathbf{z}_2$ are vectors of appropriate dimensions. The second is based on first computing and storing $\mathbf{B}^T\mathbf{B}$ in order to compute $(\mathbf{B}^T\mathbf{B})\mathbf{z}$. We show that the computational complexity of the former approach is $O(KN^2)$ per evaluation, while the complexity of the latter is $O(N^3)$ per evaluation plus an initialisation cost of $O(KN^3)$. This indicates that each of the two approaches can be faster than the other depending on the problem dimensions (number of training pairs, K, and number of neurons, N). If $K \gg N$ then the second approach is faster that the first one, otherwise the first one is better.

The structure of matrix B

Matrix $\mathbf{B}$ is composed of many different matrix blocks which correspond to entries associated with positive or negative weights as well as different input-output training pairs. As a result we can represent $\mathbf{B}$ as:

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{+1}, & \mathbf{B}_{-1} \\ \vdots & \vdots \\ \mathbf{B}_{+K}, & \mathbf{B}_{-K} \end{bmatrix}, \mathbf{B} \in \mathbb{R}^{KN \times 2N^2} \text{ and } \mathbf{B}_{\pm k} \in \mathbb{R}^{N \times N^2},\ k = 1, ..., K \qquad (3.50)$$

where $\mathbf{B}_{+k}$ and $\mathbf{B}_{-k}$ represent the entries associated with the kth input-output training pair of the positive and negative weights respectively. These matrices are sparse and are also of particular structure, as shown below for the case that $N = 3$, when $d(i) = 0$, $i = 1, ..., N$.

$$\mathbf{B}_{+k} = \overbrace{\begin{bmatrix} 0 & q_{1k} & q_{1k} & -q_{2k} & 0 & 0 & -q_{3k} & 0 & 0 \\ 0 & -q_{1k} & 0 & q_{2k} & 0 & q_{2k} & 0 & -q_{3k} & 0 \\ 0 & 0 & -q_{1k} & 0 & 0 & -q_{2k} & q_{3k} & q_{3k} & 0 \end{bmatrix}}^{\text{values corresponding to } w^+(i,j)}$$

$$\mathbf{B}_{-k} = \overbrace{\begin{bmatrix} q_{1k}+q_{1k}^2 & q_{1k} & q_{1k} & q_{1k}q_{2k} & 0 & 0 & q_{1k}q_{3k} & 0 & 0 \\ 0 & q_{1k}q_{2k} & 0 & q_{2k} & q_{2k}+q_{2k}^2 & q_{2k} & 0 & q_{2k}q_{3k} & 0 \\ 0 & 0 & q_{1k}q_{3k} & 0 & 0 & q_{2k}q_{3k} & q_{3k} & q_{3k} & q_{3k}+q_{3k}^2 \end{bmatrix}}^{\text{values corresponding to } w^-(i,j)}$$

Note that the structure of the above matrices allows their further decomposition into:

$$\mathbf{B}_{+k} = \mathbf{C}_k + \mathbf{D}_{+k} \tag{3.51}$$

$$\mathbf{B}_{-k} = \mathbf{C}_k + \mathbf{D}_{-k} \tag{3.52}$$

For example, for the case that $N = 3$ matrices $\mathbf{C}_k$, $\mathbf{D}_{+k}$ and $\mathbf{D}_{-k}$ take the form:

$$\mathbf{C}_k = \begin{bmatrix} q_{1k} & q_{1k} & q_{1k} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_{2k} & q_{2k} & q_{2k} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & q_{3k} & q_{3k} & q_{3k} \end{bmatrix}$$

$$\mathbf{D}_{+k} = \begin{bmatrix} -q_{1k} & 0 & 0 & -q_{2k} & 0 & 0 & -q_{3k} & 0 & 0 \\ 0 & -q_{1k} & 0 & 0 & -q_{2k} & 0 & 0 & -q_{3k} & 0 \\ 0 & 0 & -q_{1k} & 0 & 0 & -q_{2k} & 0 & 0 & -q_{3k} \end{bmatrix}$$

$$\mathbf{D}_{+k} = \begin{bmatrix} q_{1k}q_{1k} & 0 & 0 & q_{2k}q_{1k} & 0 & 0 & q_{3k}q_{1k} & 0 & 0 \\ 0 & q_{1k}q_{2k} & 0 & 0 & q_{2k}q_{2k} & 0 & 0 & q_{3k}q_{2k} & 0 \\ 0 & 0 & q_{1k}q_{3k} & 0 & 0 & q_{2k}q_{3k} & 0 & 0 & q_{3k}q_{3k} \end{bmatrix}$$

Notice that matrices $\mathbf{C}_k$, $\mathbf{D}_{+k}$ and $\mathbf{D}_{-k}$ also have a special structure while all can be decomposed further into $N \times N$ sized submatrices. In particular, we have that:

$$\mathbf{C}_k = [\mathbf{C}_{k1}, \ldots, \mathbf{C}_{ki}, \ldots, \mathbf{C}_{kN}]$$

$$\mathbf{D}_{+k} = [\mathbf{D}_{+k1}, \ldots, \mathbf{D}_{+ki}, \ldots, \mathbf{D}_{+kN}]$$

$$\mathbf{D}_{-k} = [\mathbf{D}_{-k1}, \ldots, \mathbf{D}_{-ki}, \ldots, \mathbf{D}_{-kN}]$$

where sub-matrices $\mathbf{C}_{ki}$, $\mathbf{D}_{+ki}$, $\mathbf{D}_{-ki} \in \mathbb{R}^{N\times N}$ are given by:

$$\mathbf{C}_{ki}{=}q_{ik}(\mathbf{e}_i\mathbf{1}^T) \tag{3.53}$$

$$\mathbf{D}_{+ki} = \text{diag}([-q_{ik}, -q_{ik}, \ldots, -q_{ik}]) = -q_{ik}\mathbf{I} \tag{3.54}$$

$$\mathbf{D}_{-ki} = q_{ik}\text{diag}([q_{1k},\, q_{2k}, \ldots, q_{Nk}]) = q_{ik}\text{diag}(\mathbf{q}_k) \tag{3.55}$$

where $\mathbf{I}$ is the $N \times N$ identity matrix and $\text{diag}(\mathbf{x})$ is a diagonal matrix with elements of the main diagonal given by the entries of vector $\mathbf{x}$, while all the other elements of the matrix are equal to zero.

The first approach for the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$

As discussed in section 3.3.1, functions $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ can be computed according to Eqs. (3.35) and (3.36) respectively, which can be written as:

$$\begin{aligned} f(\mathbf{w}) &= \frac{1}{2}(\mathbf{B}\mathbf{w}-\mathbf{b})^T(\mathbf{B}\mathbf{w}-\mathbf{b}) \\ &= \frac{1}{2}(\hat{\mathbf{z}}-\mathbf{b})^T(\hat{\mathbf{z}}-\mathbf{b}) \\ &= \frac{1}{2}\mathbf{z}^T\mathbf{z} \end{aligned} \tag{3.56}$$

$$\begin{aligned} \nabla f(\mathbf{w}) &= \mathbf{B}^T(\mathbf{B}\mathbf{w}-\mathbf{b}) \\ &= \mathbf{B}^T\mathbf{z} \end{aligned} \tag{3.57}$$

where we have defined $\hat{\mathbf{z}} = \mathbf{B}\mathbf{w}$, $\hat{\mathbf{z}} \in \mathbb{R}^{NK\times 1}$ and $\mathbf{z} = \hat{\mathbf{z}} - \mathbf{b}$. As a result, for the computation of $f(\mathbf{w})$ the only expensive step is the calculation of $\hat{\mathbf{z}} = \mathbf{B}\mathbf{w}$. Similarly, the expensive steps for the computation of $\nabla f(\mathbf{w})$ are the calculation of $\hat{\mathbf{z}} = \mathbf{B}\mathbf{w}$ and $\widetilde{\mathbf{z}} = \mathbf{B}^T\mathbf{z}$, where $\widetilde{\mathbf{z}} \in \mathbb{R}^{2N^2\times 1}$. Note that the matrix-vector product $\mathbf{B}\mathbf{w}$ appears in both terms. As a result, only two matrix-vector products are needed for the evaluation of both functions at the same point $\mathbf{w}_c$: $\hat{\mathbf{z}}_c = \mathbf{B}\mathbf{w}_c$ and $\widetilde{\mathbf{z}} = \mathbf{B}^T\mathbf{z}_c$, where $\mathbf{z}_c = \hat{\mathbf{z}}_c - \mathbf{b}$.

As the naive calculation of these matrix-vector products is not efficient, we manipulate the special structure and sparsity of matrix $\mathbf{B}$ to derive expressions of low computational complexity.

Let us first examine the term $\hat{\mathbf{z}} = \mathbf{B}\mathbf{w}$. Expanding $\mathbf{B}$ and $\mathbf{w}$ we obtain:

$$\begin{aligned}
\hat{\mathbf{z}} = \mathbf{B}\mathbf{w} &= \begin{bmatrix} \mathbf{C}_1 + \mathbf{D}_{+1}, & \mathbf{C}_1 + \mathbf{D}_{-1} \\ \vdots & \vdots \\ \mathbf{C}_K + \mathbf{D}_{+K}, & \mathbf{C}_K + \mathbf{D}_{-K} \end{bmatrix} \begin{bmatrix} \mathbf{w}^+ \\ \mathbf{w}^- \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{C}_1\mathbf{w}^+ + \mathbf{C}_1\mathbf{w}^- + \mathbf{D}_{+1}\mathbf{w}^+ + \mathbf{D}_{-1}\mathbf{w}^- \\ \vdots \\ \mathbf{C}_K\mathbf{w}^+ + \mathbf{C}_K\mathbf{w}^- + \mathbf{D}_{+K}\mathbf{w}^+ + \mathbf{D}_{-K}\mathbf{w}^- \end{bmatrix} \\
&= \begin{bmatrix} \hat{\mathbf{z}}_1 \\ \vdots \\ \hat{\mathbf{z}}_K \end{bmatrix}, \hat{\mathbf{z}}_k \in \mathbb{R}^{N\times 1} \qquad (3.58)
\end{aligned}$$

where $\mathbf{w}^+$ represents the positive weights so that value $w^+(iN-N+j) \equiv w^+(i,j)$ and $\mathbf{w}^-$ the negative weights such that $w^-(iN-N+j) \equiv w^-(i,j)$. Note that to evaluate $\hat{\mathbf{z}}$ it is sufficient to derive expressions for terms $\hat{\mathbf{z}}_k$:

$$\hat{\mathbf{z}}_k = \mathbf{C}_k\mathbf{w}^+ + \mathbf{C}_k\mathbf{w}^- + \mathbf{D}_{+k}\mathbf{w}^+ + \mathbf{D}_{-k}\mathbf{w}^- \qquad (3.59)$$

Hence, the computation of $\hat{\mathbf{z}}_k$ requires the efficient evaluation of $\mathbf{C}_k\mathbf{w}^+$, $\mathbf{C}_k\mathbf{w}^-$, $\mathbf{D}_{+k}\mathbf{w}^+$ and $\mathbf{D}_{-k}\mathbf{w}^-$. Manipulation of these terms (see Appendix B.1.1) yields:

$$\hat{\mathbf{z}}_k = \mathbf{q}_k \bigodot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) - (\mathbf{W}^+)^T\mathbf{q}_k + \mathbf{q}_k \bigodot \left((\mathbf{W}^-)^T\mathbf{q}_k\right) \qquad (3.60)$$

where operator $\bigodot$ denotes *element-wise multiplication*. In the above expression the $N \times 1$ vectors $\boldsymbol{\sigma}_{W^+}$ and $\boldsymbol{\sigma}_{W^-}$ are given by:

$$\boldsymbol{\sigma}_{W^+} = \mathbf{W}^+\mathbf{1} \qquad (3.61)$$

$$\boldsymbol{\sigma}_{W^-} = \mathbf{W}^-\mathbf{1} \qquad (3.62)$$

where $\mathbf{1}$ is a $N \times 1$ vector of all ones, $\mathbf{1} = [1, ..., 1]^T$. This definition implies that ith element of $\boldsymbol{\sigma}_{W^+}$ or $\boldsymbol{\sigma}_{W^-}$ is equal to the sum of the elements belonging to the ith row of the associated matrix. Having computed $\hat{\mathbf{z}}$ and hence $\mathbf{z}$, we can now proceed with the computation of $\widetilde{\mathbf{z}} = \mathbf{B}^T\mathbf{z}$. If we define $\mathbf{z}^T = \left[\mathbf{z}_1^T, ..., \mathbf{z}_K^T\right]$, where $\mathbf{z}_k \in \mathbb{R}^{N\times 1}$,

and use Eq. (3.50) to expand matrix $\mathbf{B}$ we obtain:

$$\begin{aligned}\widetilde{\mathbf{z}} &= \mathbf{B}^T\mathbf{z} = \begin{bmatrix} \mathbf{B}_{+1}^T & \cdots & \mathbf{B}_{+K}^T \\ \mathbf{B}_{-1}^T & \cdots & \mathbf{B}_{-K}^T \end{bmatrix} \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_K \end{bmatrix} \\ &= \begin{bmatrix} \sum_{k=1}^K \mathbf{B}_{+k}^T\mathbf{z}_k \\ \sum_{k=1}^K \mathbf{B}_{-k}^T\mathbf{z}_k \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^K (\mathbf{C}_k^T\mathbf{z}_k + \mathbf{D}_{+k}^T\mathbf{z}_k) \\ \sum_{k=1}^K (\mathbf{C}_k^T\mathbf{z}_k + \mathbf{D}_{-k}^T\mathbf{z}_k) \end{bmatrix} \end{aligned} \tag{3.63}$$

It Appendix B.1.2 we derive expressions for the appearing terms $\mathbf{C}_k^T\mathbf{z}_k$, $\mathbf{D}_{+k}^T\mathbf{z}_k$ and $\mathbf{D}_{-k}^T\mathbf{z}_k$ and show that $\widetilde{\mathbf{z}}$ is given by:

$$\widetilde{\mathbf{z}} = \begin{bmatrix} \left(\sum_{k=1}^K (\mathbf{q}_k \bigodot \mathbf{z}_k)\right) \bigotimes \mathbf{1} - \sum_{k=1}^K (\mathbf{q}_k \bigotimes \mathbf{z}_k) \\ \left(\sum_{k=1}^K (\mathbf{q}_k \bigodot \mathbf{z}_k)\right) \bigotimes \mathbf{1} - \sum_{k=1}^K (\mathbf{q}_k \bigotimes (\mathbf{q}_k \bigodot \mathbf{z}_k)) \end{bmatrix} \tag{3.64}$$

where, operator $\bigotimes$ denotes the *Kronecker product*. When this operator is applied on $\mathbf{x} \in \mathbb{R}^{N\times 1}$ and $\mathbf{y} \in \mathbb{R}^{M\times 1}$ the result is:

$$\mathbf{x} \bigotimes \mathbf{y} = [x_1\mathbf{y}^T, ..., x_N\mathbf{y}^T]^T \in \mathbb{R}^{NM\times 1}$$

Having derived expressions to efficiently derive functions $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ we will now examine the computational complexity of this approach. For the computation of $\mathbf{Bw}$ the most costly operations are the evaluation of the matrix-vector products $(\mathbf{W}^+)^T\mathbf{q}_k$ and $(\mathbf{W}^-)^T\mathbf{q}_k$ that appear in vectors $\hat{\mathbf{z}}_k$ in Eq. (3.60). The time complexity of these operations is $O(N^2)$, as $2N^2$ operations are required (N^2 multiplications and N^2 additions), and as there are K such terms to be computed, the total complexity of evaluating $\mathbf{Bw}$ is $O(KN^2)$. For the computation of term $\mathbf{B}^T\mathbf{z}$, for each k we need to evaluate $\boldsymbol{\delta}_k = \mathbf{q}_k \bigodot \mathbf{z}_k$, $\mathbf{q}_k \bigotimes \mathbf{z}_k$ and $\mathbf{q}_k \bigotimes \boldsymbol{\delta}_k$ which take $O(N)$, $O(N^2)$ and $O(N^2)$ time respectively. In addition, summation of the latter two terms for all k requires $O(KN^2)$, as we need to perform K additions of N^2 elements. If we perform the required multiplications naively, then the computation of both $\mathbf{Bw}$ and $\mathbf{B}^T\mathbf{z}$ matrix-vectors products would require $O(2KN^3)$, as the dimensions of $\mathbf{B}$ are $KN \times 2N^2$, while the dimensions of $\mathbf{w}$ and $\mathbf{z}$ are $2N^2 \times 1$ and $KN \times 1$ respectively. Hence this approach provides an $O(N)$ complexity reduction compared to naive matrix-vector multiplication.

With respect to memory requirements, this approach involves the storage of the

necessary vectors i.e. matrices $\mathbf{W}^+$ and $\mathbf{W}^-$ which have N^2 elements, and matrix $\mathbf{Q}$ which have KN elements in total, as well as a small number of auxiliary vectors of the same dimensions. Naively storing $\mathbf{B}$ requires memory for $2KN^3$ elements which is $min\{KN, 2N^2\}$ times larger than the memory required for our approach.

In sum, the computational complexity of computing $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ is $O(KN^2)$. In terms of arithmetic operations, these functions require approximately, $4KN^2$ and $8KN^2$ operations respectively. Finally, the approach does not require the storage of additional matrices other than the necessary $\mathbf{W}^+$, $\mathbf{W}^-$ and $\mathbf{Q}$ which require $KN + 2N^2$ memory in total.

The second approach for the computation of $\nabla f(\mathbf{w})$

A second approach for the evaluation of functions $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ is based on computing (during the initialisation phase) the quantities $\mathbf{\Gamma} = \mathbf{B}^T\mathbf{B}$ and $\boldsymbol{\beta} = \mathbf{B}^T\mathbf{b}$. Then, functions $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ can be expressed with respect to these quantities as:

$$\begin{aligned} f(\mathbf{w}) &= \frac{1}{2}(\mathbf{B}\mathbf{w} - \mathbf{b})^T(\mathbf{B}\mathbf{w} - \mathbf{b}) \\ &= \frac{1}{2}\mathbf{w}^T\mathbf{B}^T\mathbf{B}\mathbf{w} - \mathbf{w}^T\mathbf{B}^T\mathbf{b} + \frac{1}{2}\mathbf{b}^T\mathbf{b} \\ &= \frac{1}{2}\mathbf{w}^T(\mathbf{\Gamma}\mathbf{w}) - \mathbf{w}^T\boldsymbol{\beta} + \frac{1}{2}\mathbf{b}^T\mathbf{b} \\ &= \mathbf{w}^T(\frac{1}{2}\mathbf{\Gamma}\mathbf{w} - \boldsymbol{\beta}) + \frac{1}{2}\mathbf{b}^T\mathbf{b} \end{aligned} \tag{3.65}$$

$$\begin{aligned} \nabla f(\mathbf{w}) &= \mathbf{B}^T\mathbf{B}\mathbf{w} - \mathbf{B}^T\mathbf{b} \\ &= \mathbf{\Gamma}\mathbf{w} - \boldsymbol{\beta} \end{aligned} \tag{3.66}$$

Based on the above expressions only the matrix-vector product $\mathbf{\Gamma}\mathbf{w}$ is required for their evaluation and hence at a particular point both functions can be computed by just evaluating $\mathbf{\Gamma}\mathbf{w}$. Notice that $\mathbf{B} \in \mathbb{R}^{KN\times 2N^2}$ and $\mathbf{\Gamma} \in \mathbb{R}^{2N^2\times 2N^2}$ so that the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ are depended both on K, N in the first approach and only on N in the second. Expansion of matrix $\mathbf{B}$ according to Eq. (3.50) yields:

$$\begin{aligned}\boldsymbol{\Gamma} &= \mathbf{B}^T\mathbf{B} = \begin{bmatrix} \mathbf{B}_{+1}^T & \cdots & \mathbf{B}_{+K}^T \\ \mathbf{B}_{-1}^T & \cdots & \mathbf{B}_{-K}^T \end{bmatrix} \begin{bmatrix} \mathbf{B}_{+1} & \mathbf{B}_{-1} \\ \vdots & \vdots \\ \mathbf{B}_{+K} & \mathbf{B}_{-K} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{k=1}^K \mathbf{B}_{+k}^T\mathbf{B}_{+k} & \sum_{k=1}^K \mathbf{B}_{+k}^T\mathbf{B}_{-k} \\ \sum_{k=1}^K \mathbf{B}_{-k}^T\mathbf{B}_{+k} & \sum_{k=1}^K \mathbf{B}_{-k}^T\mathbf{B}_{-k} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Gamma}_{11} & \boldsymbol{\Gamma}_{12} \\ \boldsymbol{\Gamma}_{21} & \boldsymbol{\Gamma}_{22} \end{bmatrix} \end{aligned} \tag{3.67}$$

In order to derive expressions for the comprising terms of $\boldsymbol{\Gamma}_{lm}$, $l, m = 1, 2$, we substitute Eqs. (3.51) and (3.52), associated with $\mathbf{B}_{+k}$ and $\mathbf{B}_{-k}$, into Eq. (3.67) to obtain:

$$\boldsymbol{\Gamma}_{11} = \sum_{k=1}^K \mathbf{C}_k^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{C}_k^T\mathbf{D}_{+k} + \sum_{k=1}^K \mathbf{D}_{+k}^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{D}_{+k}^T\mathbf{D}_{+k} \tag{3.68}$$

$$\boldsymbol{\Gamma}_{12} = \sum_{k=1}^K \mathbf{C}_k^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{D}_{+k}^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{C}_k^T\mathbf{D}_{-k} + \sum_{k=1}^K \mathbf{D}_{+k}^T\mathbf{D}_{-k} \tag{3.69}$$

$$\boldsymbol{\Gamma}_{21} = \sum_{k=1}^K \mathbf{C}_k^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{C}_k^T\mathbf{D}_{+k} + \sum_{k=1}^K \mathbf{D}_{-k}^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{D}_{-k}^T\mathbf{D}_{+k} \tag{3.70}$$

$$\boldsymbol{\Gamma}_{22} = \sum_{k=1}^K \mathbf{C}_k^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{C}_k^T\mathbf{D}_{-k} + \sum_{k=1}^K \mathbf{D}_{-k}^T\mathbf{C}_k + \sum_{k=1}^K \mathbf{D}_{-k}^T\mathbf{D}_{-k} \tag{3.71}$$

In Appendix B.2.1 we examine each of the terms of $\boldsymbol{\Gamma}_{lm}$ and derive expressions that can be used for the efficient computation of $\boldsymbol{\Gamma}\mathbf{w}$. Our analysis shows that in order to reproduce these terms we only need to store five vectors/matrices that can be computed during the initialisation phase. Specifically, we define vector $\boldsymbol{\sigma}_q \in \mathbb{R}^{N\times 1}$ and matrices $\mathbf{M} \in \mathbb{R}^{N\times N}, \mathbf{M}^s \in \mathbb{R}^{N\times N}, \mathbf{R}^i \in \mathbb{R}^{N\times N}$ and $\mathbf{R}^{s,i} \in \mathbb{R}^{N\times N}, i = 1, ..., N$ with elements:

$$\sigma_q(i) = \sum_{k=1}^K q_{ik}^2 \tag{3.72}$$

$$M_{i,j} = \sum_{k=1}^K q_{ik}q_{jk} \tag{3.73}$$

$$M_{i,j}^{s} = \sum_{k=1}^{K} q_{ik}^{2} q_{jk} \tag{3.74}$$

$$R_{j,l}^{i} = \sum_{k=1}^{K} q_{ik} q_{jk} q_{lk} \tag{3.75}$$

$$R_{j,l}^{s,i} = \sum_{k=1}^{K} q_{ik} q_{jk} q_{lk}^{2} \tag{3.76}$$

Let us now examine the derivation of $\mathbf{\Gamma w}$. We have that:

$$\overline{\mathbf{z}} = \mathbf{\Gamma w} = \begin{bmatrix} \mathbf{\Gamma}_{11} & \mathbf{\Gamma}_{12} \\ \mathbf{\Gamma}_{21} & \mathbf{\Gamma}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{w}^{+} \\ \mathbf{w}^{-} \end{bmatrix} = \begin{bmatrix} \mathbf{\Gamma}_{11}\mathbf{w}^{+} + \mathbf{\Gamma}_{12}\mathbf{w}^{-} \\ \mathbf{\Gamma}_{21}\mathbf{w}^{+} + \mathbf{\Gamma}_{22}\mathbf{w}^{-} \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{z}}_{1} \\ \overline{\mathbf{z}}_{2} \end{bmatrix} \tag{3.77}$$

where vectors $\mathbf{w}^{+}$and $\mathbf{w}^{-}$ have already been defined in the first approach, while vectors $\overline{\mathbf{z}}_{l} = \mathbf{\Gamma}_{l1}\mathbf{w}^{+} + \mathbf{\Gamma}_{l2}\mathbf{w}^{-}$, $\overline{\mathbf{z}}_{l} \in \mathbb{R}^{N^2 \times 1}, l = 1, 2$ can be further decomposed into $\overline{\mathbf{z}}_{l}^{T} = [\overline{\mathbf{z}}_{l1}^{T}, ..., \overline{\mathbf{z}}_{lN}^{T}]$ with elements $\overline{\mathbf{z}}_{li} \in \mathbb{R}^{N \times 1}$. In order to obtain low complexity expressions for these terms, we take advantage of the expressions derived for the composing matrices of $\mathbf{\Gamma}_{lm}$, and of Eqs. (3.72)-(3.76). As shown in Appendix B.2.2 terms $\overline{\mathbf{z}}_{1i}$ and $\overline{\mathbf{z}}_{2i}$, $i = 1, ..., N$ are given by the following expressions:

$$\overline{\mathbf{z}}_{1i} = \sigma_{\overline{z}}(i)\mathbf{1} - \mathbf{m}_{i}^{c} \bigodot (\boldsymbol{\sigma}_{W^{+}} + \boldsymbol{\sigma}_{W^{-}}) + (\mathbf{W}^{+})^{T}\mathbf{m}_{i}^{r} - (\mathbf{R}^{i} \bigodot \mathbf{W}^{-})^{T}\mathbf{1} \tag{3.78}$$

$$\overline{\mathbf{z}}_{2i} = \sigma_{\overline{z}}(i)\mathbf{1} + \mathbf{m}_{i}^{s,c} \bigodot (\boldsymbol{\sigma}_{W^{+}} + \boldsymbol{\sigma}_{W^{-}}) - (\mathbf{R}^{i} \bigodot \mathbf{W}^{+})^{T}\mathbf{1} + (\mathbf{R}^{s,i} \bigodot \mathbf{W}^{-})^{T}\mathbf{1} \tag{3.79}$$

where vectors $\boldsymbol{\sigma}_{W^{+}}$ and $\boldsymbol{\sigma}_{W^{-}}$ have already been defined in the first approach, vectors $\mathbf{m}_{i}^{c}$, $\mathbf{m}_{i}^{r} \in \mathbb{R}^{N \times 1}$ are the ith column and row of matrix $\mathbf{M}$, while the vector $\boldsymbol{\sigma}_{\overline{z}} \in \mathbb{R}^{N \times 1}$ is defined as:

$$\boldsymbol{\sigma}_{\overline{z}} = \boldsymbol{\sigma}_{q} \bigodot (\boldsymbol{\sigma}_{W^{+}} + \boldsymbol{\sigma}_{W^{-}}) - (\mathbf{M} \bigodot (\mathbf{W}^{+})^{T})\mathbf{1} + (\mathbf{M}^{s} \bigodot (\mathbf{W}^{-})^{T})\mathbf{1} \tag{3.80}$$

Let us now examine the computational complexity of computing $\overline{\mathbf{z}}$ which corresponds to the computational complexity of computing $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$. As mentioned above, it is required to compute a number of vectors/matrices given by Eqs. (3.72)-(3.76) in the initialisation phase. Specifically, the computation of vector $\boldsymbol{\sigma}_{q}$ requires $O(KN)$ operations, the computation of matrices $\mathbf{M}$ and $\boldsymbol{M}^{s}$ $O(KN^2)$ operations while the derivation of matrices $\mathbf{R}^{i}$ and $\mathbf{R}^{s,i}$, $i = 1, ..., N$ are the most

expensive requiring $O(KN^3)$ multiplications; nevertheless, these terms are computed once. Furthermore, every time we need to obtain the gradient we have to compute vectors $\overline{\mathbf{z}}_{1i}$ and $\overline{\mathbf{z}}_{2i}$, for $i = 1, ..., N$ given by Eqs. (3.78) and (3.79), as well as vector $\boldsymbol{\sigma}_{\overline{z}}$ according to Eq. (3.80). The most expensive operations for the computation of $\boldsymbol{\sigma}_{\overline{z}}$ are $(\mathbf{M} \bigodot (\mathbf{W}^+)^T)\mathbf{1}$ and $(\mathbf{M}^s \bigodot (\mathbf{W}^-)^T)\mathbf{1}$ each requiring $O(N^2)$ operations for the computation of the matrix element-wise products and $O(N^2)$ for the multiplication of the resulting matrix by $\mathbf{1}$. Hence, the computational complexity of $\boldsymbol{\sigma}_{\overline{z}}$ is $O(N^2)$. The most computationally expensive terms for the computation of $\overline{\mathbf{z}}_{1i}$ are the last two terms $(\mathbf{W}^+)^T\mathbf{m}_i^r$ and $(\mathbf{R}^i \bigodot \mathbf{W}^-)^T\mathbf{1}$ which are of complexity $O(N^2)$. Similarly, for the evaluation of $\overline{\mathbf{z}}_{2i}$ the most expensive terms are $(\mathbf{M} \bigodot (\mathbf{W}^+)^T)\mathbf{1}$ and $(\mathbf{M}^s \bigodot (\mathbf{W}^-)^T)\mathbf{1}$, which are of the same complexity. Hence, the computation of either $\overline{\mathbf{z}}_1$ or $\overline{\mathbf{z}}_2$ is $O(N^3)$ as there are N terms $\overline{\mathbf{z}}_{li}$, for $l = 1, 2$. Hence, the time complexity of evaluating $\boldsymbol{\Gamma}\mathbf{w}$ is $O(N^3)$; in terms of arithmetic operations this matrix-vector product requires approximately $8N^3$ multiplications or additions. If we perform the required multiplications naively, then the computation of $\boldsymbol{\Gamma} = \mathbf{B}^T\mathbf{B}$ and $\boldsymbol{\Gamma}\mathbf{w}$ are of time complexity $O(2N^2 \times KN \times 2N^2)$ or $O(KN^5)$, and $O(2N^2 \times 2N^2 \times 1)$ or $O(N^4)$ respectively. Hence, our approach achieves an $O(N^2)$ complexity reduction of the initialisation phase and an $O(N)$ complexity reduction per objective function or gradient evaluation.

In terms of memory requirements, this approach requires the storage of $\boldsymbol{\sigma}_q$, $\mathbf{M}$, $\boldsymbol{M}^s$, $\mathbf{R}^i$ and $\mathbf{R}^{s,i}$ for i=1,...,N apart from the necessary $\mathbf{W}^+$, $\mathbf{W}^-$ and $\mathbf{q}_k$, $k = 1, ..., K$. As each matrix $\mathbf{R}^i$ or $\mathbf{R}^{s,i}$ has N^2 elements, the total storage space required for this approach is $O(N^3+KN)$ which is limiting for large values of N. As a result, this approach is more suitable for cases that $K > N$ and N is small enough so that we can store at least $2N^3 + KN$ elements. Notice that if the above matrices are not used, then $\boldsymbol{\Gamma}$ requires the storage of $4N^4$ elements.

In sum, the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ has time complexity $O(KN^3)$ for initialisation and $O(N^3)$ per evaluation and requires approximately $4KN^3$ arithmetic operations for initialisation and $8N^3$ operations for evaluating each or both functions, while it requires the storage of approximately $O(2N^3 + KN)$ elements.

3.3.4. Computational complexity

Having derived efficient expressions for the computation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ we will now examine the computational complexity of the PGNNLS and RNN-NNLS

algorithms.

To examine the computational complexity of PGNNLS, we must count the number of objective function and gradient evaluations required at each iteration of the algorithm. At the start of iteration τ, the functions $f(\mathbf{w}^\tau)$ and $\nabla f(\mathbf{w}^\tau)$ need to be computed once; additionally, the step-size trial of the line search procedure requires the evaluation of $f(\mathbf{w}_{cand}^{\tau+1})$. If the total number of iterations performed with the PGNNLS algorithm is NI_{PGNNLS} and the average number of trials is NT_{avg} then the total time complexity of the PGNNLS algorithm is equal to:

$$NI_{NNLS} \times (complexity(\nabla f(\mathbf{w})) + NT_{avg} complexity(f(\mathbf{w}))) \tag{3.81}$$

where $complexity(f(\mathbf{w}))$ and $complexity(\nabla f(\mathbf{w}))$ denote the time complexity for computing the cost function and the gradient respectively. The reason for not including the evaluation of $f(\mathbf{w})$ at the start of each iteration is that when the gradient is evaluated at one point the cost function can be computed with no additional cost. Due to the efficiency of the employed hyper-exponential line-search, NT_{avg} is a small number which in our experiments was always smaller than 3.

According to expression (3.81), the number of arithmetic operations performed by the PGNNLS algorithm when the first approach proposed in section 3.3.3 is used for the evaluation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ is equal to:

$$NI_{NNLS} \times (8KN^2 + NT_{avg} 4KN^2) \tag{3.82}$$

If the second approach proposed in section 3.3.3 is used for the evaluation of $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$ the number of arithmetic operations becomes:

$$4KN^3 + NI_{NNLS} \times (8N^3 + NT_{avg} 8N^3) \tag{3.83}$$

We should highlight once more that the second approach is more appropriate when $K > N$ and N is small enough so that we can store at least $2N^3 + KN$ elements.

If we consider the computational complexity of Algorithm 3.5, the main computational task at each iteration is the execution of the PGNNLS algorithm. The computation needed for the solution of Eqs. (2.6)-(2.8) for all patterns is of the order $O(KN^2)$ which is negligible compared to the complexity of the PGNNLS algorithm: the cost of the first approach (expression (3.82)) is $NI_{NNLS} \times NT_{avg}$

times higher than $O(KN^2)$, while for the second approach even the initialisation cost is higher $O(KN^3)$. As a result the computational complexity of the proposed RNN-NNLS algorithm is:

$$NI_{RNN-NNLS} \times NI_{NNLS} \times (complexity(\nabla f(\mathbf{w})) + NT_{avg} complexity(f(\mathbf{w}))) \quad (3.84)$$

Because $NI_{RNN-NNLS}$ is small, the complexity of the RNN-NNLS algorithm is not larger than the complexity of the standard RNN supervised learning algorithm which is equal to $O(KN^3)$ per iteration. As a result RNN-NNLS can also serve as a weight initialisation algorithm for the standard RNN supervised learning algorithm.

3.4. Conclusions

In this chapter we have proposed two supervised learning algorithms related to the RNN. The first, is a gradient descent learning algorithm for the RNN with synchronised interactions. We have derived the steps of the algorithm and studied the computational complexity of the approach showing that it is the same with the standard RNN learning algorithm, although the RNNSI model is a generalisation of RNN. We have also proposed an improved version of the RNNSI gradient descent algorithm that reduces the computational complexity of the method by an order of magnitude; the approach followed for the RNNSI can also be used to reduce the complexity of the RNN gradient descent learning algorithm.

The second is a learning algorithm for the RNN that is mostly suitable for problems where the ratio of the number of output to the total number of neurons is large. This approach is based on modelling the signal-flow equations of the network as a nonnegative least squares problem which can be accomplished when all neurons have desired values. For its solution, we have developed PGNNLS, a large-scale projected gradient NNLS algorithm that employs the limited-memory BFGS formula to update the search direction. PGNNLS also combines two advanced line-search procedures to obtain an appropriate step-size in each iteration. We further examine how to efficiently obtain the cost and gradient NNLS functions which are the most costly operations in the PGNNLS algorithm. We propose two different approaches for their efficient evaluation that reduce the execution time and storage requirements of the algorithm by one or two orders of magnitude. We also develop, RNN-NNLS algorithm to deal with the case that the network is composed of

both output and non-output neurons. In RNN-NNLS we iteratively solve an NNLS problem and the RNN signal-flow equations with respect to the NNLS solution and modify accordingly the weights and the desired output neuron values. The obtained weights can serve either as the final trained weights or as a good initialisation point of an RNN supervised learning algorithm.

So far the performance of the developed learning algorithms has not been assessed experimentally. This is done in the next chapter, where a combinatorial optimization problem is considered and the learning algorithms are utilised for the off-line training of a random neural network in order to provide fast close to optimal decisions to the problem considered.

4. Assignment of emergency units to incidents

In this chapter, we start the discussion of emergency management optimisation problems by considering the assignment of emergency units to incidents. In this problem, the assignment of emergency units to incidents (AEUI), a number of emergency incidents have taken place simultaneously and there are a number of injured civilians at each incident. The objective is to dispatch the available emergency units to the incidents, which are spatially distributed, in order to collect as many people as possible and also minimise the average response time to the incidents.

Although formulation of this problem can be easily achieved through integer programming, its optimal solution relies on time-consuming non-polynomial time algorithms. Nevertheless the problem at hand needs to be solved rapidly, with limited computational resources, and preferably in real-time, so that heuristic solutions are the approach of choice.

We propose the use of random neural networks in a supervised learning context for the solution of this assignment problem. The idea is to train the RNNs using several problem instances in the same physical context as the emergency, and then use the trained system for real-time decision making. Specifically, we will consider both the gradient descent RNNSI learning algorithm and the RNN-NNLS approach, developed in Chapter 3, to assess the performance of supervised learning algorithms for the examined problem. Additionally, we will be able to examine the computational performance of several aspects of the developed learning algorithms, especially of RNN-NNLS, such as the speedup from the use of analytical expressions for the function and gradient evaluation of the NNLS problem, the benefit from using the hyper-exponential line search procedure and the convergence properties of the algorithm.

The structure of the chapter is as follows: In section 4.1 we start with the de-

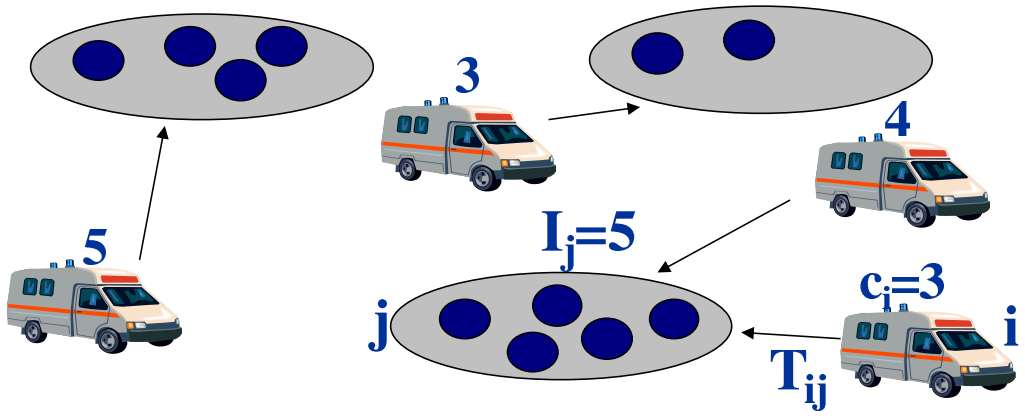


Figure 4.1.: Dispatching of emergency units to locations of injured civilians

scription and mathematical formulation of the problem followed by the proposed solution approach. In sections 4.3 and 4.4 we empirically evaluate the performance of the RNNSI and RNN-NNLS learning algorithms for the solutions of AEUI . Next, the performance of RNN-NNLS as an initialisation algorithm is examined. Section 4.6 is a summary of the chapter.

4.1. Problem description

Consider that N_L incidents occur simultaneously at different locations with I_j people injured at incident j. N_U emergency units or ambulances (say) are spatially distributed before the time of the incident with unit i being able to collect $c_i > 0$ injured and having response time to incident j given by $T_{ij} > 0$ as shown in Fig. 4.1. We also assume that decisions are irrevocable so that after a unit is allocated to some incident, it cannot be re-assigned to some other incident. An additional requirement is to have no central control unit so that decisions are taken by the emergency units in a distributed but globally consistent manner.

If the capacity of the ambulances is sufficient to collect all the injured, then our goal is not only to collect the injured, but also to minimise the average response time of the ambulances; this is essential to ensure the quick collection and treatment of the civilians. If we want to state the problem in mathematical terms then our

goal is to find the optimal allocation matrix $\mathbf{X}$ with elements:

$$X_{ij} = \begin{cases} 1 & \text{if unit } i \text{ is allocated to incident } j \\ 0 & \text{if unit } i \text{ is not allocated to incident } j \end{cases} \tag{4.1}$$

so that the total response time to the incidents is minimised

$$min\ f(\mathbf{X}) = \sum_{i=1}^{N_U}\sum_{j=1}^{N_L} T_{ij}X_{ij} \tag{4.2}$$

subject to the constraints:

$$\sum_{j=1}^{N_L} X_{ij} = 1 \quad \forall i \tag{4.3}$$

$$\sum_{i=1}^{N_U} c_i X_{ij} \geq I_j \quad \forall j \tag{4.4}$$

$$X_{ij} \in \{0, 1\} \quad \forall i, j \tag{4.5}$$

Constraint (4.3) indicates that an emergency unit must be allocated to exactly one incident, while (4.4) expresses the fact that the total capacity of the units allocated to an incident must be at least equal to the number of people injured there. The above problem is NP-hard in the strong sense since it is a generalisation of the 0-1 Multiple Knapsack Problem which is of the same complexity class [136].

This means that no known algorithms exist to solve the problem in polynomial time. Optimal solution can be achieved by enumeration or integer programming algorithms, such as branch and bound algorithms, which may potentially search all possible combinations. However, the search space increases exponentially with respect to the problem size so that these algorithms are not of polynomial time. Because AEUI needs to be solved rapidly with limited computational resources and preferably in real-time, we cannot resort to integer programming methods; for this reason we rely upon heuristic algorithms that can provide fast and close to optimal solutions to the problem. In the next section we discuss a heuristic method based on supervised learning that will be used to obtain fast and distributed decision making, as well as close to optimal results.

4.2. Supervised learning solution approach

The approach taken for the solution of AEUI problem is to train a random neural network using numerous instances of the optimisation problem, with exact solutions which are obtained off-line. Then, if a problem instance is presented to the trained neural network, it will be able to provide a solution that is close to optimal, due to its ability to generalise well. As a result, the trained RNN can be "handed out" to all decision agents (emergency units) to serve as an "oracle" for decision making. When the emergency happens, each individual agent uses its "oracle" to obtain fast and distributed decisions. Since all agents have the same "oracle", if they have the same information there will be no conflicts in their decisions; the "oracle" provides the same allocation matrix $\mathbf{X}$ to each agent, so that agent i is allocated to incident j' with $X_{ij'} = 1$. Information about the emergency (whereabouts of the incidents, estimate of the number of victims) can be broadcast to the agents if they do not have access to the information. If the decision agents collect the information about the event themselves, then they can exchange any locally available data, so that all agents share the same information during the decision stage.

Fast decision making can be achieved because the RNN signal-flow equations can be solved by an analytical polynomial time algorithm which scales quadratically to the network size. Because the solution to the RNN signal-flow equations always exists and it is unique, decision consistency is accomplished. It is important to mention, that consistent decision making cannot be achieved by all heuristic methods for combinatorial optimization problems. A large number of methods such as simulated annealing [113], genetic algorithms [94] and chaotic Hopfield neural networks [118], rely on stochastic search procedures which may result in different decisions for each agent. Furthermore, even if the same random number generator is used at different agents, so that the agent outcomes are the same at all iterations, there is no way to ensure that a good solution will be found fast enough.

Fixing the T_{ij} and c_i parameters, the problem can be mapped to a supervised learning context by representing the inputs to the network by I_j and the outputs by X_{ij}. Because $I_j \geq 0 \; \forall j$, in the RNN they will be represented by the parameters $\Lambda(j)$ of the input neurons. The output variables are associated with the excitation probabilities of output neurons. Specifically, output neuron with index (i, j) represents decision variable X_{ij}. During the training phase, we represent decision $X_{ij} = 1$ with $q_{(i,j)} = 1 - \epsilon_q$, $0 \leq \epsilon_q < 1/2$ and decision $X_{ij} = 0$ with $q_{(i,j)} = \epsilon_q$.

During the testing phase, if the value of the particular neuron is $q_{(i,j)} > 0.5$ then we assume that $X_{ij} = 1$, otherwise we take $X_{ij} = 0$.

With respect to the number of hidden neurons, we consider three different configurations: (a) there are no hidden neurons (b) the number of hidden neurons is equal to the number of output neurons, and (c) the number of hidden neurons is twice the number of output neurons. Furthermore, we always assume that our network is fully connected in terms of the $\mathbf{W}^+$ and $\mathbf{W}^-$ weight matrices, while for the weight matrix $\mathbf{A}$ we take either that it is fully connected (RNNSI case) or that $\mathbf{A} = \mathbf{0}$ (RNN case).

For the solution of the problem we considered two general NN architectures. In the "collective" NN architecture we construct a single neural network for all decisions which is comprised of $N_O = N_U N_L$ output neurons. As the output of the neural network provides the actions for all agents, each agent only performs the action corresponding to him/her. In the "individual" NN architecture we construct and train a different NN for each agent's decision, so that we need to train N_U architectures of N_L output neurons. In this case, the ith NN is trained using as outputs only the variables $X_{ij}, j = 1, ..., N_L$ to advise agent i. Despite the fact that each NN provides a single action, decision making is still consistent because training is performed using the optimal solutions to the problem instances which are globally consistent.

To train the NNs, we have first generated at random 1000 problem instances for different numbers of emergency units and locations of incidents. The remaining parameters have been chosen at random with $T_{ij} = U(0, 1)$ and $c_i = U^{int}(1, 4)$, where $U(a, b)$ and $U^{int}(a, b)$ represent real and integer numbers generated from the uniform distribution in the interval $[a, b]$ respectively. For each problem instance, the number of injured at location I_j is also chosen from the uniform distribution such that: $I_j = U^{int}(0.5c_t/N_L, c_t/N_L)$, where $c_t = \sum_i c_i$ is the total capacity of the emergency units.

To evaluate the performance of the RNNSI and RNN-NNLS learning algorithms (the results are presented in subsequent sections) we have performed experiments with the following numbers of emergency units and incidents: $N_U = \{5, 10, 15, 20\}$ and $N_L = \{3, 5\}$. Among the test cases considered, we only chose those whose required capacity was within the total available capacity of the emergency units. The optimal solution in each case was then obtained accurately by solving the com-

binatorial optimisation problem in Matlab using function *bintprog* which employs a branch and bound procedure for the solution of binary combinatorial optimisation problems.

Testing after training was performed using a distinct but similarly generated set of 250 test cases so that the training and testing were disjoint, but with the same probability distributions for all parameters.

The effectiveness of the learning algorithms was evaluated on the basis of the following metrics:

- The percentage of instances that were solved so that all of the injured were evacuated
- The percentage of people collected averaged over all testing instances
- The average relative percentage deviation from the optimal, σ_{opt},which evaluates the closeness of the solution to optimality, taken over the number of problem instances that the emergency units covered all casualties N_F, defined as:

$$\sigma_{opt} = \frac{1}{N_F} \sum_{i=1}^{N_F} \frac{f_{NN}^i(\mathbf{X}) - f_{opt}^i(\mathbf{X})}{f_{opt}^i(\mathbf{X})} \times 100 \tag{4.6}$$

  where $f_{NN}^i(\mathbf{X})$ and $f_{opt}^i(\mathbf{X})$ are the cost function values obtained from the heuristic neural network learning algorithm and the exact algorithm for instance i respectively.

In the following sections we present the results obtained from the RNNSI and RNN-NNLS learning algorithms for the neural network architectures describe above.

4.3. Performance evaluation of the RNNSI learning algorithm

To solve AEUI using the RNNSI tool, we have employed the algorithm developed in section 3.2. For training we have used the input-output mapping between parameters of the optimisation problem and the RNNSI discussed in the previous section. We have also normalised the inputs of the RNNSI so that $\Lambda(i) \in [0.2, 1]$, while for the output neurons we have chosen $\epsilon_q = 1/3$, so that "low" and "high" neurons take values $1/3$ and $2/3$ respectively. In order to select the best set of weights for each

data set, $\mathbf{W}^+$, $\mathbf{W}^-$ and $\mathbf{A}$, we perform 2000 iterations and check every 10 iterations the quality of the weights for the training set (chosen based on empirical experience); the weights selected are those producing the best results for the training set in terms of percentage of problem instances were all injured have been collected.

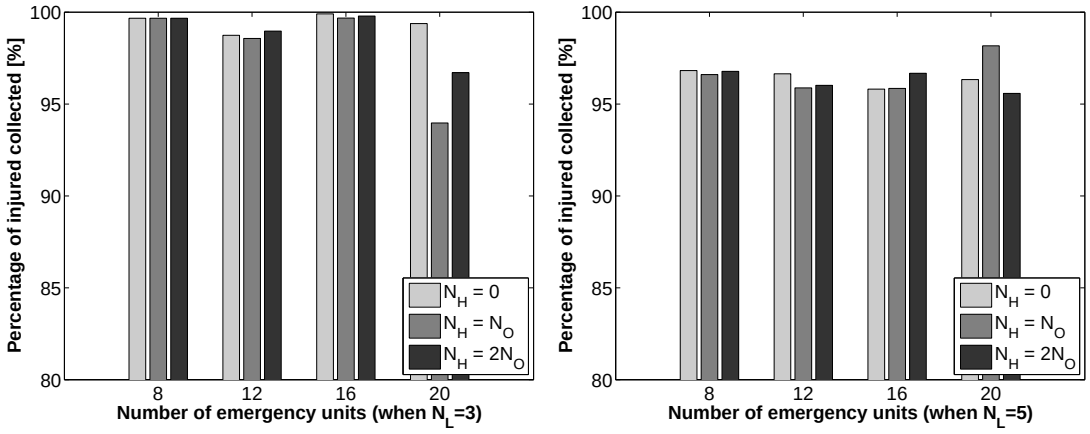
Figs. 4.2 and 4.3 illustrate the results obtained for the "collective" NN architecture for varying number of hidden neurons when all synchronised interaction weights are trained (RNNSI case) and when they are set to zero (RNN case) respectively. As can be seen from Fig. 4.2 the performance of the RNNSI learning algorithm is similar for different numbers of hidden neurons except for the ($N_U = 20$, $N_L = 3$) and ($N_U = 20$, $N_L = 5$) cases where having 0 and N_O hidden neurons respectively, produce the best results for all solution quality metrics considered. Furthermore, in all cases the percentage of injured collected is more than 95%, while for a large percentage of problem instances complete collection is achieved especially for $N_L = 3$. Additionally, the best average relative percentage deviation from optimality achieved for different cases is less than 3.5%.

The results obtained with the RNN learning algorithm depicted in Fig. 4.3 are of similar and slightly better quality to those of Fig. 4.2 for small problem instances, while for large problems with RNNSI model produces better results. Similar to the previous case, no clear conclusion can be drawn on the best number of hidden neurons to use in the neural network architecture to achieve the best collection results. The only conclusion that can be drawn, is that the RNN architecture with no hidden neurons achieves the best results in terms of the response time to the incidents.

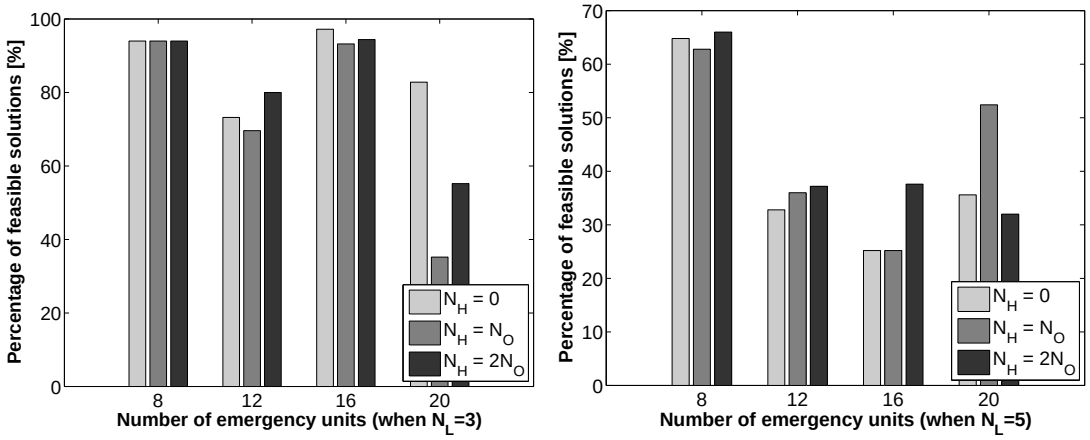
Figs. 4.4 and 4.5 illustrate the results obtained for different numbers of hidden neurons for the "individual" RNNSI and RNN architectures respectively. Concerning the RNNSI case, this architecture produces similar results in terms of the first two solutions metrics and better results for the σ_{opt} metric, as in all cases the best result is at most within 1% from the optimal solution. This architecture is also more robust in the sense that the performance for all three metrics is almost identical irrespective of the number of hidden neurons.

On the contrary, the performance of the "individual" RNN architecture fluctuates significantly for different number of hidden neurons for all three metrics, while its overall performance is worse than the other cases considered so far.

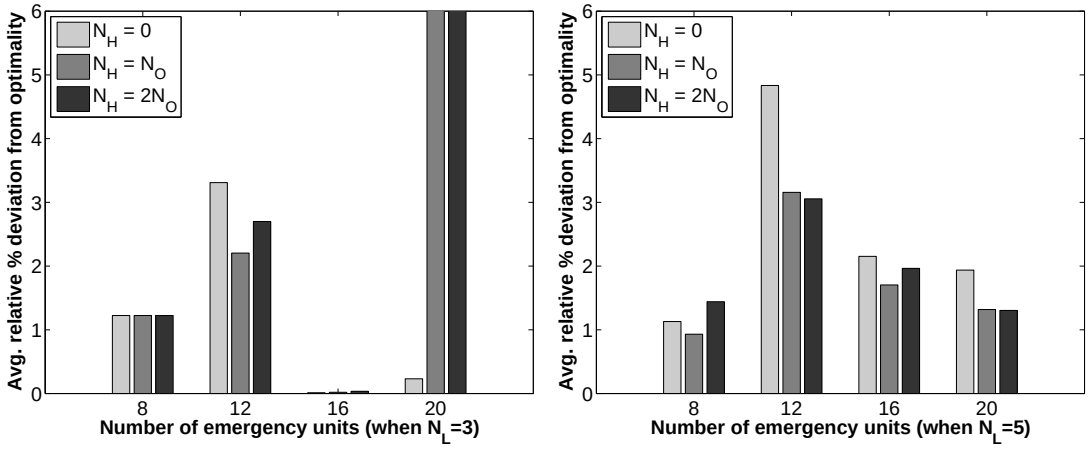
Fig. 4.6 depicts the best results obtained for each (N_U, N_L) pair for the two



(a) Percentage of injured that are collected

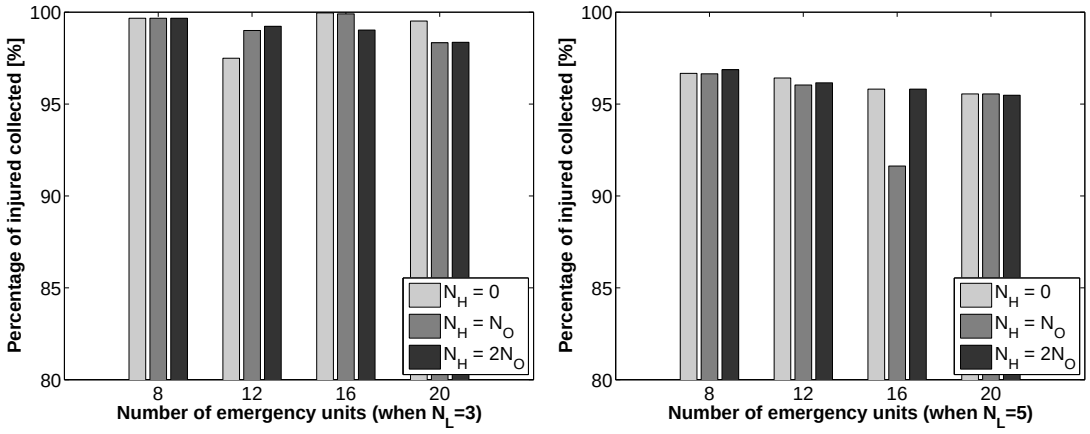


(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term

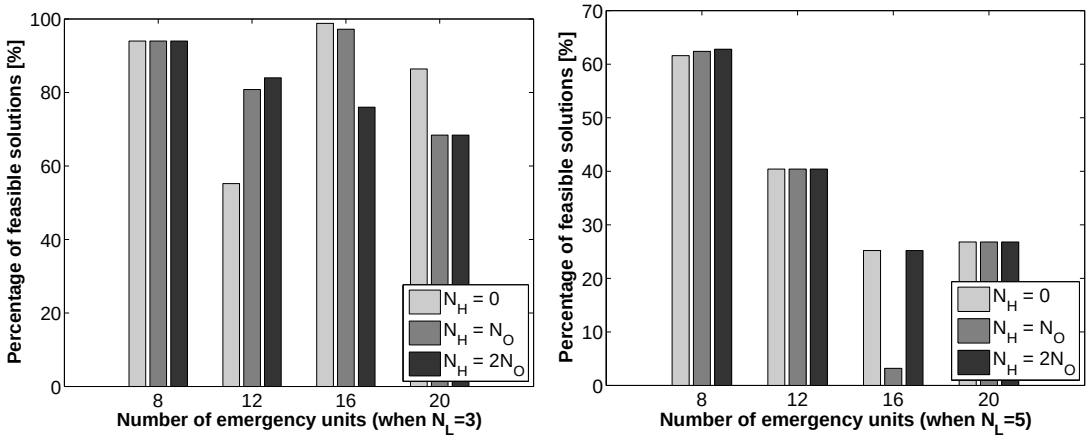


(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

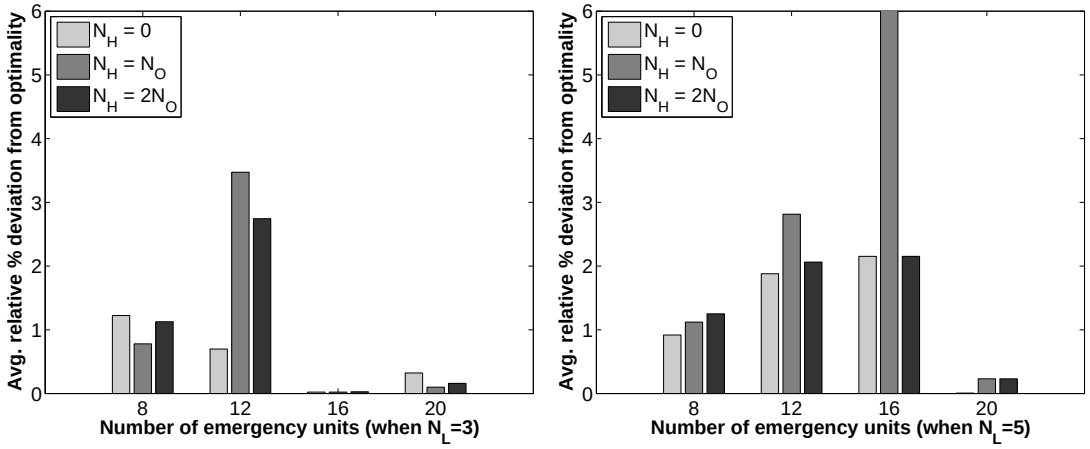
Figure 4.2.: Performance of the RNNSI gradient descent algorithm for the "collective" neural network architecture



(a) Percentage of injured that are collected

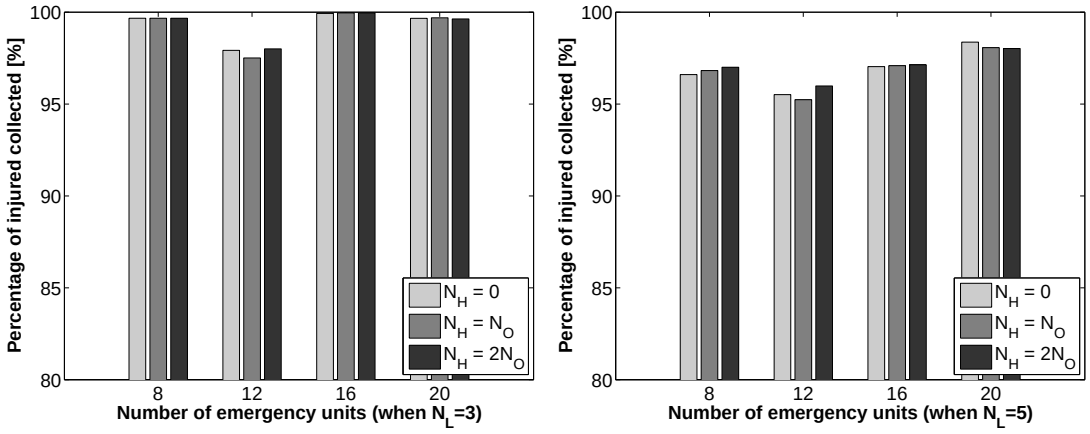


(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term

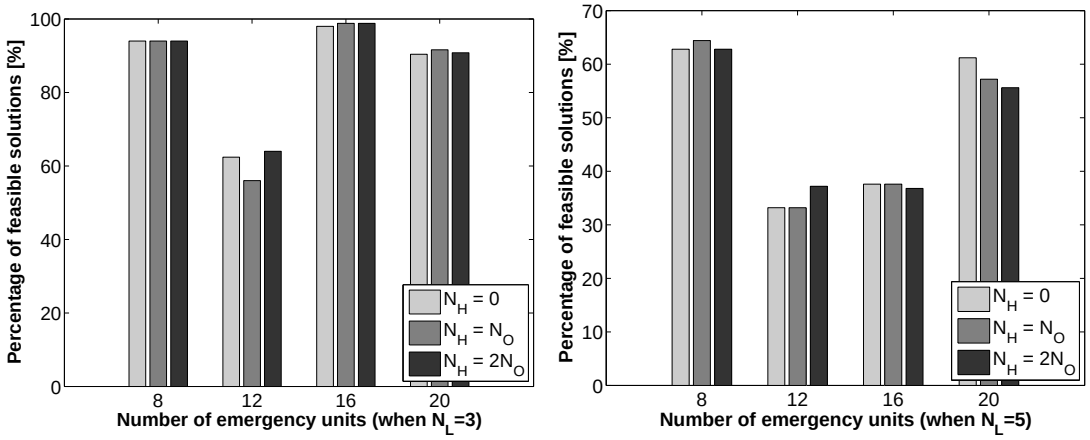


(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

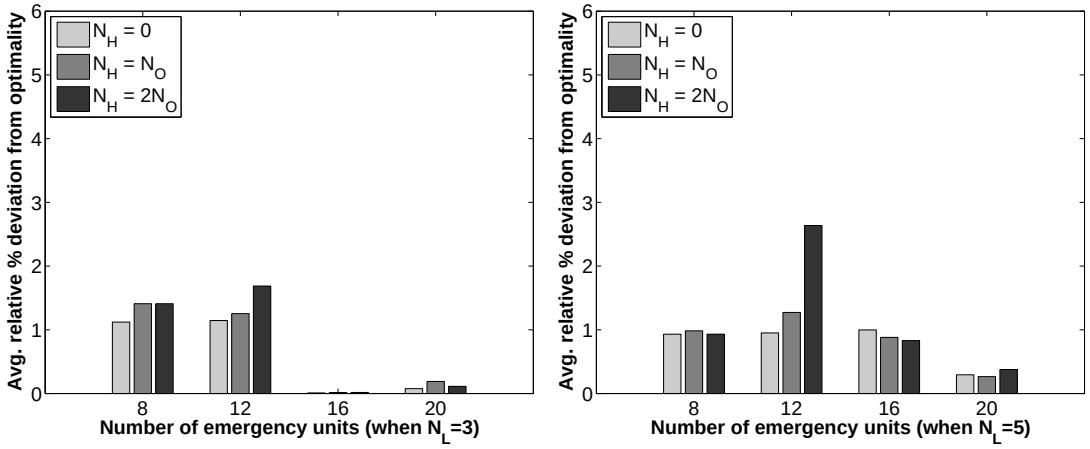
Figure 4.3.: Performance of the RNN gradient descent algorithm for the "collective" neural network architecture



(a) Percentage of injured that are collected

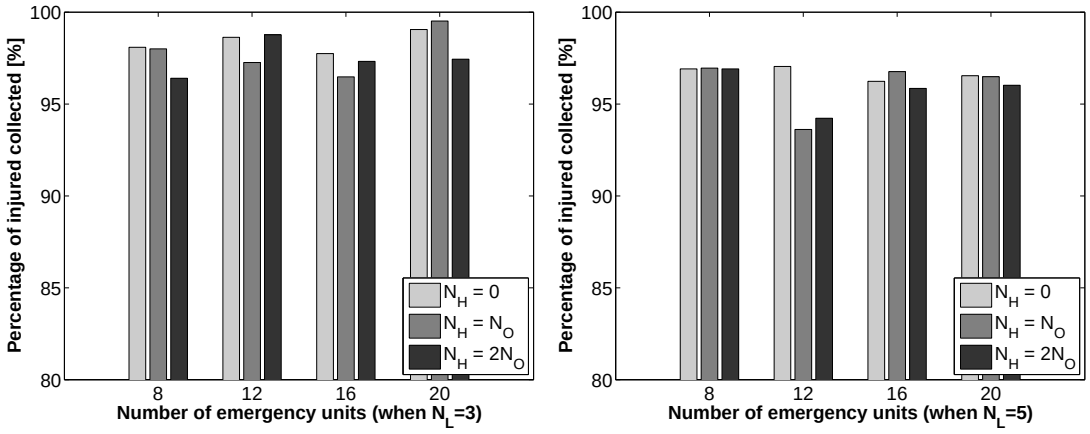


(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term

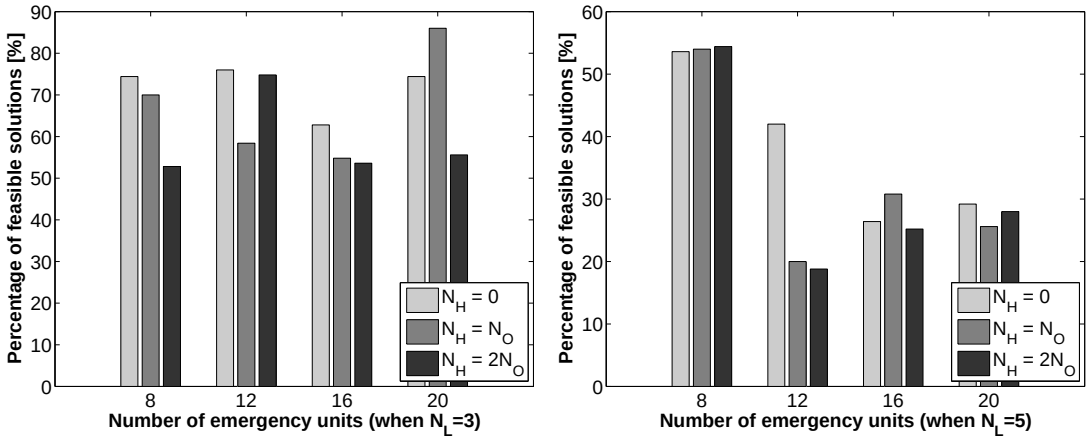


(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

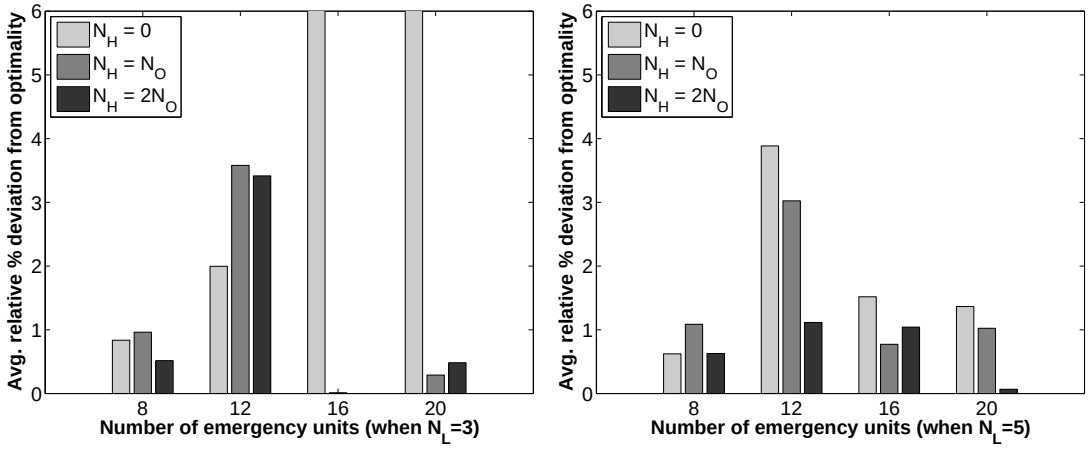
Figure 4.4.: Performance of the RNNSI gradient descent algorithm for the "individual" neural network architecture



(a) Percentage of injured that are collected

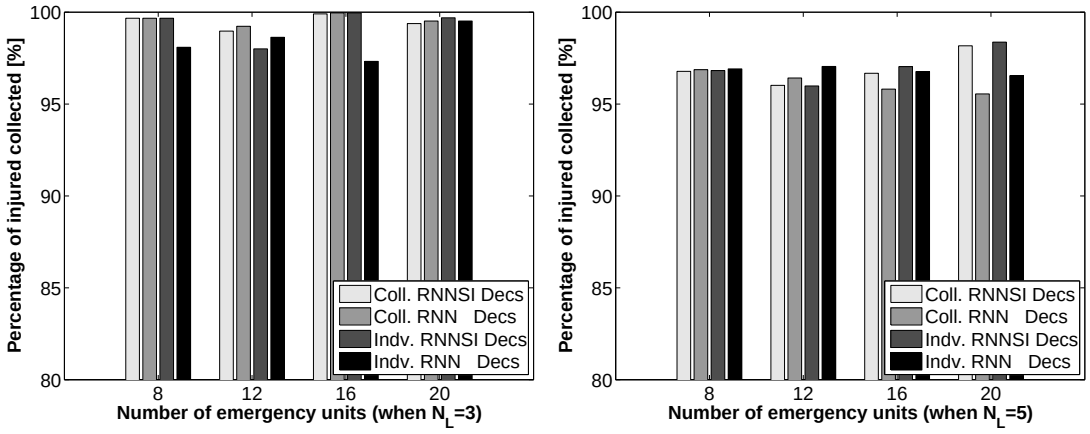


(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term

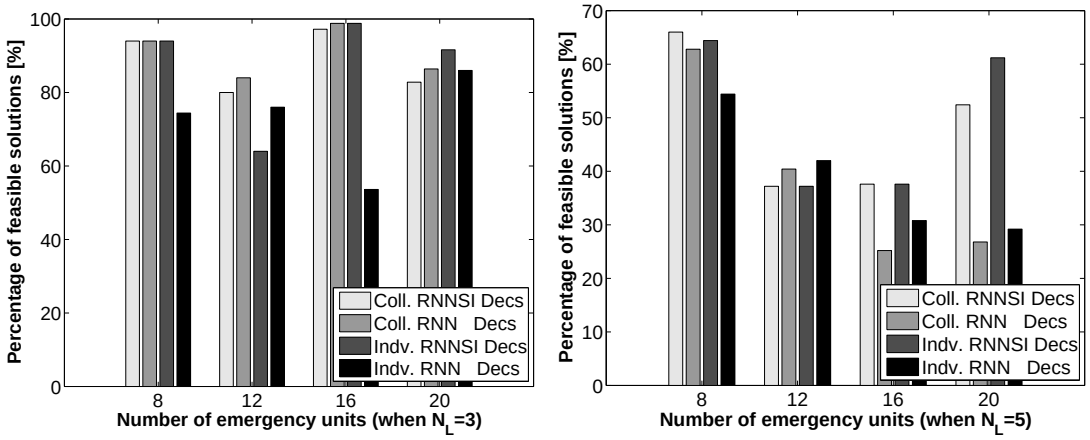


(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

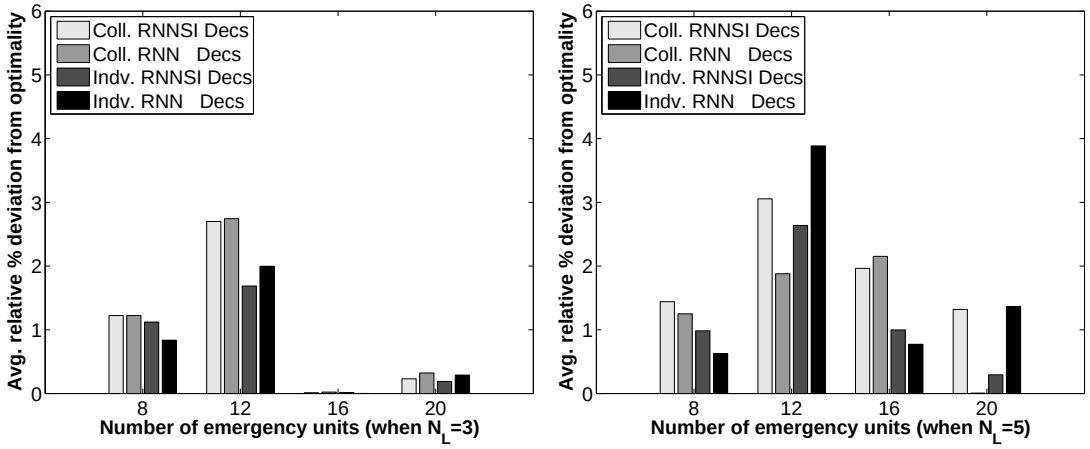
Figure 4.5.: Performance of the RNN gradient descent algorithm for the "individual" neural network architecture



(a) Percentage of injured that are collected



(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term



(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

Figure 4.6.: Performance evaluation of the four architectures considered: (a) "Collective" RNNSI , (b) "Collective" RNN, (c) "Individual" RNNSI, and (d) "Individual" RNN

architectures under investigation. With respect to the percentage of injured collected and percentage of instances where all injured were collected, the "collective" architectures are better for small problems, while the "individual" ones are better for the larger problems. This is possibly due to the large number of local minima that can appear on the error surface when the number of variables is large, which hinders the discovery of good solutions. On the other hand, having a "collective" architecture for all agent solutions also captures the dependencies among different agents' actions which leads to better solutions for small problems. Regarding the performance of the RNN and RNNSI algorithms, the former is better for small problems ($N_U = 8, 12$), while for large problems the "individual" RNNSI architecture provides the best results. This architecture also yields excellent results in terms of optimality, as in all cases $\sigma_{opt} < 2.5\%$. In fact, we could argue that the "individual" RNNSI architecture has the best overall performance, as it provides the best results for large problems ($N_U = 16, 20$), while for small problems it always leads to good solutions for all solution quality metrics considered.

Let us now examine the computational efficiency of RNN and RNNSI gradient descent algorithms. As discussed in section 3.2.5, each iteration of the RNNSI gradient descent algorithm requires the computation of one matrix-matrix product and the solution of a linear system of equations, while the RNN algorithm only requires the solution of a linear system of equations. If the linear system is solved using Gaussian elimination then RNNSI requires approximately $2N^3 + 2N^3/3$ arithmetic operations while RNN requires only $2N^3/3$ operations which means that the RNN learning algorithm can be at most four times faster. Figure 4.7 depicts the execution time ratio between the RNNSI and RNN learning algorithms for the "collective" architecture when $N_H = 2N_O$. It is evident that the RNN is on average two times faster for the considered cases. The main reason why the RNNSI to RNN execution time ratio is not close to four, is because highly optimised routines are used for the computation of matrix-matrix products.

4.4. Performance evaluation of the RNN-NNLS algorithm

To solve AEUI using the RNN-NNLS learning algorithm, we have employed the regularised NNLS formulation (3.37). In order to find a good pair of regularisation

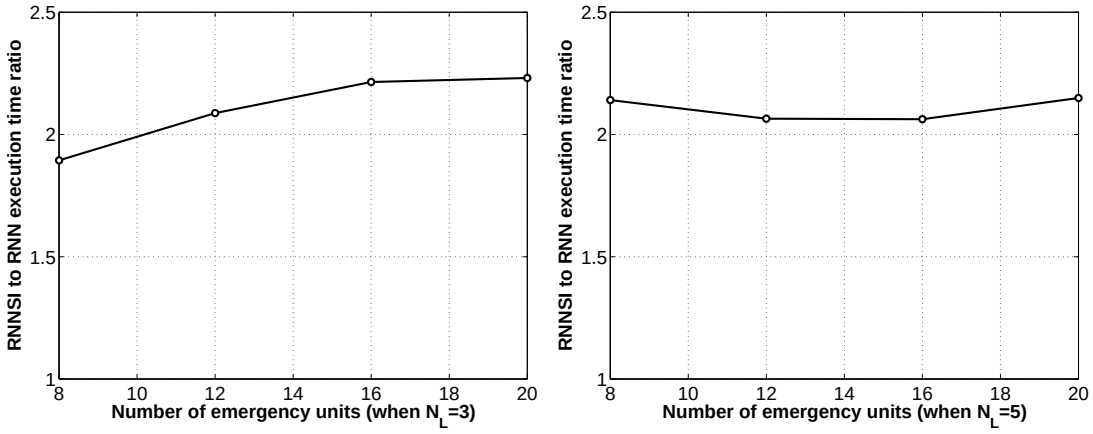


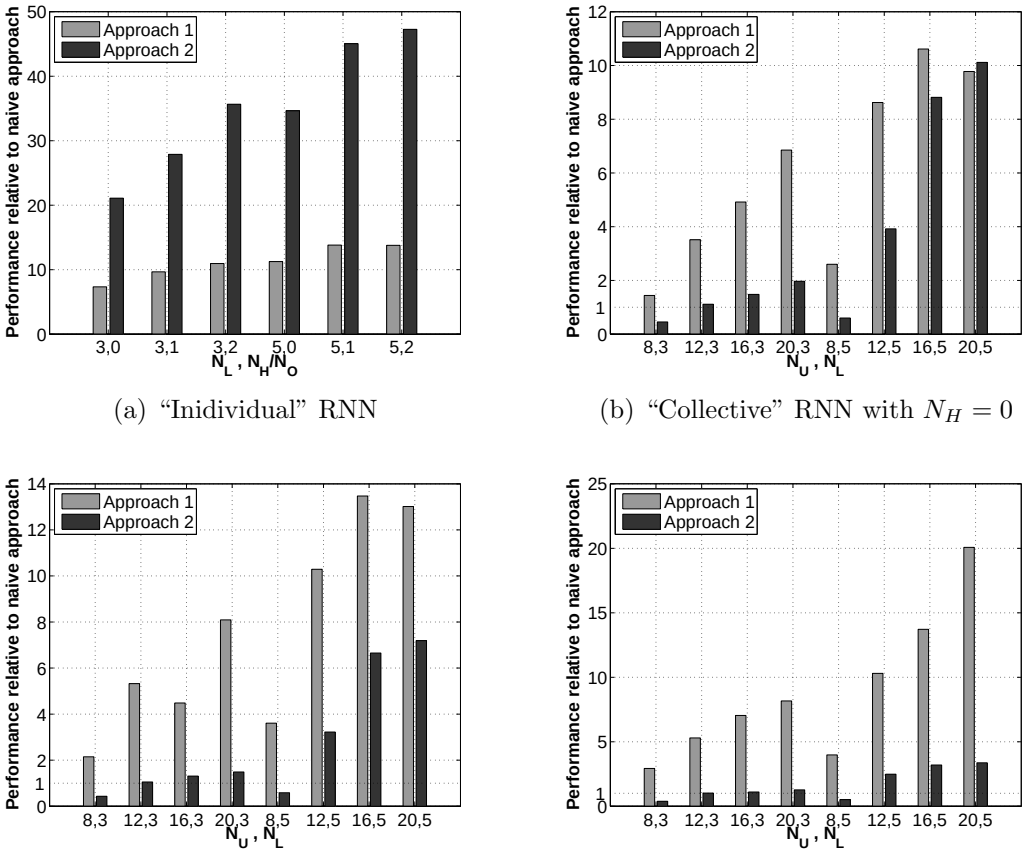
Figure 4.7.: RNNSI to RNN execution time ratio

weights $\{\theta_1, \theta_2\}$ we have repeated Algorithm 3.5 for several combinations of the two parameters. Specifically, we considered $\theta_1, \theta_2 \in \{0, 2^i\}$, $i = -5, -4, ..., 2$ and examined all possible combinations between the two weights. For each regularisation pair we have performed ten iterations ($NI_{RNN-NNLS} = 10$), checking the solution quality after each iteration and storing the weights corresponding to the largest percentage of instances where all injured were collected. For updating the desired values of the non-output and output weights we have set $\alpha_{no} = 0.75$ and $\alpha_o = 0.9$ respectively. Furthermore, the inputs and outputs to the RNN model were set according to the approach discussed in the previous section, while the initial desired excitation probabilities of the non-output neurons were generated according to the uniform distribution in the interval [0.25, 0.75]. Concerning the PGNNLS algorithm itself, we have chosen to perform 100 iterations with five correction vectors ($M = 5$) and hyper-exponential line search with parameters $\beta = 0.4$, $\sigma_{APA} = 0.25$ and $\mathbf{S}_0^\tau = \mathbf{I}$, $\forall \tau$ (see Eqs. (3.49) and (3.48)).

Before discussing the effectiveness of the RNN-NNLS algorithm for the solution of the investigated problem, we present preliminary results concerning the computational efficiency and convergence speed of the developed PGNNLS and RNN-NNLS algorithms.

4.4.1. Preliminary results

In this section we investigate the computational performance of two PGNNLS procedures, and present convergence results. Specifically, we investigate the compu-



(a) "Inidividual" RNN

(b) "Collective" RNN with $N_H = 0$

(c) "Collective" RNN with $N_H = N_O$

(d) "Collective" RNN with $N_H = 2N_O$

Figure 4.8.: Performance of approaches for computing the objective and gradient NNLS function compared to a "naive" one; the metric used is the ratio of execution times between the naive and another approach

tational performance of the approaches developed to speedup the computation of the objective and gradient NNLS functions, as well as the efficiency of the hyper-exponential line search procedure in terms of matrix vector products. Furthermore, we examine the convergence of PGNNLS and RNN-NNLS for the solution of the NNLS and supervised learning problems respectively.

To evaluate the efficiency of the two developed approaches for the computation of the costly NNLS functions, we have measured the execution time required for the evaluation of 200 $\mathbf{B}^T(\mathbf{Bw})$ operations which involve two matrix-vector products; the execution time also includes the initialisation time. To demonstrate the benefit from using these approaches, a "naive" method for the computation of these products was implemented, that takes into consideration the sparsity of $\mathbf{B}$, but performs no

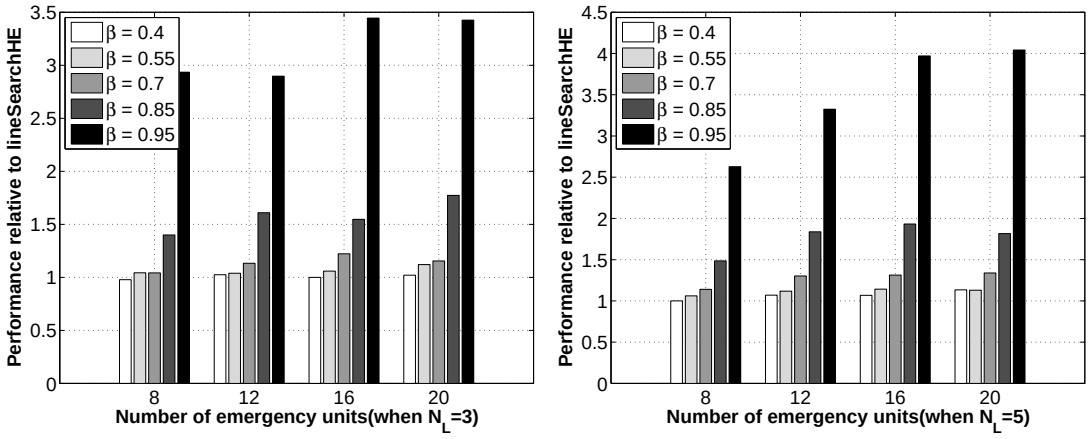


Figure 4.9.: Comparison of line search procedures lineSearchLin and lineSearchHE in terms of matrix vector products (the y-axis is their ratio) for $N_L = 3$ and $N_L = 5$ when $N_H = N_O$

analytical manipulation.

Fig. 4.8, illustrates the execution time ratio of the "naive" against the developed approaches (speedup) for different RNN architectures. In the figures, Approaches 1 and 2 correspond to the first and second computation methods discussed in section 3.3.3. Fig. 4.8(a), shows the results of the "individual" RNN architecture when $N_L = \{3, 5\}$ and various ratios of hidden to output neurons. In this case the N_U parameter is not important as the size of the network for each emergency unit depends only on N_L. Because the constructed neural network for the "individual" architecture is small, Approach 2 is significantly better than Approach 1, while both approaches have an order of magnitude speedup compared to the "naive" implementation. In fact, Approach 2 reaches an overall speedup of fifty for $N_L = 5$ and $N_H/N_O = 2$. On the contrary, for the "collective" architecture the number of neurons is significantly larger that the "individual" one, which is in favour of Approach 1. Indeed, this is verified by the results which show that Approach 1 is better than Approach 2 by up to seven times. Also as the network size increases, with the addition of more hidden neurons, Approach 1 becomes more efficient and Approach 2 less efficient. These results show that both architectures are useful, as they perform better in different cases, while they both provide a significant speedup over a "naive" implementation, as discussed in the derivation of these approaches.

To examine the efficiency of the hyper-exponential search we have measured the number of matrix-vector product evaluations during the first 10 iterations of

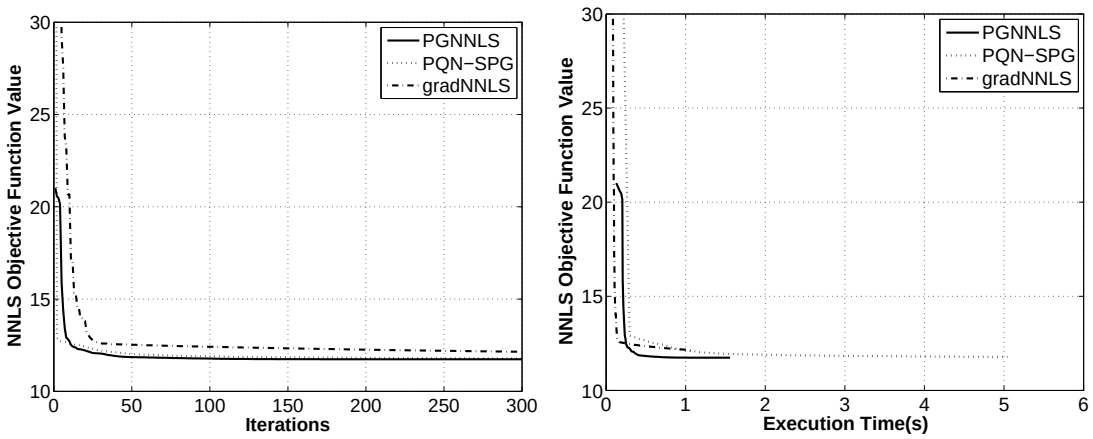
PGNNLS with or without the use of the particular line search. Experiments were conducted for several values of β under the "collective" RNN architecture with $N_H = N_O$, for all N_U, N_L pairs. Fig. 4.9, illustrates the ratio of the number of matrix-vector product evaluations required for lineSearchLin and lineSearchHE. It is clear that for $\beta < 0.7$ the benefit from using lineSearchHE is small but for larger values the benefit grows exponentially; in fact for $\{16, 5\}$ and $\{20, 5\}$ the number of matrix-vector product evaluations with lineSearchLin is four times larger. It is important to mention that in all cases considered the objective function value was always the same at the end of the 10th iteration. Additionally, the smallest number of matrix-vector product evaluations is obtained for $\beta = 0.4$, without affecting the convergence of PGNNLS, and for this reason we have adopted the particular β value in our experiments.

To examine the efficiency of the proposed PGNNLS algorithm, we have compared its convergence in terms of iterations and execution time with two other algorithms, gradNNLS [124] and PQN-SPG [157]. The former is a projected gradient algorithm with first-order information ($\mathbf{S}^\tau = \mathbf{I}$) that employs the efficient line-search described in Algorithm 3.3. The latter is a limited-memory projected quasi-Newton algorithm introduced in section 3.3.2. It is evident from Fig. 4.10 that the PGNNLS algorithm outperforms gradNNLS and PQN-SPG both in terms of iterations and execution time. In fact, the larger the problem under consideration the better the performance of PGNNLS is compared to the other algorithms.

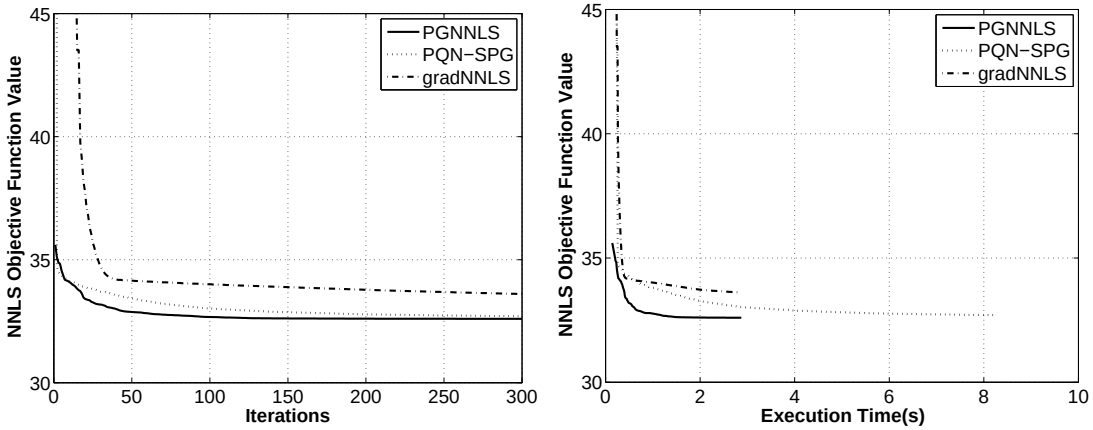
Finally, Fig. 4.11 depicts the mean squared error (MSE) with respect to the desired and attained excitation probabilities for the output neurons for ten iterations of the RNN-NNLS algorithm with $\theta_1 = \theta_2 = 0$. It is clear that the MSE error decreases for subsequent iterations leading to the converge of the algorithm. In fact, stabilisation of the MSE is accomplished after a very small number of iterations (around five). Although monotonic convergence cannot be guaranteed, the observed behaviour is sufficient to produce good trained weights that will derive high quality solutions.

4.4.2. Solving the AEUI problem

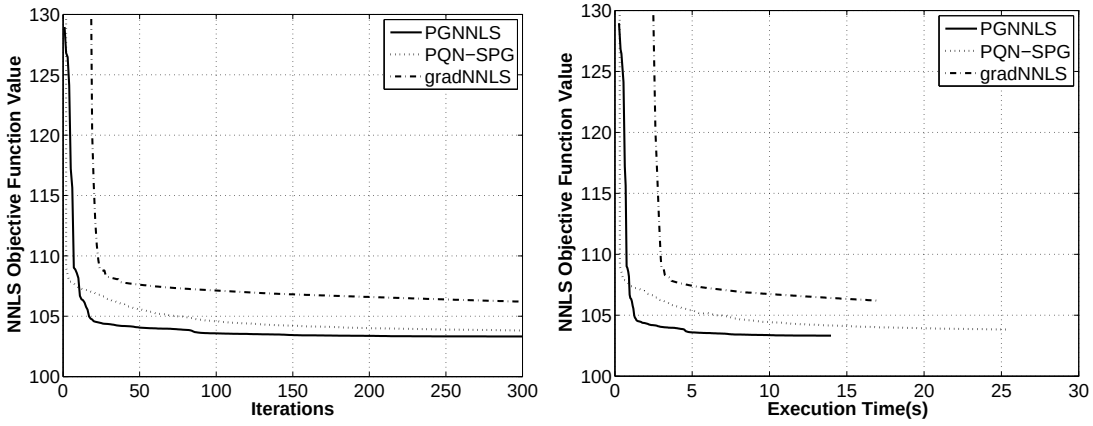
In this section the performance of the RNN-NNLS algorithm for the solution of the AEUI problem is evaluated. Fig. 4.12 summarises the results for the "collective" RNN architecture. As can be seen, the configuration with $N_H = 2N_O$ neurons is the



(a) Results for $N_U = 16$ and $N_L = 3$



(b) Results for $N_U = 12$ and $N_L = 5$



(c) Results for $N_U = 20$ and $N_L = 5$

Figure 4.10.: Comparison of convergence between algorithms gradNNLS, PQN-SPG and PGNNLS with respect to iterations and execution time

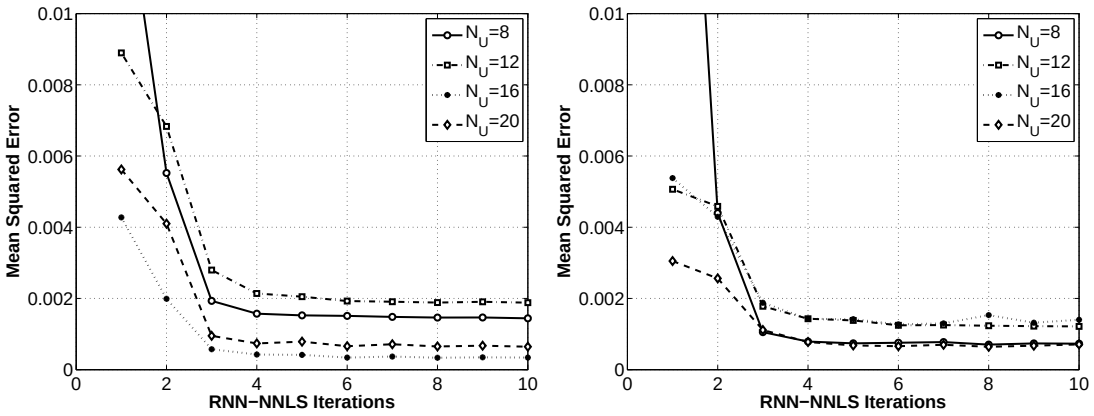


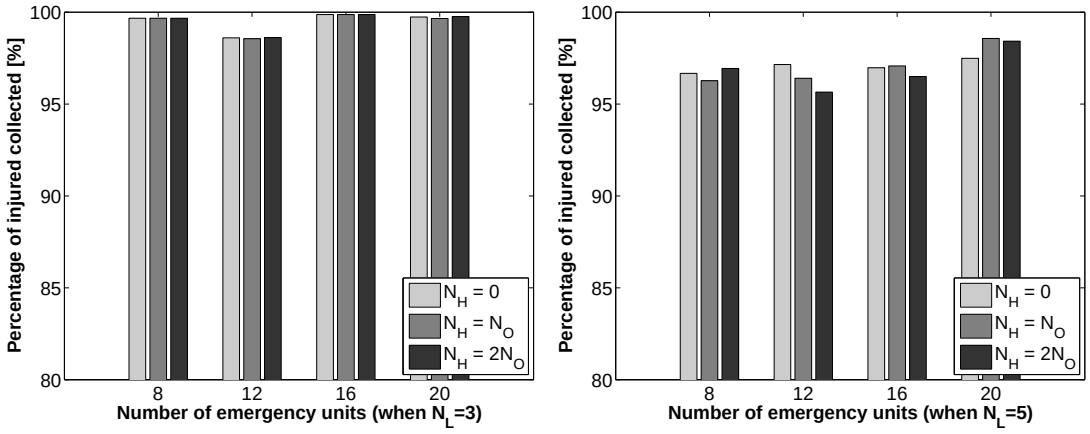
Figure 4.11.: Convergence of RNN-NNLS algorithm for $N_L = 3$ and $N_L = 5$ when $N_H = 2N_O$

most effective in finding solutions where all injured have been collected as in almost all cases it provides the best performance (apart from the $N_U = 16, N_L = 5$ case). However, in terms of percentage of injured collected and deviation from optimality it is not so efficient. The most robust performance is observed for the $N_H = N_O$ configuration, as it is always close to the best performing in terms of collecting injured, while in term of solution quality, σ_{opt} is smaller that 3% in all cases.

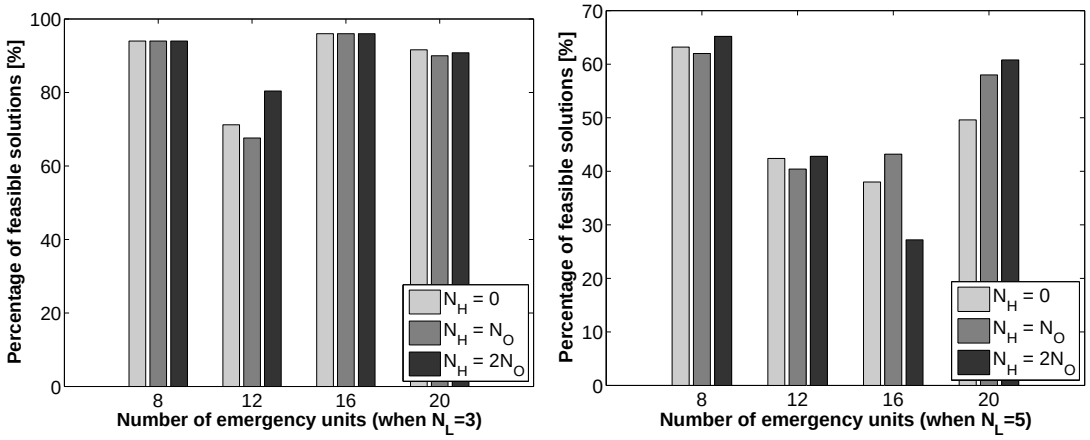
The particular configuration is the most robust in term of solution quality for the "individual" RNN architecture as well (Fig. 4.13), always achieving σ_{opt} within 4%. Another interesting characteristic is that the small configurations ($N_H = 0$ and $N_H = N_O$) have the best performance in terms of percentage of instances where all injured were collected in almost all cases.

Comparing the efficiency of the RNN-NNLS to the RNN learning algorithm, it is evident that the RNN-NNLS algorithm is more robust as it provides no low quality results for any of the metrics considered. On the contrary, the RNN learning algorithm provides low quality results for the deviation from optimality and the percentage of solutions were all injured were collected in some problem sets.

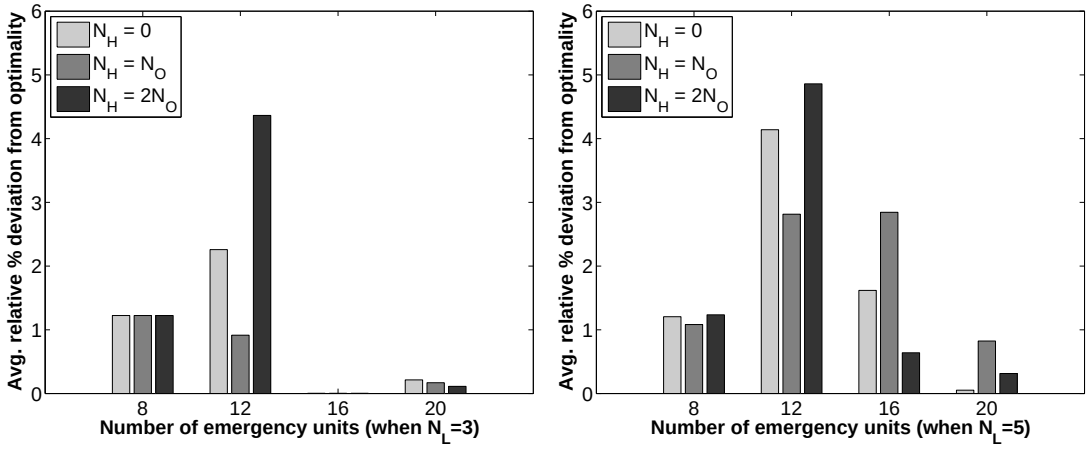
Fig. 4.14 summarises the best results for the "collective" and "individual" NN architectures of the RNN-NNLS and RNN approaches. It is evident that the "collective" RNN-NNLS algorithm yields the best results in terms of percentage of instances were all injured were collected, as it is the most effective for $N_L = 5$ and highly competitive for $N_L = 3$. On the other hand, the "individual" RNN architecture produces the best results in terms of deviation from the optimal but has



(a) Percentage of injured that are collected

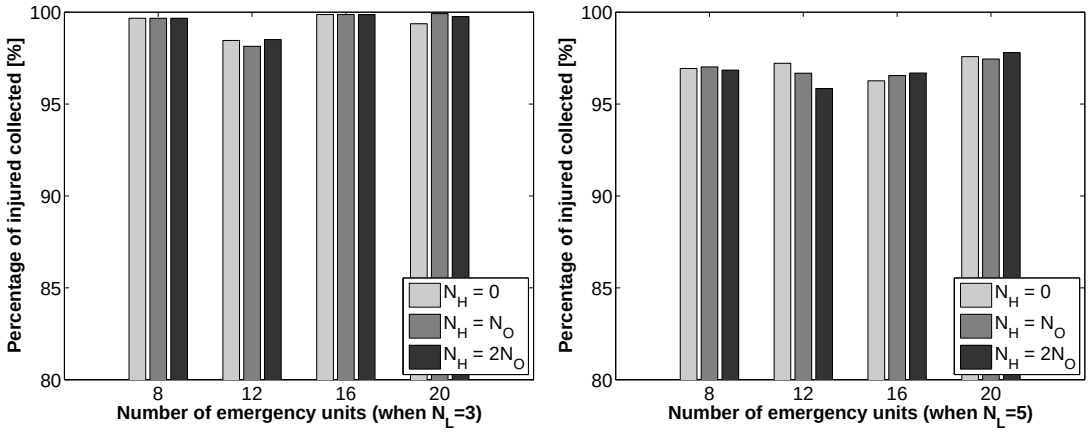


(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term

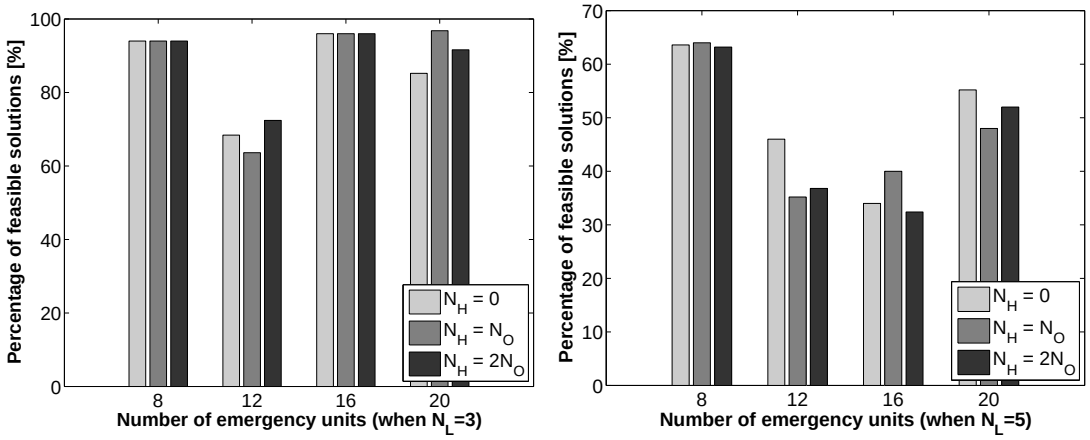


(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

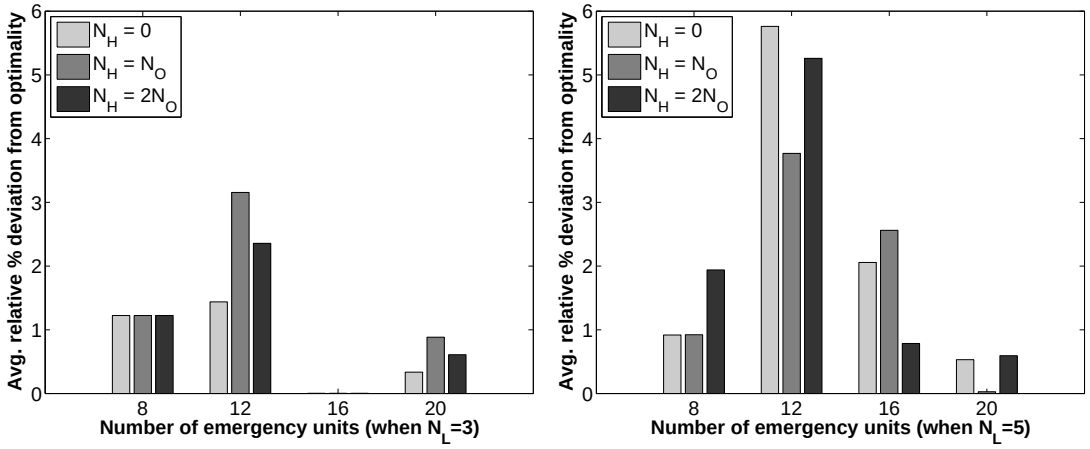
Figure 4.12.: Performance of the NNLS-RNN algorithm for the "collective" neural network architecture



(a) Percentage of injured that are collected



(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term



(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

Figure 4.13.: Performance of the NNLS-RNN algorithm for the "individual" neural network architecture

		$N_H = 0$		$N_H = N_O$		$N_H = 2N_O$	
N_U	N_L	θ_1	θ_2	θ_1	θ_2	θ_1	θ_2
8	3	0.000	0.031	0.000	0.000	0.000	0.000
12	3	0.125	0.250	0.031	0.000	0.250	0.063
16	3	0.031	0.250	0.000	0.000	0.016	0.125
20	3	0.016	0.063	0.063	0.250	0.016	0.063
8	5	0.000	0.016	0.000	0.063	0.031	0.250
12	5	0.063	0.125	0.500	0.031	0.125	0.031
16	5	0.063	0.000	0.250	0.063	0.031	1.000
20	5	0.000	0.500	0.125	0.063	0.125	2.000

Table 4.1.: Optimal regularisation weights for the "collective" RNN architecture

the worst performance in terms of the other two metrics. Among the other three architectures the "collective" RNN is the one with the best performance in terms of σ_{opt} but it is not as effective in collecting injured, especially for larger problems.

Finally, to illustrate the importance of the regularisation terms, the values of the regularisation parameters corresponding to the selected trained weights for the collective "architecture" are presented on Table 4.1. Three observations that can be drawn from this table:

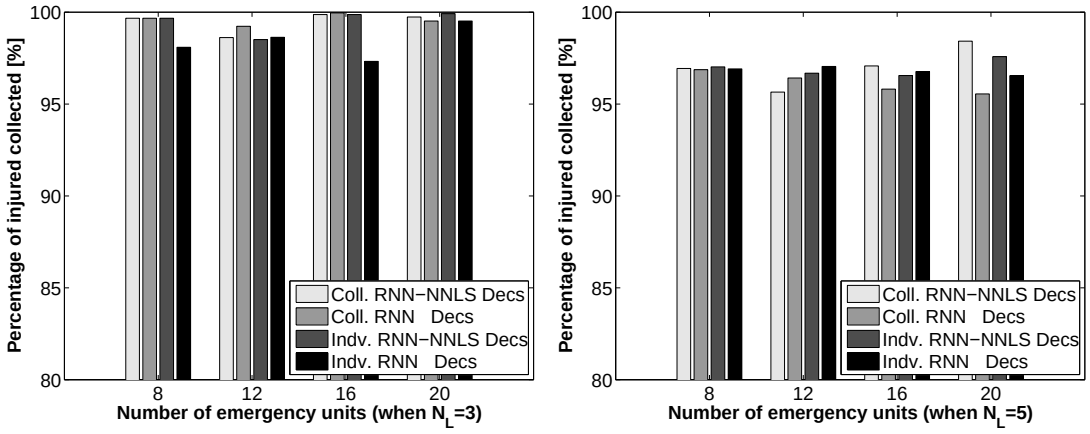
- The case $\{\theta_1 = 0,\ \theta_2 = 0\}$ is scarcely the optimal combination.
- Both parameters are important as in most of the cases both are non-zero.
- Most of the optimal regularisation coefficients are not larger than 0.5.

These observations support the decision to include these two regularisation terms in the objective function of the NNLS problem.

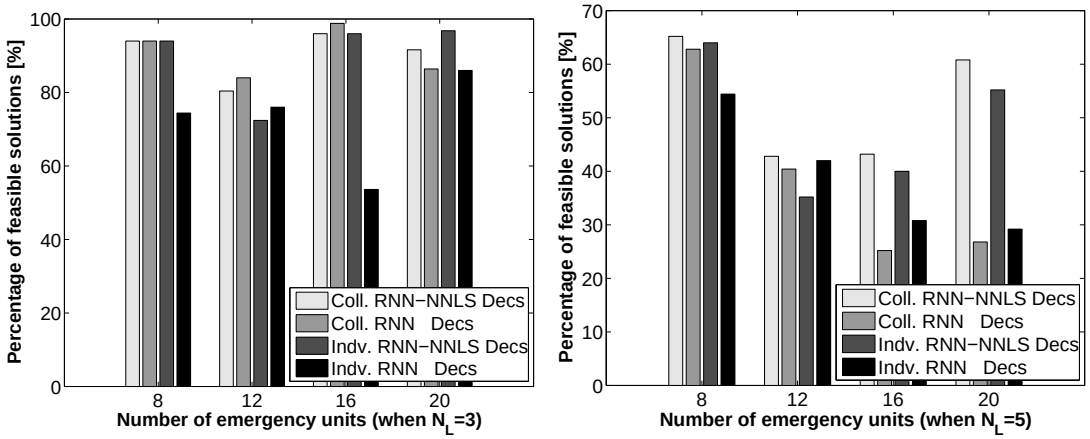
4.5. Using RNN-NNLS algorithm for weight initialisation

As has been shown in the previous section, the RNN-NNLS algorithm yields small MSE in few iterations, while it can also lead to good quality solutions. For this reason it would be interesting to examine the case where RNN-NNLS is used to initialise the weights for the RNN algorithm, instead of using random initialisation.

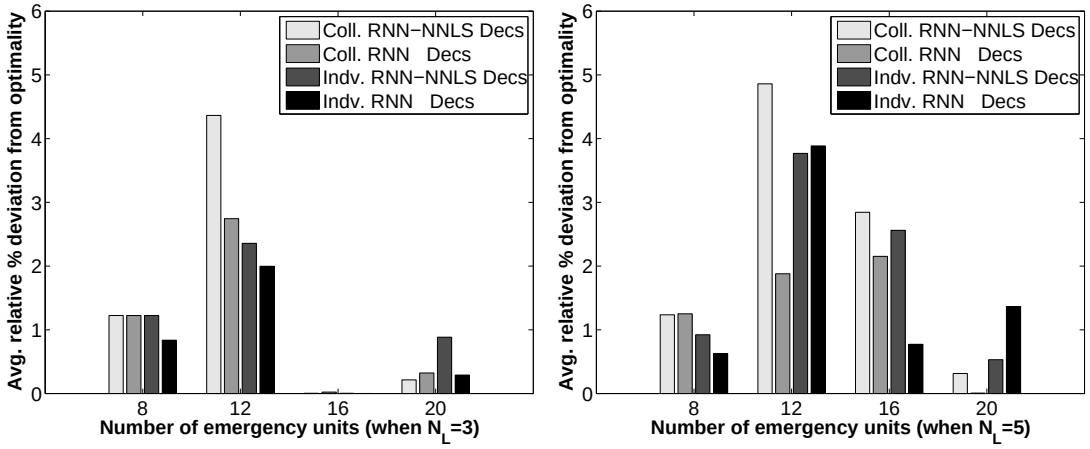
To compare the performance of the RNN gradient descent algorithm with random or NNLS initialisation the "collective" RNN architecture with no hidden neurons



(a) Percentage of injured that are collected



(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term



(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

Figure 4.14.: Comparison between the RNN-NNLS and RNN learning algorithms. The four architectures considered are: (a) "Collective" RNN-NNLS, (b) "Collective" RNN, (c) "Individual" RNN-NNLS, and (d) "Individual" RNN

was employed. For the initialisation of the weights according to the NNLS formulation, we used the best set of weights obtained after running the RNN-NNLS algorithm with the following parameters: $NI_{RNN-NNLS} = 10$; $NI_{NNLS} = 50$; $\{\theta_1, \theta_2\} \in \{\{0,0\}, \{0,2^{-i}\}, \{2^{-i},0\}\}$ where, $i = 4,5,6$.

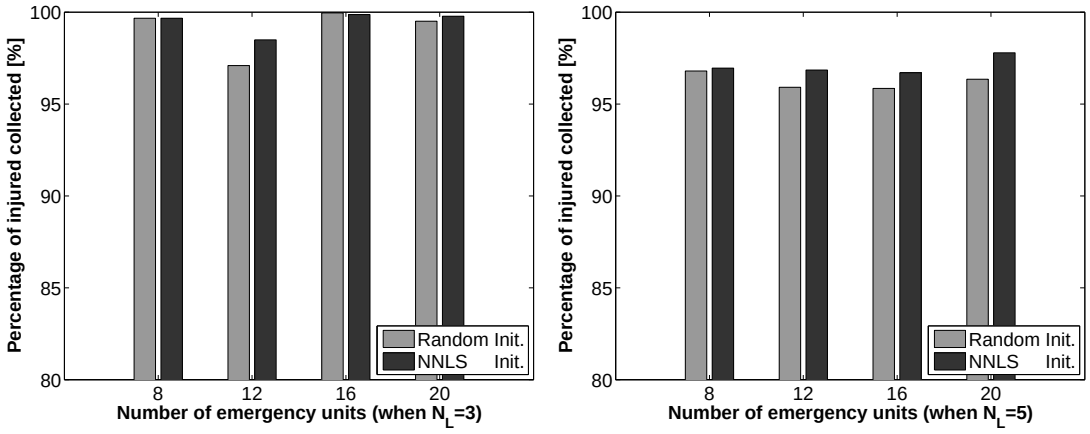
Fig. 4.15 demonstrates the performance of the two weight initialisation methods for the three solution quality metrics, while Fig. 4.16 depicts the time required to derive the best weights from the RNN gradient descent algorithm; in the NNLS initialisation case, the time to execute RNN-NNLS is also included. It is clear that weight initialisation with the NNLS approach is superior to random weight initialisation both in terms of solution quality and execution time. The NNLS initialisation scheme achieves significantly better performance for the first two metrics especially for the case that $N_L = 5$ and at the same time it provides solution with small deviation from optimality. Furthermore, the specific scheme results in faster derivation of the best weights found by the RNN gradient descent algorithm. In fact, the performance improvement is higher as the size of the network increases, which shows that the NNLS initialisation scheme is appropriate for large networks.

4.6. Conclusions

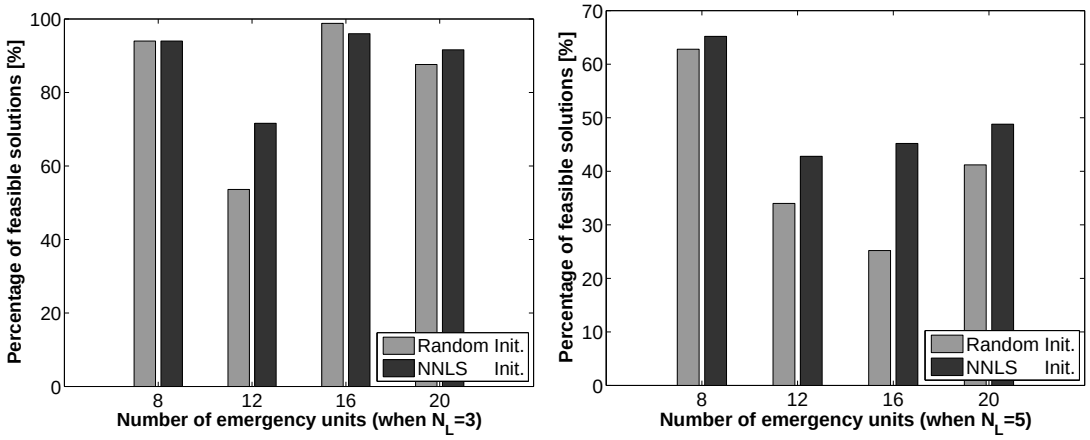
In this chapter, we have studied the AEUI problem which is a combinatorial optimisation problem associated with the dispatching of emergency units to locations of injured civilians, to collect as many as possible in the least possible time. For its solution, we have proposed the use of random neural networks in a supervised learning context. To train the neural network model, instances of the optimisation problem and the corresponding optimal solutions are supplied as training patterns. In this way, the emergency units can employ the trained neural network tool as an "oracle" that provides fast, globally consistent and close to optimal solutions in a distributed manner.

Specifically, we have considered two different neural network models, RNN and RNNSI, which were trained using the related supervised learning algorithms developed in the previous chapter. Performance evaluation of the particular algorithms concentrated on two different aspects: (a) solution quality of the AEUI problem, and (b) supervised learning efficiency.

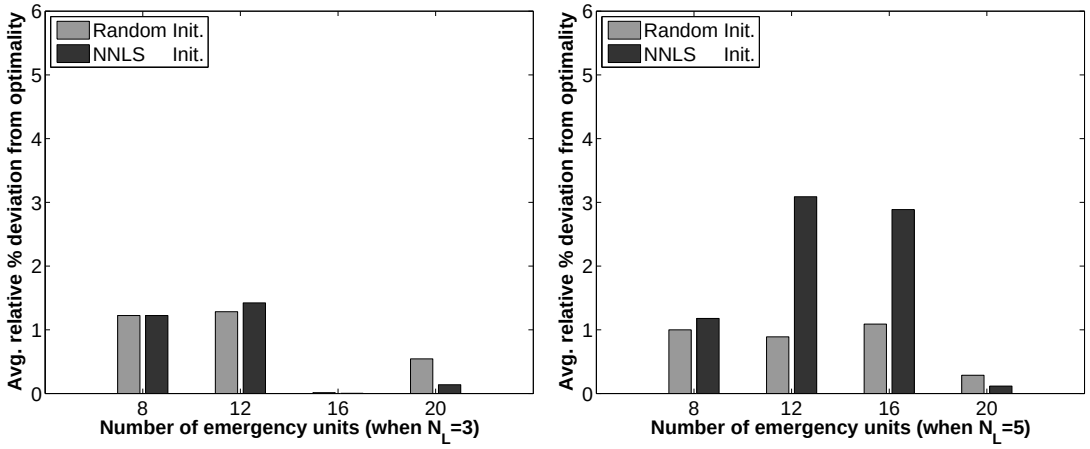
With respect to the solution of the AEUI problem, the RNNSI model that uses



(a) Percentage of injured that are collected



(b) Percentage of solutions in which all injured civilians are collected; these solutions are called "feasible" in the graphs, for want of a better term



(c) Average relative deviation from optimality for the solutions where the units are able to remove all the casualties (i.e. the "feasible" ones)

Figure 4.15.: Performance of the RNN learning algorithm with random or NNLS initialisation for the "collective" NN architecture, with no hidden neurons.

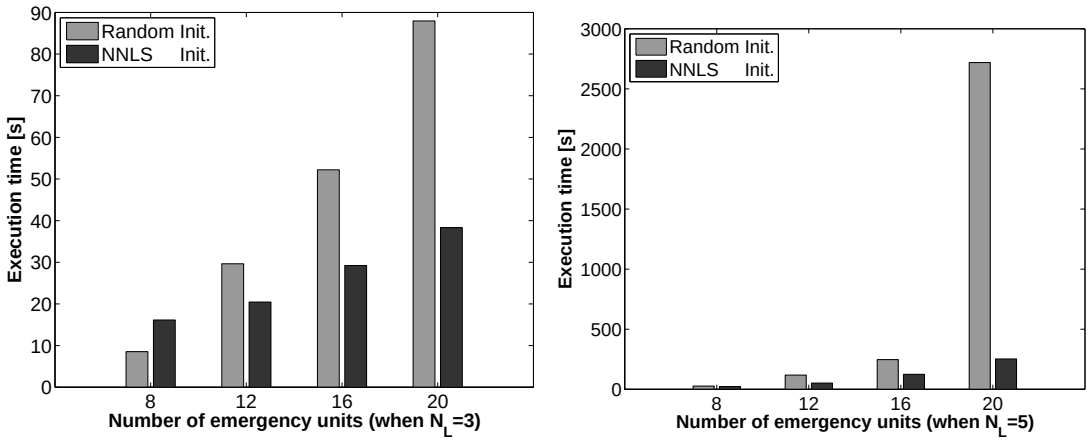


Figure 4.16.: Execution times of the RNN learning algorithm with random or NNLS initialisation

a different network for the decision of one agent, called "individual", was the most successful. This model resulted in collecting more than 95% of the injured with less than 3% deviation from the optimal in all cases, while for a large percentage of instances all injured were collected. Nonetheless, as RNNSI is a generalised version of RNN, it is more complex and hence requires more train training time (roughly speaking, RNN training is two times faster).

The RNN-NNLS algorithm, developed in the previous chapter for the training of RNN, also exhibited good performance in solving AEUI . In addition, this algorithm was the most robust in the sense that in all different topologies considered it obtained good results with no large variations for the solution metrics considered. Moreover, it was shown that the algorithm converges in a very small number of iterations (around five).

We also examined the efficiency of three different aspects of PGNNLS, the core of the RNN-NNLS algorithm that solves the NNLS problem. Firstly, we compared the efficiency of the two proposed approaches for the objective and gradient NNLS function evaluation to a "naive" implementation; it was shown that both approaches are useful because they provide better results for different cases, while they are up to fifty times faster compared to the "naive" implementation. Secondly, we demonstrated that the proposed hyper-exponential line-search can be up to four times faster than another efficient line-search. Thirdly, convergence speed of the PGNNLS algorithm was compared against two other algorithms; the results illustrated the superiority of PGNNLS both in terms of iterations and time.

Finally, the use of RNN-NNLS to provide an initial set of weights for the RNN gradient descent algorithm was investigated. Performance evaluation showed that the use of RNN-NNLS algorithm for weight initialisation is beneficial both in terms of solution quality and time required to obtain the best set of weights.

5. Asset-task assignment under execution uncertainty

In this chapter, we investigate the asset-task assignment under execution uncertainty (ATAU) problem, where each asset can potentially execute any of the tasks, but assets execute tasks with a probabilistic outcome of success. There is a cost associated with each possible assignment of an asset to a task, and if a task is not executed there is also a cost associated with the non-execution of the task. Thus any assignment of assets to tasks will result in an expected overall cost which we wish to minimise. We formulate the allocation of assets to tasks in order to minimise this expected cost, as a nonlinear combinatorial optimisation problem. A neural network approach for its approximate solution is proposed based on selecting parameters of an RNN, solving the network in equilibrium, and then identifying the assignment by selecting the neurons whose probability of being active is highest. We also propose the use of network flow algorithms which are based on solving a sequence of minimum cost flow problems on appropriately constructed networks with estimated arc costs. We introduce three different schemes for the estimation of the arc costs and we investigate their performance. We also develop an approach for obtaining tight lower bounds to the optimal solution based on transforming the problem into an equivalent form and approximating the nonlinear terms of the latter with piecewise linear functions.

The structure of the chapter is as follows. We start with a general introduction to the problem in section 5.1. Then, we provide the description and the mathematical formulation of the ATAU problem, followed by a brief discussion of related problems in section 5.3. Next, we describe the RNN parameter association and network flow approaches for the solution of ATAU , in sections 5.4 and 5.5 respectively. In section 5.6 we describe an algorithm for obtaining tight lower bounds to the studied problem, while in the last two sections we examine the performance of the proposed

approaches and conclude.

5.1. Introduction

Assignment problems is a fundamental class of combinatorial optimisation problems which involve assigning assets to tasks to minimise a desired cost function. Several variations of these problems have been studied over the years finding widespread application in diverse fields such as telecommunications, transportation systems and signal processing [30]. Nevertheless, an important assumption made in such problems is that the desired result of an assignment always occurs, e.g. a job assigned to a particular machine is executed successfully.

Clearly, in real-world applications assigning an asset to a task does not necessarily imply successful execution. The outcome of an assignment can depend on several factors such as the surrounding environment and an asset's ability in achieving its task. For example, when dispatching emergency personnel to treat injured civilians, one may fail either because of difficulty in accessing the location of the injured from the route followed, or because his/her skills do not suffice to treat the injured successfully.

Additionally, uncertainty in successfully completing a task can even be inherent. For instance in cancer therapy, the possibility of destroying a targeted tumour with a certain therapeutic tool (chemotherapy, radiotherapy, immunotherapy) can only be expressed probabilistically. Therefore, in such cases it is beneficial to apply more than one therapy tools to targeted areas to minimise not only the possibility of failure but also the cost and the side effects of the overall therapy [50].

In this chapter we investigate a general assignment problem where the outcome of any assignment is uncertain. We model uncertainty by assuming that one asset has a certain probability in executing a particular task. In the examined problem, one asset suffices to execute one task, while more than one assignments can be made for the same purpose to increase the probability of success. We further assume that the assignments made to the same task have an independent overall effect so that the total failure probability of the particular task is given by the product of the individual failure probabilities. The objective is to minimise the overall expected cost, given that there is a cost for each asset-task assignment, and a cost for the non-execution of each task.

We have two additional requirements with respect to any developed algorithms. First, we want all decisions to be made separately for all the assets; i.e. the assets are allocated independently of each other. Second, any developed algorithm should, in all cases, be fast. This implies that the decision algorithm cannot be based on enumerating all possible solutions and selecting the one that has the least cost among the enumerated solutions, but rather be of low polynomial complexity.

Apart from the dispatching of emergency personnel to treat injured and the cancer therapy problems discussed above other application areas that are covered by this abstract representation include examples where:

- Tasks represent "jobs", and assets represent "resources" and the goal is to find an assignment matrix in order to minimise the expected cost of not executing successfully the "jobs" as well as the assignment cost,
- Tasks are "targets to be detected" and assets are sensors and the goal is to find as many targets as possible when a sensor can detect a target in its monitoring sector with a certain probability [135].
- Tasks are "entities that need to communicate" and assets are "communication channels or frequencies" and the objective is to maximise the expected number of entities that will successfully communicate when communication in each of the channels is uncertain, etc.

The contribution of this chapter is three-fold:

1. We develop an RNN solution approach by associating parameters of the optimisation problem with parameters of the neural network model, solving the network in equilibrium, and then identifying assignments by selecting the neurons whose probability of being active is highest, which is discussed in section 5.4.
2. We develop network flow algorithms for the solution of the considered asset-task assignment problem. These algorithms are based on solving a sequence of minimum cost flow problems on appropriately constructed networks with estimated arc costs. Specifically, we consider three different estimation schemes MCFmax, MCFmin and MCFrnn as described in section 5.5.

3. We propose an approach for obtaining tight lower bounds to the optimal solution by transforming the problem to an equivalent one and approximating the latter using piecewise linear functions. We derive analytical expressions for the upper and lower bounds of the approximation intervals and introduce an adaptive scheme for the selection of the piecewise linear segments that restricts the maximum approximation error to a desired value, as explained in section 5.6.

5.2. Problem description and mathematical formulation

Consider a set of tasks $\mathcal{T}$ that need to be executed by a set of assets $\mathcal{A}$. Task t carries a penalty $U(t)$ if it is not executed, while there is also a cost $C_a(a,t)$ for assigning asset a to task t. We assume that any one of the tasks can be executed by any one of the assets and that one asset suffices to execute one task. It is also possible that the task execution may fail despite the fact that an asset has been assigned to it, and this will be represented by the probability $0 \leq p_f(a,t) \leq 1$ that asset a will fail in executing task t when it is assigned to it.

To compensate task execution failures more than one assets can be assigned to one task to increase the probability of successful execution. It is assumed that the assets assigned to the same task t have an independent overall effect so that the overall failure probability for the particular task, $p_f^o(t)$, is given by the product of the failure probabilities of the assets assigned to it. For example, if a particular task is associated with three assignments with failure probabilities 0.4, 0.2 and 0.1, then the total failure probability for the particular task will be equal to $p_f^o(t) = 0.4 \times 0.2 \times 0.1 = 0.008$. We also assume that after an asset is allocated to some task, it cannot be re-assigned again to some other task; this corresponds to cases where the assets are expendable or to real-time situations where, for the given time epoch considered, decisions are irrevocable. It is also possible for one asset not to be assigned to any of the tasks, as the incurred assignment cost can increase the overall cost instead of decreasing it. Our objective if to find an allocation matrix $\mathbf{X}$ with elements $X(a,t) \in \{0,1\}$, representing whether asset a is assigned to task

t, that minimises our cost function defined below:

$$C = \sum_{t \in \mathcal{T}} \sum_{a \in \mathcal{A}} C_a(a,t) X(a,t) + \sum_{t \in \mathcal{T}} U(t) p_f^o(t) \tag{5.1}$$

where $p_f^o(t) = \prod_{a \in \mathcal{A}} \{1 - p_s(a,t) X(a,t)\}$ is the overall failure probability of task t and $p_s(a,t) = 1 - p_f(a,t)$ is the probability that asset a will successfully execute task t if it is assigned to it.

In Eq. (5.1) the first term is the total cost of the assignments made, while the second term expresses the expected remaining cost of task t. Expression $\{1 - p_s(a,t)X(a,t)\}$ in $p_f^o(t)$ denotes the failure probability from assigning asset a to task t without knowing if the assignment will take place. If the assignment is made, $X(a,t) = 1$, the failure probability is equal to $p_f(a,t) = 1 - p_s(a,t)$, otherwise if $X(a,t) = 0$ then the failure probability for that assignment is equal to 1. Note that $p_f^o(t)$ is given by the product of the aforementioned failure probabilities as we have assumed that assignments to the same task have an independent overall effect.

Moreover, when $X(a,t) \in \{0,1\}$, expressions $\{1 - p_s(a,t)X(a,t)\}$ are equivalent to $p_f(a,t)^{X(a,t)}$ so that the cost function can be written with decision variables in the exponents of assignment failure probabilities, resulting in the following problem formulation which will be used throughout the chapter:

$$\begin{aligned} \min C \quad &= \quad \sum_{t \in \mathcal{T}} \sum_{a \in \mathcal{A}} C_a(a,t) X(a,t) + \sum_{t \in \mathcal{T}} U(t) \prod_{a \in \mathcal{A}} p_f(a,t)^{X(a,t)} \\ s.t. \quad & \quad \sum_{t \in \mathcal{T}} X(a,t) \leq 1, \quad a \in \mathcal{A} \\ & \quad X(a,t) \in \{0,1\} \end{aligned} \tag{5.2}$$

The constraint shows that one asset can be assigned to at most one task.

5.3. Related problems

In our formulations, the decision variables appear either in the exponent of parameters or as a product and hence the examined problem is a nonlinear combinatorial optimisation problem which belongs to the general class of nonlinear assignment problems [147]. A related problem with a product of terms that also have the decision variables in their exponent is the Weapon Target Assignment (WTA) problem.

In the WTA problem, we have a set of weapons $\mathcal{A}$ and a set of targets $\mathcal{T}$ and the goal is to find an optimal allocation of the weapons to the targets so that the expected damage on the targets is maximised or equivalently the expected leakage of the targets is minimised. It is assumed that weapon a has a probability of successfully intercepting target t equal to $p_s(a,t)$, while each target has a cost equal to $U(t)$. Therefore, the mathematical formulation of the WTA problem is as follows:

$$\begin{aligned} \min C_{WTA} &= \sum_{t\in\mathcal{T}} U(t) \prod_{a\in\mathcal{A}} p_f(a,t)^{X(a,t)} \\ s.t. \quad & \sum_{t\in\mathcal{T}} X(a,t) = 1, \quad a \in \mathcal{A} \\ & X(a,t) \in \{0,1\}, \quad a \in \mathcal{A},\ t \in \mathcal{T} \end{aligned} \tag{5.3}$$

There are two main differences between formulations (5.2) and (5.3). The first is that in formulation (5.2) each asset has an associated cost $C_a(a,t)$, while in (5.3) the weapons carry no cost. The second difference is that in our formulation not all assets need to be assigned to tasks. This stems from the fact that each asset has an associated cost and hence a particular assignment might not be beneficial.

In the general case, the WTA-problem is NP-complete [125] and hence exact algorithms have been mostly proposed for solving special optimally solvable cases of the problem. One such special case is when the probabilities $p_f(a,t)$ are independent of the weapon, i.e. $(p_f(a,t) = p_f(t),\ \forall a)$, while a second one is when we want to assign at most one weapon to each target. The first special case can be solved either using the Maximum Marginal Return (MMR) algorithm [46, 114], where the weapons are assigned in a greedy fashion to the targets that result in the maximum decrease of the cost function, or using a local search algorithm to identify and swap any weapon-target pairs that reduce the overall cost [104]. The second special case results in a linear assignment problem that can be efficiently solved using for example a network flow algorithm [39].

Exact algorithms for the solution of the general WTA problem have been recently introduced in [12]. The authors proposed several lower bounding schemes based on general network flow approximations of the problem and developed a branch and bound algorithm achieving the exact solution of medium size problems (80 weapons and 80 targets). However, the time required for the exact solution of the general

WTA problem is very large and a great amount of research has focus on developing heuristic algorithms primarily based on metaheuristic techniques such as Hopfield neural networks [174], ant colony optimisation, [182, 120], genetic algorithms [121] and very large scale neighbourhoods [12].

Apart from the WTA problem, other problems related to our formulation are assignment problems where the objective function involves the product of two or more variables such as the quadratic and biquadratic assignment problems [126]. However, in these problems not only each asset must be assigned once, but also each task must be associated to only one asset which is not in agreement with (5.2), where more than one assets can be assigned to one task. The problems with the particular constraint relaxed are called semi-assignment problems of which the most widely studied nonlinear problem is the quadratic semi-assignment problem (QSAP) [151] defined as:

$$\begin{aligned} \min C_{QSAP} \;\; = \;\; & \sum_{t\in\mathcal{T}}\sum_{a\in\mathcal{A}} C_a(a,t)X(a,t) \\ & + \sum_{t\in\mathcal{T}}\sum_{t'\in\mathcal{T}}\sum_{a'\in\mathcal{A}}\sum_{a\in\mathcal{A}} b_{ata't'}X(a,t)X(a',t') \qquad (5.4) \\ s.t. \qquad & \sum_{t\in\mathcal{T}} X(a,t) = 1, \;\; \forall a \\ & X(a,t) \in \{0,1\} \end{aligned}$$

Note that in this problem we have to assign all assets to tasks, whereas the products in the cost function involve only two decision variables, contrary to our formulations where we have $|\mathcal{A}|$. Nevertheless, as in our formulation (5.2), the cost function in (5.4) has also a linear term associated with the cost of the asset assignments apart from the nonlinear term.

In the general case, QSAP is NP-hard [154] and in practice optimal solutions cannot be obtained even for small size problems [129]. As a result exact algorithms have been developed only for special cases of QSAP that result in algorithms with polynomial time complexity [131, 132].

5.4. The RNN parameter association approach

In this section we will develop and evaluate an RNN-based formulation of the asset-to-task assignment problem (5.2). The solution is based on an algorithm that uses an RNN whose parameters, including the weights associated with the connections between neurons, are selected directly by translation of the parameters of the optimisation problem. Then the RNN model is solved numerically, and a sequence of subsequently smaller RNN models are solved where each subsequent RNN is obtained by a greedy reduction of the previous network. The steady-state solutions of the first and of each subsequent RNN are used to decide on the asset to task assignments that are chosen. A similar approach has previously been used successfully in other optimisation problems [67, 87].

In the approach that we propose, each allocation decision (a, t) is represented by a neuron $N(a, t)$ of a RNN, so that $X(a, t)$ corresponds to the probability $q_{(a,t)}$ that this particular neuron is excited. Thus the computational size of the problem to be considered will depend on $|\mathcal{A}| \times |\mathcal{T}|$ as indicated below. To specify the RNN used for the heuristic solution to the optimisation problem, we must specify the arrival rates of excitation and inhibition signals to each of the neurons $N(a, t)$, and the excitatory and inhibitory weights between neurons. These parameters are chosen as follows:

$$\Lambda_{(a,t)} = \max\{0, b(a, t)\}$$

$$\lambda_{(a,t)} = \max\{0, -b(a, t)\}$$

where

$$b(a, t) = U(t)p_s(a, t) - C_a(a, t)$$

Parameter $b(a, t)$ represents the net expected reduction in the objective function when asset a is allocated to task t, since $U(t)p_s(a, t)$ is the expected remaining cost of task t if this allocation is made and $C_a(a, t)$ is the cost of allocating this asset to the given task. To discourage the allocation of distinct assets to the same task, we also set the inhibitory weights to:

$$w^-(a, t; a', t) = \max\{0, b(a, t)\}, \text{ if } a \neq a'$$

Similarly we wish to avoid the same asset being assigned to distinct tasks :

$$w^-(a,t;a,t') = \max\{0, b(a,t)\}, \text{ if } t \neq t'$$

To keep matters as simple as possible, we choose not to reinforce or weaken any of the assignments other than what is already done via the incoming excitatory signals, so that we choose $w^+(a,t;a,t') = 0$ and $w^-(a,t;a',t') = 0$ for all other a, a' and t, t', and we end with:

$$r_{(a,t)} = \sum_{a',t'} w^-(a,t;a',t') \tag{5.5}$$

Based on the above parameters the excitation level of each neuron satisfies:

$$q_{(a,t)} = \frac{\Lambda_{(a,t)}}{\lambda_{(a,t)} + r_{(a,t)} + \sum_{a' \neq a} q_{(a',t)} w^-(a',t;a,t) + \sum_{t' \neq t} q_{(a,t')} w^-(a,t';a,t)} \tag{5.6}$$

and the system of equations (5.6) is then solved iteratively in the following manner to obtain the assignments of assets to tasks:

1. Initialisation: $\mathcal{A}_{rem} \leftarrow \mathcal{A}$, $S \leftarrow \emptyset$ and $U_{cur}(t) \leftarrow U(t),\ t \in \mathcal{T}$.
2. Compute the RNN parameters based on $U_{cur}(t), \forall t$ and construct the neural network for $a \in \mathcal{A}_{rem}$ and $t \in \mathcal{T}$.
3. Solve the system of Eqs. (2.6)-(2.8) to obtain $q_{(a,t)}$.
4. Select asset-task pair (a^*, t^*) that corresponds to the neuron with the largest positive $q_{(a,t)}$; if all $q_{(a,t)} = 0$, $a \in \mathcal{A}_{rem}$ and $t \in \mathcal{T}$ stop: there is no assignment that reduces the cost of the objective function.
5. Set $S \leftarrow S \cup (a^*, t^*)$.
6. Set $\mathcal{A}_{rem} \leftarrow \mathcal{A}_{rem} \backslash \{a^*\}$.
7. Set $U_{cur}(t^*) \leftarrow U_{cur}(t^*) p_f(a^*, t^*)$.
8. If $\mathcal{A}_{rem} \neq \emptyset$ go to step (2) otherwise stop: all assets has been assigned.

In the algorithm, $\mathcal{A}_{rem}$ represents the assets remaining to be assigned, while the solution set where the assigned asset-task pairs are stored is S. $U_{cur}(t)$ is the current

expected cost of task t given any already made assignments. Note that using this algorithm, the assignment of some asset a^* to a task t^* always results in reducing the cost of the objective function; otherwise if $b(a,t) < 0$ then $q_{(a,t)} = \Lambda_{(a,t)} = 0$ and the neuron is not selected.

One interesting feature of the algorithm is that once all assets have acquired the parameters of the problem, then they can decide in a decentralised manner and arrive at a non-conflicting decision even though their actions are not coordinated. This is possible because the RNN algorithm is deterministic, while the solution to the RNN signal-flow equations is unique. Knowledge of the problem parameters can be accomplished in an initial phase prior to decision making, in which each asset exchange with other assets nay information associated with it.

The complexity of solving the system of equations (5.6) in the general case is $O(N^2)$, where $N = |\mathcal{A}||\mathcal{T}|$ is the number of neurons in the network. Therefore, the complexity of solving the system of equations in the general case is $O(|\mathcal{A}|^2|\mathcal{T}|^2)$ and the overall complexity of the algorithm is $O(|\mathcal{A}|^3|\mathcal{T}|^2)$ as we need to execute $|A|$ iterations. However, we can take advantage of the special structure of the system and improve the efficiency of the computation. Note that Eq. (5.6) can be written as:

$$q_{(a,t)} = \frac{\Lambda_{(a,t)}}{\lambda_{(a,t)} + r_{(a,t)} + \sum_{a' \neq a} q_{(a',t)} \max\{0, b(a',t)\} + \sum_{t' \neq t} q_{(a,t')} \max\{0, b(a,t')\}}$$

Based on the above equation the calculation of $q_{(a,t)}$ is of complexity $O(|\mathcal{A}| + |\mathcal{T}|)$ so that the complexity of calculating the values of all neurons once is: $O(|\mathcal{A}||\mathcal{T}|(|\mathcal{A}| + |\mathcal{T}|))$. We can further reduce the complexity if we define the variables:

$$\sigma_1^{ATAU}(t) = \sum_{a' \in \mathcal{A}} q_{(a',t)} \max\{0, b(a',t)\}, \ \forall t$$

$$\sigma_2^{ATAU}(a) = \sum_{t' \in \mathcal{T}} q_{(a,t')} \max\{0, b(a,t')\}, \ \forall a$$

which require $O(|\mathcal{A}| \cdot |\mathcal{T}|)$ multiplications and need to be computed only once. Then we have:

$$q_{(a,t)} = \frac{\Lambda_{(a,t)}}{\lambda_{(a,t)} + r_{(a,t)} + \sigma_1^{ATAU}(t) + \sigma_2^{ATAU}(a) - 2q_{(a,t)} \max\{0, b(a,t)\}}$$

Note that in the above equation the computation of $q_{(a,t)}$ requires $O(1)$ time so

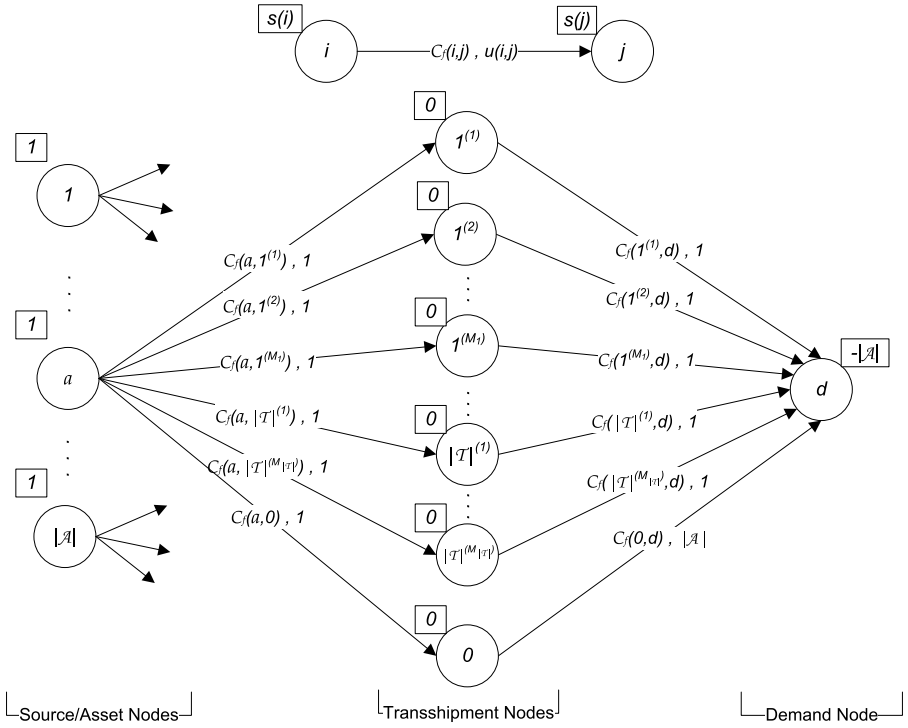


Figure 5.1.: Flow network for the solution of problem (5.2)

that the computation of all $q_{(a,t)}$ values is of complexity $O(|\mathcal{A}||\mathcal{T}|)$. Because the solution of system (5.6) dominates the computation of each iteration, and we need to perform at most $|\mathcal{A}|$ iterations, the complexity of the algorithm is $O(|\mathcal{A}|^2|\mathcal{T}|)$.

5.5. Network flow algorithms

Network flow problems are an important class of linear programming problems. They can be utilised for the solution of many optimisation problems such as the maximum flow, assignment, transportation and shortest path problems. Due to their special structure, network flow problems can be solved tens of times faster than linear programming problems while there are strong polynomial algorithms that put bounds on their worst case performance. In addition, when a network flow problem has specific properties, it results in an integer solution without using an integer programming approach for its solution [11].

The most fundamental problem in network flows is the minimum cost flow (MCF); most other network flow problems are either special cases or generalisation of the

MCF. The MCF problem considers a directed graph or network $G = (\mathcal{N}, \mathcal{E})$ which consists of a set of vertices or nodes $\mathcal{N}$ and a set of directed edges or arcs $\mathcal{E}$ connecting the nodes. Each arc $(i, j) \in \mathcal{E}$ is characterised by two parameters: the capacity $u(i, j)$ of the particular arc which is the upper bound of flow $X_f(i, j)$ allowed through (i, j) and an associated cost per unit of flow $C_f(i, j)$. Each node $i \in \mathcal{N}$ has a supply $s(i)$ that is interpreted as the amount of flow that enters the node from the outside. Node i is a *source* or *supply node* if $s(i) > 0$, a *sink* or *demand node* if $s(i) < 0$ and a *transshipment node* if $s(i) = 0$. Flow networks are governed by the flow conservation constraint which states that at each node the incoming and outgoing flows are equal. Note that the conservation constraint can hold only if $\sum_i s(i) = 0$. In the MCF problem, the objective is to find the cheapest flows that satisfy the nodes' supply, under the flow conservation constraint and the capacity constraint. The mathematical formulation of MCF is given by Eq. (5.7).

$$\min \quad \sum_{(i,j)\in\mathcal{E}} C_f(i,j) X_f(i,j) \tag{5.7a}$$

$$\text{s.t.} \quad \underbrace{s(i) + \sum_{j:(j,i)\in\mathcal{E}} X_f(j,i)}_{\text{incoming flow to } i} = \underbrace{\sum_{j:(i,j)\in\mathcal{E}} X_f(i,j)}_{\text{outgoing flow from } i}, \forall i \tag{5.7b}$$

$$0 \leq X_f(i,j) \leq u(i,j), (i,j) \in \mathcal{E} \tag{5.7c}$$

The approach that we take is to construct a flow network with flow costs associated with the net expected reduction in the objective function when assigning assets to tasks.

Fig. 5.1 depicts the network used for the solution of problem (5.2). The network is comprised of three layers of nodes: the first layer contains the source nodes, the second layer contains the transshipment nodes and the third layer the demand node that aggregates the flows sent by the source nodes. Each source node a has supply $s(a) = 1$ and corresponds to asset a. Each transshipment node $t^{(m_t)}$ denotes the m_tth asset assignment to task t, while node 0 corresponds to the case that an asset is not assigned to any task. At most M_t assets can be assigned to task t. The role of the demand node d is to aggregate the flows sent in the network and its demand is equal to the total supply of the assets, $s(d) = -|\mathcal{A}|$.

A source node a is connected to all transshipment nodes $t^{(m_t)}$ and the capacity of all arcs is equal to 1, so that the associated flows $X_f(a, t^{(m_t)})$ represent the fact that

asset a is the m_tth assignment to task t. Even though there are $|\mathcal{A}|$ arcs arriving at each transshipment node there is only one arc leaving each such node towards the demand node d. These arcs also have capacity 1 except from the arc $(0, d)$ whose capacity is equal to $|\mathcal{A}|$ so that even if no assignments are made the source nodes' supply reaches the demand node via node 0. Thus, flow $X_f(t^{(m_t)}, d), t \in \mathcal{T}$ denotes whether the m_tth assignment for task t has been made.

The resulting configuration guarantees that at most one asset can be assigned to a particular transshipment node. Moreover, as all arc capacities and supplies/demands of the nodes are integers, the integrality property guarantees that in the MCF solution all flows $X_f(i, j)$ will be integer (see. p.318 of [11]). Actually, $X_f(a, t^{(m_t)})$ flows have unit capacity so that the final value of the particular flows will be 0 or 1. We also need to ensure that the assignment of assets to a particular task t is contiguous in the sense that if there are already $m_t - 1$ assignments, then the next one should be the m_tth one. The contiguous property will be established after we discuss about the costs of the arcs.

The costs of the arcs represent the net reduction in the cost function from assigning a particular asset to a task so our aim is to maximise the net reduction in the objective function. Thus, to solve the problem as a minimum cost flow problem we need to negate all the costs associated with the network.

The cost of the arcs associated with a first assignment $(a, t^{(1)})$ is given by:

$$C_f(a, t^{(1)}) = \max\{0, U(t)p_s(a, t) - C_a(a, t)\}, \ \forall a, \ t \tag{5.8}$$

In order to be able to correctly determine the arc costs associated with the second assignment $C_f(a, t^{(2)})$ we need to know the expected cost of the task t after the first assignment, $U^{(1)}(t)$. If we assume that an oracle provides the first asset allocated to task t, $a_{t^{(1)}}$, then $U^{(1)}(t) = U(t)p_f(a_{t^{(1)}}, t)$. Similarly, if the oracle provides the first m_t assigned assets to task t, $a_{t^{(1)}}, ..., a_{t^{(m_t)}}$ then the arc costs for the next assignment of task t will be given by:

$$\begin{aligned} C_f(a, t^{(m_t+1)}) &= \max\{0, U^{(m_t)}(t)p_s(a, t) - C_a(a, t)\} \\ &= \max\{0, U(t) \prod_{m=1}^{m_t} p_f(a_{t^{(m)}}, t)p_s(a, t) - C_a(a, t)\} \end{aligned} \tag{5.9}$$

Note that each assigned asset carries a cost so it is possible that a particular as-

signment will result in a negative net expected reduction; in that case we assign a zero cost to that arc. The maximum number of transshipment nodes associated with a task t, M_t can be determined from the fact that $C_f(a, t^{(M_t+1)}) = 0, \forall a$. Alternatively, we can assign a maximum value to M_t to limit the network size.

Concerning the cost of the arcs towards node 0 we take $C_f(a, 0) = \epsilon > 0, \ \forall a$, where ϵ is a sufficiently small positive value. The use of a positive value for $C_f(a, 0)$ is important for the avoidance of unbounded solutions due to the zero arc costs present in the network. Additionally, ϵ should be sufficiently small in order not to be considered as a beneficial assignment. The arc costs from the transshipment nodes to the demand node are not important so $C_f(t^{(m_t)}, d) = 0, \forall t, \ m_t = 1, ..., M_t$; their role is to ensure that at most one asset is related to one task.

In practice, the asset assignments are not known beforehand and hence we cannot determine the cost values $C_f(a, t^{(m_t)}), m_t > 1$. As a result, we need to develop approximation schemes. A conservative approach, which we call *MCFmax*, is to always assume that the previously assigned asset to a particular task is the least effective one i.e. the one with the largest execution failure probability $p_{f,max}(t) = \max_{a \in \mathcal{A}} p_f(a, t)$. Hence, every term $p_f(a_{t^{(m)}}, t)$, $m = 1, ..., m_t$ in Eq. (5.9) will be replaced by $p_{f,max}(t)$. An optimistic approach, called *MCFmin*, is to always consider the most effective asset for previous assignments. If $p_{f,min}(t) = \min_{a \in \mathcal{A}} p_f(a, t)$ then we set $p_f(a_{t^{(m)}}, t) \equiv p_{f,min}(t)$. A third approximation scheme, called *MCFrnn*, is to solve the problem using the RNN association approach and then use the derived allocations to obtain the arc costs for the MCF network. Hence, the terms $p_f(a_{t^{(m)}}, t)$ are changed to $p_{f,rnn}(t^{(m)})$ which denote the probability of execution failure for the mth asset assigned to task t according to the RNN approach. Because the complexity of the RNN algorithm is small compared to the solution of an MCF problem the overall execution time is not significantly affected.

An important property of the described flow network is that because $0 < p_f(a, t) < 1$, the following relationship holds for all cost determination approaches described above:

$$C_f(a, t^{(1)}) > \ldots > C_f(a, t^{(m_t)}) > C_f(a, 0) > 0 \tag{5.10}$$

The fact that inequality (5.10), does not include $C_f(a, t^{(M_t)})$ implies that $C_f(a, t^{(m)}) = 0$, $m = m_t+1, ..., M_t$, so that after the m_tth assignment asset a cannot be assigned to task t. Inequality (5.10) guarantees the contiguous property as the most beneficial assignment for every asset-task pair is always the first available. For example if for

task $\widehat{t}$ $X_f(a_{\widehat{t}^{(1)}}, \widehat{t}^{(1)}) = 1$ and $X_f(a_{\widehat{t}^{(2)}}, \widehat{t}^{(2)}) = 1$, then the next assignment to be made with be $X_f(a_{\widehat{t}^{(3)}}, \widehat{t}^{(3)}) = 1$. That is because $C_f(a, \widehat{t}^{(3)}) > C_f(a, t^{(m_t)}), m_t = 4, 5, \ldots$ for all assets a.

The procedure for the solution of problem (5.2) using the proposed MCF approach is outlined below:

1. Initialise $\mathcal{A}_{rem} \leftarrow \mathcal{A}$, $S \leftarrow \emptyset$ and $U_{cur}(t) \leftarrow U(t),\ t \in \mathcal{T}$.
2. Compute $C_f(a, t^{(m_t)}), a \in \mathcal{A}_{rem}, t \in \mathcal{T}$ and $m_t = 1, ..., M_t$ according to Eq. (5.9) and the desired cost approximation scheme.
3. Construct the flow network for $a \in \mathcal{A}_{rem}$ and $t \in \mathcal{T}$ as in Fig. 5.1.
4. Solve the MCF problem with negated arc costs to obtain the optimal flows $X_f(a, t^{(k)})$.
5. Set $\mathcal{A}_{ass} \leftarrow \{a : X_f(a, t^{(1)}) = 1,\ a \in \mathcal{A}_{rem},\ t \in \mathcal{T}\}$.
6. Set $S_{cur} \leftarrow \{(a, t) : X_f(a, t^{(1)}) = 1,\ a \in \mathcal{A}_{rem},\ t \in \mathcal{T}\}$ and $S \leftarrow S \cup S_{cur}$.
7. Set $\mathcal{A}_{rem} \leftarrow \mathcal{A}_{rem} \backslash \mathcal{A}_{ass}$.
8. Set $U_{cur}(t) \leftarrow U_{cur}(t) \prod_{a:(a,t)\in S_{cur}} p_f(a, t),\ t \in \mathcal{T}$.
9. If $\mathcal{A}_{ass} \neq \emptyset$ and $\mathcal{A}_{rem} \neq \emptyset$ go to step (2) otherwise stop:the objective function cannot be reduced further.

The procedure involves the solution of a sequence of MCF problems. At the beginning, we initialise the assignment pairs stored in set S to zero as well as the remaining assets set $\mathcal{A}_{rem}$ to $\mathcal{A}$. At each iteration, we first compute the arc costs for the remaining asset nodes according to our desired cost estimation scheme and the current task costs $U_{cur}(t)$ and then we construct the network. After solving the MCF problem with negated arc costs, we select the optimal flows with value one that indicate first assignments, store the associated asset-task pairs in the solution set S and remove the assigned assets from the set $\mathcal{A}_{rem}$ so that in subsequent iterations they are not considered again. Finally, the task costs are reduced accordingly and the solution procedure is repeated until either we have assigned all assets or no beneficial assignment exist.

The largest MCF problem to be minimised dominates the time required for the solution of problem (5.2). As already mentioned there are several polynomial time algorithms for the solution of the MCF problem. Currently the best strong polynomial time algorithm is the *enhanced capacity scaling algorithm* [146], which has a time complexity of $O(|\mathcal{E}|^2 log(|\mathcal{N}|) + |\mathcal{E}||\mathcal{N}|(log(|\mathcal{N}|))^2)$. Our network is comprised of $|\mathcal{N}| = |\mathcal{A}| + \overline{M}|\mathcal{T}| + 2$ nodes and $|\mathcal{E}| = (|\mathcal{A}| + 1)(\overline{M}|\mathcal{T}| + 1)$ arcs, where $\overline{M} = 1/|\mathcal{T}| \sum_{t \in \mathcal{T}} M_t$ is the average number of nodes per task. As a result, the time complexity for the solution of the largest MCF problem is:

$$O((\overline{M}|\mathcal{A}||\mathcal{T}|)^2 log(|\mathcal{A}| + \overline{M}|\mathcal{T}|) + \overline{M}|\mathcal{A}||\mathcal{T}|(|\mathcal{A}| + \overline{M}|\mathcal{T}|)(log(|\mathcal{A}| + \overline{M}|\mathcal{T}|))^2)$$

It is easy to verify that the time complexity of the RNN approach is better than the one of the MCF approach by at least a factor of $O(\overline{M}^2|\mathcal{T}|log(|\mathcal{A}| + \overline{M}|\mathcal{T}|))$.

The approach proposed in this section is a modified version of the MCF construction based heuristic that was proposed for the solution of the WTA problem in [12]. However, our approach is different in several ways. Firstly, the arc costs in the network are different due to the incurred asset assignment costs $C_a(a, t)$ which are not present in the WTA problem. Secondly, we have modified the network structure to address the possibility of not assigning a particular asset to any task. Thirdly, instead of using a predefined constant number of transshipment nodes for each task, we have chosen to use a different number of transhipment nodes for each task according to the maximum possible number of assignments for a specific task. Finally, we have introduced the MCFrnn approach to efficiently estimate the network arc costs. As will be shown in section 5.7, this approach leads to the best results overall without adding on the time complexity of the MCF method.

Finally, we should emphasise that the MCF approaches are deterministic. This means that for given input parameters to the optimisation problem, they will always produce the same output if executed several times. As a result, these approaches can be used for decision making in a distributed manner, similar to the RNN parameter association approach discussed in section 5.4.

5.6. Obtaining tight lower bounds

To assess the performance of the discussed algorithms we have developed a tight lower bounding scheme which is based on deriving a piecewise linear approximation

of the cost function of alternative formulation (5.11) of our problem:

$$\begin{aligned} \min C \quad &= \quad \sum_{t\in\mathcal{T}} U(t)2^{-z_t} + \sum_{t\in\mathcal{T}}\sum_{a\in\mathcal{A}} C_a(a,t)X(a,t) \\ s.t. \quad & \sum_{t\in\mathcal{T}} X(a,t) \leq 1,\ \forall a \\ & \sum_{a\in\mathcal{A}} h(a,t)X(a,t) = z_t,\ \forall t \\ & X(a,t) \in \{0,1\}, \forall a,t \text{ and } z_t \geq 0, \forall t \end{aligned}$$

To derive formulation (5.11) let us assume that:

$$2^{-z_t} = \prod_{a\in\mathcal{A}} p_f(a,t)^{X(a,t)} \tag{5.11}$$

Taking the logarithm in both sides of Eq. (5.11) gives:

$$z_t = -\sum_{a\in\mathcal{A}} X(a,t)\log_2 p_f(a,t) = \sum_{a\in\mathcal{A}} X(a,t)h(a,t) \tag{5.12}$$

where $h(a,t) = -\log_2 p_f(a,t)$. Because $0 < p_f(a,t) \leq 1$, it is true that $h(a,t) \geq 0$ and $z_t \geq 0$. Substitution of Eq. (5.12) into (5.2) yields formulation (5.11).

Although the objective function remains nonlinear we can approximate the terms $2^{-z_t},\ \forall t$ by a piecewise linear approximation function $\phi(z_t)$. In fact, we can obtain a lower bound to the problem's cost C_{LB}, if we make sure that $2^{-z_t} \geq \phi(z_t),\ z_t \geq 0$. This can be achieved by taking the upper envelop of a number of lines tangent to 2^{-z_t}.The piecewise linear approximation approach described above, was firstly proposed in [134] for the solution of the WTA problem.

Next, we will discuss how we can reduce the approximation interval for the 2^{-z_t} terms and describe a piecewise linear approximation scheme that restricts the maximum error to a desired value. Then, we will show how we can even remove $\phi(z_t),\ \forall t$ from the formulation transforming it into a mixed integer optimisation problem with a linear objective function [141].

One problem with the piecewise linear approximation is the large approximation range which in our case is $z_t \geq 0$. We show that we only need to approximate the function for a specific range of values. Clearly, the lower bound for variable z_t is 0, attained when no asset is assigned to task t. If there is at least one allocation for

task t then $z_t \geq z_t^{min} > 0$, where z_t^{min} is the smallest positive value of z_t. This value is acquired by assigning the asset with the largest execution failure probability to task t,

$$z_t^{min} = -\log_2(\max_a\{p_f(a,t)\}) \tag{5.13}$$

We can also limit the approximation range by deriving an upper bound for z_t. Due to the asset cost $C_a(a,t)$, assignments are made only when the net expected reduction in the objective function is positive. In other words, a new asset a can be assigned to task t if $U_{cur}(t)p_s(a,t) - C_a(a,t) > 0$, where $U_{cur}(t)$ is the expected cost of the task due to other assignments. If we consider the marginal case $U_{mar}(t) = C_a(a,t)/p_s(a,t)$, then the expected task cost after the assignment is made will be equal to $U_{mar}(t)p_f(a,t) = C_a(a,t)p_f(a,t)/p_s(a,t)$. Hence the smallest expected cost that we can obtain for task t is $U_{min}(t) = \min_{a\in\mathcal{A}}\{C_a(a,t)p_f(a,t)/p_s(a,t)\}$ so that:

$$\begin{aligned} z_t^{max} &= -\log_2(U_{min}(t)/U(t)) \\ &= -\log_2\left(\frac{\min_{a\in\mathcal{A}}\{\frac{C_a(a,t)p_f(a,t)}{p_s(a,t)}\}}{U(t)}\right) \end{aligned} \tag{5.14}$$

One approach for obtaining the piecewise linear approximation of a function is to take the upper envelop of the lines tangent to it at integer multiples of a parameter κ. Nonetheless, using this approach we do not have any knowledge of the approximation error, while the maximum error for different segments varies. Our approach is to create lines whenever needed so that the approximation error does not exceed a predefined value e_{max}. Although this approach may lead to a large number of segments we can adjust e_{max} to achieve the desirable number of segments and at the same time maintain the error less than a constant known value.

Fig. 5.2 depicts an example of piecewise linear approximation of term 2^{-z_t} when $e_{max} = 0.02$, $z_t^{min} = 1.94$ and $z_t^{max} = 8.62$. The solid thin line represents 2^{-z_t} while the dashed lines represent the various approximating lines. The thick line, which is the upper envelop of the approximating lines, corresponds to $\phi(z_t)$. On Fig. 5.2 the piecewise linear approximation function is tangent to 2^{-z_t} at points P_1, P_5 and P_7, whereas P_2, P_4 and P_6 are points of maximum approximation error.

In order to create $\phi(z_t)$ we follow an iterative procedure which involves the numer-

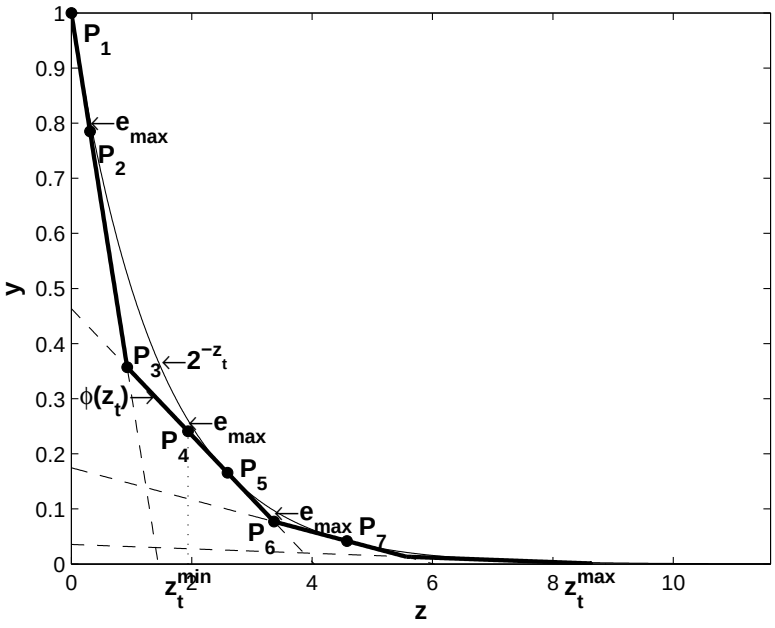


Figure 5.2.: Piecewise linear approximation

ical solution of two equations. The first equation corresponds to the case that a line segment is tangent to 2^{-z_t} at a known point $(z_t^{tp},\ 2^{-z_t^{tp}})$ with slope $\alpha = -2^{-z_t^{tp}} \ln 2$; our aim is to find the point $z_t^{ep} > z_t^{tp}$ where the approximation error is equal to e_{max}, as described in Eq. (5.15).

$$2^{-z_t^{ep}} - \alpha(z_t^{ep} - z_t^{tp}) + 2^{-z_t^{tp}} = e_{max} \tag{5.15}$$

The second equation corresponds to the case that one point of maximum approximation error, $(z_t^{ep},\ 2^{-z_t^{ep}} - e_{max})$, of a particular line segment is known and we want to obtain the point $z_t^{tp} > z_t^{ep}$ that is tangent to 2^{-z_t}. This point is determined from the numerical solution of Eq. (5.16):

$$\frac{2^{-z_t^{tp}} - (2^{-z_t^{ep}} - e_{max})}{z_t^{tp} - z_t^{ep}} = -2^{-z_t^{tp}} \ln 2 \tag{5.16}$$

We now outline the approach followed to derive $\phi(z_t)$. Starting from point (0,1) we solve Eq. (5.15) to obtain the point $z_t^{P_2}$ where the approximation error is equal to e_{max}. Since no approximation is needed in the interval $0 < z_t < z_t^{min}$, we set

the start of the new segment to $z_t^{P_2}$ only when $z_t^{P_2} > z_t^{min}$ otherwise we impose point $(z_t^{min}, 2^{-z_t^{min}} - e_{max})$ to be a point on the second line segment (point P_4). Then the procedure alternates between the solution of Eq. (5.16) and (5.15). The second segment is fully determined by solving Eq. (5.16) to obtain point P_5. Then, point P_5 is used to find P_6 by solving Eq. (5.15); point P_6 denotes the end of the particular line segment and the start of the next one. The described procedure is repeated until an attained point is larger than z_t^{max}.

Having obtained an approximating function $\phi(z_t), t \in \mathcal{T}$, we now describe how to obtain a problem formulation with a linear cost function. Let us assume that the term 2^{-z_t} is approximated by L_t linear segments with slopes $\alpha_t^{(1)}$,...,$\alpha^{(L_t)}$ and start-points $z_t^{(1)}$,...,$z_t^{(L_t)}$. Let us also assume that $z_t^{(L_t+1)} = z_t^{max}$. Because 2^{-z_t} is convex, the envelop approximation $\phi(z_t)$ will also be convex and the slopes will have monotone increasing values: $\alpha_t^{(1)} < \alpha_t^{(2)} < ... < \alpha^{(L_t)}$. Let $\xi_t^{(l)}, l = 1, ..., L_t$ be the value of z_t corresponding to the lth linear segment so that $0 \leq \xi_t^{(l)} \leq z_t^{(l+1)} - z_t^{(l)},\ l = 1, ..., L_t$. Under the assumption that $\xi_t^{(i)} = z_t^{(i+1)} - z_t^{(i)}, i = 1, ..., l-1$ when $\xi_t^{(l)} > 0$, it is true that $z_t = \sum_{l=1}^{L_t} \xi_t^{(l)}$ and also that $\phi(z_t) = \sum_{l=1}^{L_t} \alpha_t^{(l)} \xi_t^{(l)}$. In other words, z_t can be replaced by the sum of variables $\xi_t^{(l)}, l = 1, ..., L_t$ if we can ensure that the solution of the optimisation problem will always be such that each $\xi_t^{(l)}$ is nonzero only when the variables $\xi_t^{(i)},\ i = 1, \ldots, l-1$ have obtained their maximum value. As mentioned earlier, $\alpha_t^{(1)}$ has the smallest slope value and hence $\xi_t^{(1)}$ will be the first variable associated with z_t to be assigned a nonzero value. Only when $\xi_t^{(1)}$ has been assigned its maximum value variable $\xi_t^{(2)}$ will be assigned a nonzero value and this procedure will continue until z_t becomes equal to the sum of the nonzero variables. Thus, the assumption stated above is satisfied and formulation (5.11) becomes:

$$
\begin{aligned}
\min \quad & \sum_{t \in \mathcal{T}} U(t) \sum_{l=1}^{L_t} \alpha_t^{(l)} \xi_t^{(l)} + \sum_{t \in \mathcal{T}} \sum_{a \in \mathcal{A}} C_a(a,t) X(a,t) \qquad (5.17) \\
s.t. \quad & \sum_{t \in \mathcal{T}} X(a,t) \leq 1,\ \forall a \\
& \sum_{a \in \mathcal{A}} h(a,t) X(a,t) = \sum_{l=1}^{L_t} \xi_t^{(l)},\ \forall t \\
& 0 \leq \xi_t^{(l)} \leq z_t^{(l+1)} - z_t^{(l)},\ l = 1, ..., L_t,\ \forall t \\
& X(a,t) \in \{0,1\}, \forall a, t
\end{aligned}
$$

$\|\mathcal{A}\|$	$\|\mathcal{T}\|$	$\|\mathcal{A}\|/\|\mathcal{T}\|$	RNN	MMR	MCFmax	MCFmin	MCFrnn
6	3	2.0	2.23	4.12	4.40	**0.61**	0.74
6	6	1.0	4.40	7.98	0.98	**0.05**	**0.05**
6	9	0.67	1.05	4.84	0.18	**0.00**	**0.00**
6	12	0.5	0.78	2.83	**0.00**	**0.00**	**0.00**
6	15	0.4	0.51	1.85	**0.00**	**0.00**	**0.00**
3	6	0.5	0.38	1.66	0.01	**0.00**	**0.00**
9	6	1.5	2.02	5.56	4.87	**0.28**	0.31
12	6	2.0	1.69	4.08	8.15	**0.56**	0.73
15	6	2.5	1.98	4.05	9.38	**0.84**	0.87
Overall Perf.			1.67	4.11	3.11	**0.26**	0.30

Table 5.1.: Average relative percentage deviation from the optimal solutions of data family 1

Formulation (5.17) is a linear mixed integer program that can be solved using a standard combinatorial optimisation solver.

We are now ready to outline the steps of the lower bounding algorithm (LBA) for problem (5.2):

1. For each task compute z_t^{min} and z_t^{max} using Eqs. (5.13) and (5.14) respectively.

2. Follow the proposed piecewise linear approximation scheme to compute the L_t linear segments with slopes $\alpha_t^{(1)}$,...,$\alpha^{(L_t)}$ and start-points $z_t^{(1)}$,...,$z_t^{(L_t)}$, $z_t^{(L_t+1)} = z_t^{max}$ based on the desired value of e_{max}.

3. Use an integer programming solver to solve problem (5.17). The obtained objective function cost corresponds to a lower bound of the original problem.

5.7. Evaluation

The effectiveness of the proposed algorithms was tested with respect to two generated data families. In data family 1, the problem parameters are independently generated, while in data family 2 there is positive correlation between the cost of an asset and its associated execution success probabilities, so that "better" assets are more expensive.

In both data families, parameters $U(t)$ for each task in $\mathcal{T}$ are generated from the uniform distribution in the interval [10,200]. In data family 1 the other two problem parameters also follow the uniform distribution. The cost of assignment

$\|\mathcal{A}\|$	$\|\mathcal{T}\|$	$\|\mathcal{A}\|/\|\mathcal{T}\|$	RNN	MMR	MCFmax	MCFmin	MCFrnn
6	3	2.0	4.70	4.21	4.21	**2.24**	2.55
6	6	1.0	0.95	2.02	0.49	1.05	**0.39**
6	9	0.67	0.04	0.27	**0.01**	0.05	**0.01**
6	12	0.5	0.02	0.05	**0.00**	**0.00**	**0.00**
6	15	0.4	0.01	0.03	**0.00**	**0.00**	**0.00**
3	6	0.5	0.03	0.06	**0.00**	0.01	0.02
9	6	1.5	2.45	3.16	2.69	1.50	**1.36**
12	6	2.0	2.94	3.23	5.00	**2.72**	2.81
15	6	2.5	4.14	4.29	6.65	**4.10**	4.25
Overall Perf.			1.70	1.92	2.12	1.30	**1.26**

Table 5.2.: Average relative percentage deviation from the optimal solutions of data family 2

$C_a(a) \in [4, 30],\ a \in \mathcal{A}$ is taken to be independent from its assigned task, while for the execution failure probabilities we have that $p_f(a,t) \in [0.05, 0.4]$. In data family 2, the execution failure probabilities are taken to be independent from the tasks, i.e. $p_f(a) \in [0.05, 0.4]$, while the associated asset costs $C_a(a)$ are drawn from the normal distribution with mean $\overline{C}_a(a)$ and variance $0.1\overline{C}_a(a)$. The parameters $\overline{C}_a(a)$ are calculated from linear equation (5.18) that connects points ($p_{f,max} = 0.4, \overline{C}_{a,min} = 4$) and ($p_{f,min} = 0.05, \overline{C}_{a,max} = 30$).

$$\overline{C}_a(a) = \frac{(\overline{C}_{a,max} - \overline{C}_{a,min})}{(p_{f,min} - p_{f,max})}(p_f(a) - p_{f,max}) + \overline{C}_{a,min} \tag{5.18}$$

To test the effectiveness of the proposed algorithms, we have performed two sets of experiments. The first set of experiments is conducted with small size asset-task problem sets where the optimal solution can be obtained, while the second set is conducted with large size problems and the results are compared against the lower bounding algorithm (LBA).

For the small size problems, we have generated problem instances for constant number of assets, $|\mathcal{A}| = 6$, and varying number of tasks $|\mathcal{T}| = \{3,\ 6,\ 9,\ 12,\ 15\}$ as well as for constant number of tasks, $|\mathcal{T}| = 6$, and varying number of assets $|\mathcal{A}| = \{3,\ 9,\ 12,\ 15\}$, for both data families described above. To compare the performance of the various algorithms we have used as performance criterion the average relative percentage deviation from the optimal solution (obtained by enumeration), σ_{opt}, defined as:

$$\sigma_{opt} = \frac{1}{N_{PI}} \sum_{i=1}^{N_{PI}} \frac{C_{alg,i} - C_{opt,i}}{C_{opt,i}} \times 100 \tag{5.19}$$

where $C_{opt,i}$ is the cost of the optimal solution obtained from the solution of problem (5.2) for instance i, $C_{alg,i}$ is the cost obtained from the heuristic algorithm used, and N_{PI} is the total number of problem instances considered in each case. The number of problem instances used was $N_{PI} = 300$ in all cases.

The results are summarised in Tables 5.1 and 5.2 for data family 1 and 2 respectively. The bold letters denote the best performing approach or approaches for each $(|\mathcal{A}|, |\mathcal{T}|)$ pair. For data family 1, clearly the best performing method is the MCFmin with average relative percentage deviation from the optimal not exceeding 1% in all cases, as is the case for the MCFrnn which achieves slightly worse results. Remarkably, both MCFmin and MCFrnn obtain optimal results in all cases that the number of assets is smaller than the number of tasks. MCFmax performs well only when $|\mathcal{A}| > |\mathcal{T}|$, while its performance is the worst for the other cases. The RNN approach clearly outperforms the maximum marginal return algorithm and in all cases $\sigma_{opt} < 5\%$.

The results obtained for data family 2 are overall worse than those of data family 1. The MCFmin algorithm is better only for the cases that $|\mathcal{A}| \geq 2|\mathcal{T}|$, while for the other cases the best performing algorithm is MCFrnn. The RNN approach again outperforms the MMR algorithm, and its performance remains within 5% of optimality. Moreover, RNN's effectiveness is comparable to the best performing methods MCFmin and MCFrnn. From this first set of experiments we can conclude that the MCFmin and MCFrnn approaches are the best performing algorithms, whilst the performance of the algorithms significantly depends on the ratio $|\mathcal{A}|/|\mathcal{T}|$.

We have performed a second set of experiments with large problem instances, for several $(|\mathcal{A}|, |\mathcal{T}|)$ pairs with up to 200 assets and 200 tasks. Due to the large size of the problems, the optimal solution is difficult to be derived, so the performance of the algorithms is compared against tight lower bounds obtained from the solution of (5.17). To obtain tight lower bounds in relatively short execution period, the parameter e_{max} has been carefully selected for each $(|\mathcal{A}|, |\mathcal{T}|)$ pair to accomplish a good trade-off between execution time and optimality. The performance measure that we use is the average relative percentage deviation from the lower bound, σ_{LB},

$\|\mathcal{A}\|$	$\|\mathcal{T}\|$	$\|\mathcal{A}\|/\|\mathcal{T}\|$	RNN	MMR	MCFmax	MCFmin	MCFrnn	LBA
20	80	0.25	0.94	1.31	**0.35**	**0.35**	**0.35**	0.37
40	120	0.33	1.55	2.15	**0.67**	**0.67**	**0.67**	0.75
10	20	0.5	1.14	3.31	0.16	**0.15**	**0.15**	0.15
20	40	0.5	1.95	4.07	**0.61**	**0.61**	**0.61**	0.63
40	80	0.5	2.37	4.16	**0.92**	**0.92**	**0.92**	0.98
80	160	0.5	2.54	3.91	**1.24**	**1.24**	**1.24**	1.37
100	200	0.5	2.44	3.66	**1.18**	**1.18**	**1.18**	1.29
10	10	1.0	4.56	8.77	1.62	**0.12**	**0.12**	0.09
20	20	1.0	3.32	8.10	2.07	**0.10**	**0.10**	0.10
40	40	1.0	2.42	8.56	2.04	**0.11**	**0.11**	0.11
80	80	1.0	2.20	7.47	1.82	**0.12**	**0.12**	0.12
100	100	1.0	2.14	7.02	1.77	**0.12**	**0.12**	0.13
10	5	2.0	1.82	3.67	6.96	**0.49**	0.66	0.05
20	10	2.0	2.30	6.04	11.07	**0.39**	0.47	0.06
40	20	2.0	2.68	6.72	14.80	**0.24**	0.28	0.07
80	40	2.0	2.82	7.20	17.55	**0.09**	0.12	0.08
100	50	2.0	2.93	7.25	18.27	**0.09**	0.11	0.09
200	100	2.0	2.87	6.55	20.06	**0.09**	**0.09**	0.10
40	10	4.0	2.60	4.13	12.63	**1.09**	1.16	0.09
80	20	4.0	2.74	5.05	14.19	0.80	**0.79**	0.10
Overall Perf.			2.42	5.50	6.50	**0.45**	0.47	0.34

Table 5.3.: Average relative percentage deviation from the lower bound of data family 1

$\|\mathcal{A}\|$	$\|\mathcal{T}\|$	$\|\mathcal{A}\|/\|\mathcal{T}\|$	RNN	MMR	MCFmax	MCFmin	MCFrnn	LBA
20	80	0.25	**0.27**	0.28	**0.27**	**0.27**	**0.27**	0.29
40	120	0.33	**0.42**	0.43	**0.42**	**0.42**	**0.42**	0.45
10	20	0.5	0.49	0.53	**0.48**	**0.48**	**0.48**	0.51
20	40	0.5	**0.49**	0.54	**0.49**	**0.49**	**0.49**	0.53
40	80	0.5	**0.51**	0.56	**0.51**	**0.51**	**0.51**	0.55
80	160	0.5	**0.68**	0.74	**0.68**	**0.68**	**0.68**	0.75
100	200	0.5	0.72	0.77	**0.64**	0.72	0.72	0.80
10	10	1.0	0.83	2.55	0.64	1.34	**0.47**	0.07
20	20	1.0	0.71	3.03	0.77	1.58	**0.60**	0.09
40	40	1.0	0.67	3.38	0.82	1.71	**0.65**	0.14
80	80	1.0	**0.78**	3.72	1.01	1.83	0.81	0.52
100	100	1.0	**0.93**	3.93	1.19	2.02	1.64	0.79
10	5	2.0	3.30	3.61	4.87	2.64	**2.54**	0.04
20	10	2.0	2.92	3.38	5.58	3.04	**2.89**	0.05
40	20	2.0	**2.62**	3.43	5.98	3.15	3.04	0.08
80	40	2.0	**2.32**	3.32	6.27	3.26	3.09	0.80
100	50	2.0	**2.38**	3.39	6.44	3.43	3.31	0.58
200	100	2.0	**2.12**	3.30	6.33	3.53	3.32	1.21
40	10	4.0	6.84	**6.76**	9.93	7.47	7.68	0.06
80	20	4.0	6.28	**6.08**	9.86	7.13	7.45	0.09
Overall Perf.			**1.81**	2.69	3.16	2.28	2.05	0.42

Table 5.4.: Average relative percentage deviation from the lower bound of data family 2

defined as:

$$\sigma_{LB} = \frac{1}{N_{PI}} \sum_{i=1}^{N_{PI}} \frac{C_{alg,i} - C_{LB,i}}{C_{LB,i}} \times 100 \tag{5.20}$$

where $C_{LB,i}$ is the cost of the lower bound obtained from the solution of (5.17).

We report σ_{LB} for the various algorithms in Tables 5.3 and 5.4. Column LBA corresponds to the cost of the original problem (5.2), computed using the solution obtained from the lower bounding algorithm. LBA is only considered to demonstrate the tightness of the lower bounds and is not compared with the other methods, as it is not of polynomial complexity. Moreover, the MMR is a greedy heuristic, where in each iteration we select the assignment corresponding to the maximum marginal return, represented by term $max\{0, b(a,t)\}$.

For data family 1 the most effective algorithms are the network flow approaches MCFmin and MCFrnn for all $(|\mathcal{A}|, |\mathcal{T}|)$ pairs, which have almost the same efficiency and achieve $\sigma_{LB} < 1.3\%$ in all cases. In addition, these algorithms even outperform the LBA approach for the cases that $|\mathcal{A}|/|\mathcal{T}| \leq 1$. Additionally, MCFmax only performs well when $|\mathcal{A}|/|\mathcal{T}| \leq 1$. The RNN approach is worse than the network flow heuristics, but performs by approximately 4% better than the MMR algorithm.

Interestingly, for data family 2 the best overall performing algorithm is the RNN. RNN performs better than the other approaches for large problem instances when $|\mathcal{A}| = |\mathcal{T}|$ and $2|\mathcal{A}| = |\mathcal{T}|$, while for the other problems its performance is highly competitive. MCFrnn achieves better results than the MCFmin approach, especially for the problem sets with equal number of assets and tasks. The performance of the MMR approach is well improved compared to data family 1; in fact the MMR obtains the best results for problem instances with $|\mathcal{A}|/|\mathcal{T}| = 4$. Finally, the MCFmax is again the least effective approach.

Summarising the results, we could argue that the best performing method is the MCFrnn since it performs similar to the MCFmin for data family 1, whilst it is more effective on data family 2. At this point, it is important to highlight that the RNN solution attained in MCFrnn is not considered to improve the performance of the MCFrnn. If this is taken into account, then the outcome from the MCFrnn algorithm will always be at least equal to the outcome of RNN, as we can choose the best solution amongst the two approaches; in that case, the MCFrnn would be the best performing method for data family 2. Finally, the performance of the RNN approach should also be emphasised because even though its overall performance is

by 2% worse than the MCF approaches in the data family 1, it is the most efficient method for data family 2, while it is of lower polynomial time complexity compare to the MCF approaches.

5.8. Conclusions

In this chapter we have studied an asset-task allocation problem when an asset may fail to execute an assigned task. For its solution, we have proposed two approaches based on the random neural network and the minimum cost flow problem. We have also proposed an algorithm for obtaining tight lower bounds to the optimal solution of the examined problem. The proposed RNN and MCF approaches are deterministic and of polynomial time complexity so that they can be utilised by the assets (the decision makers) to obtain fast, distributed, close to optimal and consistent solutions if all assets are provided with the same information.

We have tested the efficiency of the various algorithms with respect to two different data families, the one with all parameters being independent and the other with the assignment costs having a positive correlation to the execution success probabilities. The results that we have obtained confirm the usefulness of a simple RNN based heuristic for the fast and efficient solution of problem (5.2), especially when there is positive correlation between the asset assignment costs and the success probabilities, achieving on average performance within 5% of optimality for almost all problem sets examined. With respect to the proposed network flow algorithms, we could argue that MCFmin and MCFrnn have better overall performance than the MMR and RNN heuristics, with the MCFrnn being the most successful; it performs equally well with the MCFmin for the independent data family and it is more effective on the dependent data. This indicates that the incorporation of RNN into the MCF approach has a positive effect. Although the MCF approaches are more effective than the RNN approach overall, the time complexity of the latter is better by a factor of $O(|\mathcal{T}|\overline{M}^2 log(|\mathcal{A}| + \overline{M}|\mathcal{T}|))$.

6. Connecting trapped civilians to a wireless ad-hoc robotic network

During a disaster, emergency response operations can benefit from the establishment of a wireless ad-hoc network. We propose the use of autonomous robots that move inside a disaster area and establish a network for two-way communication between trapped civilians with *a priori* known or uncertain locations and an operations centre. Our aim is to maximise the number of civilians connected to the network. We present a centralised approach which is based on network flows, as well as distributed algorithms which involve clustering possible locations of civilians; clustering facilitates both connectivity within groups of civilians and exploration when the civilian locations are uncertain. We proposed an algorithm for the case that the locations of the civilians are known, as well as two algorithms for the case that the locations of the civilians are uncertain. The first is based on performing dynamic clustering of the locations of the civilians as the exploration-connectivity process evolves and the other is based on constructing a minimum spanning tree of the initially identified clusters which facilitates the movement of the robots and the parallel exploration of more than one clusters simultaneously.

The chapter is structured as follows. We start with a general description and the motivation for the examined problem. In section 6.2 we review related research topics to set the theoretical ground, followed by the assumptions that we make for the solution of the problem. In section 6.4, we describe a centralised mathematical formulation of the problem that can be applied both for *a priori* known or uncertain locations of civilians and provide numerical results to demonstrate the validity and the efficiency of the formulation. Apart from the centralised approach we also describe three versions of a distributed heuristic algorithm for the solution of the

problem, in section 6.5. The first version of the algorithm, deals with the problem when the locations of the civilians are *a priori* known; its efficiency is compared against that of the centralised approach in terms of the percentage of civilians connected. The second version, which is a modified version of the first one, tackles the problem with uncertain civilian locations using a risk measure for economic theory called, *expected shortfall*. The third version extends the algorithm for uncertain civilian locations to improve its efficiency in terms of the time needed to connect the civilians and the total energy spend by the robots. Section 6.6 provides a summary of the conclusions.

6.1. Introduction

Mobile robots are routinely used in disaster management operations to reach areas that are inaccessible to humans. Usually, they are designed to search for victims, inspect the structural integrity of buildings, or detect hazardous materials, but with recent advances in small-size robotics and wireless communications, emergency response robots can also be used to form ad-hoc networks. Examples of robots, where this ad hoc networking paradigm applies include the Soryu III [171], which provides live video streaming and two-way voice connection with trapped civilians, and the Packbot that is designed for military operations [128]. The following typical large-scale emergency situation indicates the usefulness of such a robotic network:

An earthquake has demolished a large building block in a city; the rescuers have arrived and need to assess the situation. Traditionally, the best case scenario is that the trapped civilians use whistles or some more sophisticated radio-transmitting personal emergency device that facilitates their detection. From detection to rescue however, a long period may pass during which establishing and maintaining communication between the rescuers and trapped civilians is vital. During this period, the rescuers' job would be immensely assisted if instead of a simple notification device the civilians carried a device that would provide wireless connection with the rescuers, in the form of VoIP, live video streaming or even environmental and biomedical sensor data. In this way, the rescuers would be in position to better assess the health condition of the victims and the state of their local environment long before locating them. Given that the existing communication infrastructure may be partially or completely destroyed, a promising approach would be to em-

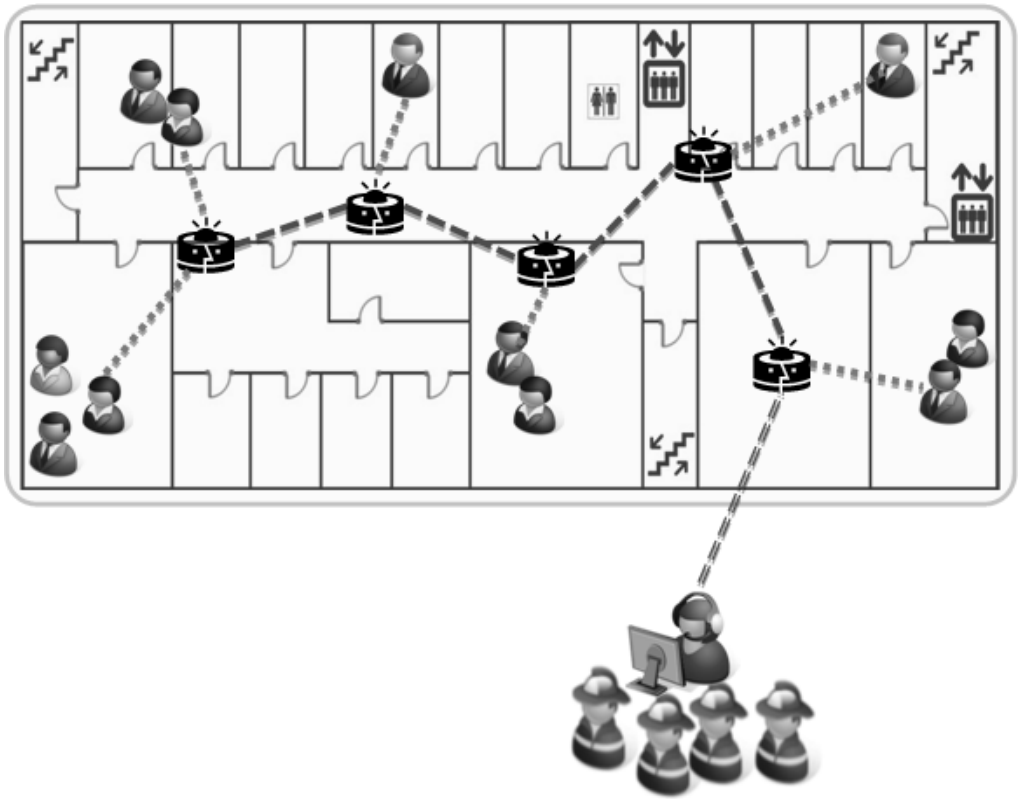


Figure 6.1.: Example scenario: A group of robots establish communication with trapped civilians

ploy mobile robots to act as wireless routers and form a network with the wireless devices of the trapped civilians (Fig. 6.1) to establish communication between the civilians and the operations centre.

For this emergency communication paradigm, the fact that we have a limited amount of robots means that they need to be deployed efficiently to optimise different key objectives, such as time for the formation of the network or energy limitations. Yet, the most important objective is to maximise the number of civilians that are connected to the operations centre via the robotic network while maintaining multi-hop connectivity between the robots; this is the problem that we deal with in this chapter.

6.2. Related work

Ad hoc networking for the collaboration of search and rescue robotic operations was first suggested in [176] and further investigated in [47, 145], but their authors assumed star topology with a control station in the centre of the search area, which is usually impractical during a disaster. A more general system was proposed in [168, 169], where networked robots collaborated to detect a single injured civilian and maintained their connection while moving. In this chapter, we tackle the fundamental problem of the optimal allocation of such robots that need to form an ad hoc network with all or as many static civilians as possible while at the same time being connected to a wireless sink over multiple hops. Related problems can be found in the field of network design, but they usually refer to wired networks and their goal is to select or add links to achieve some network objectives [90, 130]. In the mobile wireless case that we deal with, the addition of links is done implicitly with the location selection of the robots. A popular related problem found in the field of sensor networks is the positioning or scheduling of sensors to maximise area coverage while maintaining connectivity, which has produced a few interesting heuristics, such as [183, 179]. In our case, we deal with the connection of civilians instead of area coverage, and we provide both an exact centralised formulation and a distributed heuristic. Finally, mobility-assisted relocation has also been explored in sensor networks, but the focus in such networks is either the area coverage or the degree of connectivity, and not the connection of specific targets [178].

6.3. Assumptions

Our focus is on the constrained environments. Thus, instead of a continuous or grid representation of the area, we choose a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ representation, which is preferable for environments where there are limited movement options for the robots. We assume that the civilians carry a wireless device of range R_{civ}, which is used to connect with the robots. The wireless device of each robot has range R_{rob}. Several wireless coverage models have been proposed in the literature, but for the sake of simplicity we will use the euclidian distance as the connectivity criterion. A civilian $c \in \mathcal{C}$ is considered to be in two-way connection with a robot $r \in \mathcal{R}$ if their euclidian distance is smaller than the minimum of their respective ranges: $d(c, r) < \min\{R_{rob}, R_{civ}\}$. The robots need to maintain multi-hop connectivity

with a static wireless sink d_s, which represents the operations centre. A civilian can either be directly connected to the sink or via a robot that is in turn connected to it. Also, we assume that the movement graph $\mathcal{G}$ is known to the robots. Concerning the civilian locations, we examine two cases: (a) they are *a priori* known and (b) they are uncertain but we know the probability distribution of the number of civilians at each vertex.

6.4. A centralised approach

In this section we describe a centralised approach for the solution of the problem under examination which is suitable both for known locations as well as for uncertain locations of civilians. We also present numerical results to demonstrate the effectiveness and validity of the formulation. Finally, we examine composite cost functions which apart from the maximisation of connected civilians achieve the minimisation of the total number of robots used or the energy spent by the robots.

6.4.1. Formulation

Our goal is to find appropriate positions for the robots so that the number of connected civilians is maximised and the connectivity of the formed network is maintained. The robots have identical characteristics and hence do not need to be explicitly distinguished; we only need to find optimal locations for them. The binary decision variables $\mathbf{x} = \{x_i : i \in \mathcal{N}\}$ denote whether a robot should be positioned at vertex i.

To capture connectivity relationships, we use the binary matrices $\mathbf{C}^{RR}$ and $\mathbf{C}^{RC}$ to represent robot-to-robot and robot-to-civilian connectivity respectively. $\mathbf{C}^{RR}$ is a $|\mathcal{N}| \times |\mathcal{N}|$ matrix with its $C^{RR}_{i,j}$ element representing whether vertex $j \in \mathcal{N}$ is in range R_{rob} of vertex $i \in \mathcal{N}$. The $|\mathcal{C}| \times |\mathcal{N}|$ matrix $\mathbf{C}^{RC}$ has elements $C^{RC}_{c,i}$ equal to 1 if a robot at vertex i would be connected with civilian c.

To formulate the problem, we also need to introduce the binary variables $y_c, c \in \mathcal{C}$ which represent whether civilian c is connected to at least one robot. The problem

can be formulated as follows:

$$\text{maximize} \sum_{c \in \mathcal{C}} y_c \tag{6.1a}$$

$$\sum_{i \in \mathcal{N}} x_i <= |\mathcal{R}|, \tag{6.1b}$$

$$y_c = \min\{1, \sum_{i \in \mathcal{N}} x_i C^{RC}_{c,i}\}, c \in \mathcal{C} \tag{6.1c}$$

$$u(i,j) = \max\{0, C^{RR}_{i,j} x_i + C^{RR}_{j,i} x_j - 1\}, \ \forall i < j, \tag{6.1d}$$

$$u(j,i) = u(i,j), \ \forall i < j, \tag{6.1e}$$

$$u(i,i) = 0, \ i \in \mathcal{N} \tag{6.1f}$$

$$-\frac{1}{|\mathcal{R}|} x_i + \sum_{j \in \mathcal{N}} X_f(j,i) = \sum_{j \in \mathcal{N}} X_f(i,j), i \in \mathcal{N} \setminus \{d_s\} \tag{6.1g}$$

$$\frac{1}{|\mathcal{R}|} \sum_{j \in \mathcal{N} \setminus \{d_s\}} x_j + \sum_{j \in \mathcal{N}} X_f(j,i) = \sum_{j \in \mathcal{N}} X_f(i,j), i = d_s \tag{6.1h}$$

$$0 \leq X_f(i,j) \leq u(i,j), \ i,j \in \mathcal{N} \tag{6.1i}$$

$$x_i \in \{0,1\}, \ i \in \mathcal{N} \tag{6.1j}$$

where $|\mathcal{R}|$ is the number of robots, while $X_f(i,j)$ and $u(i,j)$ are continuous variables that denote the amount of flow and the capacity of link (i,j). A link capacity is nonzero if there are robots at vertices i and j and they are connected. The auxiliary network flow variables $X_f(i,j)$ and $u(i,j)$ have been employed to help us examine whether the robots maintain connectivity to each other for a given configuration, as will be explained below.

In the above formulation, Eq. (6.1b) indicates that we cannot use more than $|\mathcal{R}|$ robots, while Eq. (6.1c) shows whether civilian c is connected to the network. To formulate the connectivity matrix of the robots for given robot positions $\mathbf{x}$, we have employed a network flow formulation. If two robots at i and j are connected, then according to Eq. (6.1d) - (6.1f) the capacities $u(i,j)$ and $u(j,i)$ are equal to 1; otherwise they are equal to zero. Hence, the actual links of the formed network for specific robot locations are represented by the nonzero capacities. Connectivity is ensured if small hypothetical traffic flows from the sink d_s can reach all robot nodes. Transmission of traffic flows in the network implies that the flow conservation equations (6.1g)-(6.1i) are satisfied so that the total incoming traffic to any vertex

is equal to its total outgoing traffic.

In Eq. (6.1g) and (6.1h), the first term represents the supply $s(i)$ of vertex i which accounts for the amount of flow that enters the network from the outside. Note that a source vertex has positive supply $s(i) > 0$, a sink vertex negative supply $s(i) < 0$, and transshipment vertices have $s(i) = 0$ [26]. If the amount of traffic sent from the sink vertex to each of the robots is equal to $1/|\mathcal{R}|$, the total amount of traffic received by a link never exceeds 1 and the capacity constraint (6.1i) is never violated. Consequently, the set of Eq. (6.1g), (6.1h) is feasible only when all robots are connected to the sink. Also, the sink's supply must be equal to the robots' total demand so that $\sum_{i\in\mathcal{N}} s(i) = 0$; hence, the sink supply is equal to $\frac{1}{|\mathcal{R}|}\sum_{j\in\mathcal{N}\setminus\{d_s\}} x_j$.

Due to the presence of the *min* and *max* terms in Eq. (6.1c) and (6.1d) respectively, the above formulation is not linear. However, both expressions can be transformed into equivalent linear ones [1], which facilitates to a great extend the solution of the problem. In our case, Eq. (6.1c) is equivalent to the combination of Eq. (6.2b), (6.2c) and the goal function (6.2a). This is because when $\sum_{i\in\mathcal{N}} x_i C^{RC}_{c,i} = 0$, then Eq. (6.2b) and (6.2c) force y_c to zero. Additionally, when $\sum_{i\in\mathcal{N}} x_i C^{RC}_{c,i} \geq 1$, then $0 \leq y_c \leq 1$ and expression (6.2a) ensure that y_c will take the maximum value in that interval, i.e. $y_c = 1$.

$$\text{maximize} \sum_{c\in\mathcal{C}} y_c \tag{6.2a}$$

$$\sum_{i\in\mathcal{N}} x_i C^{RC}_{c,i} \geq y_c,\ c \in \mathcal{C} \tag{6.2b}$$

$$0 \leq y_c \leq 1,\ c \in \mathcal{C} \tag{6.2c}$$

Furthermore, constraint (6.1d) is equivalent to constraints (6.3a)-(6.3d). Eq. (6.3b) - (6.3d) force $u(i,j)$ to zero when not both terms $C^{RR}_{i,j}x_i$ and $C^{RR}_{j,i}x_j$ are equal to 1. In addition, when both terms are equal to 1 then from Eq. (6.3a) $u(i,j) \geq 1$ and from Eq. (6.3b) and (6.3c) $u(i,j) \leq 1$, implying that $u(i,j) = 1$.

[1] By linear, we mean here that if we relax the integrality constraints, use $x_i \in [0,1]$ instead of $x_i \in \{0,1\}$, we get a linear programming formulation

$$C^{RR}_{i,j}x_i + C^{RR}_{j,i}x_j - 1 \leq u(i,j), \forall i < j \tag{6.3a}$$
$$C^{RR}_{i,j}x_i \geq u(i,j), \forall i < j \tag{6.3b}$$
$$C^{RR}_{j,i}x_j \geq u(i,j), \forall i < j \tag{6.3c}$$
$$0 \leq u(i,j) \leq 1,\ i,j \in \mathcal{N} \tag{6.3d}$$

The described centralised formulation provides an optimal solution to the problem when the civilian positions are known. However, a slightly modified formulation can provide an optimal solution also in the case of uncertain locations.

Let $E[Z_i]$ represent the expected number of civilians at vertex i and y_i show if the civilians at vertex i are connected. Let also the $|\mathcal{N}| \times |\mathcal{N}|$ binary matrix $\mathbf{C}^{RL}$ represent the connectivity of robots with possible locations of civilians, where element $C^{RL}_{i,j}$ shows whether a robot at location i can connect civilians at location j. To obtain a formulation for the uncertain locations case we need only to replace expressions (6.1a) and (6.1c) in formulation (6.1) by expressions (6.4a) and (6.4b) respectively.

$$\text{maximize} \sum_{i \in \mathcal{N}} E[Z_i]y_i \tag{6.4a}$$
$$y_i = \min\{1, \sum_{j \in \mathcal{N}} x_j C^{RL}_{i,j}\},\ i \in \mathcal{N} \tag{6.4b}$$

6.4.2. Numerical results using the general centralised approach

We have performed numerical evaluation of our approach for varying robot and civilian ranges as well as number of robots. In all cases with constant number of robots and civilians we have used $|\mathcal{R}| = 20$ and $|\mathcal{C}| = 20$ respectively. For the solution of problem (6.1) a standard mixed integer programming solver was employed.

Fig. 6.2 shows three instances of optimally allocated robots for combinations of robot and civilian ranges. When the range of the wireless devices carried by the

civilians is too small, then in practice there must be one robot per each civilian in order to connect them, but for larger ranges the robots move to areas of higher civilian density. Fig. 6.3 illustrates the maximum number of connected civilians for different R_{rob} and R_{civ} values. It can be observed that in all cases the maximum possible number of connected civilians is achieved for $R_{civ} = 8m$. On the contrary, Fig. 6.4 shows that there is a threshold for R_{rob} in the range [6m,8m); below this threshold no civilian is connected, while above it the number of connected civilians increases substantially and reaches a maximum. In Fig. 6.5 the effect of the number of robots was examined for different combinations of civilian and robot ranges. Interestingly, the civilian connectivity increases almost linearly with the number of robots until full coverage is achieved.

6.4.3. Numerical results for uncertain civilian locations

For the case of uncertain civilian locations, we have generated 20 different problem instances with the number of civilians at each vertex obtained using the truncated exponential distribution with mean 0.25 civilians per vertex. In this case, the optimisation problem can be solved in a similar manner, by replacing expressions (6.1a) and (6.1c) in formulation (6.1) with expressions (6.4a) and (6.4b) respectively.

Figs. 6.6 and 6.7 illustrate the average percentage of civilians connected when the robot range and the number of robots vary respectively, for different combinations of civilian and robot ranges. In Fig. 6.6, the transition from no connectivity to high connectivity is much smoother than in the case where the civilian locations are known. In addition, for large robot ranges full connectivity is not guaranteed, as in Fig. 6.4, because due to the uncertainty the robots may take positions where no civilians can be connected, especially when $R_{civ} = 6m$ where the best connectivity achieved is only half of what is achieved for *a priori* known locations.

Interestingly, when the number of robots changes, the behaviour observed is similar to the case of certainty but the same connectivity is achieved for a larger number of robots, as illustrated on Fig. 6.7.

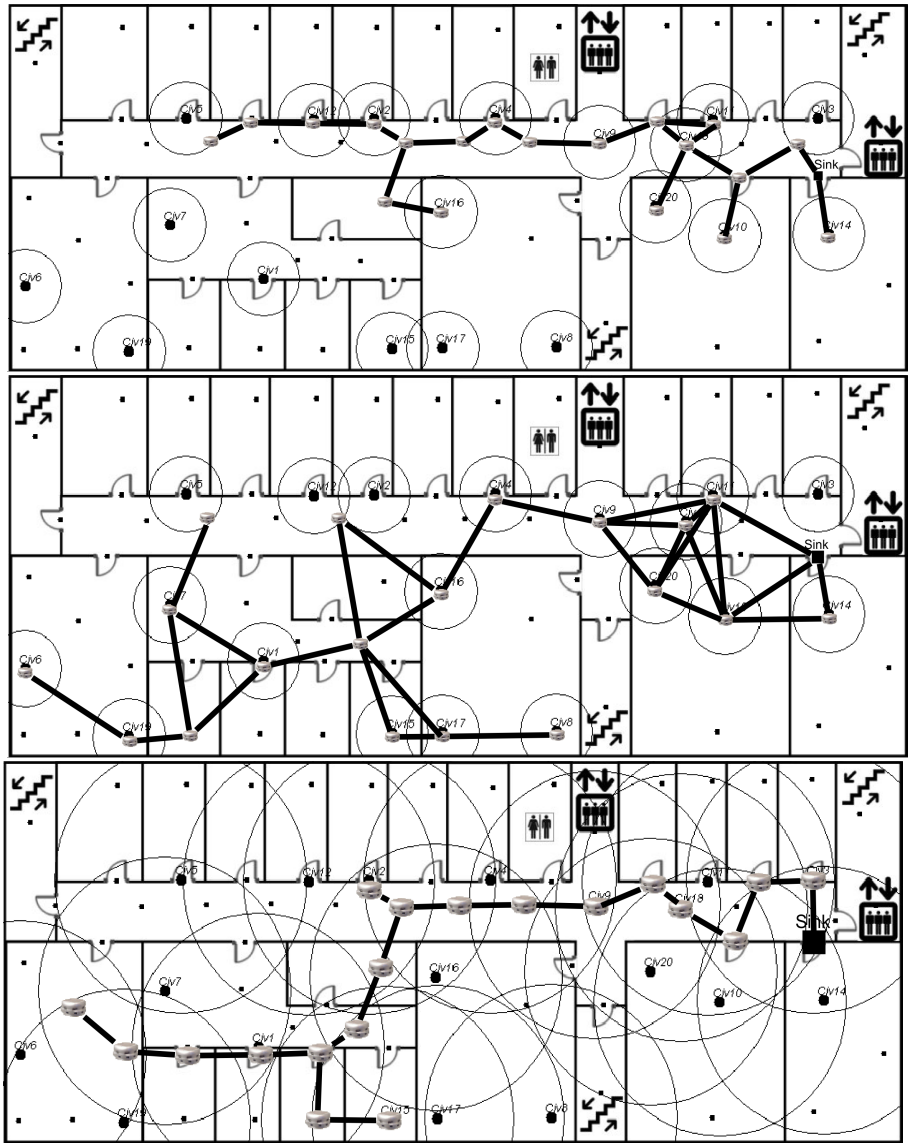


Figure 6.2.: Robot allocations according to the centralised solutions for (i)$R_{rob} = 8m$ and $R_{civ} = 4m$, (ii)$R_{rob} = 14m$ and $R_{civ} = 4m$, (iii)$R_{rob} = 8m$ and $R_{civ} = 14m$: When the civilian range is small one robot must be dedicated to each civilian, whereas when it is large one robot may suffice to connect multiple civilians. In addition, when the robot range is not large enough, a significant number of the robots must be used for maintaining connectivity; hence, they move to locations of high density of civilians for optimal performance.

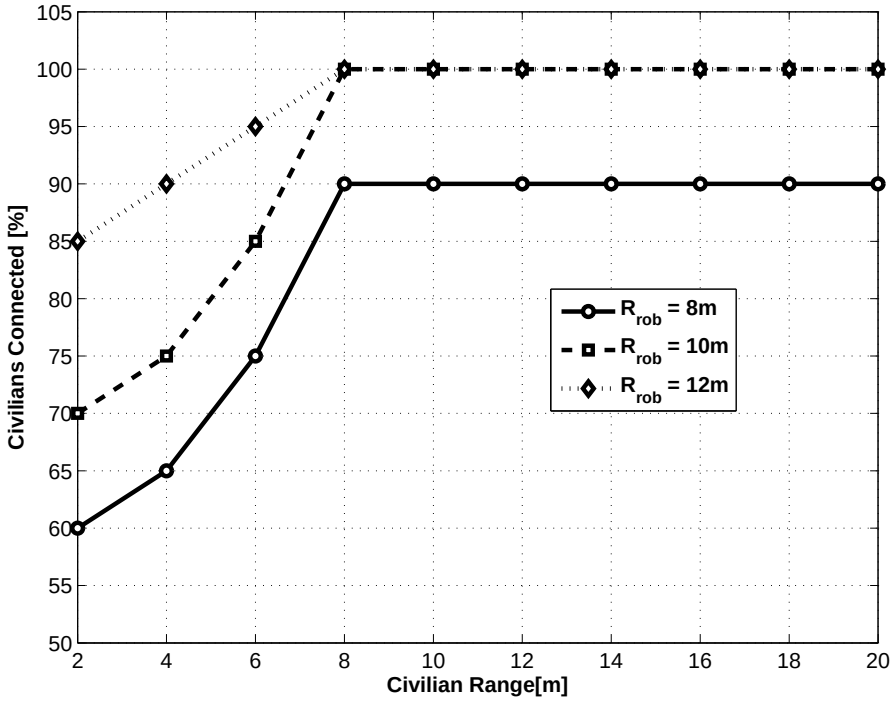


Figure 6.3.: Maximum number of connected civilians for varying R_{civ} and $R_{rob} = 8m, 10m, 12m$

6.4.4. Minimising the number of robots and the energy consumption

Although our primary goal is to maximise the number of connected civilians, it is also important to minimise the number of required robots or the locomotion energy used by the robots to achieve maximum connectivity. By locomotion energy we refer to the energy consumed by the robots to travel to their destinations. For this, we include these goals in the objective function and compare the results for the cases that emphasis is given in minimising a) the number of robots and b) the average locomotion energy consumed by the robots. We denote $E_{i,j}$ the normalised energy consumed by a robot to travel from vertex i to vertex j so that $0 \leq E_{i,j} \leq 1, \forall i, j$. We assume that the locomotion energy consumed by the robots while moving is analogous to the distance travelled.

Concerning the robot minimisation, the new objective function is the following:

$$\text{maximize } M_y \sum_{c \in \mathcal{C}} y_c - \sum_{i \in \mathcal{N}} x_i \tag{6.5}$$

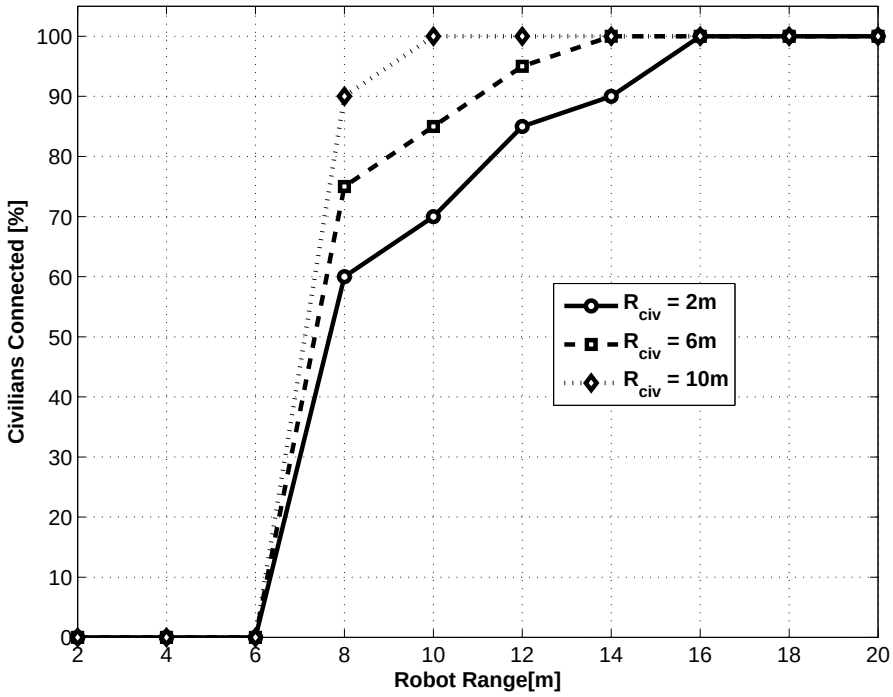


Figure 6.4.: Maximum number of connected civilians for varying R_{rob} and $R_{civ} = 2m, 6m, 10m$

where M_y is sufficiently large so that any increase of the civilian maximisation term is larger than the maximum possible value of the robot minimisation term. In this way, the optimal solution corresponds to the lexicographically greatest of the feasible ordered sets $(\sum_{c\in\mathcal{C}} y_c, -\sum_{i\in\mathcal{N}} x_i)$.

When we are interested in minimising the energy instead of the number of robots the objective function becomes:

$$\text{maximize } M_y \sum_{c\in\mathcal{C}} y_c - \sum_{i\in\mathcal{N}} E_{d_s,i} x_i \tag{6.6}$$

where again M_y must be sufficiently large so that the solution is the best lexicographically ordered feasible set $(\sum_{c\in\mathcal{C}} y_c, -\sum_{i\in\mathcal{N}} E_{d_s,i} x_i)$.

We have solved the centralised problem for objectives (6.5) and (6.6) for R_{rob} ranging from $10m$ to $20m$ and $R_{civ} = 12m$, computing the minimum number of robots required and the total consumed energy. We have selected the particular robot and civilian ranges because they have feasible solutions for the connection of all civilians as already shown in Fig. 6.3 - 6.4.

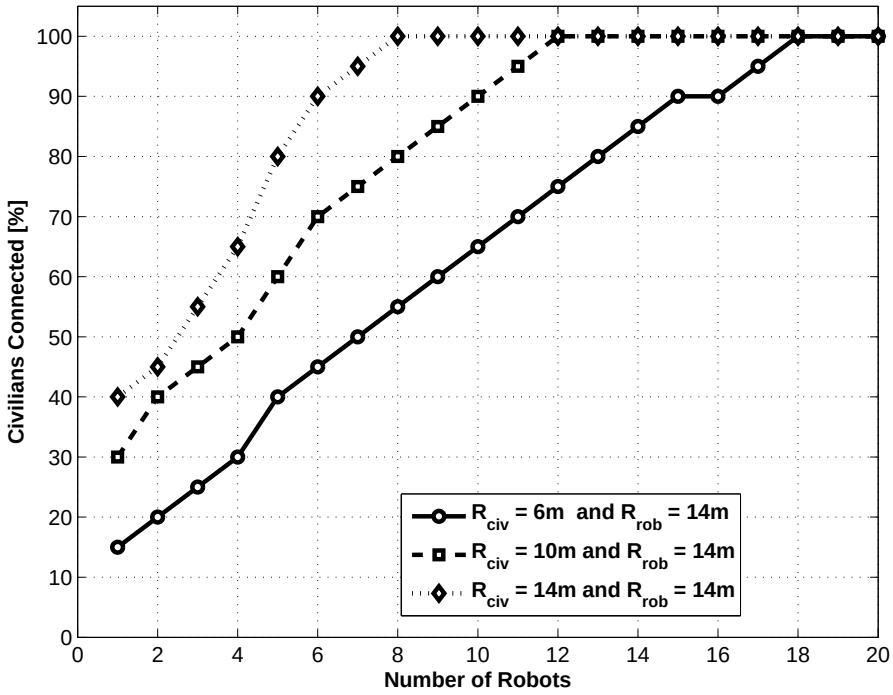


Figure 6.5.: Maximum number of connected civilians for varying number of robots and different combinations of ranges

A comparison of the efficiency of the objectives (6.2a), (6.5) and (6.6) in terms of the minimum number of robots used and the locomotion energy consumed, is illustrated on Fig. 6.8 and 6.9. In Fig. 6.8, it is clear that when using the objective function (6.2a) the results are significantly worse because no effort is put on minimising the number of robots. Interestingly, the results for the case of minimising the energy are very close to the optimum values which are obtained using objective (6.5).

The numerical results presented in Fig. 6.9 correspond to the locomotion energy consumption being linear with the distance, at $2.0J/m$, which is a typical value for small wireless-equipped robots, such as robomote [163]. Similar observations to the ones associated with Fig. 6.8 are valid for this case as well. Nonetheless, the energy consumed when utilising objective (6.5) is always more than the optimal in contrast to Fig. 6.8 where in some cases minimising the energy resulted in minimizing the number of robots required as well. Based on the above observations, we can argue that minimising the number of robots or the locomotion energy consumed are highly associated goals.

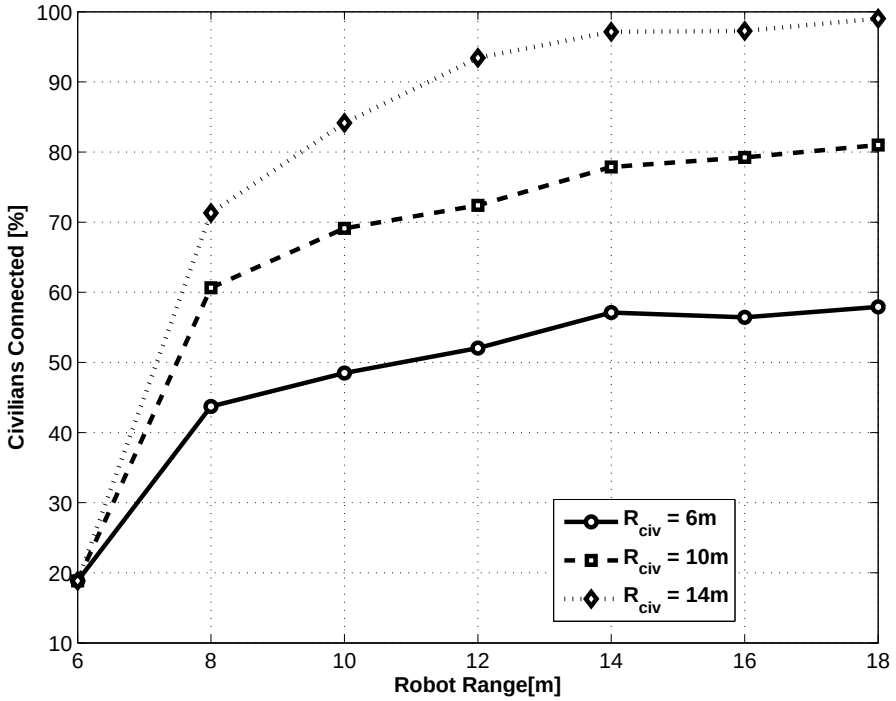


Figure 6.6.: Average percentage of connected civilians for varying R_{rob} and $R_{civ} = 2m, 6m, 10m$, for uncertain civilian locations

6.5. A distributed heuristic

The above problem is particularly challenging because connectivity between the robots must be maintained, which constraints their movements and requires their efficient cooperation to achieve their common goal. We have developed a distributed heuristic algorithm with which the robots can autonomously relocate in the disaster area and take appropriate actions independently and in a timely fashion. A general flow diagram of the algorithm is shown on Fig. 6.10.

The problem can be significantly simplified if the locations of the civilians are clustered so that their maximum radius is smaller than $R_{rob} + R_{civ}$. In that case, by locating a robot at the centre of this cluster, the connectivity constraint is always satisfied within the cluster (Fig. 6.14). Clustering the locations of the civilians in a disaster scenario is not unrealistic, because the civilians are naturally clustered in groups, either because they were together when the disaster occurred or grouped with others in their effort to survive. The robot that settles on the cluster centre acts as a cluster leader and is responsible to issue an exploration announcement to

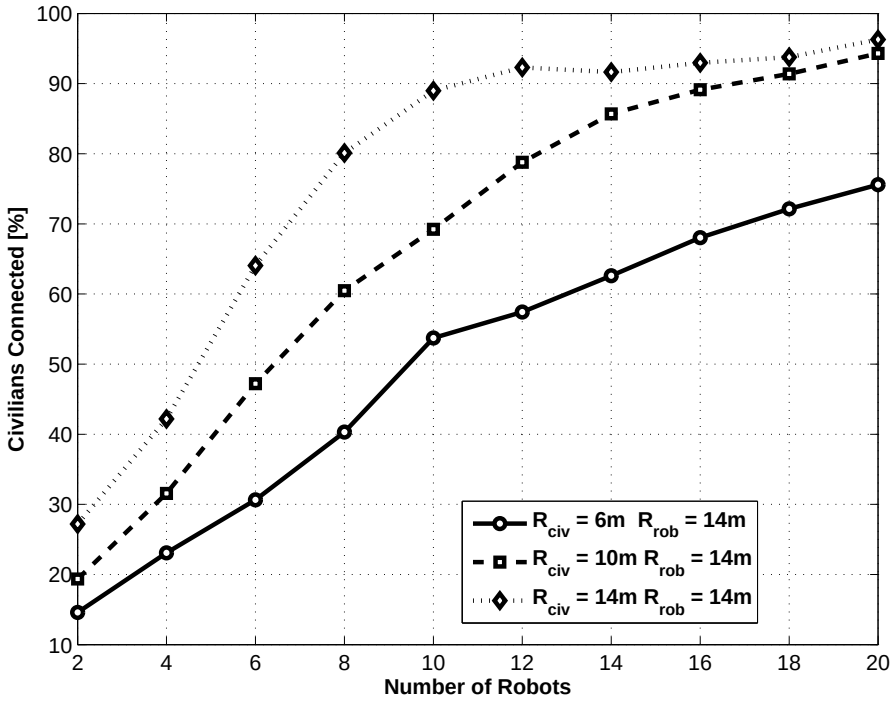


Figure 6.7.: Average percentage of connected civilians for varying number of robots, for uncertain civilian locations

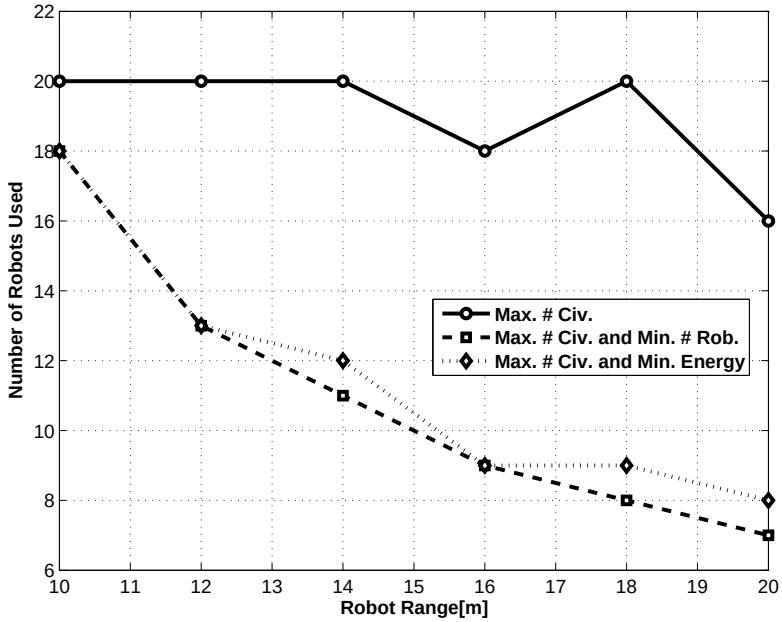


Figure 6.8.: Minimum number of robots required to connect all civilians when objectives (6.2a), (6.5) and (6.6) are utilised.

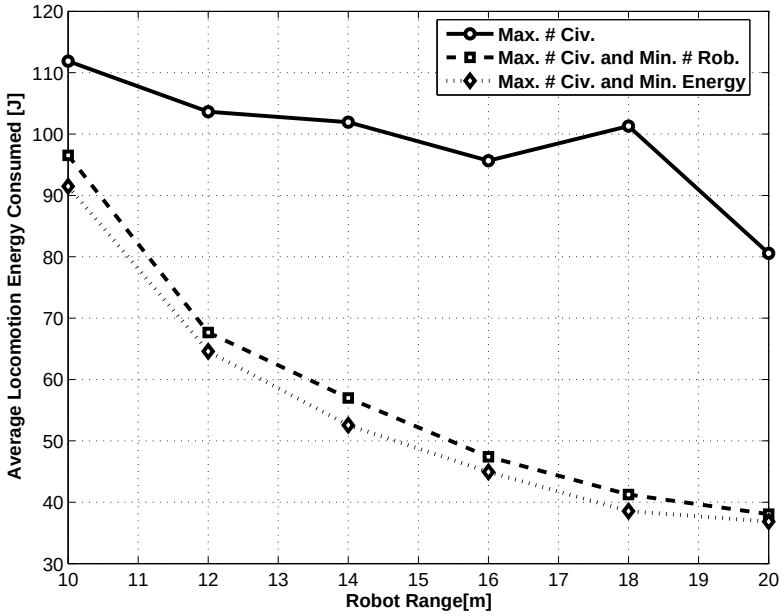


Figure 6.9.: Average locomotion energy consumed per robot when objectives (6.2a), (6.5) and (6.6) are utilised.

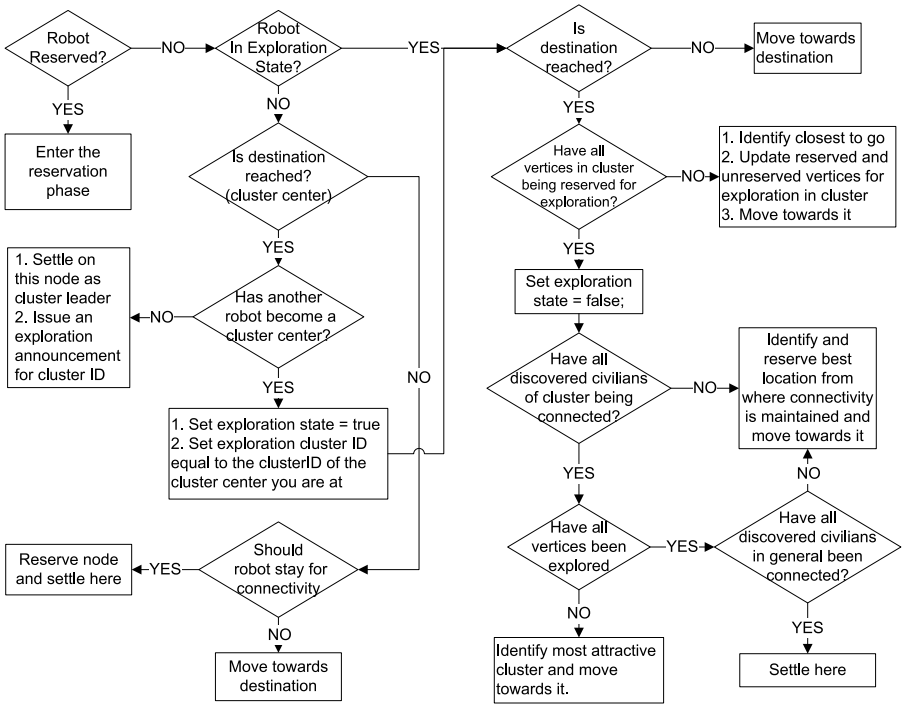


Figure 6.10.: General flow diagram of the distributed heuristic algorithm

all available robots in the network, which in turn explore the cluster for civilians and connect the ones that they find. Between clusters, chains of robots are formed to ensure connectivity. Within a cluster, a robots chooses to move to the location from which a maximum number of discovered and unconnected civilians will become connected.

Essentially, our heuristic approach is composed of two stages:

- Move to the most attractive cluster of civilians forming a chain of robots to maintain connectivity between clusters
- Discover and connect the civilians of this cluster and move to the next one

Each of the robots greedily selects the cluster to which it is attracted the most. Several attractiveness metrics can be used such as the number of civilians in each cluster, the distance between the robot and each cluster, or a combination of the two. A good trade-off is to use the ratio of these two metrics so as to maximise the number of connected civilians and minimise the number of robots that settle to maintain connectivity between clusters.

In order to avoid having multiple robots at the same location, each one reserves the location where it intends to settle to act as a cluster leader, to connect civilians, or to maintain multi-hop connectivity between cluster leaders. A robot does not reserve a location from where it would lose connectivity, and this ensures that the final robotic network will be connected.

6.5.1. Simulation results for the distributed algorithm

In order to evaluate our distributed algorithm, we implemented it as the movement decision model of robot agents in the Building Evacuation Simulator [92]. Figure 6.11 shows the final allocation of the robots for the given civilian clustering, where the larger circles represent the clusters and the smaller circles are the ranges of the civilians. The robots moved first to cluster 1, which was closer to their starting point and had the most civilians, connected all its civilians, and the remaining robots moved to clusters 2, 3 and 4 according to the attractiveness metric. The distributed heuristic reached comparable results with the centralised solution (Fig. 6.12 and 6.13), but with an average increase in the locomotion energy consumption of approximately 40%. This was expected since with the centralised approach,

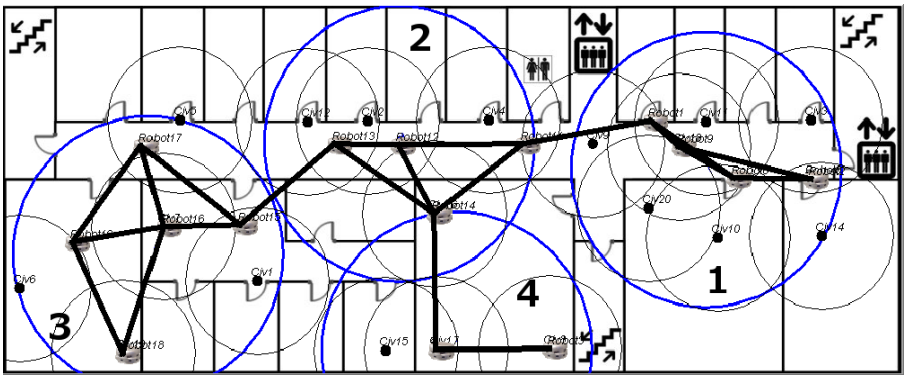


Figure 6.11.: Solution of the distributed algorithm for the given clustering of civilians

the robots know their destination before they start moving and go there directly, while in the distributed case, the robots need to move inside the area and gradually identify their final positions.

6.5.2. Introducing uncertainty

If the robots do not have complete knowledge of the locations of the civilians they must move so that they first cover areas of high probability of existence of civilians with low risk. For this reason, we employ a risk measure for the number of civilians on each location, the *Expected Shortfall* $ES_q(i)$, borrowed from the field of financial risk management. $ES_q(i)$ shows the expected number of civilians on location i in the worst $q\%$ cases [6]: $ES_q(i) = \mathbf{E}(Z_i | Z_i < m)$ where m is determined by $Prob(Z_i < m) = q$ and q is the given threshold. Note that for $q = 100\%$, the expected shortfall is equal to the expected number of civilians on the particular vertex. In practice, by using the ES_q measure, the operation centre determines the risk with which the autonomous robots will perform their exploration and connection tasks.

The introduction of uncertainty naturally leads to the need for dynamic exploration. At the beginning, the robots set a predetermined threshold l and take into account only the locations where $ES_q > l$. They cluster these locations according to the k-means clustering algorithm [177], where the value of k is the smallest feasible value for which all clusters have radius smaller than $R_{rob} + R_{civ}$. When the cluster leader issues the exploration announcement, the available robots move to unexplored areas within the range of the cluster leader and identify locations of civilians until all have been explored. Each time a robot moves to the nearest

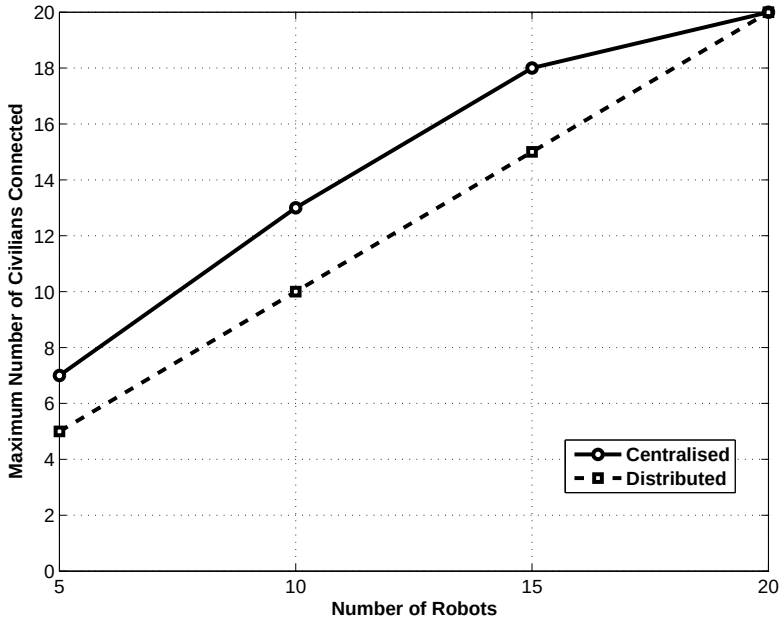


Figure 6.12.: Comparison between the distributed and centralised approach in terms of number of connected civilians against the number of robots

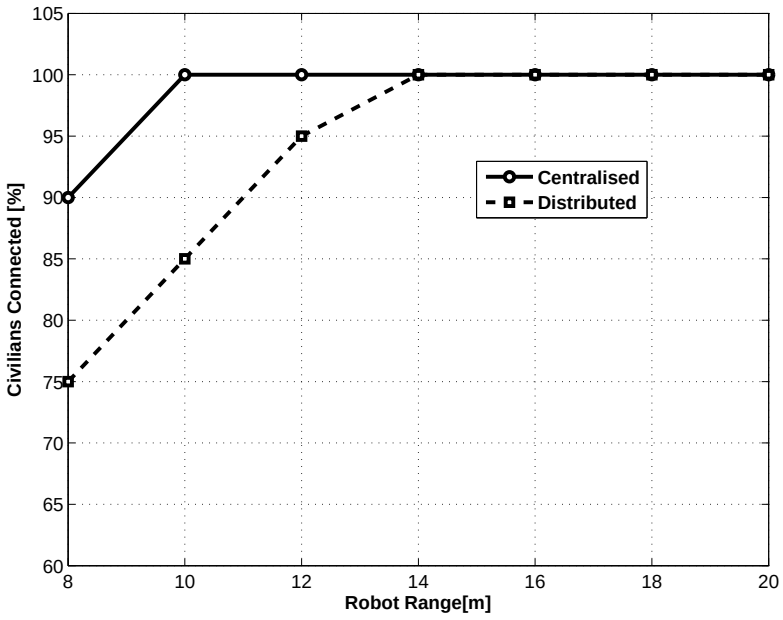


Figure 6.13.: Comparison between the distributed and centralised approach in terms of number of connected civilians against the wireless range of the robots

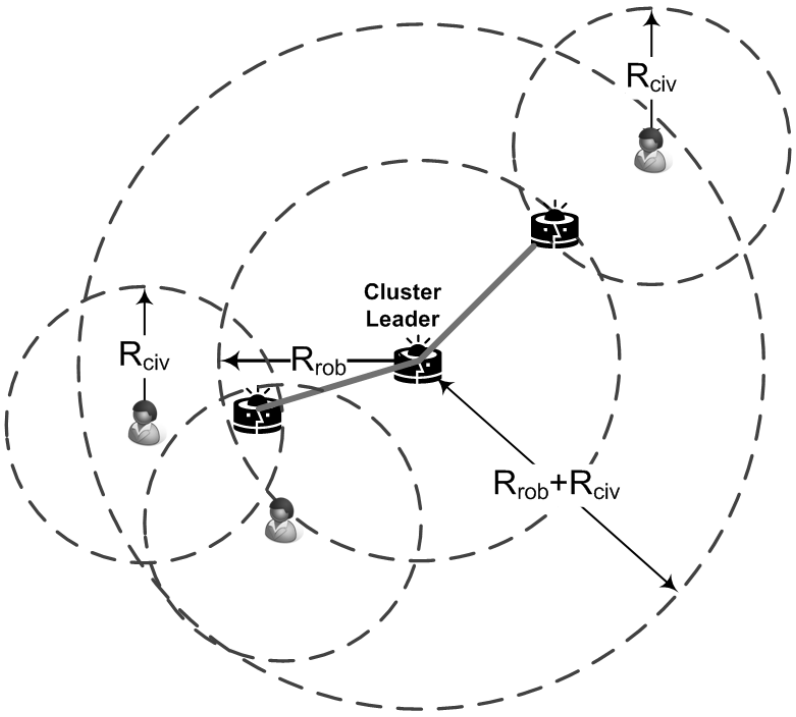


Figure 6.14.: Connectivity is guaranteed within a cluster if its radius is smaller than $R_{rob} + R_{civ}$

candidate location for exploration. A necessary number of robots stay to connect these civilians and the rest continue to the next cluster centre. When both the exploration of the cluster and the connection of its civilians are completed, the cluster leader dynamically computes a new set of clusters by reducing the ES_q limit. It then informs the other robots so that they select a new cluster. Thus, as soon as high-ES_q locations are completed, less probable locations start to be considered by the robots. In other words, the robots tend to move from locations of high probability of finding civilians towards less probable ones, until they connect them all.

We have evaluated our heuristic for 10, 15 and 20 robots and for varying risk parameter q for truncated exponential distribution of civilians with mean 0.25 civilians per vertex (Fig. 6.15).

Moderate-risk strategies ($q = 80 - 90\%$) yielded better results than the high-risk ($q = 100\%$) and low-risk strategies ($q = 70\%$). The latter on several occasions missed distant high-reward clusters, while the former allocated robots to locations where their expectations were not met.

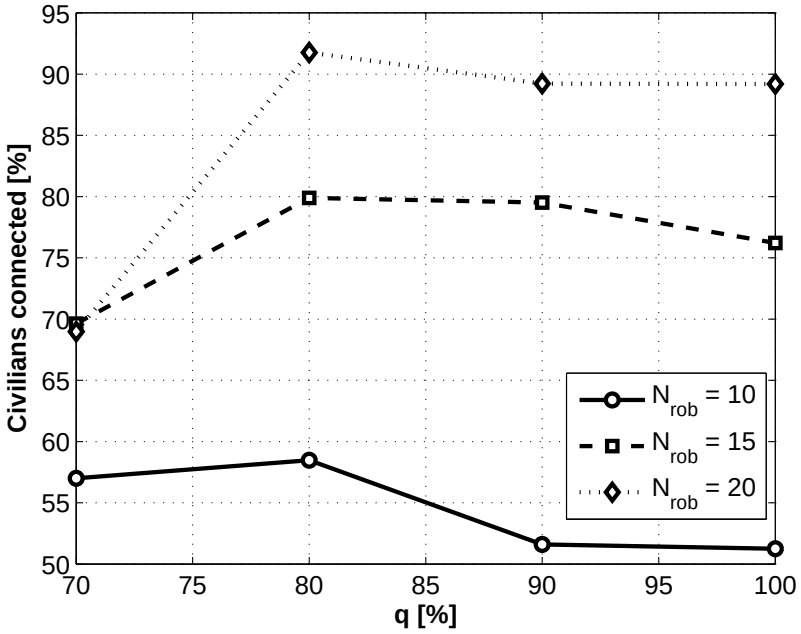


Figure 6.15.: Evaluation of the distributed heuristic for different numbers of robots N_{rob} and varying risk parameter q

6.5.3. MST-based modification

The above algorithm has two significant weaknesses: (i) it does not allow the robots to move towards different clusters at the same time if they start from the same location because they all adhere to the same rules and (ii) it selects the next best cluster without planning its next steps. To address these issues we look at the clusters of the civilian locations as the vertices of an overlay graph. The robots again need to move from cluster to cluster to connect the civilians, but this time they do so probabilistically. Whenever they are to choose a new cluster, they consult the minimum spanning tree (MST) of the overlay cluster graph as seen in Fig. 6.16. The MST is computed by one of the robots at the beginning and is communicated to the rest of the robots. Whenever a cluster is completely explored and all its civilians connected to the network, the remaining robots need to move to a new cluster. If more than one options exist, the probability of choosing each new direction is proportional to the estimated ratio of robots needed at the specific direction. The estimation is based simply on the expected number of civilians and the distances between clusters. By choosing probabilistically, the robots do not have to negotiate

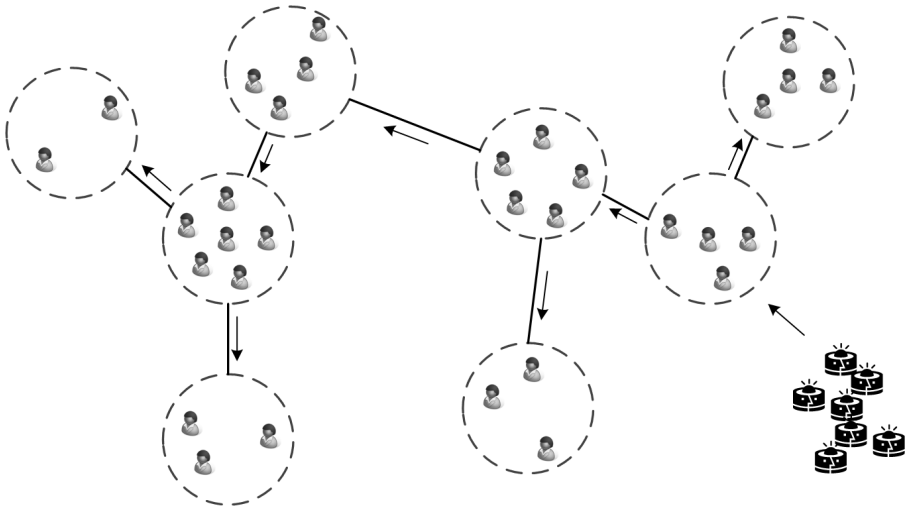


Figure 6.16.: Illustration of the Minimum Spanning Tree formed by the clusters and the movement options of the robots

with each other, they spread more quickly in the area and essentially they decide by taking into account their future options. This is a significant improvement over the algorithm shown in Section 6.5.2, where the robots were looking for the most attractive cluster ignoring their future options. However, unlike the previous algorithm, the MST-based approach dictates the use of static clustering, since the MST will not provide dependable paths if the overlay cluster graph keeps changing.

We investigate this tradeoff with experiments for 10, 15 and 20 robots, the results of which are shown in Fig. 6.17. For 10 robots, the first approach presented in section 6.5.2 performs better than the MST-based one. The reason is that the number of robots is not sufficient to connect many civilians and the planning of the MST-based algorithm does not bring the desired outcome. On the contrary, for 15 and 20 robots the MST-based approach outperforms the first one since there is sufficient number of robots that can deploy effectively using the MST. We can argue that the first approach is more efficient when the number of robots as well as the connectivity achieved is small, while the MST-based modification is better when the robots suffice to achieve high connectivity.

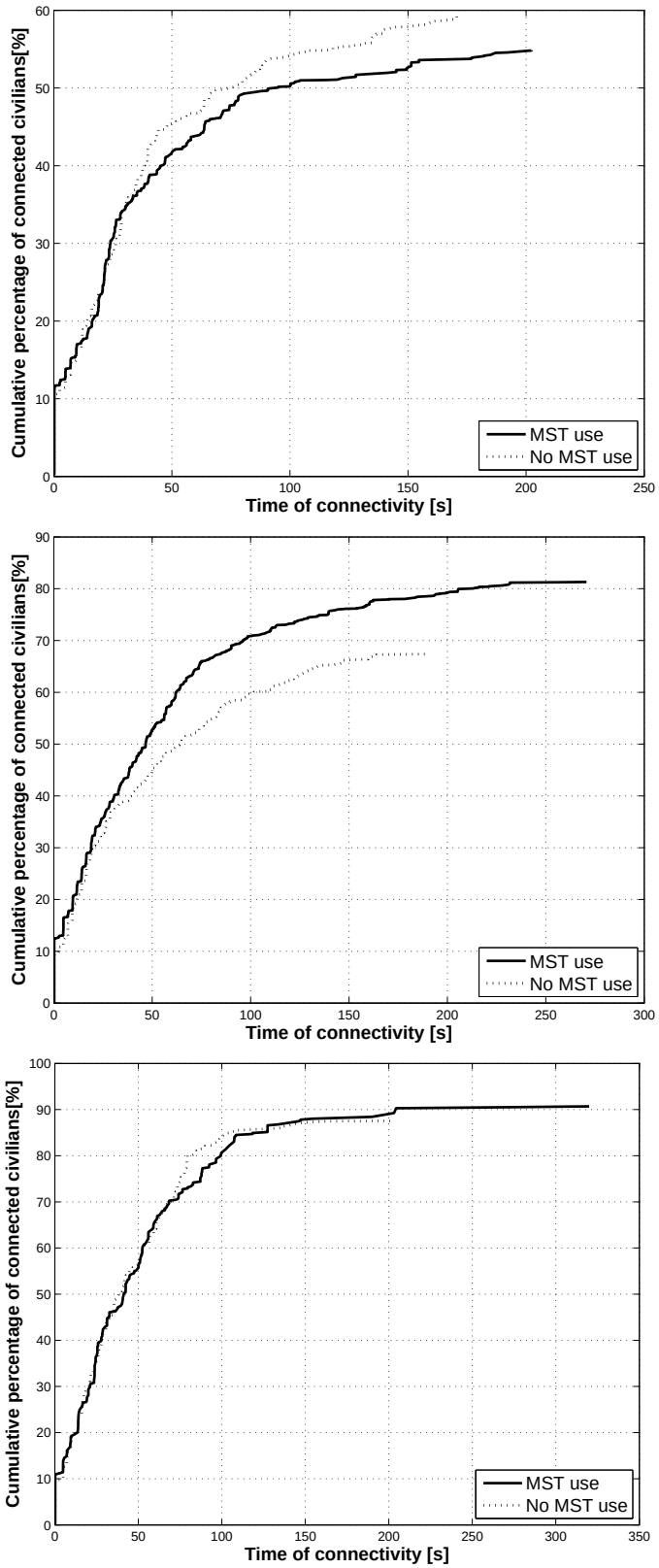


Figure 6.17.: Percentage of trapped civilians connected against time for 10, 15 and 20 robots.

6.6. Conclusions

In this chapter we have examined the problem of maximising the number of connected injured civilians to an operations centre via a wireless ad-hoc robotic network when the locations of the civilians are known or uncertain. We have introduced a centralised formulation which is based on network flows to find the optimal positions of the robots which is useful in both certain and uncertain civilian locations. We have also examined the effect of composite terms and we have seen that we can significantly reduce the total number of robots used or the energy consumed without affecting our primary goal which is the maximisation of the civilian connectivity to the network. We have also introduced distributed algorithms to deal with the problem under examination. We have shown that for the case of known civilian locations the distributed algorithm, which is based on clustering the locations of civilians, achieves results close to optimality. For the case of uncertain civilian locations we have introduced two algorithms. The one is based on dynamically clustering the possible civilian locations, after one cluster is explored and the discovered civilians are connected. The other is based on building the minimum spanning tree of the clusters centres so that the robots can explore and connect more than one clusters at the same time. Numerical evaluation showed that the first approach performs better when the number of robots is small and only a small percentage of civilians are finally connected, while the second is better when the number of robots is sufficiently large to connect almost all civilians.

7. Conclusions and future work

In this thesis we have aimed at the introduction of new learning algorithms for RNN, as well as the development of RNN-based and other algorithms for the solution of combinatorial optimisation problems arising in emergency management. With respect to supervised learning, we have introduced two new approaches. The first is a gradient descent algorithm for RNNSI, an extension model that we have introduced inspired by observations in vitro. The second is based on approximating the RNN equations into a nonnegative least squares problem, when all neurons have desired values, which is solved using the PGNNLS algorithm. We also develop the RNN-NNLS algorithm to deal with networks composed of both output and non-output neurons.

The developed supervised learning algorithms have been applied to the solution of emergency management problems that require fast, distributed and close to optimal solution, despite their large complexity (NP-hard problems) and uncertainty of information. Specifically, three assignment problems with the above characteristics are considered that are associated with: (1) the dispatching of emergency units to locations of injured civilians, (2) the allocation of assets to tasks under execution uncertainty, and (3) the deployment of a team of networked robots for establishing communication with trapped civilians in a disaster area. Apart from supervised learning algorithms, for the solution of these problems we considered fast approaches based on RNN parameter association, network flows and greedy heuristics. In the remainder of this chapter, we summarise the thesis contributions and discuss open issues and directions for future work.

7.1. Thesis contributions

The contributions of this thesis can be divided into two main categories:

a. Theoretical developments for the random neural network model

i. We have introduced an extension of the RNN model that incorporates synchronous interactions (RNNSI), which are based on two neurons that jointly excite a third one, inspired by observations of synchronised firing in cortical and retinal neuronal circuits. In addition, a gradient descent learning algorithm for the RNNSI model has been developed, which retains the computational complexity of the respective RNN algorithm for updating one weight ($O(N^3)$), despite the fact that RNNSI is a generalisation of the RNN model. Analytical manipulation of the related learning expressions, showed that all the weights of the network can be updated in $O(N^3)$, reducing the complexity of updating all weights in one iteration by an order of magnitude. This result extends to the RNN learning algorithm as well. Details can be found in Chapter 3.

ii. We have proposed RNN-NNLS, an RNN method that aids supervised learning both in terms of learning and weight initialisation. The method relies on approximating the signal-flow equations of the network to obtain a linear system of equations with nonnegativity constraints, which can be solved as a nonnegative least squares problem (NNLS) when all neurons in the network have desired values. For the solution of NNLS, a quasi-Newton projected gradient algorithm with limited-memory requirements has been developed. In this algorithm we have introduced two novel characteristics: (a) a hyper-exponential line-search procedure that reduces the number of matrix vector products required for the completion of one iteration, and (b) we derived efficient analytical expressions for the evaluation of the NNLS objective and gradient functions, resulting in an order of magnitude speedup. Apart from the computational speedup, the developed NNLS algorithm is suitable for large-scale problems, as it only requires the storage of a few vectors of the same size as the decision vector. We have also addressed successfully the case that the neural network is composed of both output and non-output neurons. More details can be found in Chapter 3.

iii. We have conducted an extended survey of the RNN which covers the main theory, as well as other extension models, associated learning algorithms and applications. This is the first extended survey on RNN since its introduction two decades ago. More details can be found in Chapter 2.

It is important to highlight that the applicability of the developed supervised learning RNN algorithms is not limited to the problems studied in this thesis; they can be used for the solution of other appropriate supervised learning problems, augmenting the pool of RNN learning algorithms. Furthermore, the RNNSI learning algorithms could be particularly useful in applications that combine learning with modelling. The reason is that the RNNSI model incorporates second order interactions, which could be useful in modelling complex systems such as gene regulatory networks and synchronously firing neuronal brain circuits. Finally, the RNN survey is a rich source of information about the RNN theory and different RNN tools that can be utilised for the solution of practical problems, contributing to the dissemination of scientific knowledge.

b. Investigation of optimisation problems in Disaster Management

During a large-scale disaster the allocation of the right emergency services to the right place in order to minimise the impact of the disaster is an extremely challenging operation. The evolving situation is complex, the collected information is incomplete, the communications can be disrupted and at the same time instant decisions are required. Therefore, it is imperative that decision-makers utilise mathematical algorithms to help them take appropriate actions. Any developed algorithms should be able to provide close to optimal solutions to the underlying problems in real-time, deal efficiently with information uncertainty, and operate in a distributed manner when communication with a central coordinator is not possible. In the context of my research, three problems that may arise in disaster response have been identified and solved. Next, the investigated problems are briefly described and the associated research contributions are outlined:

i. *Dispatching of emergency units to incidents*: In this problem, we investigate how to collect a number of spatially distributed injured civilians in the least possible total response time, when there is a limited number of emergency units with different capabilities. For the solution of this problem, we have proposed the use of supervised learning when distributed and fast decision making is required. The idea is to provide each of the emergency units, which are the decision makers, with a trained random neural network tool, an "oracle" that advises each agent on what actions to take. The "oracle" is trained prior to the actual event using as input the parameters of numerous

problem instances in the same physical context as the disaster and, as output, the optimal solutions of those instances. When the real-world problem occurs, each agent consults its "oracle" on the actions to be taken. If all the agents share the same information then the decision taken will be non-conflicting and their actions will be coordinated as if all agents had cooperated to find a consensus solution. We have employed RNN-NNLS and the RNNSI gradient descent learning algorithms for its solution. Extensive performance evaluation showed that by using these algorithms we can collect almost all the injured with very small deviation from optimality, while for a large number of problem instances we can collect all the injured. More details about this problem can be found in Chapter 4.

ii. *Asset-task assignment with uncertain execution*: We investigate a general problem associated with the assignment of assets to tasks when each asset can potentially execute any of the tasks, but assets execute tasks with a probabilistic outcome of success, so as to minimise the cost of the tasks. Assets can be emergency personnel that need to be dispatched to treat injured civilians, when there is possibility of failure either because of difficulty in accessing the location of the injured from the route followed, or due to insufficient skills. For this problem, we have developed an RNN parameter association approach, where the parameters of the optimisation problem are associated with parameters of the RNN model. In addition, we have introduced network flow algorithms that are based on solving a sequence of minimum cost flow problems on appropriately constructed networks with estimated arc costs and introduced three different estimation schemes. Performance evaluation indicates that both algorithms are useful, as each performs better than the other for different data sets, while both algorithms can solve large-scale problems in real-time and within 5% from optimality. We have also designed an approach for obtaining tight lower bounds to the optimal solution based on a piecewise linear approximation of the considered problem. More details about this problem can be found in Chapter 5.

iii. *Connecting trapped civilians to a wireless ad hoc robotic network*: We investigate the use of autonomous robots that move inside a disaster area and establish a network for two-way communication between trapped civilians with *a priori*

known or uncertain locations and an operations centre. The specific problem considered is to find optimal locations for the robots in order to maximise the number of civilians connected to the network, given that each civilian carries a short-range communication device. This problem is important because it provides the means for locating and assessing the condition of the injured. For its optimal solution, we have derived and solved a mixed-integer linear programming formulation based on network flows. We have also designed and implemented distributed heuristic algorithms based on clustering possible locations of civilians for both *a priori* known and uncertain civilian locations. Simulation results indicate that the performance of the distributed algorithms is close to the optimal one. More details about this problem can be found in Chapter 6.

From a practical applications perspective, the above investigated problems could help the emergency services improve their effectiveness in dealing with large scale disasters.

7.2. Future work

There are several interesting directions for future research based on the work undertaken in this thesis. Next, we provide suggestions for further work and discuss open issues for Chapters 3-6 in which the research contributions of this thesis have been elaborated.

We begin our suggestions from the developed supervised learning algorithms discussed in Chapter 3. One direction of future research involves the development of new learning algorithms for RNN or its extension models. Learning algorithms related to RNN are mostly gradient descent based although it is well known that these methods are not the most efficient. It would be interesting to investigate rigorous nonlinear programming techniques such as interior point or Newton trust-region methods that guarantee convergence to a local optimum and examine their computational and memory efficiency, as well as their ability in finding good local optima. Another interesting direction is to apply the idea of the RNN-NNLS algorithm for the RNNSI model. This problem is even more challenging as it is a nonconvex problem and further considerations are needed for the development of an efficient algorithm. An open issue related to the RNN-NNLS algorithm is

the selection of the desired excitation probabilities of the non-output neurons. To address this issue, either analytical techniques could be employed to derive the optimal distribution of the excitation probabilities of the non-output neurons, or a metaheuristic technique could be exploited to efficiently explore the search-space of these values. The developed algorithms could also be exploited in other applications. For example, RNN-NNLS is ideal for image processing applications where each pixel is represented by a neuron and has a desired output value [69]. Moreover, the ability of the RNNSI model to represent in a close manner the interactions in complex systems could be combined with its learning capabilities to investigate open questions arising in biological networks. One such question is related to the understanding of the operation of gene regulatory networks. In fact, the RNNSI model is already under investigation for the specific research goal [112].

Concerning the assignment of emergency units to incidents, further research could be undertaken to consider generalisations of it or relax the problem assumptions. In the problem, we have made the assumption that one unit can be allocated to exactly one incident. Nevertheless, if the capacity of the emergency units is less than the number of injured, then each unit should perform multiple trips delivering the injured each time to collection points. Another assumption that has been made is that emergency units allocated to one incident cannot be re-assigned to some other incident. However, it would be possible to assign one emergency unit to more than one incidents if excess capacity is available or dynamically re-assign it to incidents that have more injured than expected. In addition, each injured civilian could have an expected remaining lifetime so that an emergency unit would have to be able to arrive in time in order to save the particular person. These conditions impose significant challenges both in terms of modelling and solving the problem.

The assets to tasks assignment with execution uncertainty problem and the developed algorithms could also be extended in several directions. One such direction is associated with the allocation of the assets in stages. First some assets can be allocated to tasks, and then the resulting outcome, in terms of the set of tasks which are actually executed, is observed. As a consequence of the outcome the remaining assets can then be used in the next stage, based on the work that was not completed in the first stage. This would correspond to an approach in which assets are maintained in reserve and re-allocated during the course of the process that one is studying. Moreover, because the particular problem is quite general, it

would be interesting to address other real-world problems in diverse fields that can be modelled with the specific formulation. As briefly discussed in Chapter 5, potential applications include, for example, communication over an unreliable medium, cancer therapy treatment and sensor coverage. Another interesting extension of this work would be to examine complete decentralisation of the decision process, so that each asset decides on what task to execute, based only on local information and communication with its neighbours. It is the author's belief that this could be accomplished by the considered network flow algorithms, as each network flow subproblem is a linear program that can be solved in a completely decentralised manner [51].

Finally, regarding the problem of connecting trapped civilians to a wireless ad-hoc robotic network, there are a number of research challenges that need to be addressed. Firstly, the employment of clusters for civilian exploration and connectivity leads to interesting optimisation problems, such as the optimal exploration choices within a cluster in order to minimise the exploration time or the energy expenditure. Secondly, robust approaches that take into account any robot or communication failures should be developed to ensure the uninterrupted connectivity of the robotic network. Thirdly, in many situations, especially in disaster management, the assumption that the robots know the building graph may not be valid so that robots will have to rely on pure exploration to navigate through the building and connect civilians. Fourthly, the centralised approach should be exploited for the development of efficient heuristic algorithms similar to the network flow algorithms developed in the asset-task assignment problem.

Bibliography

[1] H. Abdelbaki, E. Gelenbe, and S.E. El-Khamy. Analog hardware implementation of the random neural network model. In *Proceedings of the IEEE/INNS/ENNS International Joint Conference on Neural Networks (IJCNN 2000), Como, Italy, 24-27 July*, pages 197–201. IEEE Computer Society, Los Alamitos, California, 2000.

[2] H. Abdelbaki, E. Gelenbe, and T. Kocak. Matched neural filters for EMI based mine detection. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 99), Washington, DC, 10-16 July*, pages 3236–3240. IEEE, Piscataway, NJ, 1999.

[3] H. Abdelbaki, E. Gelenbe, and T. Kocak. Neural Algorithms and Energy Measures for EMI Based Mine Detection. *Journal of Differential Equations and Dynamical Systems*, 13(1-2):63–86, 2005.

[4] A. M. Abdelbar. Designing high order recurrent networks for bayesian belief revision for bayesian belief revision. In L.R. Medsker and L.C. Jain, editors, *Recurrent neural networks: design and applications*, pages 77–98. CRC Press, 1999.

[5] Ashraf M. Abdelbar, Emad A. M. Andrews, and Donald C. Wunsch. Abductive reasoning with recurrent neural networks. *Neural Networks*, 16(5-6):665 – 673, 2003.

[6] Carlo Acerbi and Dirk Tasche. Expected shortfall: a natural coherent alternative to value at risk. *Economic Notes*, 31(2):379–388, 2002.

[7] J. Aguilar and A. Colmenares. Resolution of Pattern Recognition Problems using a Hybrid Genetic/Random Neural Network Learning Algorithm. *Pattern Analysis and Applications*, 1(1):52–61, 1998.

[8] J. Aguilar and E. Gelenbe. Task assignment and transaction clustering heuristics for distributed systems. *Information Sciences – Informatics and Computer Science*, 97(1 & 2):199–221, 1997.

[9] Jose Aguilar. Definition of an energy function for the random neural to solve optimization problems. *Neural Networks*, 11(4):731–737, 1998.

[10] Jose Aguilar. Learning Algorithm and Retrieval Process for the Multiple Classes Random Neural Network Model. *Neural Processing Letters*, 13(1):81–91, 2001.

[11] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, United States, 1993.

[12] Ravindra K. Ahuja, Arvind Kumar, Krishna C. Jha, and James B. Orlin. Exact and Heuristic Algorithms for the Weapon-Target Assignment Problem. *OPERATIONS RESEARCH*, 55(6):1136–1146, 2007.

[13] G. Aiello, S. Gaglio, G. Lo Re, and A. Urso. A Random Neural Network for the Dynamic Multicast Problem. *WSEAS Transactions on Computers*, 3(5):1545–1550, 2004.

[14] Bilge E. S. Akgul, Lakshmi N. Chakrapani, Pinar Korkmaz, and Krishna V. Palem. Probabilistic CMOS Technology: A Survey and Future Directions. In *Proceedings of the 14th IFIP International Conference on Very Large Scale Integration of System-on-Chip, Nice, France, 16-18 October*, pages 1–6. IEEE, Piscataway, NJ, 2006.

[15] J. R. Artalejo. G-networks: A versatile approach for work removal in queueing networks. *European Journal of Operational Research*, 126(2):233–249, 2000.

[16] V. Atalay. Learning by Optimization in Random Neural Networks. In *Proceedings of the 13th International Symposium on Computer and Information Sciences, Antalya, Turkey, 26-28 October*, pages 143–148. IOS Press, Amsterdam, 1998.

[17] V. Atalay and E. Gelenbe. Parallel algorithm for colour texture generation using the random neural network model. *International Journal of Pattern Recognition and Artificial Intelligence*, 6(2 & 3):437–446, 1992.

[18] V. Atalay, E. Gelenbe, and N. Yalabik. The random neural network model for texture generation. *International Journal of Pattern Recognition and Artificial Intelligence*, 6(1):131–141, 1992.

[19] Alex Aussem. Call admission control in ATM networks with the random neural network. In *Proceedings of the IEEE International Conference on Neural Networks, Orlando, FL, USA, 28 June - 2 July*, pages 2482–2487. IEEE, Piscataway, NJ, 1994.

[20] H. Bakircioglu and E. Gelenbe. Random Neural Network recognition of shaped objects in strong clutter. In *Proceedings of the 3rd Conference on Applications of Artificial Neural Networks in Image Processing, San Jose, CA, USA,26-27 January*, pages 22–28. SPIE, Bellingham, WA, 1998.

[21] H. Bakircioglu, E. Gelenbe, and T. Koçak. Image Enhacement and Fusion with the Random Neural Network. *ELEKTRIK, Turkish Journal of Electrical Engineering and Computer Sciences*, 5(1):65–77, 1997.

[22] H. Bakircioglu and T. Kocak. Survey of random neural network applications. *European Journal of Operational Research*, 126(2):319–330, 2000.

[23] J. L. Barlow and G. Toraldo. The effect of diagonal scaling on projected gradient methods for bound constrained quadratic programming problems. *Optimization Methods and Software*, 5(3):235–245, 1995.

[24] D. P. Bertsekas. Projected Newton Methods for Optimization Problems with Simple Constraints. *SIAM Journal on Control and Optimization*, 20(2):221–246, 1982.

[25] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[26] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific Series in Optimization and Neural Computation, 1997.

[27] Friedrich Biegler-Konig and Frank Barmann. A learning algorithm for multilayered neural networks based on linear least squares problems. *Neural Networks*, 6(1):127–131, 1993.

[28] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[29] R. Bro and S. D. Long. A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics*, 11(5):393–401, 1997.

[30] Rainer E. Burkard and Eranda Cela. Linear assignment problems and extensions. In P.M. Pardalos and D.-Z. Du, editors, *Handbook of Combinatorial Optimization - Supplement Volume A*, pages 75–149. Kluwer Academic Publishers, 1999.

[31] R. H. Byrd, P. Lu, and J. Nocedal. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific and Statistical Computing*, 16(5):1190–1208, 1995.

[32] Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

[33] Paul Calamai and Jorge Moré. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39:93–116, 1987.

[34] Hector Cancela, Franco Robledo, and Gerardo Rubino. A GRASP algorithm with RNN based local search for designing a WAN access network. *Electronic Notes in Discrete Mathematics*, 18(1):59 – 65, 2004.

[35] Enrique Castillo, Oscar Fontenla-Romero, Bertha Guijarro-Berdinas, and Amparo Alonso-Betanzos. A global optimum approach for one-layer neural networks. *Neural Computation*, 14(6):1429–1449, 2002.

[36] Enrique Castillo, Bertha Guijarro-Berdinas, Oscar Fontenla-Romero, and Amparo Alonso-Betanzos. A very fast learning method for neural networks based on sensitivity analysis. *Journal of Machine Learning Research*, 7:1159–1182, 2006.

[37] C. Cerkez, I. Aybay, and U. Halici. A digital neuron realization for the random neural network model. In *Proceedings of the International Conference on Neural Networks, Houston, TX, USA, 09- 12 June*, pages 1000–1004. IEEE, Piscataway, NJ, 1997.

[38] Lakshmi N. Chakrapani, Pinar Korkmaz, Bilge E. S. Akgul, and Krishna V. Palem. Probabilistic system-on-a-chip architectures. *ACM Transactions on Design Automation of Electronic Systems*, 12(3):1–28, 2007.

[39] Shi-Chung Chang, Ronald M. James, and Jonh J. Shaw. Assignment algorithm for kinetic energy weapons in boost phase defence. In *Proceedings of the 26th IEEE Conference on Decision and Control, Los Angeles, CA, 9-11 December*, pages 1678–1683. IEEE, New York, NY, 1987.

[40] B. Cooper. Stability analysis of higher-order neural networks for combinatorial optimization. *International Journal of Neural Systems*, 12(3-4):177–186, 2002.

[41] C. Cramer, , E. Gelenbe, and P. Gelenbe. Image and video compression. *IEEE Potentials*, 17(1):29–33, Feb/Mar 1998.

[42] C. Cramer and E. Gelenbe. Video quality and traffic QoS in learning-based subsampled and receiver-interpolated video sequences. *IEEE Journal on Selected Areas in Communications*, 18(2):150–167, 2000.

[43] C. Cramer, E. Gelenbe, and H. Bakircioglu. Low bit rate video compression with neural networks and temporal subsampling. *Proceedings of the IEEE*, 84(10):1529–1543, 1996.

[44] Ana Paula Couto da Silva, Martin Varela, Edmundo de Souza e Silva, Rosa M. M. Leao, and Gerardo Rubino. Quality assessment of interactive voice applications. *Computer Networks*, 52(6):1179–1192, 2008.

[45] Achiya Dax. On computational aspects of bounded linear least squares problems. *ACM Transactions on Mathematical Software*, 17(1):64–73, March 1991.

[46] G. G. den Broeder, R. E. Ellison, and L. Emerling. On Optimum Target Assignments. *OPERATIONS RESEARCH*, 7(3):322–326, 1959.

[47] Frauke Driewer, Herbert Baier, Klaus Schilling, Jiri Pavlicek, Libor Preucil, Niramon Ruangpayoongsak, Hubert Roth, Jari Saarinen, Jussi Suomela, Aarne Halme, and Miroslav Kulich. Hybrid telematic teams for search and rescue operations. In *IEEE International Workshop on Safety, Security, and Rescue Robotics, SSRR 2004*, 2004.

[48] A. Nunez E. Gelenbe, R. Lent. Self-aware networks and QoS. *Proceedings of the IEEE*, 92(9):1478 – 1489, 2004.

[49] D. Erdogmus, O. Fontenla-Romero, J.C. Principe, A. Alonso-Betanzos, and E. Castillo. Linear-least-squares initialization of multilayer perceptrons through backpropagation of the desired response. *IEEE Transactions on Neural Networks*, 16(2):325–337, 2005.

[50] Oner Esen, Eyup Cetin, and Seda Tolun Esen. A mathematical immunochemoradiotherapy model: A multiobjective approach. *Nonlinear Analysis: Real World Applications*, 9(2):511 – 517, 2008.

[51] B. Feinberg. Coercion functions and decentralized linear programming. *Mathematics of Operations Research*, 14(1):177–187, 1989.

[52] Oscar Fontenla-Romero, Deniz Erdogmus, J. C. Principe, Amparo Alonso-Betanzos, and Enrique Castillo. Linear least-squares based methods for neural networks learning. In *Proceedings of the International Conference on Artificial Neural Networks and Neural Information Processing, Istanbul, Turkey, June 26-29*, pages 84–91, Heidelberg, 2003. Springer-Verlag.

[53] J.-M. Fourneau and E. Gelenbe. Flow equivalence and stochastic equivalence in g-networks. *Computational Management Science*, 1(2):179 – 192, 2004.

[54] J. M. Fourneau, E. Gelenbe, and R. Suros. G-networks with multiple classes of positive and negative customers. *Theoretical Computer Science*, 155:141–156, 1996.

[55] J.N. Fourneau. Computing the Steady State Distribution of Networks with Positive and Negative Customers. In *Proceedings of the 13-th IMACS World Congress on Computational and Applied Mathematics, Dublin, Ireland, July.* North-Holland, Amsterdam, 1991.

[56] E. Gelenbe. Random Neural Networks with Negative and Positive Signals and Product Form Solution. *Neural Computation*, 1(4):502–510, 1989.

[57] E. Gelenbe. Stability of the random neural network. *Neural Computation*, 2(2):239–247, 1990.

[58] E. Gelenbe. A probabilistic generalization of the sat problem. *Comptes Rendus de l'Academie des Sciences Series I*, 315(3):339–342, 1992.

[59] E. Gelenbe. G-networks with triggered customer movement. *Journal of Applied Probability*, 30(3):742–748, 1993.

[60] E. Gelenbe. Learning in the recurrent random network. *Neural Computation*, 5:154–164, 1993.

[61] E. Gelenbe. Users and services in intelligent networks. *Proceedings IEE (ITS)*, 153(3):213–220, 2006.

[62] E. Gelenbe. Steady-state solution of probabilistic gene regulatory networks. *Physical Review E*, 76(1):031903, 2007.

[63] E. Gelenbe and C. Cramer. Oscillatory corticothalamic response to somatosensory input. *Biosystems*, 48(1–3):67–75, 1998.

[64] E. Gelenbe, C. Cramer, M. Sungur, and P. Gelenbe. Traffic and video quality in adaptive neural compression. *Multimedia Systems*, 4:357–369, 1996.

[65] E. Gelenbe, T. Feng, and K. R. R. Krishnan. Neural network methods for volumetric magnetic resonance imaging of the human brain. *Proceedings of the IEEE*, 84(1):1488–1496, 1996.

[66] E. Gelenbe and J. M. Fourneau. Random neural networks with multiple classes of signals. *Neural Computation*, 11(4):953–963, 1999.

[67] E. Gelenbe, A. Ghanwani, and V. Srinivasan. Improved neural heuristics for multicast routing. *IEEE Journal of Selected Areas of Communications*, 15(2):147–155, 1997.

[68] E. Gelenbe and K. Hussain. Learning in the multiple class random neural network. *IEEE Trans. on Neural Networks*, 13(6):1257–1267, 2002.

[69] E. Gelenbe, K. Hussain, and H. Abdelbaki. Random neural network texture model. In *Proceedings of the 5th Conference on Applications of Artificial Neural Networks in Image Processing, San Jose, CA, USA, 27-28 January*, pages 104–111. SPIE, Bellingham, WA, 2000.

[70] E. Gelenbe, K. Hussain, and V. Kaptan. Simulating autonomous agents in augmented reality. *Journal of Systems and Software*, 74(3):255–268, 2005.

[71] E. Gelenbe and T. Kocak. Wafer surface reconstruction from top-down scanning electron microscope images. *Microlectronic Engineering*, 75:216–233, 2004.

[72] E. Gelenbe, V. Koubi, and F. Pekergin. Dynamical random neural network approach to the traveling salesman problem. *ELEKTRIK*, 2(2):1–10, 1994.

[73] E. Gelenbe and R. Lent. Power-aware ad hoc cognitive packet networks. *Ad Hoc Networks Journal*, 2(3):205–216, 2004.

[74] E. Gelenbe, R. Lent, and Z. Xu. Design and performance of cognitive packet networks. *Performance Evaluation*, 46(2-3):155–176, 2001.

[75] E. Gelenbe, R. Lent, and Z. Xu. Measurement and performance of a cognitive packet network. *Computer Networks*, 37:691–791, 2001.

[76] E. Gelenbe, Z.-H. Mao, and Y.-D. Li. Function approximation with spiked random networks. *IEEE Transactions on Neural Networks*, 10(1):3–9, 1999.

[77] E. Gelenbe, Z.-H. Mao, and Y.-D. Li. Function approximation by random neural networks with a bounded number of layers. *J. Differential Equations and Dynamical Systems*, 12(1 & 2):143–170, 2004.

[78] E. Gelenbe, G. Sakellari, and M. D' Arienzo. Admission of QoS aware users in a smart network. *ACM Transactions on Autonomous and Adaptive Systems*, 3(1):4:1–4:28, 2008.

[79] E. Gelenbe, E. Seref, and Z. Xu. Simulation with learning agents. *Proceedings of the IEEE*, 89(2):148–157, 2001.

[80] E. Gelenbe and H. Shachnai. On G-networks and resource allocation in multimedia systems. *European Journal of Operational Research*, 126(2):308–318, 2000.

[81] E. Gelenbe, A. Stafylopatis, and A. Likas. Associative memory operations of the random neural network. In *Proceedings of the International Conference*

on Artificial Neural Networks, Espoo, Finland, 24-28 June, pages 307–312. North-Holland, Amsterdam, 1991.

[82] Erol Gelenbe. Theory of the Random Neural Network. In E. Gelenbe, editor, *NEURAL NETWORKS: Advances and Applications*, pages 1–20. Elsevier Science Publishers B.V., 1991.

[83] Erol Gelenbe. G-networks: a unifying model for neural and queueing networks. *Annals of Operations Research*, 48(5):433–461, 1994.

[84] Erol Gelenbe. A class of genetic algorithms with analytical solution. *Robotics and Autonomous Systems*, 22(1):59–64, 1997.

[85] Erol Gelenbe. The first decade of G-networks. *European Journal of Operational Research*, 126(2):231–232, 2000.

[86] Erol Gelenbe. Product-Form Queueing Networks with Negative and Positive Customers. *Journal of Applied Probability*, 28(3):656–663, Sep. 1991.

[87] Erol Gelenbe and F. Batty. Minimum Graph Covering with the Random Neural Network Model. In E. Gelenbe, editor, *NEURAL NETWORKS: Advances and Applications, 2*, pages 215–222. Elsevier Science Publishers B.V., 1992.

[88] Erol Gelenbe and Jean-Michel Fourneau. G-Networks with resets. *Performance Evaluation*, 49(1-4):179–191, 2002.

[89] Erol Gelenbe and Ali Labed. G-networks with multiple classes of signals and positive customers. *European Journal of Operational Research*, 108(2):293–305, 1998.

[90] B. Gendron, T.G. Crainic, and A. Frangioni. Multicommodity capacitated network design. In *Telecommunications Network Planning*, pages 1–19. Kluwer Academic Publisher, 1999.

[91] A. Ghanwani. A qualitative comparison of neural network models applied to the vertex covering problem. *ELEKTRIK, Turkish Journal of Electrical Engineering and Computer Sciences*, 2(1):11–18, 1994.

[92] D. Gianni, G. Loukas, and E. Gelenbe. A simulation framework for the investigation of adaptive behaviours in largely populated building evacuation

scenarios. In *Organised Adaptation in Multi-Agent Systems workshop at the 7th International Conference on Autonomous Agents and Multiagent Systems*, 2008.

[93] C. L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[94] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[95] M.W. Goudreau, C.L. Giles, S.T. Chakradhar, and D. Chen. First-order versus second-order single-layer recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(3):511–513, May 1994.

[96] Bertha Guijarro-Berdinas, Oscar Fontenla-Romero, Beatriz Perez-Sanchez, and Paula Fraguela. A fast semi-linear backpropagation learning algorithm. In *Proceedings of the International Conference on Artificial Neural Networks, Porto, Portugal, September 9-13*, pages 190–198, Heidelberg, 2007. Springer-Verlag.

[97] U. Halici. Reinforcement learning in random neural networks for cascaded decisions. *BIOSYSTEMS*, 40(1-2):83–91, 1997.

[98] U. Halici and E. Karaoz. A Linear Approximation for Training Recurrent Random Neural Networks. In *Proceedings of the 13th International Symposium on Computer and Information Sciences, Antalya, Turkey, 26-28 October*, pages 149–156. IOS Press, Amsterdam, 1998.

[99] Ugur Halici. Reinforcement learning with internal expectation for the random neural network. *European Journal of Operational Research*, 126(2):288–307, 2000.

[100] Ugur Halici. Reinforcement learning with internal expectation in the random neural networks for cascaded decisions. *Biosystems*, 63(1-3):21–34, 2001.

[101] L.A. Hey. Power Aware Smart Routing in Wireless Sensor Networks. In *Proceedings of the Conference on Next Generation Internet Networks, Krakow, Poland, 28-30 April*, pages 195–202. IEEE, Piscataway, NJ, 2008.

[102] Sepp Hochreiter and Jorgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[103] J.J. Hopfield and D.W. Tank. "Neural" Computation of Decisions in Optimization Problems. *Biological Cybernetics*, 52(3):141–152, 1985.

[104] P. A. Hosein. *A Class of Dynamic Nonlinear Resource Allocation Problems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA.,, 1989.

[105] C. Hubert. Learning internal representations with the N-M-N random neural network. *Comptes Rendus de l'Academie des Sciences Series II*, 317(4):451–456, 1993.

[106] C. Hubert. Supervised learning and recall of simple images with the random neural network. *Comptes Rendus de l'Academie des Sciences Series I*, 316(1):93–96, 1993.

[107] Khaled Hussain and Ghada S. Moussa. Laser intensity vehicle classification system based on random neural network. In *Proceedings of the 43rd Annual Southeast Regional Conference, Kennesaw, Georgia, Alabama, USA, 18-19 March*, pages 31–35. ACM, New York, 2005.

[108] Herbert Jaeger. Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. GMD report 159, German National Research Center for Information Technology, 2002.

[109] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[110] L.C. Jia, M. Sano, P.-Y. Lai, and C.K. Chan. Connectivities and synchronous firing in cortical neuronal networks. *Physical Review Letters*, 93(8):088101–1–088101–4, 2004.

[111] D. Kim, S. Sra, and I.S. Dhillon. A new projected quasi-newton approach for the nonnegative least squares problem. Technical report, Dept. of Computer Sciences, The University of Texas at Austin, Dec. 2006.

[112] Haseong Kim and Erol Gelenbe. Anomaly detection in gene expression via stochastic models of gene regulatory networks. *BMC Genomics*, 10(Suppl 3):S26, 2009.

[113] S. Kirkpatrick, Jr. Gelatt, C. D., and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[114] S. E. Kolitz. Analysis of a maximum marginal return assignment algorithm. In *Proceedings of the 27th IEEE Conference on Decision and Control, Austin, Texas, USA, 7-9 December*, pages 2431–2436. IEEE, New York, NY, 1988.

[115] P. Konig and A. K. Engel. Correlated firing in sensory-motor systems. *Current Opinion in Neurobiology*, 5(4):511–519, 1995.

[116] E. B. Kosmatopoulos, M. M. Polycarpou, M.A. Christodoulou, and P.A. Ioannou. High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks*, 6(2):422–431, Mar 1995.

[117] Vassilada Koubi. *Reseaux de neurones et optimisation combinatoire*. PhD thesis, Universit de Paris 05, Paris, France, 1994.

[118] Terence Kwok and Kate A. Smith. A unified framework for chaotic neural-network approaches to combinatorial optimization. *IEEE Transactions on Neural Networks*, 10(4):978–981, 1999.

[119] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Prentice Hall, 1987.

[120] Zne-Jung Lee, Chou-Yuan Lee, and Shun-Feng Su. An immunity-based ant colony optimization algorithm for solving weapon-target assignment problem. *Applied Soft Computing*, 2(1):39 – 47, 2002.

[121] Zne-Jung Lee, Shun-Feng Su, and Chou-Yuan Lee. Efficiently solving general weapon-target assignment problem by genetic algorithms with greedy eugenics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 33(1):113–121, Feb 2003.

[122] A. Likas and A. Stafylopatis. High capacity associative memory based on the random Neural Network model. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(8):919–937, 1996.

[123] Aristidis Likas and Andreas Stafylopatis. Training the random neural network using quasi-Newton methods. *European Journal of Operational Research*, 126(2):331–339, 2000.

[124] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779, 2007.

[125] S. P. Lloyd and H. S. Witsenhausen. Weapons allocation is NP-complete. In *Proceedings of the 1986 Summer Computer Simulation Conference, Reno, Nevada, USA, 28-30 July*, pages 1054–1058. Society for Computer Simulation, 1986.

[126] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657 – 690, 2007.

[127] Rong Lu and Yi Shen. Image Segmentation Based on Random Neural Network Model and Gabor Filters. In *Proceedings of the 27th Annual International Conference of the Engineering in Medicine and Biology Society, Shanghai, China, 1-4 September*, pages 6464–6467. IEEE, Piscataway, NJ, 2005.

[128] B. Luu, B. O' Brien, D. Baran, and R. Hardy. A Soldier-Robot Ad Hoc Network. pages 558–563, March 2006.

[129] V. F. Magirou and J. Z. Milis. An algorithm for the multiprocessor assignment problem. *Operations research letters*, 8(6):351–356, 1989.

[130] T. Magnanti and S. Raghavan. Strong formulations for network design problems with connectivity requirements. *Networks*, 45(2):61–79, 2005.

[131] Federico Malucelli. A polynomially solvable class of quadratic semi-assignment problems. *European Journal of Operational Research*, 91(3):619 – 622, 1996.

[132] Federico Malucelli and Daniele Pretolani. Lower bounds for the quadratic semi-assignment problem. *European Journal of Operational Research*, 83(2):365 – 375, 1995.

[133] P. Mann-Metzer and Y. Yarom. Electrotonic coupling interacts with intrinsic properties to generate synchronized activity in cerebellar networks of inhibitory interneurons. *J. Neuroscience*, 19(9):3298–3306, 1999.

[134] Alan Manne. A Target-Assignment Problem. *Operations Research*, 6(3):346–351, 1958.

[135] Jason R. Marden and Adam Wierman. Distributed welfare games with applications to sensor coverage. In *Proceedings of the 47th IEEE Conference on Decision and Control, Cancún, México.*

[136] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* Wiley, 1990.

[137] S. Mohamed and G. Rubino. A study of real-time packet video quality using random neural networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(12):1071–1083, 2002.

[138] Samir Mohamed, Gerardo Rubino, and Martin Varela. Performance evaluation of real-time speech through a packet network: a random neural networks-based approach. *Performance Evaluation*, 57(2):141–161, 2004.

[139] Jorge J. Moré and Gerardo Toraldo. On the Solution of Large Quadratic Programming Problems with Bound Constraints. *SIAM Journal on Optimization*, 1(1):93–113, 1991.

[140] K. Muramoto, K. Kobayashi, S. Nakanishi, Y. Matsuda, and Y. Kuroda. Functional Synapse Formation between Cultured Neurons of Rat Cerebral Cortex. In *Proc. Japan Academy 1988*, pages 319–322, 1988.

[141] K.G. Murty. *Linear and Combinatorial Programming.* Krieger Publishing Company, Malabar, FL, 1976.

[142] National Fire Protection Association (NFPA). *Standard on Disaster/Emergency Management and Business Continuity Programs.* NFPA, 2007.

[143] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer-Verlag, 1999.

[144] Gulay Oke and Georgios Loukas. A Denial of Service Detector based on Maximum Likelihood Detection and the Random Neural Network. *The Computer Journal*, 50(6):717–727, 2007.

[145] Anibal Ollero, Gunter Hommel, Jeremi Gancet, Luis-Gonzalo Gutierrez, D.X. Viegas, Per-Erik Forssen, and M.A. Gonzalez. COMETS: A multiple heterogeneous UAV system. In *IEEE International Workshop on Safety, Security, and Rescue Robotics, SSRR 2004*, 2004.

[146] James B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[147] P.M. Pardalos and L. S. Pitsoulis. *Nonlinear Assignment Problems: Algorithms and Applications*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2000.

[148] F. Pekergin. Combinatorial optimization by random neural network model - Application to the independent set problem. In *International Conference on Artificial Neural Networks (ICANN'92), Brighton, England*, 1992.

[149] Fernando J. Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *Neural Information Processing Systems, Denver, Colorado, USA*, pages 602–611. American Institue of Physics, New York, 1987.

[150] Fernando J. Pineda. Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation*, 1(2):161–172, 1989.

[151] Leonidas S. Pitsoulis. Quadratic semi-assignment problem. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 3170–3171. Springer, 2009.

[152] H. P. Robinson, M. Kawahara, Y. Jimbo, K. Torimitsu, Y. Kuroda, and A. Kawana. Periodic synchronized bursting and intracellular calcium transients elicited by low magnesium in cultured cortical neurons. *J. Neurophysiology*, 70(4):1606–1616, 1993.

[153] G. Rubino, M. Varela, and J.M. Bonnin. Controlling multimedia QoS in the future home network using the PSQA metric. *The Computer Journal*, 49(2):137–155, 2006.

[154] Sartaj Sahni and Teofilo Gonzalez. P-Complete Approximation Problems. *Journal of the ACM*, 23(3):555–565, 1976.

[155] Georgia Sakellari. The Cognitive Packet Network: A Survey. *The Computer Journal*, 53(3):268–279, 2010.

[156] Jorgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez. Training Recurrent Networks by Evolino. *Neural Computation*, 19(3):757–779, 2007.

[157] M. Schmidt, E. van den Berg, M. P. Friedlander, and K. Murphy. Optimizing Costly Functions with Simple Constraints: A Limited-Memory Projected Quasi-Newton Algorithm. In *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS), Florida, US 16-18 April 2009*, pages 456–463, 2009.

[158] R. Segev, M. Benveniste, Y. Shapira, and E. Ben-Jacob. Formation of electrically active clusterized neural networks. *Physical Review Letters*, 90(16):168101–1–168101–4, 2003.

[159] R. Segev, Y. Shapira, M. Benveniste, and E. Ben-Jacob. Observations and modeling of synchronized burst in two-dimensional neural networks. *Physical Review E*, 64(1):011920–1–011920–9, 2001.

[160] Jonathon Shlens, Greg D. Field, Jeffrey L. Gauthier, Martin Greschner, Alexander Sher, Alan M. Litke, and E. J. Chichilnisky. The structure of large-scale synchronized firing in primate retina. *The Journal of Neuroscience*, 29(15):5022–5031, April 2009.

[161] Jonathon Shlens, Greg D. Field, Jeffrey L. Gauthier, Matthew I. Grivich, Dumitru Petrusca, Alexander Sher, Alan M. Litke, and E. J. Chichilnisky. The structure of multi-neuron firing patterns in primate retina. *The Journal of Neuroscience*, 26(32):8254–8266, 2006.

[162] Jonathon Shlens, Fred Rieke, and EJ Chichilnisky. Synchronized firing in the retina. *Current Opinion in Neurobiology*, 18(4):396 – 402, 2008.

[163] G.T. Sibley, M.H. Rahimi, and G.S. Sukhatme. Robomote: a tiny mobile robot platform for large-scale ad-hoc sensor networks. pages 1143 –1148, 2002.

[164] Siu-yeung Cho and Tommy W. S. Chow. Training multilayer neural networks using fast global learning algorithm - least-squares and penalized optimization methods. *Neurocomputing*, 25(1-3):115 – 131, 1999.

[165] Kate Smith. Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research. *INFORMS Journal on Computing*, 11(1):15–34, 1999.

[166] Kate Smith, Marimuthu Palaniswami, and M. Krishnamoorthy. Neural techniques for combinatorial optimization with applications. *IEEE Transactions on Neural Networks*, 9(6):1301–1318, 1998.

[167] A. Stafylopatis and A. Likas. Pictorial information-retrieval using the random neural network. *IEEE Transactions on Software Engineering*, 18(7):590–600, 1992.

[168] Hisayoshi Sugiyama, Tetsuo Tsujioka, and Masashi Murata. Victim detection system for urban search and rescue based on active network operation. In *Design and application of hybrid intelligent systems*, pages 1104–1113. 2003.

[169] Hisayoshi Sugiyama, Tetsuo Tsujioka, and Masashi Murata. Collaborative movement of rescue robots for reliable and effective networking in disaster area. In *First Intern. Conf. on Collaborative Computing*, 2005.

[170] Jo Sungho, J. Yin, and Mao Zhi-Hong. Random neural networks with state-dependent firing neurons. *IEEE Transactions on Neural Networks*, 16(4):980–983, 2005.

[171] S. Tadokoro. Special project on development of advanced robots for disaster response (DDT project). pages 66–72, 2005.

[172] Ken Tanaka and Daigo Hasegawa. On computational capability of recurrent higher order neural networks. *Systems and Computers in Japan*, 32(10):42–50, 2001.

[173] A. Teke and V. Atalay. Texture classification and retrieval using random neural network model. *Computational Management Science*, 3(3):193–205, 2006.

[174] Eitan Wacholder. A Neural Network-Based Optimization Algorithm for the Static Weapon-Target Assignment Problem. *INFORMS JOURNAL ON COMPUTING*, 1(4):232–246, 1989.

[175] Thomas Wahle and Gregg Beatty. *Emergency Management Guide for Business & Industry*. U.S Federal Emergency Management Agency (FEMA), 1993.

[176] Zhigang Wang, MengChu Zhou, and N. Ansari. Ad-hoc robot wireless communication. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 4045–4050, 2003.

[177] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[178] X. Wu, B.J. D' Auriol, and S. Lee. Mobility-assisted relocation for self-deployment in wireless sensor networks. *IEICE Transactions on Communications*, 90-B(8):2056–2069, 2007.

[179] Guoliang Xing, Xiaorui Wang, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill. Integrated coverage and connectivity configuration for energy conservation in sensor networks. *ACM Transactions in Sensor Networks*, 1(1):36–72, 2005.

[180] Jim Y. F. Yam and Tommy W. S. Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1-4):219–232, 2000.

[181] Y. F. Yam, Tommy W. S. Chow, and C. T. Leung. A new method in determining initial weights of feedforward neural networks for training enhancement. *Neurocomputing*, 16(1):23 – 32, 1997.

[182] Wang Yanxia, Qian Longjun, Guo Zhi, and Ma Lifeng. Weapon target assignment problem satisfying expected damage probabilities based on ant colony algorithm. *Journal of Systems Engineering and Electronics*, 19(5):939 – 944, 2008.

[183] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *International Journal of Wireless Ad Hoc and Sensor Networks*, 1(1-2):89–124, 2005.

[184] YongJun Zhong, DonChuan Sun, and JianJun Wu. Dynamical Random Neural Network Approach to a Problem of Optimal Resource Allocation. In *Proceedings of the 8th International Work-Conference on Artificial Neural Networks, Barcelona, Spain, 8-10 June*, pages 1157–1163. Springer, Berlin, 2005.

[185] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67:301–320, 2005.

Appendices

A. Derivative calculations of RNNSI gradient descent algorithm

We present the calculations required to derive equations (3.15)-(3.17).

Firstly we differentiate $q_i = N(i)/D(i)$ with respect to the generic variable $w(u,v)$, which yields:

$$\frac{\partial q_i}{\partial w(u,v)} = \frac{\frac{\partial N(i)}{\partial w(u,v)} D(i)}{D^2(i)} - \frac{\frac{\partial D(i)}{\partial w(u,v)} N(i)}{D^2(i)} = \frac{1}{D(i)} \left[\frac{\partial N(i)}{\partial w(u,v)} - \frac{\partial D(i)}{\partial w(u,v)} q_i \right] \tag{A.1}$$

Next we derive expressions for the derivatives $\frac{\partial N(i)}{\partial w(u,v)}$ and $\frac{\partial D(i)}{\partial w(u,v)}$ for all three parameters $w^+(u,v)$, $w^-(u,v)$ and $a(u,v)$.

$$\begin{aligned} \frac{\partial N(i)}{\partial w^+(u,v)} &= \sum_{j=1}^{N} \frac{\partial q_j}{\partial w^+(u,v)} w^+(j,i) \\ &\quad + q_u \mathbf{1}_{\{v=i\}} + \sum_{j=1}^{N} \frac{\partial q_j}{\partial w^+(u,v)} \sum_{m=1}^{N} w^-(j,m)\, q_m a(m,i) \\ &\quad + \sum_{j=1}^{N} q_j \sum_{m=1}^{N} \frac{\partial q_m}{\partial w^+(u,v)} w^-(j,m)\, a(m,i) \end{aligned} \tag{A.2}$$

$$\begin{aligned} \frac{\partial N(i)}{\partial w^-(u,v)} &= \sum_{j=1}^{N} \frac{\partial q_j}{\partial w^-(u,v)} w^+(j,i) + \sum_{j=1}^{N} \frac{\partial q_j}{\partial w^-(u,v)} \sum_{m=1}^{N} w^-(j,m)\, q_m a(m,i) \\ &\quad + \sum_{j=1}^{N} q_j \sum_{m=1}^{N} \frac{\partial q_m}{\partial w^-(u,v)} w^-(j,m)\, a(m,i) + q_u q_v a(v,i) \end{aligned} \tag{A.3}$$

$$\begin{aligned}\frac{\partial N(i)}{\partial a(u,v)} &= \sum_{j=1}^{N}\frac{\partial q_j}{\partial a(u,v)}w^{+}(j,i)+\sum_{j=1}^{N}\frac{\partial q_j}{\partial a(u,v)}\sum_{m=1}^{N}w^{-}(j,m)\,q_m a(m,i)\\ &\quad+\sum_{j=1}^{N}q_j\sum_{m=1}^{N}\frac{\partial q_m}{\partial a(u,v)}w^{-}(j,m)\,a(m,i)\\ &\quad+\sum_{j=1}^{N}q_j q_u w^{-}(j,u)\mathbf{1}_{\{v=i\}} \qquad \text{(A.4)}\end{aligned}$$

$$\frac{\partial D(i)}{\partial w^{+}(u,v)}=\frac{\mathbf{1}_{\{u=i\}}}{1-d(i)}+\sum_{j=1}^{N}\frac{\partial q_j}{\partial w^{+}(u,v)}\left[w^{-}(j,i)+\sum_{m=1}^{N}w^{-}(j,i)a(i,m)\right] \qquad \text{(A.5)}$$

$$\begin{aligned}\frac{\partial D(i)}{\partial w^{-}(u,v)} &= \mathbf{1}_{\{u=i\}}\frac{1+\sum_{m=1}^{N}a(v,m)}{1-d(i)}+q_u\mathbf{1}_{\{v=i\}}+q_u\sum_{m=1}^{N}a(v,m)\mathbf{1}_{\{v=i\}}\\ &\quad+\sum_{j=1}^{N}\frac{\partial q_j}{\partial w^{-}(u,v)}\left[w^{-}(j,i)+\sum_{m=1}^{N}w^{-}(j,i)a(i,m)\right] \qquad \text{(A.6)}\end{aligned}$$

$$\begin{aligned}\frac{\partial D(i)}{\partial a(u,v)} &= \frac{w^{-}(i,u)}{1-d(i)}+\sum_{j=1}^{N}q_j w^{-}(j,u)\mathbf{1}_{\{u=i\}}\\ &\quad+\sum_{j=1}^{N}\frac{\partial q_j}{\partial a^{(}u,v)}\left[w^{-}(j,i)+\sum_{m=1}^{N}w^{-}(j,i)a(i,m)\right] \qquad \text{(A.7)}\end{aligned}$$

Substituting (A.2) and (A.5) into (A.1) gives:

$$\begin{aligned}\frac{\partial q_i}{\partial w^{+}(u,v)} &= \frac{1}{D(i)}\sum_{j=1}^{N}\frac{\partial q_j}{\partial w^{+}(u,v)}\left\{w^{+}(j,i)+\sum_{m=1}^{N}q_m w^{-}(j,m)a(m,i)\right.\\ &\quad\left.+a(j,i)\sum_{m=1}^{N}q_m w^{-}(m,j)-q_i w^{-}(j,i)\sum_{m=1}^{N}a(i,m)-q_i w^{-}(j,i)\right\}\\ &\quad+\frac{q_u}{D(i)}\left[\mathbf{1}_{\{v=i\}}-\frac{\mathbf{1}_{\{u=i\}}}{1-d(i)}\right] \qquad \text{(A.8)}\end{aligned}$$

To derive the above equation we used the fact that:

$$\begin{aligned}\sum_{j=1}^{N} q_j \sum_{m=1}^{N} \frac{\partial q_m}{\partial w^+(u,v)} w^-(j,m)a(m,i) &= \sum_{m=1}^{N} \frac{\partial q_m}{\partial w^+(u,v)} \sum_{j=1}^{N} q_j w^-(j,m)a(m,i) \\ &= \sum_{j=1}^{N} \frac{\partial q_j}{\partial w^+(u,v)} \sum_{m=1}^{N} q_m w^-(m,j)a(j,i)\end{aligned}$$

Similarly, for $\frac{\partial q_i}{\partial w^-(u,v)}$ and $\frac{\partial q_i}{\partial a(u,v)}$ we obtain:

$$\begin{aligned}\frac{\partial q_i}{\partial w^-(u,v)} &= \frac{1}{D(i)} \sum_{j=1}^{N} \frac{\partial q_j}{\partial w^-(u,v)} \Bigg\{ w^+(j,i) + \sum_{m=1}^{N} q_m w^-(j,m)a(m,i) \\ &\quad + a(j,i) \sum_{m=1}^{N} q_m w^-(m,j) - q_i w^-(j,i) \sum_{m=1}^{N} a(i,m) - q_i w^-(j,i) \Bigg\} \\ &\quad + \frac{1}{D(i)} \Bigg[q_u q_v a(v,i) - q_u \mathbf{1}_{\{u=i\}} \frac{1 + \sum_{m=1}^{N} a(v,m)}{1 - d(i)} \\ &\quad - q_v q_u \mathbf{1}_{\{v=i\}} [1 + \sum_{m=1}^{N} a(v,m)] \Bigg] \qquad \text{(A.9)}\end{aligned}$$

$$\begin{aligned}\frac{\partial q_i}{\partial a(u,v)} &= \frac{1}{D(i)} \sum_{j=1}^{N} \frac{\partial q_j}{\partial a(u,v)} \Bigg\{ w^+(j,i) + \sum_{m=1}^{N} q_m w^-(j,m)a(m,i) \\ &\quad + a(j,i) \sum_{m=1}^{N} q_m w^-(m,j) - q_i w^-(j,i) \sum_{m=1}^{N} a(i,m) - q_i w^-(j,i) \Bigg\} \\ &\quad + \frac{1}{D(i)} \Bigg[q_u (\mathbf{1}_{\{v=i\}} - \mathbf{1}_{\{u=i\}}) \sum_{j=1}^{N} q_j w^-(j,u) - \frac{q_i w^-(i,u)}{1 - d(i)} \Bigg] \qquad \text{(A.10)}\end{aligned}$$

B. Derivation of expressions associated with the efficient computation of NNLS costly functions

In this appendix, we elaborate on the equations used without derivation in section 3.3.3 for the efficient computation of the objective and gradient NNLS functions, $f(\mathbf{w})$ and $\nabla f(\mathbf{w})$, given by Eqs. (3.35) and (3.36) respectively. For the first evaluation approach, we show in detail how Eqs. (3.60) and (3.64) are derived. For the second, we obtain low complexity expressions both for the composing matrices of $\mathbf{\Gamma}$ and for vector $\mathbf{\Gamma w}$.

B.1. The first approach

B.1.1. Derivation of Eq. (3.60)

According to Eq. (3.58) each of the terms $\hat{\mathbf{z}}_k$, $k = 1, ..., K$ can be computed independently, so that the computation of $\hat{\mathbf{z}}$ requires the efficient evaluation of $\mathbf{C}_k(\mathbf{w}^+ + \mathbf{w}^-)$, $\mathbf{D}_{+k}\mathbf{w}^+$ and $\mathbf{D}_{-k}\mathbf{w}^-$.

For the examination of expression $\mathbf{C}_k(\mathbf{w}^+ + \mathbf{w}^-)$ we use the fact that $\mathbf{C}_k = [\mathbf{C}_{k1}, \cdots, \mathbf{C}_{kN}]$ and define $(\mathbf{w}^\pm)^T = \left[\left(\mathbf{w}_1^\pm\right)^T, ..., \left(\mathbf{w}_N^\pm\right)^T\right]$ so that the jth element of $\mathbf{w}_i^\pm$ is the same as $w^\pm(i, j)$. By performing the appropriate calculations and

exploiting the fact that $\mathbf{C}_{ki}$ is given by Eq. (3.53) yields:

$$\begin{aligned}\mathbf{C}_k(\mathbf{w}^+ + \mathbf{w}^-) &= [\mathbf{C}_{k1}, \cdots, \mathbf{C}_{kN}] \begin{bmatrix} \mathbf{w}_1^+ + \mathbf{w}_1^- \\ \vdots \\ \mathbf{w}_N^+ + \mathbf{w}_N^- \end{bmatrix} = \sum_{i=1}^{N} \mathbf{C}_{ki}(\mathbf{w}_i^+ + \mathbf{w}_i^-) \\ &= \sum_{i=1}^{N} q_{ik}(\mathbf{e}_i \mathbf{1}^T)(\mathbf{w}_i^+ + \mathbf{w}_i^-) = \sum_{i=1}^{N} q_{ik}\mathbf{e}_i \sum_{j=1}^{N}(w^+(i,j) + w^-(i,j)) \\ &= \mathbf{q}_k \bigodot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-})\end{aligned}$$

where $\boldsymbol{\sigma}_{W^+}$ and $\boldsymbol{\sigma}_{W^-}$ are $N \times 1$ vectors given by Eqs. (3.61) and (3.62). This definition implies that their ith element is equal to the sum of the ith row elements of the associated matrix. Similarly for the other two terms we obtain:

$$\mathbf{D}_{+k}\mathbf{w}^+ = \sum_{i=1}^{N} \mathbf{D}_{+ki}\mathbf{w}_i^+ = -\sum_{i=1}^{N} q_{ik}\mathbf{w}_i^+ = -\begin{bmatrix} \sum_{l=1}^{N} q_{lk} w^+(l,1) \\ \vdots \\ \sum_{l=1}^{N} q_{lk} w^+(l,N) \end{bmatrix} = -(\mathbf{W}^+)^T \mathbf{q}_k$$

$$\begin{aligned}\mathbf{D}_{-k}\mathbf{w}^- &= \sum_{i=1}^{N} \mathbf{D}_{-ki}\mathbf{w}_i^- = \sum_{i=1}^{N} q_{ik}\mathbf{q}_k \bigodot \mathbf{w}_i^- = \begin{bmatrix} q_{1k} \sum_{l=1}^{N} q_{lk} w^-(l,1) \\ \vdots \\ q_{Nk} \sum_{l=1}^{N} q_{lk} w^-(l,N) \end{bmatrix} \\ &= \mathbf{q}_k \bigodot \left((\mathbf{W}^-)^T \mathbf{q}_k\right)\end{aligned}$$

The above two expressions have been derived by replacing $\mathbf{D}_{+k}$ and $\mathbf{D}_{-k}$ with the expressions of Eqs. (3.54) and (3.55) respectively. Combining the above three expressions into Eq. (3.59) yields Eq. (3.60).

$$\hat{\mathbf{z}}_k = \mathbf{q}_k \bigodot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) - (\mathbf{W}^+)^T \mathbf{q}_k + \mathbf{q}_k \bigodot \left((\mathbf{W}^-)^T \mathbf{q}_k\right)$$

B.1.2. Derivation of Eq. (3.64)

In order to derive Eq. (3.64) we need to analyse the terms $\mathbf{C}_k^T \mathbf{z}_k$, $\mathbf{D}_{+k}^T \mathbf{z}_k$ and $\mathbf{D}_{-k}^T \mathbf{z}_k$ that appear in Eq. (3.63). Utilising Eqs. (3.53), (3.54) and (3.55) and performing the appropriate calculations yields:

$$\mathbf{C}_k^T\mathbf{z}_k = \begin{bmatrix} \mathbf{C}_{k1}^T \\ \vdots \\ \mathbf{C}_{kN}^T \end{bmatrix}\mathbf{z}_k = \begin{bmatrix} \mathbf{C}_{k1}^T\mathbf{z}_k \\ \vdots \\ \mathbf{C}_{kN}^T\mathbf{z}_k \end{bmatrix} = \begin{bmatrix} q_{1k}(\mathbf{1}\mathbf{e}_1^T)\mathbf{z}_k \\ \vdots \\ q_{Nk}(\mathbf{1}\mathbf{e}_N^T)\mathbf{z}_k \end{bmatrix} = (\mathbf{q_k}\bigodot\mathbf{z}_k)\bigotimes\mathbf{1}$$

$$\begin{aligned}\mathbf{D}_{+k}^T\mathbf{z}_k &= \begin{bmatrix} \mathbf{D}_{+k1}^T \\ \vdots \\ \mathbf{D}_{+kN}^T \end{bmatrix}\mathbf{z}_k = \begin{bmatrix} \mathbf{D}_{+k1}\mathbf{z}_k \\ \vdots \\ \mathbf{D}_{+kN}\mathbf{z}_k \end{bmatrix} = -\begin{bmatrix} q_{1k}\mathbf{I}_N\mathbf{z}_k \\ \vdots \\ q_{Nk}\mathbf{I}_N\mathbf{z}_k \end{bmatrix} \\ &= -\begin{bmatrix} q_{1k}\mathbf{z}_k \\ \vdots \\ q_{Nk}\mathbf{z}_k \end{bmatrix} = -\mathbf{q}_k\bigotimes\mathbf{z}_k\end{aligned}$$

$$\begin{aligned}\mathbf{D}_{-k}^T\mathbf{z}_k &= \begin{bmatrix} \mathbf{D}_{-k1}^T \\ \vdots \\ \mathbf{D}_{-kN}^T \end{bmatrix}\mathbf{z}_k = \begin{bmatrix} \mathbf{D}_{-k1}\mathbf{z}_k \\ \vdots \\ \mathbf{D}_{-kN}\mathbf{z}_k \end{bmatrix} = \begin{bmatrix} q_{1k}diag([q_{1k}, q_{2k}, \cdots, q_{Nk}])\mathbf{z}_k \\ \vdots \\ q_{Nk}diag([q_{1k}, q_{2k}, \cdots, q_{Nk}])\mathbf{z}_k \end{bmatrix} \\ &= \begin{bmatrix} q_{1k}(\mathbf{q}_k\bigodot\mathbf{z}_k) \\ \vdots \\ q_{Nk}(\mathbf{q}_k\bigodot\mathbf{z}_k) \end{bmatrix} = \mathbf{q}_k\bigotimes(\mathbf{q}_k\bigodot\mathbf{z}_k)\end{aligned}$$

If we combine the above expressions into Eq. (3.63) we obtain Eq. (3.64):

$$\begin{aligned}\widetilde{\mathbf{z}} &= \begin{bmatrix} \sum_{k=1}^K [(\mathbf{q}_k\bigodot\mathbf{z}_k)\bigotimes\mathbf{1} - \mathbf{q}_k\bigotimes\mathbf{z}_k] \\ \sum_{k=1}^K [(\mathbf{q}_k\bigodot\mathbf{z}_k)\bigotimes\mathbf{1} + \mathbf{q}_k\bigotimes(\mathbf{q}_k\bigodot\mathbf{z}_k)] \end{bmatrix} \\ &= \begin{bmatrix} \left(\sum_{k=1}^K(\mathbf{q}_k\bigodot\mathbf{z}_k)\right)\bigotimes\mathbf{1} - \sum_{k=1}^K(\mathbf{q}_k\bigotimes\mathbf{z}_k) \\ \left(\sum_{k=1}^K(\mathbf{q}_k\bigodot\mathbf{z}_k)\right)\bigotimes\mathbf{1} - \sum_{k=1}^K(\mathbf{q}_k\bigotimes(\mathbf{q}_k\bigodot\mathbf{z}_k)) \end{bmatrix}\end{aligned}$$

B.2. The second approach

In this section of the appendix we provide the details for the derivation of $\mathbf{\Gamma w}$. We start with the derivation of the individual terms of $\mathbf{\Gamma}_{lm}$ according to Eqs. (3.68)-

(3.71). Then, based on the obtained expressions for those terms we examine $\mathbf{\Gamma w}$.

B.2.1. Derivation of $\mathbf{\Gamma}_{lm}$

As discussed in section 3.3.3, in order to derive an expression for $\mathbf{\Gamma} \in \mathbb{R}^{2N^2 \times 2N^2}$ we need to examine the terms $\mathbf{\Gamma}_{lm} \in \mathbb{R}^{N^2 \times N^2}$, $l, m = 1, 2$, which are given by Eqs. (3.68)-(3.71). Let us start with the examination of term $\mathbf{\Gamma}_{11}$ given by Eq. (3.68) which is rewritten here for convenience:

$$\mathbf{\Gamma}_{11} = \sum_{k=1}^{K} \mathbf{C}_k^T \mathbf{C}_k + \sum_{k=1}^{K} \mathbf{C}_k^T \mathbf{D}_{+k} + \sum_{k=1}^{K} \mathbf{D}_{+k}^T \mathbf{C}_k + \sum_{k=1}^{K} \mathbf{D}_{+k}^T \mathbf{D}_{+k}$$

There are four terms that need to be analysed and all of them are of the form $\sum_{k=1}^{K} \mathbf{G}_k^T \mathbf{H}_k$, where $\mathbf{G}_k, \mathbf{H}_k \in \mathbb{R}^{N \times N^2}$. Moreover, these matrices can be further decomposed such that $\mathbf{G}_k = [\mathbf{G}_{k1}, \mathbf{G}_{k2}, ..., \mathbf{G}_{kN}]$ and $\mathbf{H}_k = [\mathbf{H}_{k1}, \mathbf{H}_{k2}, ..., \mathbf{H}_{kN}]$, with $\mathbf{G}_{ki}, \mathbf{H}_{ki} \in \mathbb{R}^{N \times N}$. For the general term $\sum_{k=1}^{K} \mathbf{G}_k^T \mathbf{H}_k$ we have that:

$$\begin{aligned} \sum_{k=1}^{K} \mathbf{G}_k^T \mathbf{H}_k &= \sum_{k=1}^{K} \begin{bmatrix} \mathbf{G}_{k1}^T \\ \vdots \\ \mathbf{G}_{kN}^T \end{bmatrix} [\mathbf{H}_{k1}, \mathbf{H}_{k2}, \cdots, \mathbf{H}_{kN}] \\ &= \begin{bmatrix} \sum_{k=1}^{K} \mathbf{G}_{k1}^T \mathbf{H}_{k1} & \cdots & \sum_{k=1}^{K} \mathbf{G}_{k1}^T \mathbf{H}_{kN} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{K} \mathbf{G}_{kN}^T \mathbf{H}_{k1} & \cdots & \sum_{k=1}^{K} \mathbf{G}_{kN}^T \mathbf{H}_{kN} \end{bmatrix} \end{aligned} \tag{B.1}$$

Consequently, as all terms that have to be analysed are of the form $\sum_{k=1}^{K} \mathbf{G}_k^T \mathbf{H}_k$, it is sufficient to analyse the terms $\sum_{k=1}^{K} \mathbf{G}_{ki}^T \mathbf{H}_{kj}$, when $\mathbf{G}_{ki}^T$ and $\mathbf{H}_{kj}$ represent different matrix combinations. For example, for the second term of $\mathbf{\Gamma}_{11}$, $\sum_{k=1}^{K} \mathbf{C}_k^T \mathbf{D}_{+k}$, it is sufficient to analyse $\sum_{k=1}^{K} \mathbf{C}_{ki}^T \mathbf{D}_{+kj}$. Next, we derive expressions for each one of the four terms of $\mathbf{\Gamma}_{11}$ in the order that they appear in Eq. (3.68). This is accomplished by utilising Eqs. (3.53), (3.54) and (3.55).

$$\sum_{k=1}^{K}\mathbf{C}_{ki}^{T}\mathbf{C}_{kj} = \sum_{k=1}^{K}q_{ik}q_{jk}(\mathbf{e}_i\mathbf{1}^T)^T(\mathbf{e}_j\mathbf{1}^T) = \sum_{k=1}^{K}q_{ik}q_{jk}\mathbf{1}\mathbf{e}_i^T\mathbf{e}_j\mathbf{1}^T$$
$$= \begin{cases} \mathbf{0}_{N\times N} & i\neq j \\ \left(\sum_{k=1}^{K}q_{ik}^2\right)(\mathbf{1}\times\mathbf{1}^T) & i=j \end{cases}$$

$$\sum_{k=1}^{K}\mathbf{C}_{ki}^{T}\mathbf{D}_{+kj} = \sum_{k=1}^{K}q_{ik}(\mathbf{e}_i\mathbf{1}^T)^T\mathbf{D}_{+kj} = \sum_{k=1}^{K}q_{ik}\mathbf{1}\mathbf{e}_i^T(-q_{jk}\mathbf{I}_N) = -\left(\sum_{k=1}^{K}q_{ik}q_{jk}\right)(\mathbf{1}\mathbf{e}_i^T)$$

$$\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{C}_{kj} = \sum_{k=1}^{K}\mathbf{D}_{+ki}\mathbf{C}_{kj} = \sum_{k=1}^{K}(-q_{ik}\mathbf{I}_N)(\mathbf{e}_j\mathbf{1}^T)q_{jk} = -\left(\sum_{k=1}^{K}q_{ik}q_{jk}\right)(\mathbf{e}_j\mathbf{1}^T)$$

$$\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{D}_{+kj} = \sum_{k=1}^{K}\mathbf{D}_{+ki}\mathbf{D}_{+kj} = \sum_{k=1}^{K}(-q_{ik}\mathbf{I}_N)(-q_{jk}\mathbf{I}_N) = \left(\sum_{k=1}^{K}q_{ik}q_{jk}\right)\mathbf{I}_N$$

Let us proceed now with the examination of the terms composing $\mathbf{\Gamma}_{12}$ given by Eq. (3.69). Since the first two terms of $\mathbf{\Gamma}_{12}$ are the same with the first and third terms of $\mathbf{\Gamma}_{11}$, we only have to analyse the terms $\sum_{k=1}^{K}\mathbf{C}_k^T\mathbf{D}_{-k}$ and $\sum_{k=1}^{K}\mathbf{D}_{+k}^T\mathbf{D}_{-k}$. We have that:

$$\sum_{k=1}^{K}\mathbf{C}_{ki}^{T}\mathbf{D}_{-kj} = \sum_{k=1}^{K}q_{ik}(\mathbf{e}_i\mathbf{1}^T)^T(q_{jk}\mathrm{diag}(\mathbf{q}_k)) = \sum_{k=1}^{K}q_{ik}q_{jk}\mathbf{1}(\mathbf{e}_i^T\mathrm{diag}(\mathbf{q}_k))$$
$$= \sum_{k=1}^{K}q_{ik}q_{jk}\mathbf{1}(q_{ik}\mathbf{e}_i^T) = \left(\sum_{k=1}^{K}q_{ik}^2q_{jk}\right)(\mathbf{1}\mathbf{e}_i^T)$$

$$\begin{aligned}\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{D}_{-kj} &= \sum_{k=1}^{K}\mathbf{D}_{+ki}\mathbf{D}_{-kj}=\sum_{k=1}^{K}\left(-q_{ik}\mathbf{I}_N\right)\left(q_{jk}\text{diag}(\mathbf{q}_k)\right)\\ &= -\sum_{k=1}^{K}q_{ik}q_{jk}\text{diag}(\mathbf{q}_k)=-\text{diag}([\sum_{k=1}^{K}q_{ik}q_{jk}q_{1k},\ldots,\sum_{k=1}^{K}q_{ik}q_{jk}q_{Nk}])\end{aligned}$$

Similarly, for $\mathbf{\Gamma}_{21}$ we only need to analyse $\sum_{k=1}^{K}\mathbf{D}_{-k}^{T}\mathbf{C}_k$ and $\sum_{k=1}^{K}\mathbf{D}_{-k}^{T}\mathbf{D}_{+k}$ as its first two terms are identical with the first two terms of $\mathbf{\Gamma}_{11}$. We have that:

$$\begin{aligned}\sum_{k=1}^{K}\mathbf{D}_{-ki}^{T}\mathbf{C}_{kj} &= \sum_{k=1}^{K}\mathbf{D}_{-ki}\mathbf{C}_{kj}=\sum_{k=1}^{K}\left(q_{ik}\text{diag}(\mathbf{q}_k)\right)q_{jk}(\mathbf{e}_j\mathbf{1}^T)\\ &= \sum_{k=1}^{K}q_{ik}q_{jk}(\text{diag}(\mathbf{q}_k)\mathbf{e}_j)\mathbf{1}^T=\sum_{k=1}^{K}q_{ik}q_{jk}(q_{jk}\mathbf{e}_j)\mathbf{1}^T=\left(\sum_{k=1}^{K}q_{ik}q_{jk}^2\right)(\mathbf{e}_j\mathbf{1}^T)\end{aligned}$$

$$\begin{aligned}\sum_{k=1}^{K}\mathbf{D}_{-ki}^{T}\mathbf{D}_{+kj} &= \sum_{k=1}^{K}\mathbf{D}_{-ki}\mathbf{D}_{+kj}=\sum_{k=1}^{K}q_{ik}\text{diag}(\mathbf{q}_k)\left(-q_{jk}\mathbf{I}_N\right)=-\sum_{k=1}^{K}q_{ik}q_{jk}\text{diag}(\mathbf{q}_k)\\ &= -\text{diag}([\sum_{k=1}^{K}q_{ik}q_{jk}q_{1k},\ldots,\sum_{k=1}^{K}q_{ik}q_{jk}q_{Nk}])=\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{D}_{-kj}\end{aligned}$$

Finally, for $\mathbf{\Gamma}_{22}$ the only term that has not already been analysed is $\sum_{k=1}^{K}\mathbf{D}_{-ki}^{T}\mathbf{D}_{-kj}$.

$$\begin{aligned}\sum_{k=1}^{K}\mathbf{D}_{-ki}^{T}\mathbf{D}_{-kj} &= \sum_{k=1}^{K}\mathbf{D}_{-ki}\mathbf{D}_{-kj}=\sum_{k=1}^{K}q_{ik}\text{diag}(\mathbf{q}_k)q_{jk}\text{diag}(\mathbf{q}_k)\\ &= \sum_{k=1}^{K}q_{ik}q_{jk}\text{diag}(\mathbf{q}_k\bigodot\mathbf{q}_k)=\text{diag}([\sum_{k=1}^{K}q_{ik}q_{jk}q_{1k}^2,\ldots,\sum_{k=1}^{K}q_{ik}q_{jk}q_{Nk}^2])\end{aligned}$$

An important observation with respect to the terms of $\mathbf{\Gamma}_{lm}$ is that the values of all their elements will be known if the following five expressions are known: (1) $\sum_{k=1}^{K}q_{ik}^2$, (2) $\sum_{k=1}^{K}q_{ik}q_{jk}$, (3) $\sum_{k=1}^{K}q_{ik}^2q_{jk}$, (4) $\sum_{k=1}^{K}q_{ik}q_{jk}q_{lk}$, and (5) $\sum_{k=1}^{K}q_{ik}q_{jk}q_{lk}^2$. These expressions have been defined in Eqs. (3.72)-(3.76) in vector/matrix form and are given by $\boldsymbol{\sigma}_q$, $\mathbf{M}$, $\boldsymbol{M}^s$, $\mathbf{R}^i$, $i=1,...,N$ and $\mathbf{R}^{s,i}$, $i=1,...,N$ respectively.

B.2.2. Derivation of Γw

In section 3.3.3, we have shown that in order to obtain a low complexity expression for $\overline{\mathbf{z}} = \mathbf{\Gamma}\mathbf{w}$ we need to examine the terms $\overline{\mathbf{z}}_l = \mathbf{\Gamma}_{l1}\mathbf{w}^+ + \mathbf{\Gamma}_{l2}\mathbf{w}^-$, where $\overline{\mathbf{z}}_l \in \mathbb{R}^{N^2\times 1}, l = 1, 2$ can be further decomposed into $\overline{\mathbf{z}}_l^T = [\overline{\mathbf{z}}_{l1}^T, ..., \overline{\mathbf{z}}_{lN}^T]$ with elements $\overline{\mathbf{z}}_{li} \in \mathbb{R}^{N\times 1}$(see Eq. (3.77)). However, as each of the terms composing $\mathbf{\Gamma}_{lm}$ (Eqs. (3.68)-(3.71)) are given by the general expression (B.1) we can write the product of any of these matrices with the generic vector $\mathbf{w}$, which represents either $\mathbf{w}^+$ or $\mathbf{w}^-$, as:

$$\begin{aligned}\breve{\mathbf{z}} &= \left(\sum_{k=1}^{K}\mathbf{G}_k^T\mathbf{H}_k\right)\mathbf{w} = \begin{bmatrix} \sum_{k=1}^{K}\mathbf{G}_{k1}^T\mathbf{H}_{k1} & \cdots & \sum_{k=1}^{K}\mathbf{G}_{k1}^T\mathbf{H}_{kN} \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^{K}\mathbf{G}_{kN}^T\mathbf{H}_{k1} & \cdots & \sum_{k=1}^{K}\mathbf{G}_{kN}^T\mathbf{H}_{kN} \end{bmatrix}\begin{bmatrix}\mathbf{w}_1 \\ \vdots \\ \mathbf{w}_N\end{bmatrix} \\ &= \begin{bmatrix} \sum_{j=1}^{N}(\sum_{k=1}^{K}\mathbf{G}_{k1}^T\mathbf{H}_{kj})\mathbf{w}_j \\ \vdots \\ \sum_{j=1}^{N}(\sum_{k=1}^{K}\mathbf{G}_{kN}^T\mathbf{H}_{kj})\mathbf{w}_j \end{bmatrix} = \begin{bmatrix}\breve{\mathbf{z}}_1 \\ \vdots \\ \breve{\mathbf{z}}_N\end{bmatrix}\end{aligned}$$

Based on the above expression, it is sufficient to only analyse the terms $\breve{\mathbf{z}}_i = \sum_{j=1}^{N}(\sum_{k=1}^{K}\mathbf{G}_{k1}^T\mathbf{H}_{kj})\mathbf{w}_j$, $\breve{\mathbf{z}}_i \in \mathbb{R}^{N\times 1}$, $i = 1, ..., N$, using the already derived expressions for the composing terms of $\mathbf{\Gamma}_{lm}$. Let us start our examination with the terms of $\mathbf{\Gamma}_{11}\mathbf{w}^+$. We have that:

$$\begin{aligned}\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{C}_{ki}^T\mathbf{C}_{kj}\right)\mathbf{w}_j^+ &= \left(\sum_{k=1}^{K}q_{ik}^2\right)(\mathbf{1}\times\mathbf{1}^T)\mathbf{w}_i^+ = \mathbf{1}\sigma_q(i)\sum_{j=1}^{N}w^+(i,j) \\ &= \mathbf{1}\sigma_q(i)\sigma_{W^+}(i)\end{aligned}$$

$$\begin{aligned}\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{C}_{ki}^T\mathbf{D}_{+kj}\right)\mathbf{w}_j^+ &= -\sum_{j=1}^{N}\left(\sum_{k=1}^{K}q_{ik}q_{jk}\right)(\mathbf{1}\mathbf{e}_i^T)\mathbf{w}_j^+ = -\mathbf{1}\sum_{j=1}^{N}M_{ij}w^+(j,i) \\ &= -\mathbf{1}\sigma_{MW^+}(i)\end{aligned}$$

$$\begin{aligned}
\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{C}_{kj}\right)\mathbf{w}_{j}^{+} &= -\sum_{j=1}^{N}\left(\sum_{k=1}^{K}q_{ik}q_{jk}\right)(\mathbf{e}_{j}\mathbf{1}^{T})\mathbf{w}_{j}^{+} \\
&= -\sum_{j=1}^{N}\mathbf{e}_{j}M_{ij}\sum_{l=1}^{N}w^{+}(j,l) = -\sum_{j=1}^{N}\mathbf{e}_{j}M_{ij}\sigma_{W^{+}}(j) \\
&= -\begin{bmatrix} M_{i1}\sigma_{W^{+}}(1) \\ 0 \\ \vdots \\ 0 \end{bmatrix} - \cdots - \begin{bmatrix} 0 \\ \vdots \\ 0 \\ M_{iN}\sigma_{W^{+}}(N) \end{bmatrix} \\
&= -\begin{bmatrix} M_{i1}\sigma_{W^{+}}(1) \\ \vdots \\ M_{iN}\sigma_{W^{+}}(N) \end{bmatrix} = -\mathbf{m}_{i}^{r}\bigodot\boldsymbol{\sigma}_{W^{+}}
\end{aligned}$$

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{D}_{+kj}\right)\mathbf{w}_{j}^{+} = \sum_{j=1}^{N}\left(\sum_{k=1}^{K}q_{ik}q_{jk}\right)\mathbf{I}_{N}\mathbf{w}_{j}^{+} = \sum_{j=1}^{N}M_{ij}\mathbf{w}_{j}^{+} = (\mathbf{W}^{+})^{T}\mathbf{m}_{i}^{r}$$

where $\boldsymbol{\sigma}_{MW^{+}} = (\mathbf{M}\bigodot(\mathbf{W}^{+})^{T})\mathbf{1}$. For the derivation of the above expressions we have used Eqs. (3.72) and (3.73), as well as Eq. (3.61).

Let us know examine the composing terms of $\boldsymbol{\Gamma}_{12}\mathbf{w}^{-}$. We have that:

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{C}_{ki}^{T}\mathbf{C}_{kj}\right)\mathbf{w}_{j}^{-} = \mathbf{1}\sigma_{q}(i)\sigma_{W^{-}}(i)$$

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{C}_{kj}\right)\mathbf{w}_{j}^{-} = -\mathbf{m}_{i}^{r}\bigodot\boldsymbol{\sigma}_{W^{-}}$$

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{C}_{ki}^{T}\mathbf{D}_{-kj}\right)\mathbf{w}_{j}^{-} = \sum_{j=1}^{N}\left(\sum_{k=1}^{K}q_{ik}^{2}q_{jk}\right)(\mathbf{1}\mathbf{e}_{i}^{T})\mathbf{w}_{j}^{-} = \mathbf{1}\sum_{j=1}^{N}M_{ij}^{s}w^{-}(j,i) = \mathbf{1}\sigma_{M^{s}W^{-}}(i)$$

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{D}_{-kj}\right)\mathbf{w}_{j}^{-} = -\sum_{j=1}^{N}\mathrm{diag}([\sum_{k=1}^{K}q_{ik}q_{jk}q_{1k},\ldots,\sum_{k=1}^{K}q_{ik}q_{jk}q_{Nk}])\mathbf{w}_{j}^{-}$$
$$= -\begin{bmatrix}\sum_{j=1}^{N}R_{j1}^{i}w^{-}(j,1)\\ \vdots \\ \sum_{j=1}^{N}R_{jN}^{i}w^{-}(j,N)\end{bmatrix} = -(\mathbf{R}^{i}\bigodot\mathbf{W}^{-})^{T}\mathbf{1}$$

where $\boldsymbol{\sigma}_{M^{s}W^{-}} = (\mathbf{M}^{s}\bigodot(\mathbf{W}^{-})^{T})\mathbf{1}$. For the derivation of the above expressions we have used Eqs. (3.72), (3.73), (3.74) and (3.75), as well as Eq. (3.62).

Let us now proceed to the examination of $\boldsymbol{\Gamma}_{21}\mathbf{w}^{+}$. As we have already examined terms $\left(\sum_{k=1}^{K}\mathbf{C}_{k}^{T}\mathbf{C}_{k}\right)\mathbf{w}^{+}$ and $\left(\sum_{k=1}^{K}\mathbf{C}_{k}^{T}\mathbf{D}_{+k}\right)\mathbf{w}^{+}$ we analyse the other two terms:

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{-ki}^{T}\mathbf{C}_{kj}\right)\mathbf{w}_{j}^{+} = \sum_{j=1}^{N}\left(\sum_{k=1}^{K}q_{ik}q_{jk}^{2}\right)(\mathbf{e}_{j}\mathbf{1}^{T})\mathbf{w}_{j}^{+} = \sum_{j=1}^{N}\mathbf{e}_{j}M_{ji}^{s}\sum_{l=1}^{N}w^{+}(j,l)$$
$$= \sum_{j=1}^{N}\mathbf{e}_{j}M_{ji}^{s}\sigma_{W^{+}}(j) = \mathbf{m}_{i}^{s,c}\bigodot\boldsymbol{\sigma}_{W^{+}}$$

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{-ki}^{T}\mathbf{D}_{+kj}\right)\mathbf{w}_{j}^{+} = \sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{+ki}^{T}\mathbf{D}_{-kj}\right)\mathbf{w}_{j}^{+} = -(\mathbf{R}^{i}\bigodot\mathbf{W}^{+})^{T}\mathbf{1}$$

Finally, for $\boldsymbol{\Gamma}_{22}\mathbf{w}^{-}$ we only need to analyse the terms $(\sum_{k=1}^{K}\mathbf{D}_{-k}^{T}\mathbf{C}_{k})\mathbf{w}^{-}$ and $(\sum_{k=1}^{K}\mathbf{D}_{-k}^{T}\mathbf{D}_{-k})\mathbf{w}^{-}$. We have that:

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{-ki}^{T}\mathbf{C}_{kj}\right)\mathbf{w}_{j}^{-} = \mathbf{m}_{i}^{s,c}\bigodot\boldsymbol{\sigma}_{W^{-}}$$

$$\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\mathbf{D}_{-ki}^{T}\mathbf{D}_{-kj}\right)\mathbf{w}_{j}^{-} = \sum_{j=1}^{N}\mathrm{diag}([\sum_{k=1}^{K}q_{ik}q_{jk}q_{1k}^{2},\ldots,\sum_{k=1}^{K}q_{ik}q_{jk}q_{Nk}^{2}])\mathbf{w}_{j}^{-}$$
$$= \begin{bmatrix}\sum_{j=1}^{N}R_{j1}^{i}w^{-}(j,1)\\ \vdots \\ \sum_{j=1}^{N}R_{jN}^{i}w^{-}(j,N)\end{bmatrix} = (\mathbf{R}^{s,i}\bigodot\mathbf{W}^{-})^{T}\mathbf{1}$$

For the derivation of the last two expressions we have used Eqs. (3.74) and (3.76),

as well as Eq. (3.62).

Combining the above expressions yields:

$$\begin{aligned}
\overline{\mathbf{z}}_{1i} &= \Big(\mathbf{1}\sigma_q(i)\sigma_{W^+}(i) - \mathbf{1}\sigma_{MW^+}(i) - \mathbf{m}_i^r \bigodot \boldsymbol{\sigma}_{W^+} + (\mathbf{W}^+)^T\mathbf{m}_i^r\Big) \\
&\quad + \Big(\mathbf{1}\sigma_q(i)\sigma_{W^-}(i) - \mathbf{m}_i^r \bigodot \boldsymbol{\sigma}_{W^-} + \mathbf{1}\sigma_{M^sW^-}(i) - (\mathbf{R}^i \bigodot \mathbf{W}^-)^T\mathbf{1}\Big) \\
&= \sigma_{\overline{z}}(i)\mathbf{1} - \mathbf{m}_i^r \bigodot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) + (\mathbf{W}^+)^T\mathbf{m}_i^r - (\mathbf{R}^i \bigodot \mathbf{W}^-)^T\mathbf{1}
\end{aligned}$$

where

$$\boldsymbol{\sigma}_{\overline{z}} = \boldsymbol{\sigma}_q \bigodot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) - (\mathbf{M} \bigodot (\mathbf{W}^+)^T)\mathbf{1} + (\mathbf{M}^s \bigodot (\mathbf{W}^-)^T)\mathbf{1}$$

Also for the vector $\overline{\mathbf{z}}_2$ we have that:

$$\begin{aligned}
\overline{\mathbf{z}}_{2i} &= \Big(\mathbf{1}\sigma_q(i)\sigma_{W^+}(i) - \mathbf{1}\sigma_{MW^+}(i) + \mathbf{m}_i^{s,c} \bigodot \boldsymbol{\sigma}_{W^+} - (\mathbf{R}^i \bigodot \mathbf{W}^+)^T\mathbf{1}\Big) \\
&\quad + \Big(\mathbf{1}\sigma_q(i)\sigma_{W^-}(i) + \mathbf{m}_i^{s,c} \bigodot \boldsymbol{\sigma}_{W^-} + \mathbf{1}\sigma_{M^sW^-}(i) + (\mathbf{R}^{s,i} \bigodot \mathbf{W}^-)^T\mathbf{1}\Big) \\
&= \sigma_{\overline{z}}(i)\mathbf{1} + \mathbf{m}_i^{s,c} \bigodot (\boldsymbol{\sigma}_{W^+} + \boldsymbol{\sigma}_{W^-}) - (\mathbf{R}^i \bigodot \mathbf{W}^+)^T\mathbf{1} + (\mathbf{R}^{s,i} \bigodot \mathbf{W}^-)^T\mathbf{1}
\end{aligned}$$