TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Data-driven prognostics and logistics optimisation

Citation for published version (APA):
de Oliveira da Costa, P. R. (2022). *Data-driven prognostics and logistics optimisation: A deep learning journey*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Industrial Engineering and Innovation Sciences]. Eindhoven University of Technology.

Document status and date:
Published: 03/02/2022

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 05. Oct. 2023

DATA-DRIVEN PROGNOSTICS AND LOGISTICS OPTIMISATION: A DEEP LEARNING JOURNEY

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5430-0

SIKS Dissertation Series No. 2022-2
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by ProefschriftMaken - www.proefschriftmaken.nl
Cover design by Paulo da Costa.

Copyright ©2022, Paulo da Costa. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the author.

DATA-DRIVEN PROGNOSTICS AND LOGISTICS OPTIMISATION: A DEEP LEARNING JOURNEY

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op donderdag 3 februari 2022 om 11:00 uur

door

Paulo Roberto de Oliveira da Costa

geboren te Belém, Pará, Brazilië

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. I.E.J. Heynderickx
promotor:	prof.dr.ir. U. Kaymak
copromotor(en):	dr. Y. Zhang
	dr. A.E. Akçay
leden:	prof.dr.ir. G.J.A.N. van Houtum
	prof.dr. M. Pechenizkiy
	prof.dr. O. Fink (Eidgenössische Technische Hochschule Zürich)
	dr. N. Yorke-Smith (Technische Universiteit Delft)

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Acknowledgements

The work described in this thesis is the result of the collaboration with my daily supervisors and co-promoters, dr. Yingqian Zhang and dr. Alp Akçay, who were a source of encouragement, advice and inspiration throughout this PhD trajectory. Yingqian and Alp, I am deeply thankful for all the help throughout the last few years. In particular, thanks for the support, guidance, fruitful conversations, constructive feedback, and the freedom to explore different research directions and problems. It has been a pleasure working alongside both of you.

I am also grateful to my committee members, prof.dr.ir. Geert-Jan van Houtum, prof.dr. Mykola Pechenizkiy, prof.dr. Olga Fink and dr. Neil Yorke-Smith, for the appraisal of the work presented in this thesis and invaluable feedback. I also would like to thank my promoter, prof.dr.ir. Uzay Kaymak, for the opportunity to undertake a PhD at TU Eindhoven and the freedom to pursue my interests.

I am happy to have had the opportunity to meet and collaborate with amazing people throughout my PhD. For that, I want to thank *all* members of the Information Systems group, especially Jason Rhuggennath, Sicui Zhang, Jonnro Erasmus, Rick Gilsing, Onat Ege Adalı, Reza Afshar, Ya Song, Emmy Bos, Konstantinos Traganos, Bambang Suratno, Frank Berkers, Amirreza Farahani, Robbert Reijnen, Sander Peters, Peipei Chen and Caro Fuchs. Jonnro, Rick, Onat and Jason, I cannot thank you enough for our time together and the camaraderie we have built in the last few years. I hope to continue our friendship for many years to come. I have also been fortunate to have had many prolific collaborations with members of other groups, for that I must thank Simon Voorberg, Sami Özarık, Alexandre Florio, and Willem van Jaarsveld from the Operations, Planning, Accounting and Control group, and Peter Verleijsdonk and Stella Kapodistria from the Stochastic Operations Research group.

I would like to extend my gratitude to the many institutions that made this PhD possible, in particular, the Dutch Research Council (NWO), Dutch Research School for Information and Knowledge Systems (SIKS), Graduate Program Operations Management & Logistics (GP-OML), and the Dutch Network on Mathematics and Operations Research (LNMB). I am also grateful for the collaboration with the members of my research project, Sicco Verwer, Wan-Jui Lee, Laurens Bliek, Mathijs

de Weerdt, Mauro Barbieri, Verus Pronk, Jan Korst, Bob Huisman and Wouter van Dis. Furthermore, I would like to thank the companies that were part of my project, namely, the Dutch Railways (NS), Fokker Services and Philips.

I dedicate this thesis to my parents, Socorro and Sergio, and sister Paula. Thank you for all the love, support, and sacrifices made to further my education. Your efforts made this thesis possible.

Finally, I would like to express my special gratitude to my fiancée Anasztázia for having patience, offering support and serving as an unyielding source of encouragement throughout this journey. This thesis is also dedicated to you.

Table of contents

Chapter 1

Introduction

1.1 Deep Learning for Maintenance and Logistics

The topic of this thesis is the study of *machine learning* (ML) and *operational research* (OR) methods for problems in maintenance and logistics. Our focus is on *prediction* and *optimisation* problems that arise when considering policies to maintain a *network of industrial assets* operational whilst minimising maintenance and travel costs. Traditionally, methods for solving these problems relied on handcrafted and expert-dependent solutions that might be costly to obtain, inaccurate or inefficient to the problems at hand. We move away from this traditional view and propose to use data-driven methods based on ML to improve the accuracy, efficiency, and effectiveness of previous solutions.

The overall structure of the thesis follows from breaking down the main problem into three main components. In particular, we first consider the estimation of the remaining lifetime of assets, essential for better planning of maintenance activities. Thus, we study *asset prognostics* problems (see Section 1.2), in which the objective is to arrive at accurate remaining life predictions, i.e., point-estimates of the remaining lifetime of assets, from sensor data gathered via monitoring devices. Additionally, we study the problem of learning remaining useful life forecasts when previous observed run-to-failure data is not directly available for the asset of interest, but data from other similar assets exist. We then study how to use this previously labelled failure data from pre-existing assets to make inferences for other (newer) assets with only sensor information.

Secondly, we consider optimisation problems that arise when planning the visitation of locations in the network. Specifically, we consider *routing* problems (see Section 1.3) and study how to leverage information from the network instance, such that routes can be obtained with minimum travelling costs. We study well-known combinatorial optimisation (CO) problems such as the *travelling salesman problem* and the *vehicle routing problem* and propose methods that can learn to select

tour perturbation operations requiring only information on locations and distances. Later, we consider the scenario in which previous expert heuristics to solve routing problems exist (as is often the case) and study how such heuristics can be leveraged to learn good policies more efficiently.

Lastly, we combine the first two scenarios in a *routing and maintenance* decision-making problem (see Section 1.4). In this problem, we consider the existence of partial information on asset degradation from asset prognostics algorithms for each asset in a network. The goal is to optimise preventive (prior to failure) and corrective (after failure) maintenance costs for a repairperson serving a network of assets considering their uncertain failure times. Therefore, we wish to learn policies that consider the failure time uncertainty while optimising joint maintenance and travel decisions.

In all studied problems, we consider the availability of data sources, including geographical locations, remote monitoring sensors and personnel availability. We assume access to sampled data from these problems and aim to learn predictions and decisions directly from these heterogeneous data sources. The methodologies proposed in this thesis focus on end-to-end ML, leveraging *deep learning* methods. These methods excel in the presence of high-dimensional heterogeneous data, learning new representations of its inputs needed to approximate complex functions. Thus, the methods proposed in this thesis comprise algorithmic solutions based on deep learning, leveraging the input data of each problem to arrive at final outputs requiring little human intervention.

The remainder of this chapter is organised as follows. In sections 1.2, 1.3 and 1.4 we provide some background knowledge about the main problems and research questions studied in this thesis. In particular, Section 1.2 provides background on asset prognostics, Section 1.3 on routing problems, and Section 1.4 on combined maintenance and routing problems. Next, in Section 1.5, we discuss the main contributions of this thesis for each studied problem. Lastly, Section 1.6 provides an outline of how the remainder of this thesis is organised.

1.2 Asset Prognostics

When considering the reliability of assets, it is often desired that maintenance is performed to keep equipment running for as long as possible at minimum costs. For this reason, proactive strategies such as condition-based maintenance (CBM) have been replacing classical maintenance strategies such as corrective maintenance (CM), i.e., waiting for the entire useful life period of assets to exhaust before taking any maintenance actions, and preventive maintenance (PM), e.g., performing maintenance in regular intervals. Unlike its predecessors, CBM uses direct information from the degradation of assets to make maintenance decisions. Note that CBM can be seen as a specific form of PM. The main distinction between the two relies on

the fact that CBM uses information coming from a specific component/system to plan maintenance. In contrast, PM is more general and often employs population-based information, e.g., the probability distribution of failure times, to decide on maintenance activities.

A common way to retrieve degradation information for CBM is to employ remote monitoring devices. Historically, decision-makers would decide on the best time to schedule downtime based on manual inspection of logged sensor values. With the recent abundance of interconnected devices, decision-makers need better tools to make use of vast amounts of gathered data. In such cases, models and algorithms can be used to analyse the incoming data, providing insights regarding the health condition and future operation of assets. Prognostics and health management (PHM) is the component of CBM concerned with assessing the current and the future health state of systems [47]. The main goal of PHM is to utilise health information such that maintenance policies can be optimised. PHM-fuelled policies can reduce maintenance time due to better fault identifications, minimise disruption due to better planning of downtime, and reduce costs due to more efficient asset operation and reduced repair costs. A primary component of PHM is having accurate *prognostics*, i.e., being able to tell when an asset or part will fail before it occurs.

Prognostics mainly focuses on the analysis of failure modes and the detection of early signs of degradation. The most known metric in prognostics is the remaining useful lifetime (RUL) of assets and components, i.e., the remaining time (sometimes measured in the number of cycles) that an asset/component will remain operational. To arrive at accurate RUL predictions, decision-makers need to take multiple factors into consideration, such as operating conditions, usage, and age of assets. Having accurate RUL predictions allows maintenance decision-makers to realise their part replacement plans and improve process safety, availability, and efficiency. In recent years, prognostics methods have evolved to consider advanced signal processing methods based on pattern recognition. The core idea of these methods is to analyse the remote monitoring data automatically, identify early signs of degradation and, provided this information, output RUL predictions that can be used in maintenance decision support systems.

1.2.1 Remaining Useful Lifetime Prediction

Arriving at accurate RUL predictions is no easy task. Predictions based on monitoring data need to adapt in real-time to the degradation data and the evolution of the signals over time. Additionally, monitoring devices can generate erroneous readings due to sensor malfunction, noise and wrong calibration. Thus, data-driven predictive methods need to filter the noise coming from devices and adapt the predictions considering only actual degradation experienced by assets. Assets can also have different ages, be operated in distinct operating conditions and under vari-

ous loads. Moreover, assets and components can fail due to many reasons (failure modes). As a result, models capable of detecting a specific type of failure mode may fail to identify other kinds of failures even when considering the same asset.

Data-driven methods based on ML have gained much attention due to their high expressive power and performance results surpassing other methods in prognostics [137]. These can automatically analyse heterogeneous data from remote monitoring devices such as cameras, vibration sensors, and spectrograms. Moreover, they require little expert knowledge about the underlying degradation process of assets, unlike other classes of methods based on physics and statistical models [47]. However, the main drawback of ML-based models is the requirement of available previous run-to-failure data for training. In some cases, this data can be expensive or even impractical to obtain. Additionally, in classical ML, experts still need to devise useful features from incoming data, which is time-consuming and requires expert knowledge about the available data and degradation process.

1.2.2 Problem Statement

In this thesis, we study the problem of predicting the RUL of single assets directly from acquired sensor information. This is a crucial problem when planning maintenance policies both for single and multiple assets in a network. In this problem, we consider the existence of multiple sensor readings acquired over time, which are related to monitoring values of different components of an asset. The main goal is to arrive at accurate RUL predictions directly from data with minimal human intervention. Inspired by practical problems requiring RUL prediction for assets with no previous run-to-failure information, we assume that previous run-to-failure data is only partially available and pertaining to previous assets operated in different conditions with different fault (failure) modes. To adequately address these scenarios, we separate the main problem into two sub-problems, explained below.

The first problem assumes that sensor data comprises non-independent time-varying sequential data. An effective model for this scenario is required to monitor multiple sensor readings retaining long term information from sensor data to identify faulty behaviour. An example of this scenario is when an asset starts showing faulty behaviour but continues running for an extended period of time before showing other signs of degradation and finally failing. This data can be highly complex and impractical for experts to monitor and analyse manually. Therefore, successful data-driven RUL prediction models need to retain information over extended periods to arrive at accurate RUL predictions. In this scenario, we assume that previously labelled run-to-failure data is available, and the goal is to arrive at accurate RUL predictions from this complex input data.

The second problem is to achieve accurate RUL predictions for new or replacement assets with no previous degradation data. Often, when a new part or asset is installed, only data from previously installed assets with different operating con-

ditions, fault modes and installed sensors are available. In these cases, decision-makers would need to wait for actual run-to-failure information to be gathered before producing accurate predictions. This process is undesirable due to possible high downtime costs and unplanned maintenance. Thus, we consider the problem in which previous run-to-failure data exists but are related to assets under different operating conditions and fault modes. In this problem, we assume that assets for which we aim to produce RUL predictions have available sensor information but no run-to-failure labels. In other words, we can observe the sensor information, but we have no information about the realised remaining useful life of assets.

Research Question

We study the main problem of devising data-driven RUL estimates based on sequential data gathered by monitoring devices on the degradation of assets. The central question we consider is *how to develop algorithms that can provide accurate RUL predictions directly from remote monitoring data?* Based on this central question, we focus on two critical prognostics problems and their related questions, namely:

RQ1 *How to effectively achieve accurate RUL predictions that can retain long term information about degradation data?*

RQ2 *How to adapt RUL predictions from previous run-to-failure data to new assets with only sensor information under different operating conditions and fault modes?*

This thesis then focuses on these two research questions and studies data-driven RUL prediction algorithms in chapters 3 and 4.

1.3 Route Scheduling

In practice, industrial assets are usually present at multiple locations simultaneously, constituting a network of assets. When a central logistics operator is responsible for multiple assets at different locations, it is crucial to factor in other decisions, such as travelling to specific locations to repair or inspect assets. The objective is to ensure that personnel, parts and supplies arrive at their destinations efficiently at minimum costs. Failing to do so can result in increased transportation time, delays and overall maintenance costs.

Due to the impact that travelling can have on devising efficient plans and reducing costs [68], *routing* problems have been previously studied in the OR literature and practical applications. Most routing problems aim at serving transportation requests of customers in a road network while minimising the total service costs and respecting one or several constraints. These optimisation problems usually utilise

the notation of *graphs*. These offer the necessary components to define the locations in a network as nodes and the connections between nodes as (un-)directed edges.

There exist multiple variants of routing problems fitting different situations. We focus our attention on general problems that can easily be extended to multiple applications domains. That is, we study classical routing problems encompassing our application domain when assets are assumed to need visitation, ensuring minimum travelling costs. The attention on the general problems allows us to focus on methods that can be employed in other application domains. In the following subsections, we briefly introduce two classical definitions of routing problems extensively studied in the OR literature, namely the travelling salesman problem (TSP) and the vehicle routing problem (VRP).

1.3.1 Travelling Salesman Problem

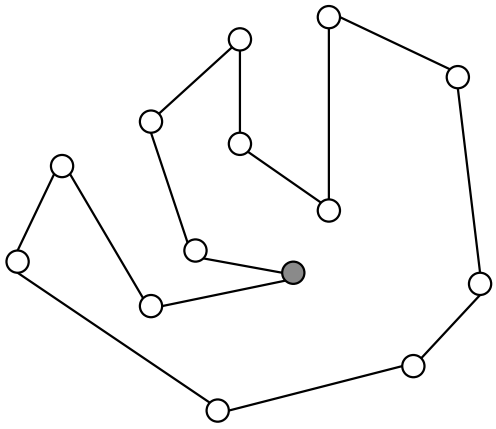


Figure 1.1: A travelling salesman problem (TSP) solution as a graph. In the TSP, a single route with the minimum travelling costs is sought.

The TSP is a well-known problem in which customer nodes in a network need to be visited exactly once by a single "salesman" ensuring the lowest travelling costs. The TSP is probably the most well-studied problem in CO, and it is commonly used to benchmark more general optimisation algorithms in OR. The problem is known to be NP-hard [76] and the attempts to solve it have resulted in many theoretical and algorithmic results in other areas of CO [134] (see Figure 1.1).

Given its broad applicability and relevance in multiple domains, several variants of the classic TSP problem exist. These include formulations with different added constraints, such as time windows, asymmetric edge costs, and tour budgets. A particular formulation of interest is when routing problems include more than one salesman in the problem formulation. This multiple TSP (mTSP) considers that

more than one salesman is available to visit nodes. Constraints can be added to break tours between the salesmen and restrain the number of nodes being visited by each salesman. An even more general formulation that includes the TSP and mTSP as particular cases is the VRP.

1.3.2 Vehicle Routing Problem

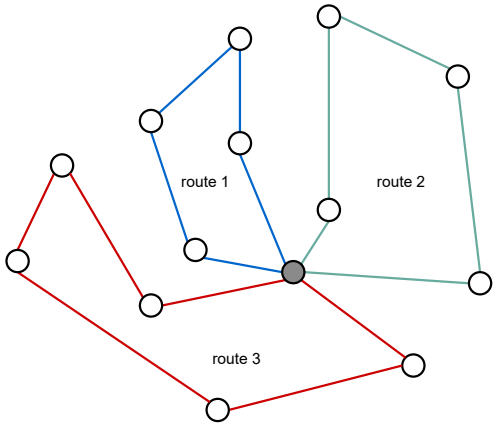


Figure 1.2: A vehicle routing problem (VRP) solution as a graph. In the VRP, the goal is to minimise the travel costs of all routes while respecting operational constraints.

The VRP [56] is a central problem in OR [133]. In its most basic formulation, i.e., the Capacitated VRP, it assumes the existence of a set of vehicles, each with a maximum capacity and the existence of nodes to be served, each with its demand. The goal of the VRP is to find the set of routes starting and ending at an initial node, named the depot, that minimises the total travelling costs of serving each node in the network exactly once without the collected customer demand of each route exceeding the total capacity of each vehicle (see Figure 1.2).

Following from its simplest form, i.e., the TSP (one vehicle with infinite capacity), the VRP is also an NP-hard problem. Moreover, like the TSP, the VRP can accept multiple variations, for example, requiring a specific vehicle type to serve certain locations due to specific customer demands, heterogeneous vehicles, and others. For a more thorough review of VRP and TSP problems, see [69, 134, 212].

1.3.3 Solving Routing Problems

The economic importance and general academic interest in CO and routing problems such as the TSP and VRP have led to many solutions. Recent successes in

solving these problems are multifaceted and can be attributed to new mathematical methods and advances in computing power. We focus our attention on algorithmic improvements, which can be broadly classified into exact methods, and approximation algorithms and metaheuristics.

Exact methods aim at finding optimal solutions via mathematical optimisation. Usually, exact algorithms for the TSP and VRP are based on mixed integer linear programming (MILP) formulations. However, other methods, such as dynamic programming (DP) [20] and constraint programming, are also suitable. While many advances in exact methods have been achieved in previous literature, exact methods are seldom utilised in practical applications given the extensive amount of computational time required to solve large instances of routing problems.

In most practical scenarios, metaheuristics are chosen as solution methods for solving CO problems. Metaheuristics implement high-level expert-based *policies* aimed at optimising the objective of a CO problem. In general, metaheuristics are not guaranteed to find the optimal solution at the end of the computation; however, they can result in sufficiently good solutions without requiring the amount of computation of exact methods. The general working of metaheuristics involves searching through a subset of candidate solutions of the optimisation problem. Although efficient and effective heuristics exist, metaheuristics often work well for specific classes of instances but may fail to generalise to instances outside of development.

In general, solving such complex problems either takes too long if optimality is the main goal or can be too far from the optimal solution if the heuristic is not well-tailored for the instances of interest. Critical to both exact methods and metaheuristics approaches are algorithmic decisions that depend on expert knowledge about the optimisation algorithm and the problem at hand. However, these decisions may be sub-optimal and fail to explore the desired search regions appropriately. For example, the branch-and-bound algorithm [136] used to solve MILP formulations requires many heuristic decisions, such as picking the next variable to branch. Moreover, replacing current algorithms usually involves designing new heuristics, a process that is time-consuming and requires specific knowledge. This knowledge may not be available, sufficient, or some of these design decisions may be unsatisfactory [24].

Furthermore, previous solvers may not take into account the regularity and the data distributions of problem instances. That is, a method that works well for specific instances may not provide the same performance in different instances of the same problem family. For example, solvers for transportation problems in a road network in *Hong Kong* may work poorly in problems that consider the road network of *Istanbul*. Moreover, when considering stochastic problems, i.e., where some of the variables of the problem are random variables, most previous methods make assumptions about the distribution of random variables without considering the actual sampled data. Thus, methods that fail to adapt to the characteristics

of instances and do not consider the observed data of the real problem can have reduced performance, especially if these instances are different and follow different distributions from those used during development.

1.3.4 Problem Statement

In this thesis, we consider the problem of automatically learning policies to solve routing problems. We assume that instance data containing location information is available and evaluate data-driven algorithms that account for statistical similarities of the instances, resulting in adaptive solvers. The main objective is to explore the space of heuristic decisions and learn from experience the best performing policies, potentially improving the current state-of-the-art of exact solvers and metaheuristics [24]. As typical in many CO problems, many heuristics containing expert knowledge about the problem structure exist and can achieve good performance requiring less computation than exact methods. Thus, we extend the original goal of learning adaptive solvers while taking advantage of previous heuristics to accelerate learning and reuse expert information embedded in expert-designed methods. In particular, we focus on improvement heuristics based on edge swap operators commonly employed in local search heuristics for routing problems.

Research Question

We consider the primary problem of improving heuristics solvers and effectively learning to solve routing problems directly from instance data. To address this problem, we focus on the following two main research questions:

RQ3 *How to learn better heuristics to solve routing problems directly from observed data of previous instances?*

RQ4 *How to take advantage of expert heuristics to learn to solve routing problems?*

We study methods to address these questions in chapters 5 and 6, respectively.

1.4 Combining Routing and Maintenance Decisions

Learning to solve routing problems is an important task of the more general problem of serving a network of assets that possibly degrade over time. When considering the existence of multiple assets at different locations, the problem of jointly planning maintenance and travelling schedules for a network of assets becomes apparent. Naturally, including the uncertainty of failures combined with maintenance and travelling decisions increases the complexity compared to the disjoint problems.

Several previous works for maintenance policy optimisation have focused on multi-asset problems without considering travelling implications [59, 171]. In most cases, they assume that assets have a series (un-)observable states representing their health conditions. Uncertainty in the failure times is considered via asset state transitions from healthy to failure states following stochastic processes, typically with the Markovian property, i.e., assuming that future degradation states' conditional probability distribution depends only on the current state. Under such assumptions, DP can be used to find optimal policies for reasonably sized state spaces.

Finding the lowest cost routes for maintenance applications have been modelled as variations of the TSP. For example, in the travelling maintainer problem (TMP), the objective is to find a route that visits assets that minimises the mean waiting times of machines [1]. A variant to the TMP studied the objective of minimising the sum of functions of response times (latency) to degrading assets [31]. This TMP variant integrates CBM prognostics with the TSP, where the predicted failure information represents a non-linear effect on the latencies.

Similarly, in recent practical applications, decision-makers are often posed with problems that involve analysing the data from multiple CBM prognostics models and deciding on the maintenance plans to best serve a network of assets. For example, hospital systems equipped with remote monitoring devices can employ prognostics models to predict the remaining life of assets. When these assets are located in different geographical locations, central operators are posed with a list of predicted failure moments to be reacted. The optimisation of maintenance policies in this scenario can lead to saved costs and improved reliability of critical assets. However, performing too much maintenance can lead to unnecessary operational costs, and failing to maintain degrading machines can lead to catastrophic failures and reduced service levels.

Thus, we consider the recent shift to use CBM prognostics and focus on problems that use health information from predictive models to plan maintenance policies. Ideally, we want to ensure that assets are maintained just before failure to ensure the highest availability at minimum costs. However, two main problems arise: an asset's failure mechanism is unknown, and assets are often part of a more extensive network of similar assets. In these cases, even when if we could somehow directly observe the health states of assets and estimate transition probabilities under the Markov property, DP methods quickly become intractable due to the curse of dimensionality [207]. That is, for large networks, it becomes intractable to solve problems to optimality under reasonable computational time. Moreover, most previous combined maintenance and routing problems have focused on penalties to the objective function that may not consider the costs of delaying and postponing maintenance in the presence of real-time data. Additionally, most proposed solutions are based on exact methods or metaheuristics, requiring expert knowledge and not necessarily considering the problem's observed data.

1.4.1 Problem Statement

We study a problem where maintenance costs, i.e., costs incurred due to repairs, are asset-dependent and increase if the decision-maker does not perform maintenance before its failure time. The failures are triggered by the assets' degradation processes, unknown and unavailable to decision-makers. Furthermore, we consider that predictions from CBM prognostics are available and represent an indication of the health conditions of assets adapting in real-time.

In practice, such as the maintenance of wind-turbine farms [120], these predictions are based on remote monitoring data and are communicated to decision-makers in the form of *alerts*, which serve as an early indication of future failures and carry censored information about the actual degradation of assets, i.e., some indication of the RUL of assets. However, alerts often carry uncertain information about the failure time due to misreadings, sensor malfunction and prediction errors.

If a predictive model indicates that a failure happens sooner than it will happen, decisions considering this information may incur unnecessary preventive maintenance costs. On the other hand, if predictions indicate that failure will happen after the real failure time, higher corrective maintenance costs and increased downtime will be incurred if the predictions are followed blindly. Ideally, if one could predict the exact moment in which failures will happen, then maintenance policies would only need to focus on scheduling the visitation of assets just prior to their failure to ensure minimum costs. In reality, there is uncertainty in the failure times, and even an accurate prediction model fails to predict the real failure time.

Thus, this thesis considers a sequential decision-making problem requiring joint travel and repair decisions in a network. In this problem, we assume the existence of alerts from prognostics models referring to the predicted RUL of assets. Additionally, we assume that this network needs to be served by the capacity of a single repairperson responsible for travelling and repairing actions to minimise the overall maintenance costs. In this setup, we consider that the degradation of assets is stochastic, and the time that failures will happen is uncertain. Thus, our goal is to learn a policy directly from available locations and alert data to serve the network and ensure that maintenance is performed just-in-time to avoid high preventive and corrective maintenance costs.

Research Question

We consider the primary problem of learning policies to solve the decision-making problem presented above directly from observed data, and we study the following research question:

RQ5 *How to automatically learn policies to solve a dynamic routing and maintenance problem in an asset network directly from observed data?*

In Chapter 7, we present the proposed methods to address this question.

1.5 Contributions of this Thesis

We split our study into three parts, i.e., prognostics, routing, and combined maintenance and routing decision-making, representing relevant problems that arise when travelling and serving a network of industrial assets. Sections 1.2, 1.3, and 1.4 provided background information on these problems and their relevance to our application domain. Considering the available data and objectives, we propose learning algorithms based on the previous ML theory on *supervised learning*, *transfer learning*, *reinforcement learning* and *imitation learning*. The methods comprised in this thesis aim at addressing the research questions posed in the previous sections. Our main contributions are detailed below.

1.5.1 Learning Asset Prognostics

Asset Prognostics from Sequential Data

Accurate real-time RUL predictions enable equipment health assessment and maintenance planning. In particular, ML methods based on neural networks have received much attention given their ability to approximate functions directly from raw data. In Chapter 3, we propose a deep learning model based on a long short-term memory (LSTM) network [97] that can handle the temporal dependencies of sequential data and provide insights on the learned relationships between sensor reading inputs and predicted RUL for single assets. In this supervised learning setup, we assume that previous labelled run-to-failure information exists, and our objective is to learn to approximate the RUL of unseen data during training of the learning algorithm. Our proposed method can produce accurate RUL predictions while not requiring knowledge of the actual physical degradation mechanism of assets. Compared to previous works, we do not incur additional pretraining or extensive optimisation of the model's hyperparameters. Additionally, a trained model can be used to visualise temporal relationships between inputs and predicted outputs providing managerial insights between the sensor inputs and RUL outputs.

Assets Prognostics under Varying Operating Conditions

When concerned with ML methods for asset prognostics, observed sensor data and RUL information are usually critical for supervised learning-based prognostics. Most previous methods assume that training data is available and labelled. However, due to different operating conditions, fault modes and noise, distribution and feature shift exist across different data sources (domains). Additionally, a new generation system might exist without enough operational data and run-to-failure in-

formation, but that still require failure prognostics. These discrepancies can reduce the performance of predictive models when no observed run-to-failure data is available or prevent the use of previous models when new asset versions contain different sensors and operate in different conditions. Chapter 4 proposes a transfer learning method for domain adaptation in prognostics employing an LSTM architecture. We assume that only sensor information is available for new assets and propose a method that leverages the data coming from different distributions to learn RUL predictions. Our method is proposed so that it can automatically learn feature spaces that attempt to reduce the discrepancy between different data domains. We show that our method can adapt remaining useful life estimates from labelled data with different operating conditions and fault modes to data containing only sensor information and no observed RUL labels.

1.5.2 Learning to Route

Learning Improvement Policies

When the goal is to find a suitable solution for routing problems such as the TSP, a common heuristic approach is to search for improving solutions in the neighbourhood of previously generated solutions. However, such approaches depend on expert knowledge to build successful policies that effectively search the solution space. ML methods can potentially use data to learn better policies reducing human interventions. Recent works using deep learning to solve CO problems have focused on learning construction heuristics, but they still require search procedures to reach good quality solutions. Chapter 5 proposes learning a local search heuristic based on two edge swap operators, namely 2-opt, via (deep) reinforcement learning, requiring no supervision. We propose a policy gradient algorithm to learn a stochastic policy that selects 2-opt operations given a current solution. Moreover, we introduce a neural network architecture to encode instance and sequence features that can be extended to more general k-opt moves. Learned policies can improve over random initial solutions and approach near-optimal solutions faster than previous state-of-the-art deep learning methods for the TSP and on par with heuristics and learning methods for the mTSP and the VRP.

Learning Improvement Policies Imitating Expert Heuristics

In Chapter 6, we propose a framework that can leverage data from previously handcrafted heuristics for the TSP. The goal is to accelerate learning and take advantage of previous expert knowledge in handcrafted heuristics. In this setting, the method first attempts at imitating previous demonstrations, i.e., from handcrafted 2-opt improvement policies. We learn policies that can surpass the quality of the demonstrations while requiring fewer samples than pure reinforcement learning. This study proposes to first learn policies with imitation learning, leveraging a small set

of demonstration data to accelerate policy learning. Afterwards, we combine policy and value approximation updates from reinforcement learning to improve the expert's performance. We show that our method learns good policies in a shorter time and uses less data than learning policies from scratch.

1.5.3 Learning to Maintain and Route

We finally consider a combined problem of maintenance and routing decisions. We assume that modern industrial assets are part of a network of assets. These assets are expected to experience minimal downtime and, ideally, are maintained right before failure to ensure maximum availability at minimum maintenance costs. As in practical scenarios, failure times of assets are unknown *a priori*. Moreover, we assume that assets are equipped with CBM prognostics models are employed to emit alerts, typically triggered by the first signs of degradation. Thus, it is crucial to plan maintenance considering the information received via alerts, asset locations and maintenance costs. This problem is referred to as the dynamic travelling maintainer problem with alerts (DTMPA). Chapter 7 proposes a modelling framework for the DTMPA, where the alerts are early and imperfect indicators of failures. The objective is to minimise discounted maintenance costs accrued over an infinite time horizon. Moreover, we propose a deep reinforcement learning method to solve this problem based on value approximation methods trained to minimise the long-term costs using the observations from the network and history of alerts. The proposed method can find effective policies for large asset networks, where computing the optimal policy is intractable and outperforms baseline greedy heuristics that rank assets based on proximity, urgency and economic risk and a TMP heuristic leveraging classical OR methods to optimise near-future costs.

1.6 Thesis Outline

The general theme of this thesis is the study of methods and problems common to ML and OR literature from a data-driven perspective. This thesis is written assuming that readers already have some familiarity with ML and OR problems. The overall view throughout this thesis is that, in some problems, expert knowledge can be automated and augmented with algorithms learning from experience and data. Each chapter of this thesis introduces the relevant problem, its formalisation and presents the proposed algorithms followed by experimental results. The chapters are based on papers published in or submitted to peer-reviewed journals and conferences in ML and OR venues. As a result, chapters can be read in any desired order, and each chapter has its notation independent from the other chapters. The remaining chapters of this thesis follow this general theme and are organised as follows.

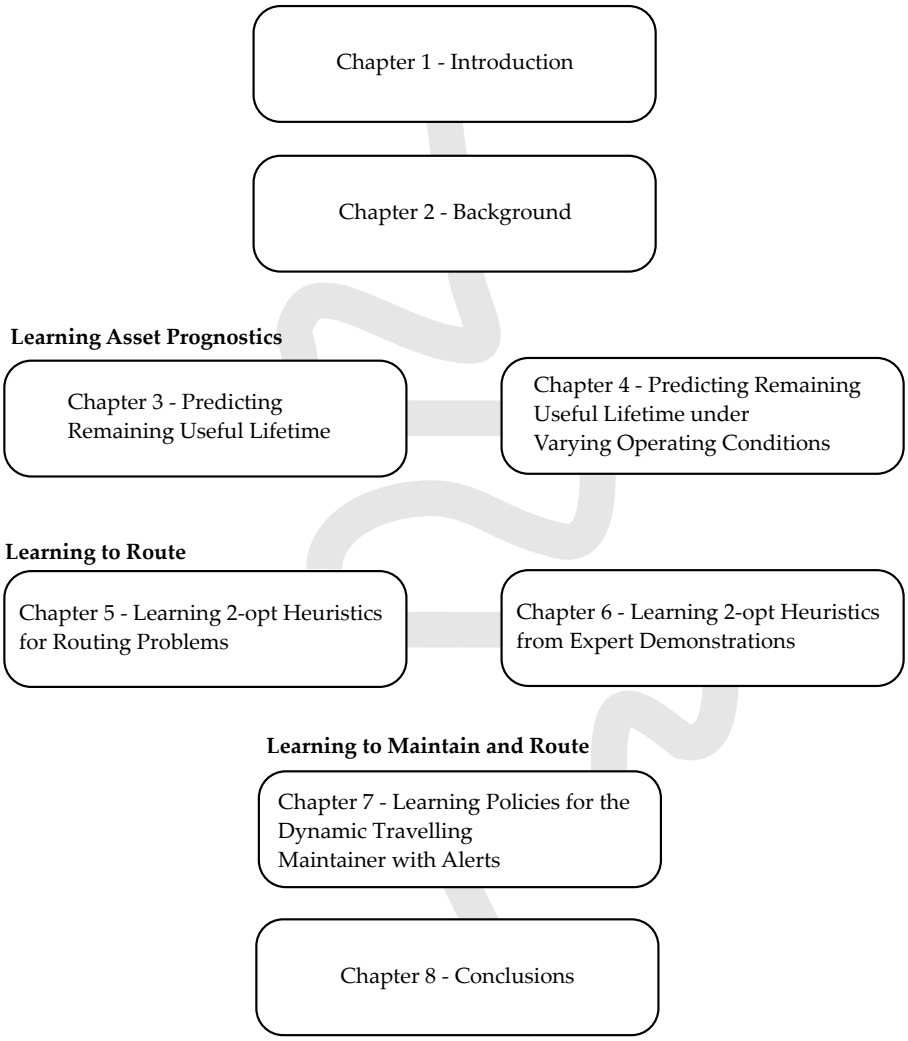


Figure 1.3: Thesis outline.

We provide a brief background and references to the ML concepts explored in this thesis in Chapter 2. Specifically, we provide background on deep learning and introduce the different learning tasks considered in the following chapters, namely supervised, transfer, reinforcement and imitation learning. The following chapters present the algorithmic methods for the problems detailed in the previous sections. Particularly, Chapter 3 and Chapter 4 focus on *learning asset prognostics* problems. Chapter 5 and 6 focus on *learning to route* and Chapter 7 focuses on *learning to maintain and route* problems. Lastly, Chapter 8 concludes the main findings of this

thesis, providing a holistic view of the main contributions of the presented work, discussing limitations and possible future research directions. Additionally, we put our contributions in a broader context, considering other application domains and previous ML and OR literature. The overall organisation of the thesis is depicted in Figure 1.3.

Chapter 2

Background

This chapter introduces the relevant background and in-depth references of the machine learning topics explored throughout this thesis. In this thesis, we consider prediction and control problems that assume different data availability and learning tasks. As presented in Figure 2.1, Chapter 3 and Chapter 4 study prediction problems based on *supervised learning* and *transfer learning* literature. Chapters 5, 6 and 7 study control problems based on *reinforcement learning* and *imitation learning*. Common to all proposed methods are *deep learning* architectures aimed at approximating functions for these machine learning tasks. Therefore, we present a review of the important background information of deep learning and the relevant machine learning tasks studied in this thesis. Those already familiar with these topics can skip this chapter entirely or several sections without affecting the remaining chapters.

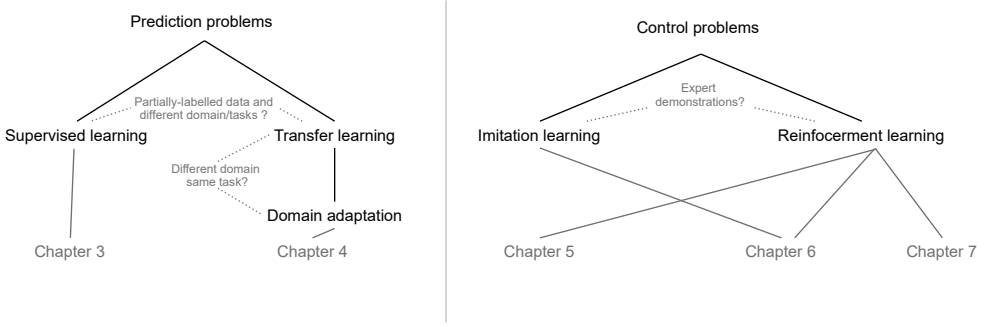


Figure 2.1: An overview of the type of problems and relevant machine learning literature studied in this thesis.

The remainder of this chapter is organised as follows, in Section 2.1 we introduce neural networks and deep learning concepts, including several neural architectures

in Section 2.2. In Section 2.3 we introduce the machine learning tasks considered in the problems of this thesis. In particular, we discuss the relevant tasks within the literature of supervised learning in Section 2.3.1, transfer learning in Section 2.3.2, reinforcement learning in Section 2.3.3 and imitation learning in Section 2.3.4.

2.1 Deep Learning

Deep learning is concerned with methods based on artificial neural networks. Neural networks are machine learning (ML) methods initially inspired by an abstraction of how information is collected and distributed in biological brains [185]. A classical neural network, also called a feed-forward neural network or multilayer perceptron (MLP), is a function $g_\theta(x)$ parameterised by learnable parameters θ trained to approximate a function between input and output vectors x and y, say $y = g^*(x)$. MLPs are called networks because they are usually represented as a directed acyclic graph with information flowing from inputs to outputs [80]. Neural

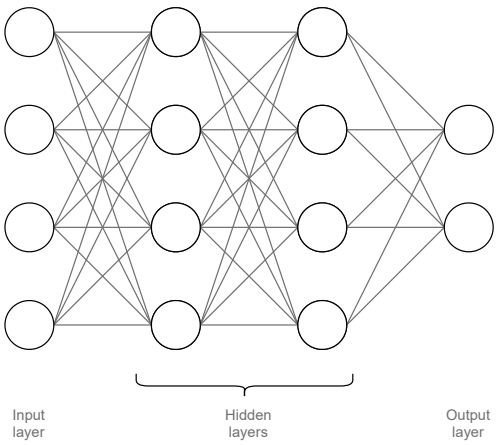


Figure 2.2: A feed-forward neural network with two hidden layers. Neurons and weights are nodes and edges in the graph, respectively.

networks are composed of a collection of individual units, named artificial neurons (nodes in the graph), responsible for simple (non-)linear computations from its inputs. Neurons receive information from previous neurons stacked in layers. The first layer (usually represented as the leftmost one) is the input layer, as it represents the inputs x. Layers following the first layer take as input the outputs of previous layers in a connected chain of functions over a number of layers L, i.e.,

$$g_\theta(x) = \sigma^{(L)}(\sigma^{(L-1)}(\dots\sigma^{(1)}(x))).$$

A neural network represented in this manner has the layers $1, \ldots, L-1$ referred to as hidden layers, and layer L as the output layer (see Figure 2.2). Neurons are responsible for computing outputs, via activation functions, taking the information from previous layers and trainable *weights* and *biases* (edges in the graph). It is common for neurons in the same layer to share the same activation function. In this case, we can represent the inputs to layer l coming from activations in the previous layer as $x^{(l-1)} := \sigma^{(l-1)}(\ldots\sigma^{(1)}(x))$, where $x^{(l-1)} \in \mathbb{R}^{d^{(l-1)}}$, $d^{(l-1)}$ represents the number of neurons (dimension) of layer $l-1$, and $x^{(0)} := x$. The output of layer l is computed as

$$x^{(l)} = \sigma^{(l)}(W^{(l)}x^{(l-1)} + b^{(l)})$$

where $W^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l-1)}}$ and $b^{(l)} \in \mathbb{R}^{d^l}$ are trainable weights and biases of the layer, and $\theta := \{W^{(l)}, b^{(l)}\}_{l=1}^{L}$ are the trainable parameters of the entire network. Activation functions are usually element-wise non-linear functions of their inputs. Well-known activation functions include the *sigmoid* function, and the *rectified linear unit* (ReLU) function [79]. A neural network in this form can be seen as the concatenation of multiple non-linear transformations of the activations in the previous layers making itself a nonlinear function.

To learn weights, we need to define a metric that measures the error between the outputs of the neural network and the expected outputs of the function we would like to learn. In a supervised learning setting, we assume access to a dataset $\mathsf{D} := \{(x_i, y_i)\}_{i=1}^{N}$, sampled from an unknown distribution $P(x, y)$, and a learning algorithm seeks to learn a function $g : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X}$ is the input space, $\mathcal{Y}$ is the output space and g belongs to a hypothesis class $\mathcal{G}$ of functions. Depending on the problem, we can have a classification problem when labels y correspond to discrete classes or regression when y are real numbers. A loss (cost) function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ measures the error between the model and outputs, and the overall objective is to find parameters θ that minimise the population risk $\mathcal{L}(g) = \mathbb{E}_{x,y\sim P(x,y)}[\ell(y, g(x))]$, usually approximated via the the empirical risk, expressed as

$$\hat{\mathcal{L}}_{\mathsf{D}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, g_\theta(x_i)).$$

It is common that we seek the parameter (model) approximation of $P(y|x;\theta)$ [166], thus $\ell(y, g_\theta(x)) = -\log P(y|x;\theta)$ reduces to the negative log-likelihood. Particular cases of the equation above include the squared error in regression, i.e., $\ell(y, g_\theta(x)) = (y - g_\theta(x))^2$ and the multiclass cross-entropy function $\ell(y, g_\theta(x)) = -\sum_{c=1}^{K} y^{(c)} \log(g_\theta(x)^{(c)})$ for classification problems with K classes, where $y_i^{(c)}$ corresponds to the dimension c of a one-hot encoded vector with dimension K. The equation above is often augmented with regularisation terms to control overfitting and provide better generalisation to unseen data. The ML objective is to minimise the generalisation gap, usually estimated by the gap associated with the empirical

risk between the data used for training, i.e., the data used to optimise the equation above, and the empirical risk when evaluating the model on unseen (test) data.

The nonlinearity of neural networks typically implies a non-convex optimisation problem [80]. Neural networks are usually trained with *stochastic gradient descent* methods that take incremental steps towards the minimisation of the risk, evaluated using samples D_k from D following

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta \hat{\mathcal{L}}_{\mathsf{D}_k}(\theta)|_{\theta=\theta_k}$$

where $|\mathsf{D}_k| \ll N$, α_k is a learning rate, and $k = 0, 1, 2, \ldots, K$, with K typically large. Stochastic gradient descent methods applied to non-convex functions have no convergence guarantees, are sensitive to initial parameters and can get stuck in local optima. Several gradient descent improvements have been proposed that can circumvent some of these issues. A thorough examination of these methods can be found in textbooks [80, 239] and surveys [188]. In most cases, the backpropagation algorithm [189] is the procedure utilised to compute the gradients of the loss w.r.t. to the learnable parameters. The algorithm is based on the repeated application of the chain rule of calculus to compute the gradient of the loss function w.r.t. the parameters in each layer sequentially. For a detailed description of the algorithm, see [80, 166]. Since this computation usually starts from the output layer and moves backwards in the network, this is often called the backward pass. In contrast, computing the outputs and the loss given the inputs is referred to as the forward pass.

An MLP with at least one hidden layer is known to be a universal function approximator [48, 100], i.e., regardless of the function we want to learn, an MLP can represent this function to arbitrarily small errors, provided it is given "enough" hidden units. However, it is not guaranteed that the training algorithm will be able to learn that function; for example, the optimisation algorithm may fail to learn the parameters of the function we want to approximate [80]. Moreover, it is not clear how large this hidden layer must be to represent this function. In many cases, using deeper (stacking more layers) networks can potentially reduce the number of parameters necessary to learn functions and lead to better generalisation. In particular, the regained interest in neural networks came from the study of the depth of these methods [129], and the strengthening of their inductive bias [161]. Various previous works, both from an experimental and theoretical perspective, have shown that deep neural networks can outperform shallow ones [165, 179, 182].

One of the reasons for the success of deep learning is attributed to learning the representation of data necessary to map inputs to outputs in a compositional and hierarchical way. That is, deep neural networks build more complex representations based on simpler learned representations in the previous layers. Thus, a deep learning model can both approximate the function from input to output and learn the necessary representation to build this approximation in a process akin to representation learning [80]. Learning this representation often results in much

better performance than those obtained with human-designed representations as in classical ML [80].

The training of neural networks is often posed with many design choices such as the activation functions, number of layers, number of neurons in each layer, optimisation algorithm [124], regularisation parameters [201], initialisation of weights and biases [78], to name a few. Recently, several advances have been made to each of these algorithmic choices leading to higher performance and faster training of deeper methods. These recent advances become especially apparent under the availability of large amounts of unstructured, complex and high dimensional data. In most of these cases, devising human-designed features is either hard or impractical. For this class of problems, deep learning has shown incredible success in learning tasks that were otherwise impractical using classic ML.

2.2 Neural Network Architectures

In this thesis, we study several problems in which large amounts of unstructured data is the norm. To take advantage of this data, we propose several neural network architectures to extract and learn features directly from raw data while solving prediction and decision-making (control) tasks. This form of strengthening the inductive bias, i.e., the assumptions embedded in the learning algorithm, of models has led to many different architectures to better take advantage of the input data regularity, reducing the sample complexity and achieving better generalisation performance [16]. Reviewing all possible architecture advances is not the scope of this chapter. Instead, we focus on the architectures studied in the problems considered in this thesis, namely, recurrent neural networks, attention methods and graph neural networks.

Recurrent Neural Networks

A recurrent neural network (RNN) [189] is a neural network that accepts sequential data as input. In general, sequential data is not independently identically distributed (i.i.d.). That is, data generated sequentially at a given point in time is typically correlated to data generated in the past. This requires particular care in taking into account the previous temporal information contained in sequences. For example, if we would like to predict the next word in a sentence, it is usually the case that we need to take into account a long sequence of words that came before the word we would like to predict.

An RNN can map input sequences to output sequences by maintaining a series of hidden states capable of retaining temporal information and ordering about the sequences. RNNs have been employed for multiple tasks, such as sequence generation, classification, and natural language processing (NLP) tasks, for example, in machine translation, i.e., translating a sentence from one language to another. The

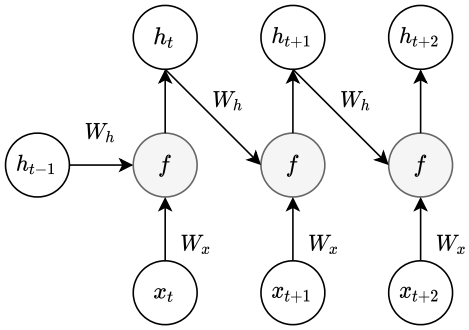


Figure 2.3: A recurrent neural network unfolded over three time steps. The recurrent network takes as input at each time step x_t and the previous hidden state h_{t-1} and computes a new hidden state h_t.

general working of an RNN is as follows, an RNN takes as input a vector x_t at time t and maintains a hidden vector h_{t-1}, representing a summary of the information received up until time $t-1$. Thus, an RNN model aimed to predict the next element of a sequence approximates $P(x_t|x_{t-1},\ldots,x_1)$ with a latent variable model $P(x_t|h_{t-1})$ where h_{t-1} depends on the past history. An RNN (see Figure 2.3) is a function defined as

$$h_t = f(h_{t-1}, x_t).$$

Classically, f has been defined as the hyperbolic tangent function (tanh) in which case the previous equation can be written as

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b)$$

where h_0 is usually a vector of zeros and $b \in \mathbb{R}^d$ is a bias term, $W_h \in \mathbb{R}^{d\times d}$ and $W_x \in \mathbb{R}^{d\times p}$ are trainable weights for previous hidden vectors $h_{t-1} \in \mathbb{R}^d$ and current input $x_t \in \mathbb{R}^p$, respectively. Unfolding the recurrence of the equations above shows that the hidden vector at time t depends on the entire past sequence by repeatedly calling the function f. The definition of the computation in this manner allows the RNN function to take always the same input size, regardless of the sequence length [80]. Moreover, the independence of the learned weights and biases from t makes the number of parameters independent of the length of the input (parameter sharing) and enables generalisation to sequences with varying lengths. Typically, RNN implementations will add further computations such as output layers responsible for mapping the hidden vectors to outputs at each time step.

In many cases, we are interested in generating another sequence from an original sequence given as input, such as in a machine translation task. In this sequence-to-sequence task, computing the loss of an RNN, i.e., the error between the generated and desired output sequence, requires aggregating the loss of each time step t in

the sequence. For example, assuming independence of previous outputs, the loss ℓ for a training example can be expressed as the sum of the individual losses at each time step ℓ_t given the input sequence $x_1, \ldots, x_t$, expressed as

$$\ell(y_t, \ldots, y_1, g_\theta(x_t, \ldots, x_1)) = \sum_{t'=1}^{t} \ell_{t'}(y_{t'}, \hat{y}_{t'})$$

where $\hat{y}_t = g_\theta(x_t, \ldots, x_1)$ is the prediction of the RNN at time t. Similar to the MLP case, computing $\ell_t(y_t, \hat{y}_t)$ when outputs y_t correspond to discrete classes can be achieved via a suitable classification loss such as the cross-entropy. For example, if y_t corresponds to words in a dictionary, we can use a classification loss function to predict the correct sequence of words from this dictionary.

In a neural network (especially in RNNs), gradients propagated over many multiplicative terms tend to vanish if they become too small or explode if they become too large. Gated architectures form a family of RNN architectures aimed at capturing long-term dependencies and reducing gradient vanishing and exploding issues. Well-known implementations include the long short-term memory (LSTM) [97] and the gated recurrent unit (GRU) [42]. These use a series of learned information gates and self-loops that allow important information to be retained in the hidden states within the RNN "cells", creating paths through time that have derivatives that do not vanish nor explode [80]. More importantly, gated architectures have shown better results in tasks that require retaining memory for long sequences [206].

Attention

Attention-based neural networks were initially introduced in RNNs for sequence-to-sequence tasks in NLP, such as machine translation [13]. After reading an input sequence, attention methods allowed for a time-shifted output sequence to refer directly to parts of the input sequence. That is, it allowed neural network modules to learn to focus on important input parts when generating (decoding) a sequence. For example, when learning to translate a word from English to French, we may be interested in specific parts of the input sentence in English that will help us translate to a word in French. Note that this is different from the vanilla RNN case, in which generating a new sequence would only use the aggregated information at the end of the input sequence of size T, i.e., h_T.

Later, self-attention [41], positional embeddings and multi-head attention were introduced in [220]. These techniques combined allowed any position of the input sequence to have access to multiple representations of the entire input sequence (context) and the relative positions of each input element. These new embeddings were followed by another attention mechanism between the learned embeddings and outputs. This architecture made it possible to replace the recurrence of RNNs in the inputs, allowing for better parallelisation of sequence models, i.e., replacing

the necessary sequential computations of RNNs.

In general, an attention function is an alignment score function between (learned) vectors. We can roughly map this function to a querying operation in which we would like to discover how related a given query is to a set of keys and value pairs. Mathematically, consider a query vector $q \in \mathbb{R}^d$ and m key-value pairs $\{(k_1, v_1), \ldots, ((k_m, v_m)\}$, where $k_i \in \mathbb{R}^n$, and $v_i \in \mathbb{R}^o$. An attention function is a learnable function $a(q, k_i)$ where a can take many forms such as additive attention [13], dot-product attention [154], scaled-dot product attention [220] among others. For example, assuming $d = n$ and the scaled dot-product case, the function above can be expressed as

$$a(q, k_i) = \frac{q^\top k_i}{\sqrt{d}}$$

where $\sqrt{d}$ is introduced to normalise the variance of the dot-product assuming that vectors q and k_i are composed of independent random variables with mean zero and unit variance. These attention scores (scalars) are then normalised and transformed into attention "weights" (not to be confused with neural network's trainable weights) using the softmax function as

$$\alpha_i = \text{softmax}(a(q, k_i)) = \frac{\exp(a(q, k_i))}{\sum_{j=1}^{m} \exp(a(q, k_j))}.$$

Finally, an attention pooling function f is defined as weighted sum of values v_i as

$$f(q, (k_1, v_1), \ldots, (k_m, v_m)) = \sum_{i=1}^{m} \alpha_i v_i$$

where α_i is the associated attention weight (scalar) between q and k_i.

Graph Neural Networks

A graph neural network (GNN) is a neural network that takes as inputs graphs $G := \{V, E\}$ where $V := \{v_i\}_{i=1}^N$ represents the set of n vertices v_i (nodes) in the graph and E the set of edges. Each edge $e_{ij} := (v_i, v_j) \in E$ is a pair of (un)directed connections between node v_i and v_j. The immediate neighbourhood of a node is defined as $\mathcal{N}(i) = \{v_j \in V | (v_j, v_i) \in E\}$. Normally, graphs carry information in the form of node and edge features, we represent the vector of node features for node v_i as $x_i \in \mathbb{R}^d$, and features for edge e_{ij} as $\bar{e}_{ij} \in \mathbb{R}^q$. In the architecture considered in this thesis, a GNN layer [16], also referred to as a message passing layer (see Figure 2.4), is a function responsible for learning and aggregating neighbouring information considering the topology of the graph. That is, a GNN layer l is responsible for

passing messages between connected nodes and edges represented as

$$x_i^{(l)} = f^{(l)}\left(x_i^{(l-1)}, \diamond_{j \in \mathcal{N}(i)} \phi^{(l)}\left(x_i^{(l-1)}, x_j^{(l-1)}, \bar{e}_{j,i}\right)\right),$$

where $\diamond$ denotes a permutation invariant (aggregation) function such as the sum, computed over the neighbourhood of node v_i; $\phi^{(l)}$ is a differentiable function that takes as input the node features from the neighbouring nodes of v_i as well as edge features, usually defined as an affine transformation of inputs followed by a non-linearity and $f^{(l)}$ is a (learnable) update function that takes as input the current node features from the previous and the representation from the node's neighbourhood to build a new representation of the node (and edge) features for the next layer. A GNN layer defined in this manner can be seen as an aggregating function

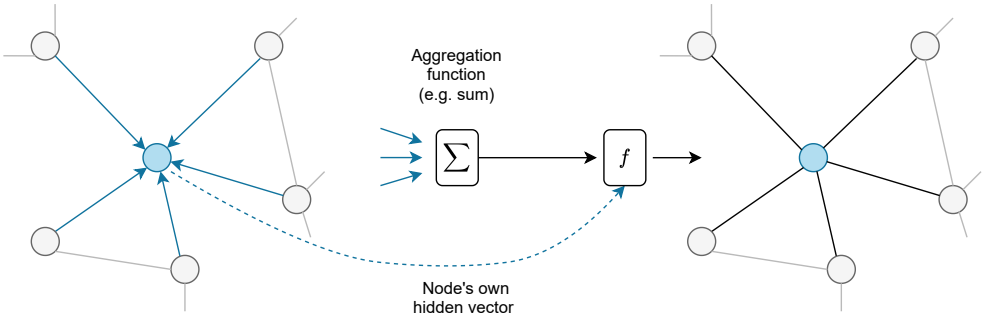


Figure 2.4: A message passing layer. The node representation for the next layer (right) aggregates information from nodes and edges on its neighbourhood as well as the previous node features (left).

that pulls information from the direct neighbourhood of a given node. This 1-hop aggregation can be extended to an l-hop aggregation by concatenating layers as $f^{(l)}(f^{(l-1)}(\ldots, f^{(2)}(f^{(1)}(\cdot))))$. Note that it is also possible to define a single GNN layer that aggregates information not only from its direct neighbours but from a larger neighbourhood. GNNs can also generalise other neural network architectures. For example, GNNs can be made equivalent to attention-based methods [220] by learning an attention function to aggregate information from neighbouring nodes [67]. A review about GNN architectures is out of the scope of this chapter but can be found in recent surveys [16, 233]. GNNs have been utilised in numerous applications in which the input data can be defined as graphs, including combinatorial optimisation problems.

2.3 Machine Learning Tasks

In ML, we are often posed with the task of learning directly from data and experience. Depending on the problem and the type of available data, different learning tasks can be defined. In this thesis, we study several learning tasks for prediction and control problems. Particularly, for prediction problems, we study supervised learning and transfer learning tasks, and for control problems, we study reinforcement learning and imitation learning tasks. We present a brief introduction of each studied ML task in the following sections.

2.3.1 Supervised Learning

Supervised learning is the most well-studied ML task. In supervised learning, and the task is to infer an otherwise unknown function $g^* : \mathcal{X} \to \mathcal{Y}$ from inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. The experience E is acquired in the form of N input-output training pairs $\mathsf{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$, known as the training set. Training is usually performed by minimising the empirical risk with a suitable loss function (see Section 2.1). A trained model should then be able to generalise this learned function to new examples not observed in the training set, i.e., have good approximated generalisation error on some test data $\mathsf{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^{M}$. The performance of a trained model is measured through this error. Common supervised learning tasks include classification and regression problems.

2.3.2 Transfer Learning

One of the assumptions of supervised learning is that training data and test data are sampled from identical distributions and feature set [174]. However, these assumptions may not hold in real-world cases where data may come from different data generating processes with different features. When these assumptions fail, previous models need to be rebuilt from scratch using newly collected data, a process that can be time-consuming and expensive.

In transfer learning, the goal is to learn to transfer knowledge present on previously acquired data (a domain) to another observed set of data and possibly different learning tasks. This general definition gives rise to many different categories of transfer learning settings by breaking one or more assumptions of classical supervised learning about the feature sets, the underlying distribution of data, the prediction problems, among others. Note that transfer learning has also been studied in reinforcement learning, for example, learning to adapt agents across different tasks. In this thesis, we focus on the case in which transfer learning is applied to classification and regression problems in a sub-category of transfer learning referred to as domain adaptation. An in-depth review of the different assumptions in transfer learning problems can be found in surveys [174, 227].

Unsupervised Domain Adaptation

We now present a general definition of the unsupervised domain adaptation problem. A domain $\mathcal{D} := \{\mathcal{X}, P(x)\}$ is defined by a feature space $\mathcal{X}$ and a marginal probability distribution $P(x)$, where $x \in \mathcal{X}$. A task $\mathcal{T}$, consists of two components, an output space $\mathcal{Y}$ and a predictive function g that we aim to learn, denoted as $\mathcal{T} := \{\mathcal{Y}, g\}$. As in supervised learning, we are usually concerned with learning the conditional probability $P(y|x)$, where $y \in \mathcal{Y}$. We consider the existence of a source domain $\mathcal{D}_S$ (the one we are transferring from) and a target domain $\mathcal{D}_T$ (the one we are transferring to) and the existence of data from the source domain, that is, $\mathsf{D}_S = \{(x_i, y_i)\}_{i=1}^{N_S}$ and data for the target domain $\mathsf{D}_T = \{(x_i, y_i)\}_{i=1}^{N_T}$ and corresponding learning tasks $\mathcal{T}_S$, $\mathcal{T}_T$. The general transfer learning task aims to improve the learning of the target function g_T for domain $\mathcal{D}_T$ and task $\mathcal{T}_T$ using knowledge from $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$.

In the unsupervised domain adaptation setting studied in this thesis, we assume that we have no access to outputs in the target domain, i.e., we have access to a dataset of the target domain $\mathsf{D}_T = \{x_i\}_{i=1}^{N_T}$ and one for the source domain $\mathsf{D}_S = \{(x_i, y_i)\}_{i=1}^{N_S}$. Furthermore, we assume that the marginal probabilities and the feature space are distinct, but the task we aim to learn is the same, i.e., $P_T(x) \neq P_S(x)$, $\mathcal{X}_S \neq \mathcal{X}_T$ and $\mathcal{T}_S = \mathcal{T}_T$. The goal is to minimise the risk of the target domain, measured via the empirical risk $\hat{\mathcal{L}}_T$. That is, we aim to find parameters θ that minimise

$$\hat{\mathcal{L}}_T(\theta) = \frac{1}{N_T} \sum_{i=1}^{N_T} \ell(y_i, g_\theta(x_i))$$

without having access to labels of the target domain y_i.

A major objective in more general domain adaptation is reducing the difference between the distributions of the source, and target domain data [174]. Intuitively, if we can find a feature representation in which the discrepancy between domains is reduced, we can use this representation for learning [23]. A feasible strategy to control the domain discrepancy is then to attempt to find a representation space at which source and target domain are as similar as possible while preserving the important statistical properties of the data, especially in the target domain [174].

Theoretical works showed that it is possible to bound the error risk of the target domain, i.e., $\mathcal{L}_T(\theta) = \mathbb{E}_{x,y\sim P_T(x,y)}\left[\ell(y, g_\theta(x))\right]$ by the risk of the source domain, i.e., $\mathcal{L}_S(\theta) = \mathbb{E}_{x,y\sim P_S(x,y)}\left[\ell(y, g_\theta(x))\right]$, adding a discrepancy between domains under the assumption that the ideal hypothesis leads to a small error on both domains, i.e., that the labelling function is similar across domains [22, 46, 156, 164]. Combining these ideas with neural networks approximation, as it is explored in Chapter 4, resulted in methods that aim at learning representation spaces for which the discrepancy between domains is minimised, for instance, using discrepancy metrics such as the maximum mean discrepancy (MMD) [198], $\mathcal{H}$-divergence [22, 123] and the discrepancy distance [156]. When employing such discrepancies resulted in ad-

versarial objectives [81], methods employing this idea define a minimax objective in which the goal is to minimise the source risk while learning to reduce the discrepancy across domains using as estimates to the target risk the empirical distributions of provided input data from target and source domains [60, 75].

2.3.3 Reinforcement Learning

Reinforcement learning (RL) is the sub-field of ML that aims to learn intelligent agents via observing the results of the interaction of such agents in an environment. A complete review of RL can be found in textbooks [27, 207] and surveys [116, 126]. This section provides a brief overview of the methods and terminology relevant to this thesis.

RL is concerned with problems in which data is generated sequentially by an agent acting in an environment and receiving as inputs the results of its actions in the form of observations and rewards (see Figure 2.5).

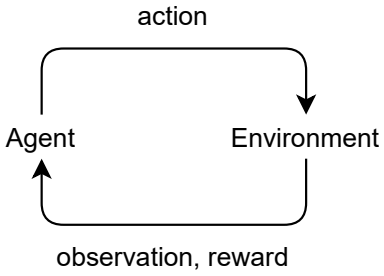


Figure 2.5: A reinforcement learning problem setup.

The main goal in RL is to learn a policy, such that the overall agent's utility, measured in the form of numerical rewards, is maximised [207]. However, unlike supervised learning, the agent is not told which actions to take by an external supervisor, and the observations depend on the agent's current policy. An RL method seeks to optimise the agent's behaviour considering the exploration (exploring actions not selected before) and exploitation (preferring actions found in the past that led to good outcomes) dilemma. The RL problem is extremely general and has been explored in multiple applications such as automated game-playing [200], robotic control [197], and combinatorial optimisation [21].

Markov Decision Process

The environment for RL problems is typically represented as a Markov decision process (MDP) [19]. An MDP is a general formalisation of sequential decision making [207]. It provides a mathematical framework to model stochastic control

processes where decisions (actions) taken by a decision-maker (agent) influence the immediate and future outcomes. There are different branches of MDPs, such as discrete-time MDPs, continuous-time MDPs, partially observable MDPs (POMDP), with extensive literature exploring each variation [103]. We focus our attention on discrete-time MDPs and POMDPs.

A MDP can be defined by the following components:

- $\mathcal{S}$: a state space, i.e., a set of states of the environment.
- $\mathcal{A}$: an action space, i.e., a set of possible actions available to the agent.
- $P(r_t = r, s_{t+1} = s' | s_t = s, a_t = a)$ the transition probabilities, i.e., the probability of arriving at next state s', receiving reward r, given previous state s and action a.
- $\gamma \in [0, 1)$, a discount factor, used to discount future reward contributions.
- r_t, a reward function, attributed to immediate transitions.

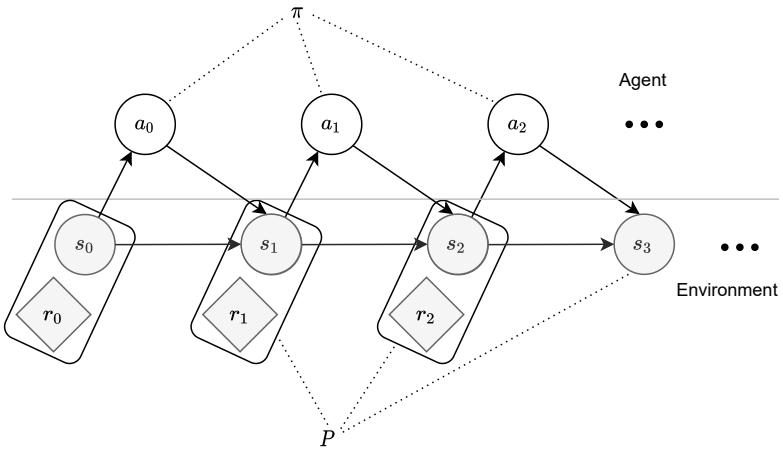


Figure 2.6: A Markov decision process environment.

Assuming an initial state probability $P(s_0)$ from which we sample initial states, the discounted infinite horizon RL problem is defined as follows. An agent (policy) produces trajectories τ over discrete time steps $t = 0, 1, 2, \ldots$. That is, an agent observes states s_t, takes actions a_t, and after an action is selected, the environment transitions to the next state according to the transition probabilities, and the agent receives a reward r_t, usually, a function of states, action and next states, representing the immediate result of that action (see Figure 2.6). Note that for the problems in this thesis with deterministic rewards we sometimes write the transition probabilities as $P(s'|s, a)$, a shorthand for $P(s_{t+1} = s' | s_t = s, a_t = a)$. The general assumption

is that the probability of arriving in the next state after selecting an action depends only on the current state and not the entire history of states and actions (Markovian property). A typical application of MDPs is to the maintenance of deteriorating assets. For example, one can define an MDP in which the states represent the current health condition assets. In this example, our goal can be to find a strategy (policy) to replace and repair an asset to maximise its uptime based on the states of the system. Other applications areas of MDPs include robotics, finance, production systems, biology, among others [228].

A (stochastic) policy π is a function mapping states to action probabilities, represented as $\pi(a|s)$, i.e., $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The objective is typically to maximise the received rewards, measured as the expected cumulative sum of (discounted) rewards, i.e.,

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \right],$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value given that the agent follows policy π in all time steps.

Partially Observable Markov Decision Process

In a POMDP, the agent has access to censored information from the (latent) states. Most real-world problems are partially observable because one seldom has access to full information. Reverting to the maintenance example from before, in the POMDP case, we do not know with full certainty the states of the system. That is, we cannot observe all the relevant aspects for optimal decision making. So instead of observing the current health condition of assets, we may observe only the age, previous repairs and indirect information about the asset such as visual wear and tear. Thus, we have to reason about the best repair policy based on partial information about the states. This partial-observability introduces a differentiation between states (full information), observations o (partial information) and introduces a probability of an observation as $P(o|s')$, an observation space O and a history $h_t = (o_0, a_0, o_1, a_1, \ldots, a_{t-1}, o_t)$. In this formulation, the agent should take actions under uncertainty of the true environment state (see Figure 2.7). POMDPs are notoriously hard to solve since observations only carry partial information for choosing an action. Therefore, information must be aggregated over time and in general, the entire history of observations must be taken into account. This history can be kept track of by retaining memory of all past transitions. Otherwise, one can estimate distributions of the latent spaces, i.e., a belief state. In fact, a POMDP can be described as a fully-observable belief-MDP with continuous state space. However, when the transitions probabilities are not available, a belief over states cannot be inferred. Luckily, if a good approximation exists for the history (or of belief states), RL methods can be employed to learn policies for POMDPs [86, 110].

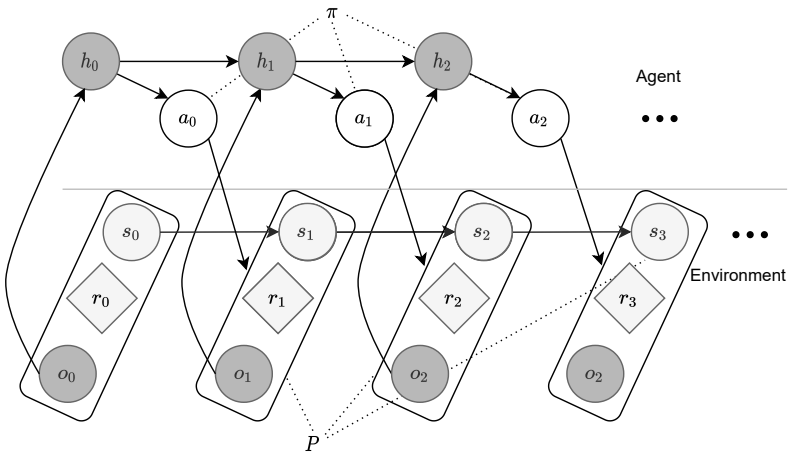


Figure 2.7: A partially observable Markov decision process environment.

Dynamic Programming

When solving MDPs, it is customary to define value functions. These are functions of states V (or state-action pairs Q) that compute the expected return of following a policy starting from a given state. That is,

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right],$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right].$$

Solving an MDP translates to finding the policy that maximises the value functions for all possible states. An optimal policy's value function must satisfy the Bellman optimality equations [20], i.e., a function that expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state [207]. That is,

$$V^*(s) = \max_a \sum_{s',r} P(s',r|s,a)\left(r + \gamma V^*(s')\right),$$

$$V^*(s) = \max_a Q^*(s,a) = \max_a \sum_{s',r} P(s',r|s,a)\left(r + \gamma \max_{a'} Q^*(s',a')\right).$$

The optimal policy can be extracted by acting greedily with respect to the optimal value functions. When transition probabilities of the environment are known, finding the optimal policy can be achieved via dynamic programming (DP). Value iter-

ation and policy iteration are well-known interactive algorithms that turn the Bellman equation into update rules for the value estimates and converge to the optimal policies for finite-state MDPs.

Reinforcement Learning Methods

DP forms the backbone of RL algorithms. The main difference between DP methods and RL methods is the assumption about the available information from the environment (model). In DP, it is typically assumed that information about the mathematical model that governs the environment is known, i.e., the results of the agent's action in the environment are known from the transition probabilities. In typical model-free RL, the assumptions about the mathematical model of the environment are dropped, and learning must emerge solely from observing the (uncertain) results of actions. There exist many different RL methods, and describing each method in detail is out of the scope of this thesis. For more information on other classical RL algorithms see [27, 207].

Value-based algorithms are based on the value functions and the Bellman optimality equations. The main objective is to learn the optimal value function using DP. Classic methods leverage the idea of temporal-difference (TD) learning - a combination of ideas of Monte Carlo methods and DP [207]. That is, like Monte Carlo methods, TD-based methods can learn directly from experience while sampling trajectories from an environment. Like DP, TD updates estimates of value functions based on previously learned estimates, i.e., they bootstrap. An important metric for TD methods is the TD-error, which measures the error of the current value function for s_t in comparison to one step ahead, i.e., $r_t + \gamma V(s_{t+1})$, represented as

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t),$$

where s_0 is the initial state and s_t is the state at the t-th step. The Monte Carlo error (for fixed V) can be written as

$$G_t - V(s_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k$$

where $G_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$ is the return (discounted sum of future rewards) from state s_t.

In the tabular case, i.e., when both state and action spaces are finite and can be represented in a tabular format, control methods based on well-known on-policy and off-policy TD methods include SARSA [190] and Q-Learning [226], respectively. On-policy methods use the transitions experienced from the current behaviour policy to update estimates of the value function. Off-policy methods can use the previous experience of other policies to improve on the value estimates. Both SARSA and Q-learning are tabular methods that seek to find the optimal state-

action value functions (Q). In Q-learning, the behaviour policy still has an effect determining which state and action-pairs are visited [207]. Nonetheless, Q-learning only requires that state-action pairs are visited enough times for correct convergence to the optimal policy. Typically, data is acquired following an ϵ-greedy policy, i.e., taking a random action with probability ϵ, otherwise choosing the action with the highest Q value. Q-learning updates its estimates as follows,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

where $\alpha \in \mathbb{R}$ is a learning rate.

Similar to DP, tabular RL methods suffer from the curse of dimensionality. Therefore, neither can be applied to combinatorial state and action spaces under reasonable assumptions about computational time. To address these issues, RL employs function approximation to learn policies or values functions without storing the entire state $\times$ action space. When the function approximation is a deep neural network, this combination is referred to as deep reinforcement learning (DRL). DRL allows for agents to be trained without feature engineering of the state representations and leverage large unstructured inputs that otherwise would be impractical with classical ML. In general, function approximation cannot represent the value function (or policy) exactly, and the resulting error can lead to poor convergence. Nevertheless, recent results have shown that function approximation methods can surpass human-based policies in multiple complex tasks.

Deep Reinforcement Learning

The recent advances in DRL have led to an ever-growing number of algorithms. We refer the reader to recent surveys [8, 111] for a detailed overview. We focus on methods that achieved recent success in various applications, that is, value-based, distributional DRL, and policy gradient methods.

Deep Q-Learning Value-based methods employ deep neural networks as function approximations for the (optimal) value functions. The use of neural networks allows for the application of RL methods in domains that depend on visual, temporal and graph inputs. Video games are a classic example in which the inputs are made of images and actions are possible combinations of controls. A seminal DRL algorithm is deep Q-learning (DQN)[1] [163]. The main idea of DQN is to adapt the value update step of Q-learning to consider a neural network function $Q_\theta(s, a)$ with trainable parameters θ and perform gradient steps towards the minimisation of the TD-error via a loss function $\ell(y_t, Q_\theta(s_t, a_t))$, where $y_t = r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$. That is, the parameters θ are optimised such that the current estimate for $Q_\theta(s_t, a_t)$

[1] Also referred to as deep Q-network

approaches the value of the one-step estimate in the future as

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s' \sim P(s'|s,a)} \left[\ell(y, Q_\theta(s,a)) \right] |_{\theta=\theta_k}$$

where the expectation is approximated with samples from a policy acting on the environment. The loss function is usually the squared error, and y considers a fixed Q_θ when updating the current values, i.e., TD targets remain fixed while updating the current estimates. Note that this is similar to the TD updates when the function Q is a lookup table. Several algorithmic improvements make training DQN possible, such as the introduction of a target neural network, i.e., a copy of the Q_θ network that is only updated after a number of computation steps and the introduction of an experience replay, responsible for storing previous transitions (s_t, a_t, r_t, s_{t+1}) in a hash table. This replay is used to sample and decorrelate past transitions, which in turn are employed to update the Q_θ estimates, allowing for more efficient use of previous data and better convergence when training a function approximator due to the i.i.d. data assumption in supervised learning. An example of neural network architecture for DQN uses the states as input features (neurons in the input layer) and have the output layer dimensions correspond to the number of possible actions an agent can perform. In this case, the learnable parameters θ correspond to the weights and biases of the neural network approximation. Other improvements to DQN include, double Q-learning [85], prioritised experience replay [193], Dueling DQN [225], among others.

Distributional Approaches The RL problem setting usually involves some stochasticity from the environment. For example, after selecting an action in a given state, the result of that action can result in multiple future states. This stochasticity can greatly affect learning and how much return is received by the agent during learning. Most value-based methods focus on learning the expectation of the random variable of returns. Nonetheless, distributional methods attempt to learn the distribution of the returns. The idea is that by modelling (and learning) the whole distribution of returns agents can better learn the result of actions in stochastic environments, especially when taking actions in this environment represents some measure of a risk-reward trade-off. Using distributional approaches can also lead to more stable learning and preserves the multi-modality in value distributions [18].

For a given MDP and policy π, we can define the value distribution of a policy as the random variable Z^π for given s and a as

$$Z^\pi(s,a) := \sum_{t=0}^{\infty} \gamma^t r_t$$

which in expectation is

$$Q^\pi(s,a) := \mathbb{E}_\pi[Z^\pi(s,a)].$$

Using this definition, the Bellman equation can be rewritten for the distributional case [18], as

$$Z^\pi(s,a) \overset{\mathrm{d}}{=} r(s,a) + \gamma Z^\pi(s',a'),$$

where $s' \sim P(s'|s,a)$, $a' \sim \pi(a'|s')$, $r(s,a)$ is the random variable of rewards, and $\overset{\mathrm{d}}{=}$ indicates that the cumulative distribution functions are equivalent (follow the same law). That is, the value distribution $Z^\pi \in \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R})$ is in a space of mappings from state-action pairs to the space of continuous distributions, shortened as $Z^\pi \in \mathcal{Z}$. An important implication is that the convergence of such a distributional equation depends on the chosen metric. In particular, we are interested in the maximal form of the Wasserstein metric, i.e.,

$$\bar{d}_p(Z_1, Z_2) := \sup_{s,a} d_p(Z_1(s,a), Z_2(s,a)),$$

which is shown to be a metric in the space of value distributions, where $d_p(X,Y)$ is the p-Wasserstein metric in space $\mathcal{P}(\mathbb{R})$, $1 \leq p \leq \infty$, and $Z_1, Z_2 \in \mathcal{Z}$ are distributions with p-bounded moments [18].

A following result, shows that the distributional Bellman operator $\mathcal{T}^\pi$, $\mathcal{T}^\pi Z(s,a) \overset{\mathrm{d}}{=} r(s,a) + \gamma Z(s',a')$, is a contraction mapping in $\bar{d}_p$ and the point iteration converges to Z^π [18]. The previous result suggests a way to learn value distributions by minimising the Wasserstein metric between a distribution Z and its Bellman update $\mathcal{T}^\pi Z$ similar to how Q learning minimises the TD error. However, we cannot directly minimise the Wasserstein metric using stochastic gradient descent [55]. Thus, several distributional approaches attempt to estimate the Wasserstein metric in the presence of gradient descent and function approximation [18, 55]. A particular method of interest in Chapter 7, named quantile regression DQN (QR-DQN), showed that selecting a family of distributions to approximate the distributions of Z reduces the Wasserstein minimisation task to search for quantiles of these distributions [55].

Note that for policy improvement, the distributional Bellman *optimality* operator $\mathcal{T}^*$, i.e, $\mathcal{T}^* Z = \mathcal{T}^\pi Z$ for some π acting greedily with respect to the expectation of Z, under point iteration may diverge. Nonetheless, it preserves means, and the mean will eventually converge to the optimal policy, i.e., it converges to $Q^*(s,a)$ with similar guarantees as Q-learning [18].

Policy Gradients In policy gradient methods, the goal is to directly approximate a policy rather than value functions. A (stochastic) policy in this context is a function with learnable parameters θ, referred to as $\pi_\theta(a|s)$. Note that we use π and π_θ interchangeably in the following equations, being explicit when computing gradients. The general objective of the RL problem remains the same, but unlike before, we have access to a differentiable policy function. Since the goal is typically

to maximise performance, policy gradient methods work by taking gradient steps towards the maximisation of a performance metric $J(\theta)$ that depends on the policy parameters θ. In this thesis, we assume, without loss of generality, that policies are neural network functions and define the performance of a policy as

$$J(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)}[G(\tau)] = \mathbb{E}_{P_\theta(\tau)}\left[\sum_{t=0}^{T-1} \gamma^t r_t | s_0\right].$$

where the expectation $\mathbb{E}_{\tau \sim P_\theta(\tau)}[\cdot]$ is taken with respect to the randomness of a full trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T)$ following the trajectory distribution $P_\theta(\tau)$ induced by policy π_θ, and $G(\tau)$ is the return of a trajectory τ. Note that, for simplicity, we derive the following results for the finite horizon case and state the more general theorem (for the discounted infinite horizon) later.

To estimate the gradient of the expectation, we make use of the log-derivative trick, i.e.,

$$\nabla_\theta p_\theta(x) = p_\theta(x) \nabla_\theta \log p_\theta(x)$$

and the score function estimator, i.e,

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{x \sim p_\theta(x)}[f(x)] &= \nabla_\theta \int p_\theta(x) f(x) dx \\
&= \int \nabla_\theta p_\theta(x) f(x) dx \\
&= \int p_\theta(x) \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} f(x) dx \\
&= \int p_\theta(x) \nabla_\theta \log p_\theta(x) f(x) dx \qquad \text{(log-derivative trick)} \\
&= \mathbb{E}_{x \sim p_\theta(x)}[\nabla_\theta \log p_\theta(x) f(x)].
\end{aligned}$$

where $p_\theta(x)$ is a probability density of random variable x and f is real function and use this result to arrive at

$$\nabla_\theta \mathbb{E}_{P_\theta(\tau)}[G(\tau)] = \mathbb{E}_{P_\theta(\tau)}[\nabla_\theta \log P_\theta(\tau) G(\tau)].$$

Using the chain rule of probabilities, the probability of a single trajectory can be written as

$$\begin{aligned}
P_\theta(\tau) = P(s_0) \pi_\theta(a_0|s_0) P(s_1, r_0|s_0, a_0) \pi_\theta(a_1|s_1) \\
P(s_2, r_1|s_1, a_1) \ldots \pi_\theta(a_{T-1}|s_{T-1}) P(s_T, r_{T-1}|s_{T-1}, a_{T-1})
\end{aligned}$$

Taking the log and differentiating with respect to θ allows us to represent the gradi-

ent of $J(\theta)$ as

$$\nabla_\theta \mathbb{E}_{P_\theta(\tau)}[G(\tau)] = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau) \right].$$

The important result from the above equation is that learning a policy requires a return but no information about the system dynamics. Intuitively, learning requires taking gradient steps towards actions that increase the chance of higher returns.

More generally, for the discounted infinite horizon case, the policy gradient theorem states that the gradient of $J(\theta)$ can be written as

Theorem 1. ***Policy gradient theorem*** *[208]. For the infinite horizon discounted reward MDP and (discounted) state visitation distribution $d^\pi(s) = \lim_{T\to\infty} \sum_{t=0}^{T-1} \gamma^t P(s_t = s|s_0, \pi)$,*

$$\nabla_\theta J(\theta) = \sum_s d^\pi(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) = \mathbb{E}_{s\sim d^\pi(s), a\sim\pi(a|s)} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right].$$

That is, the gradient of the performance $J(\theta)$ can be expressed in terms of the gradient of the policy w.r.t. θ.

In general, policy gradient methods consider (unbiased) gradients of the form

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \phi_t \right],$$

where the expectation is usually estimated using Monte Carlo samples over a number of trajectories and large T, and ϕ_t is the return of the trajectories. Other variations of ϕ_t exist to reduce the variance of the estimators, such as computing the returns considering only the future rewards at time step t and the inclusion of a baseline, i.e., an arbitrary function (conditionally independent of sampling from π_θ) that can depend on states $b(s_t)$ subtracted from the future sum of rewards and maintains the expectation unbiased, i.e., $\sum_{t'=t}^{T-1} \gamma^{t'-t} r_t - b(s_t)$. Other approaches include estimating the state-action value function $\hat{Q}^\pi(s_t, a_t)$, the TD error, i.e., $r_t + \gamma \hat{V}^\pi(s_{t+1}) - \hat{V}^\pi(s_t)$ or an advantage function $\hat{A}^\pi(s_t, a_t) = \hat{Q}^\pi(s_t, a_t) - \hat{V}^\pi(s_t)$, i.e., measuring how good is to that action a_t at state s_t in comparison to the current estimated value of state s_t [196]. Classical algorithms include REINFORCE [229] and more recent advances to reduce sample complexity include, trust-region policy optimisation (TRPO) [195] and proximal policy optimisation (PPO) [197]. When function approximation is used for both the policy and value functions these are commonly referred to as actor-critic methods [162], these are of particular interest in chapters 5 and 6.

2.3.4 Imitation Learning

In imitation learning (IL), it is assumed that learning does not have to start from scratch. In the case when IL is employed for sequential decision-making problems defined on MDPs (see Section 2.3.3), the aim is to learn to imitate a previously existing policy. Thus, it assumes that previous policies usually provided by a human expert or an existing algorithm already exist. These policies are commonly referred to as oracles or experts. The goal of imitation learning is to use the information from these policies and learn to imitate their behaviour, mapping states to expert actions instead of learning from scratch as in RL [108]. This type of learning can potentially reduce the burden of learning the desired behaviour of agents entirely from scratch [108] and reduce the need for costly interactions with the environment.

IL is related to offline RL, in which the goal is to learn optimal policies from fixed offline data without requiring interaction with an environment. Note that IL may not impose such assumptions about offline data and sometimes consider that the expert is available online, i.e., while learning via interaction with the environment. Moreover, the IL objective may be solely to reproduce the behaviour of the expert policy. When the expert policy is sub-optimal, however, the goal of imitating an expert is typically to accelerate learning, extracting knowledge from this policy before moving towards exploratory actions that improve upon the expert. This objective is similar to the offline RL case when the assumption of no interaction with the environment is dropped. One can also relate IL with transfer learning in which the source domain corresponds to expert demonstrations and the target domain to learning to generalise to unseen states in the demonstrations (see Section 2.3.2). IL is also related to apprenticeship learning and inverse RL, where the goal is to learn the typically unknown objectives of the desired behaviour from demonstrations (rewards). A complete review of IL and its relationships with other learning tasks can be found in [108, 172].

Imitation Learning from Demonstrations

In this thesis, we consider the task of learning from demonstration data in a model-free IL setup without access to trajectory returns from demonstrations. Our objective is to learn from demonstrations acquired by following an expert policy $a = \pi^e(s)$, mapping states to actions, where $s \in \mathcal{S}$ is a state in state space $\mathcal{S}$ and $a \in \mathcal{A}$ is an action in action space $\mathcal{A}$. Ideally, one would aim to learn a policy that approximates the performance of the expert as much as possible; that is, we want to learn a policy π_θ, with parameters θ that can mimic the same actions a from an expert policy π^e, given a state s.

In our setting, we assume that the acquired demonstration data is from a sub-optimal expert that is not available during the interaction with the environment (online). That is, we assume the existence of a set of expert trajectories gathered to form of a dataset $\mathsf{D}^e = \{(s_i, a_i)\}_{i=1}^N$ where s_i and a_i are states and actions performed

by the expert acquired from previous trajectories $\tau^e = s_0, a_0, s_1, a_1, \ldots$ following the environment dynamics $P(r, s'|s, a)$ induced by an MDP environment (see Section 2.3.3). Our goal is to learn to replicate the actions from the expert agent, learning from the gathered expert data D^e. A direct approach to solving this problem is to use the demonstration data to learn an approximation of the policy via supervised learning on i.i.d. training samples (behaviour cloning) [187]. Let $d^{\pi^e}(s)$ define the state distribution of the expert's policy induced by the trajectories followed by the expert. If we are learning directly from these trajectories, the objective becomes

$$\mathcal{L}(\theta) = \mathbb{E}_{s\sim d^{\pi^e}(s), a=\pi^e(s)}[\ell(a, \pi_\theta(s))]$$

where ℓ is a surrogate loss, such as the Kullback-Leibler divergence that we wish to minimise. However, this objective is slightly different from the true objective of imitation learning. That is, when sampling from the behaviour policy, any deviations from the expert policy would induce different distributions of states $d^{\pi_\theta}(s)$. Thus, in reality, we would like to minimise the objective

$$\mathcal{L}(\theta)_{\text{IL}} = \mathbb{E}_{s\sim d^{\pi_\theta}(\cdot), a=\pi^e(s)}[\ell(a, \pi_\theta(s))].$$

From the above equations, it is clear that when the two distributions do not match, i.e., the behaviour distribution of states shifts from the expert's state distribution, the approximation of the first equation may not approximate the real objective. What is worse, if no data exists on the states visited by the behaviour policy in the dataset, then the learned policy may not be able to recover from states not seen in the training data. Under mild assumptions, the bound of the imitation learning objective grows quadratically with the size of the trajectory times the generalisation error of the behaviour cloning objective [186]. To alleviate this issue, online IL methods attempt to reduce the distributional shift between the demonstration data and the behaviour policy by assuming some level of access to the expert during training [36, 187]. These interactive direct policy learning methods aim at acquiring more information about the expert policies, such as the expert's action or reward-to-go information in regions of the state space not covered by demonstrations assuming an interactive demonstrator or the property of revisiting states.

When the goal is to learn optimal policies starting solely from sub-optimal expert demonstration, as is the case in Chapter 6, we are interested in learning to surpass the quality of the expert policy. Building upon these ideas, learning can be achieved by employing IL to learn initial policies from the demonstration data and then refine these policies with RL in a second stage [40, 183]. Another option is to use the demonstration data to approximate value functions (see Section 2.3.3) of the expert and later use the learned values function in combination with policy improvement of the behaviour policy [40, 204, 205]. The overall objective of combining IL and RL is to reduce the need for exploration of bad actions, res-

ulting in much lower sampling complexity and guiding policy search to explore more promising areas that can quickly improve upon expert policies. Note that this objective is similar to offline RL, in which the objective is to learn optimal policies from demonstrations. However, offline RL normally assumes access to demonstrations of transitions including states, actions, next states and rewards and no access to online training [139].

Chapter 3

Predicting Remaining Useful Lifetime

Machine prognostics and health management is often concerned with predicting the remaining useful lifetime (RUL) of assets. Accurate real-time RUL predictions enable equipment health assessment and optimal maintenance planning. In this chapter, we consider the problem of predicting the RUL of engineering assets degrading over time. In this problem, maintenance planners have access to assets' sensor information and are interested in predicting the RUL of assets such that preventive maintenance actions can take place before failures occur.

This chapter is based on [50].

3.1 Introduction

In this chapter, we study the remaining useful lifetime (RUL) prediction problem faced by maintenance planners. In machine prognostics and health management (PHM), RUL relates to the amount of time left before a piece of equipment cannot perform its intended function [199]. The RUL of an asset (or a component) is generally a random variable that may depend on various aspects of the asset, such as age, usage, location, operating conditions, among others. Accurate RUL prognostics, i.e., *predicting point-estimates of the remaining lifetime of assets*, is critical for condition-based maintenance (CBM) and enables the interested parties to plan future maintenance actions, e.g., logistics of personnel, spare parts, and services [176]. With the introduction of remote monitoring devices, one can use raw sensor data to extract information about the current condition of assets. This information can then be used to predict the future point in time that assets may stop functioning by learning from past data of previous failures.

In PHM and CBM, physics, statistical and machine learning (ML) approaches have been proposed to address the RUL prediction problem. We focus on ML approaches because they require little to no prior knowledge about the underlying degradation of assets and have progressively outperformed other methods in prediction performance [137]. Moreover, ML methods are preferred when multi-dimensional data is present, as is usually the case of remote sensors. In particular, several deep neural networks architectures have been receiving attention given their ability to achieve high accuracy learning directly from raw data. These include convolutional neural networks (CNN) [140] and recurrent neural networks (RNN) implementations, such as, long short-term memory (LSTM) [144, 243] and gated recurrent unit (GRU) [232].

However, it can be hard to interpret the relationship between the raw sensor input signals and the prediction outputs in deep neural network methods due to the high dimensionality of these methods. In our application, it is often necessary to identify failure causes and points in time when maintenance is optimal. Therefore, one is often posed with the choice of a low-performance white box method or a black box method that performs well but gives little insight into when the assets started showing degradation signals. In this chapter, we present a deep learning method that improves the overall performance of previous methods and provides a representation between temporal inputs and RUL outputs in the well-known Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) datasets [191] (see Appendix 3.A). Our proposed method implements an attention module [13, 154] that provides a visual representation of trained weights while retaining the predictive power of deep learning. The proposed method can help decision-makers plan precise maintenance actions whilst providing information about the relationships between input sensors and RUL outputs.

Contributions and Organisation In our proposed model, we demonstrate that when combined with a variable-level attention mechanism, an LSTM [97] can learn the temporal relationship of input variables and output RUL. We show the effectiveness of the proposed method against other methods for RUL prediction of aircraft engines in the C-MAPSS datasets. The main contributions are summarised as follows.

- Our method learns accurate RUL estimates directly from raw multidimensional sensor readings.
- We use a soft attention mechanism to visualise the learned attention weights at each RUL prediction step. The learned weights offer more transparency on important input timesteps at each RUL prediction step.
- The proposed method achieves high-performance RUL prediction results in the C-MAPPS datasets compared to previous state-of-the-art methods.
- Unlike previous deep learning methods, it does not require pretraining nor extensive hyperparameter optimisation.

The remainder of this chapter is organised as follows. In the next section, we briefly discuss the state-of-the-art methods for RUL prediction in the C-MAPPS datasets. In section 3.3, we present our model detailing the learning algorithm and the architecture of our proposed LSTM. In section 3.4, we present the experimental setup and the selected hyperparameters of our method. Finally, in section 3.5, we compare and contrast the performance of proposed methods and provide an analysis of the results.

3.2 Related Literature

Deep Learning in Prognostics In the prognostics literature, several artificial intelligence methods have been proposed to predict the RUL of assets, e.g., linear regression [88], support vector regression (SVR) [26], fuzzy-logic systems [244] and (deep) neural networks [106, 210]. The latter has received much attention given their ability to approximate complex functions without prior degradation knowledge and feature engineering. In many PHM applications, sequential temporal data coming from sensors are the norm. Thus, we focus our attention on previous neural network architectures that have been successfully employed to learn RUL estimates in this scenario. In particular, RNNs are a natural fit for such problems, given their recurrent internal structure and shared parameters over time. However, due to vanishing gradients, RNNs have issues when learning long-term dependencies [25]. LSTMs [97] and GRUs [43] networks were introduced to address issues with long-term dependency by preserving memory states and reducing the vanishing gradient problem of RNNs. Other architectures, such as CNNs, have also

been successful in sequential prediction problems. In this case, the model learns to extract features by sliding multiple feature extractors over the output from the previous layers using convolution operations. We discuss recently proposed methods based on these architectures below.

In PHM, [237] recently showed that LSTMs could outperform RNNs and GRUs in RUL prediction. [243] showed that a sequence of LSTM layers followed by a feed-forward neural network (FFNN) could outperform other methods, including CNNs, in three distinct degradation datasets. In [232], the authors presented similar results by extracting features based on a dynamic difference procedure and later training an LSTM. Results showed that the LSTM also outperformed simpler RNNs and GRU architectures under similar machinery conditions. More recently, [144] showed that restricted Boltzmann machines could be used to extract useful weight information by pretraining on degradation data in an unsupervised manner. In this two-stage method, weights extracted in the first step are used in a second step to fine-tune a supervised LSTM and FFNN model. A genetic algorithm (GA) is used to select the best performing hyperparameters. The methodology holds the state-of-the-art results for the RUL prediction problem in the C-MAPSS datasets to the best of our knowledge.

CNNs are notable for extracting spatial information from 2D, and 3D data [101]. Nonetheless, CNNs can also handle 1D sequential data and extract high-level features by performing a convolution operation of feature extractors over local receptive fields of its inputs. In machine prognostics, [11] proposed a 2D deep CNN to predict the RUL of a system based on normalised variate time series from sensor signals. They show the effectiveness of the CNN in comparison to an FFNN and classical ML methods such as SVR and relevance vector regression. More recently, [140] proposed to apply 1D convolutions to the sequential input data without any pooling operations, which are commonly used in CNN architectures to reduce dimensionality. The results show that the proposed architecture can extract useful features for RUL prediction and show competitive results on the C-MAPPS dataset without incurring high training times of autoregressive methods.

Attention Mechanism Attention has been used in a wide range of neural network architectures. It was initially introduced in natural language processing (NLP) for machine translation tasks [13], but has been successfully applied in other tasks, e.g., computer vision [235]. In such tasks, we are often interested in learning to attend to specific parts of the input instead of the whole input or the last timestep in the sequential case. As it is common in recurrent architectures, the last hidden vector usually summarises the information about the entire sequence. However, we often wish to attend to different parts of the input that are more relevant to the output we want to produce at a given timestep. Attention allows the neural network to learn to focus on specific parts of the input when predicting outputs, resulting in better performance over long sequences [13]. Besides performance gains, attention

mechanisms can also be used to interpret the behaviour of neural architectures by analysing the parts of the input that the network learns to attend [73]. That is, learned attention weight could be used to visualise and investigate the relationship between inputs and outputs of the network.

Based on recent promising results showing that LSTMs and CNNs architectures can outperform traditional prognostics algorithms, we propose a deep learning architecture capable of learning RUL predictions directly from sensor data. Unlike previous works, our method does not require pretraining or the extensive search of hyperparameters to perform well on the task. Moreover, it can be used for sequences of varying sizes, unlike methods based on CNNs under the constraint of a fixed receptive field. Our method incorporates a self-attention mechanism that attends to different input sequence parts relevant to the RUL prediction at a given timestep. We show that the learned attention weights can also be used to provide more insights between the predicted RUL and the sensor information passed as input.

3.3 Attention Model

This section presents our proposed method to predict the RUL from sensor data. We first introduce the problem, notations and discuss the proposed method and its components.

3.3.1 Problem Definition

We denote our training data pairs as $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, containing $N \in \mathbb{N}$ training example pairs. Where $\mathbf{x}_i$ denotes a multivariate time-series input of length T_i and q features, i.e., $\mathbf{x}_i = [x_1, \ldots, x_{T_i}] \in \mathbb{R}^{q \times T_i}$. Moreover, $\mathbf{y}_i$ denotes the RUL values of length T_i where $\mathbf{y}_i = [y_1, \ldots, y_{T_i}]^\top \in \mathbb{R}_{\geq 0}^{T_i}$. Where for each $t \in \{1, 2, ..., T_i\}$, $x_t \in \mathbb{R}^q$ and $y_t \in \mathbb{R}_{\geq 0}$ represent the t-th measurement of q sensor inputs and RUL labels (the observed RUL values from the data), respectively. Further details about the input time-series and RUL labels are provided in Section 3.4.1. We aim to learn a function g with learnable parameters θ such that we can approximate the corresponding RUL at testing time directly from degradation data, i.e., $\mathbf{y}_i \approx g_\theta(\mathbf{x}_i)$. In our case, the learnable parameters θ are the weights and biases of our neural network model.

3.3.2 Rolling Time Windows

We employ a rolling time window approach to allow temporal sequences to influence the RUL prediction time step t and increase the number of data points used for RUL prediction. That is, we define a function that divides each sequence of size

T_i in sequential time windows of size T_w, i.e., $\phi_t(\mathbf{x}) = [x_{t-T_w+1}, ..., x_t]$ for $t \geq T_w$. In words, at time t all previous sensor data within the time window $\phi_t(\mathbf{x})$ are collected to form T_w input vectors used to predict y_{t+1}. Thus, after the transformation, each original time series will have $n_i = T_i - T_w$ training samples, where $T_i > T_w$. We denote total number of data points after the transformation as $N_{T_w} = \sum_{i=1}^{N} n_i$.

3.3.3 Long Short-Term Neural Networks

LSTMs have recurrent connections capable of learning the temporal dynamics of sensor data in prognostics scenarios. Moreover, they control how information flows within the LSTM cells by updating a series of gates capable of learning long-term relationships in the input data.

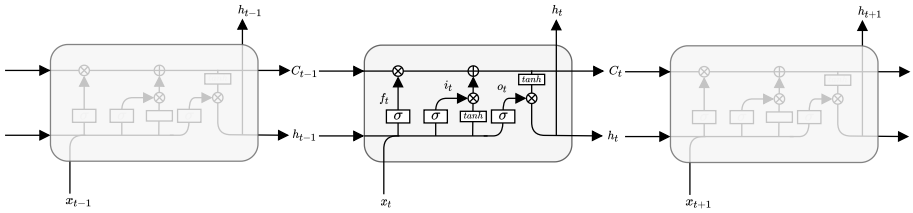


Figure 3.1: LSTM memory cell.

In our LSTM implementation, the memory cell (Figure 3.1) consists of three non-linear gating units that update a cell state $C_t \in \mathbb{R}^l$, using a hidden state vector $h_{t-1} \in \mathbb{R}^l$ and inputs $x_t \in \mathbb{R}^q$, where l is the dimension of the LSTM hidden state and q the input dimension

$$f_t = \sigma\big(W_f x_t + R_f h_{t-1} + b_f\big) \tag{3.1}$$

$$i_t = \sigma\big(W_i x_t + R_i h_{t-1} + b_i\big) \tag{3.2}$$

$$o_t = \sigma\big(W_o x_t + R_o h_{t-1} + b_o\big) \tag{3.3}$$

where σ is a sigmoid activation function responsible for squeezing the output to the 0-1 range, $W_g \in \mathbb{R}^{l \times q}$ are the input weights, $R_g \in \mathbb{R}^{l \times l}$ are the recurrent weights, and $b_g \in \mathbb{R}^l$ are biases. Where the subscript $g \in \{f, i, o\}$ can either be the forget gate f, input gate i or the output gate o, depending on the activation being calculated.

After computing f_t, i_t and $o_t \in \mathbb{R}^l$, the new cell state $\widetilde{C}_t$ candidate is computed as follows

$$\widetilde{C}_t = \tanh\big(W_C x_t + R_C h_{t-1} + b_C\big) \tag{3.4}$$

where, tanh represents the hyperbolic tangent function and similar to the gate operations, $W_C \in \mathbb{R}^{l \times q}$, $R_C \in \mathbb{R}^{l \times l}$, and $b_C \in \mathbb{R}^l$ are learnable parameters.

The previous cell state C_{t-1} is then updated to the new cell state C_t

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \tag{3.5}$$

where $\odot$ denotes the element-wise multiplication.

In the previous equations, the forget gate f_t is responsible for deciding which information will be thrown away from the cell state. Next, the input gate i_t decides which states will be updated from a candidate cell state. The input and forget gates are then used to update a new cell state for the next time step. Lastly, the output gate o_t decides which information the cell will output, and the new hidden state h_t is computed by applying a tanh function to the current cell state times the output gate activations.

$$h_t = o_t \odot \tanh(C_t) \tag{3.6}$$

3.3.4 Attention Mechanism

Our self-attention mechanism is based on [154]. At each time step t we take as input the hidden states h_t at the top layer of stacked LSTM layers. After, we decide on a context vector c_t that captures relevant information about the next target y_{t+1}. Given the target hidden state h_t and the context vector c_t, we concatenate both vectors and learn $W_c \in \mathbb{R}^{d \times 2l}$, to produce an attention vector a_t of the form

$$a_t = \tanh(W_c[c_t||h_t]) \tag{3.7}$$

where $\cdot||\cdot$ represents the concatenation operation, tanh is the hyperbolic tangent function and the context vector c_t is defined as

$$c_t = \sum_{j=1}^{t} \alpha_{t,j} h_j \tag{3.8}$$

In words, we consider all the hidden states of the LSTM encoder weighted by attention weights $\alpha_{t,j}$. In our implementation, the attention weights are derived by comparing the current hidden state h_t with the complete sequence hidden states h_j, $j \in \{1, .., t\}$. Where the attention weights $\alpha_{t,j}$ are computed via the softmax function as

$$\alpha_{t,j} = \frac{\exp(s(h_t, h_j))}{\sum_{j=1}^{t} \exp(s(h_t, h_j))} \tag{3.9}$$

and $s(h_t, h_j)$ is given by the multiplicative compatibility function [154]:

$$s(h_t, h_j) = h_t^\top W h_j \tag{3.10}$$

where $W \in \mathbb{R}^{l \times l}$. Thus, given attention weights $\alpha_{t,j}$ the context vector c_t is computed as the weighted average over all the hidden states and attention vectors a_t are passed

to the fully-connected (dense) layers in the network of the form

$$u = \text{ReLU}(W_a a_t + b_a) \tag{3.11}$$

where $\text{ReLU}(x) = \max(0, x)$ is the rectified linear unit, $W_a \in \mathbb{R}^{u \times d}$, $b_a \in \mathbb{R}^u$. Note that a_t summarises information from the context vector (including h_t) and h_t itself. Therefore, we expect that the learned attention weights will mostly focus on previous timesteps given that the information from h_t is directly available in a_t. The schematic of the proposed architecture is shown in Figure 3.2.

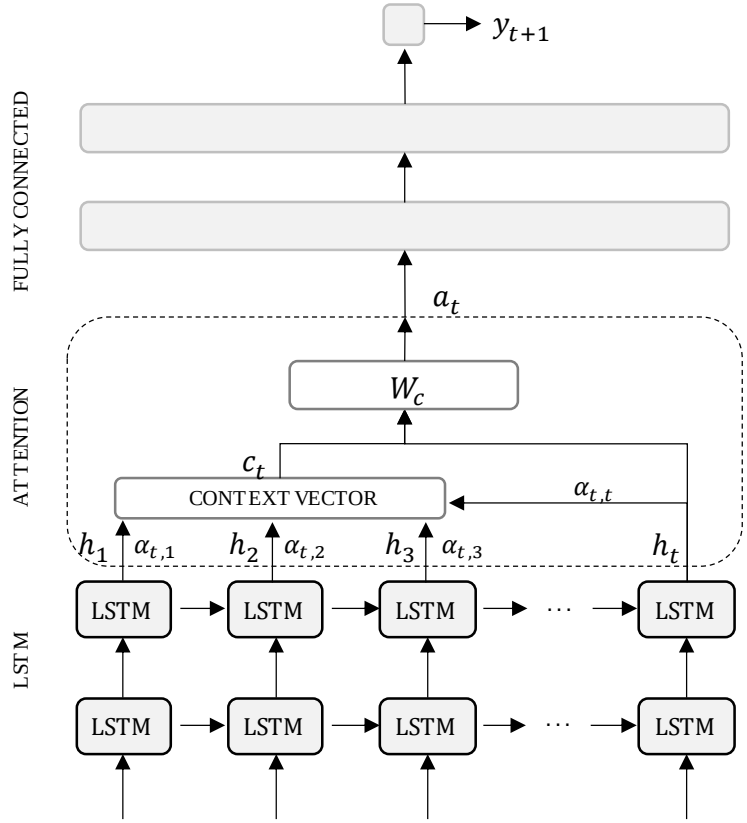


Figure 3.2: LSTM Architecture with attention. At each time step t, the model infers a variable-length alignment weight vector a_t based on the current target state h_t and a context vector c_t based on all previously seen states.

3.3.5 Loss Function

During training, we aim at minimising a regression loss based on the empirical risk minimisation using the observed RUL at time $t+1$ and inputs between $t - T_w + 1$ and t. The parameters of neural architecture are optimised towards the minimisation of the squared error loss function defined as

$$\mathcal{L}^i(\theta) = \sum_{t=T_w}^{T_i-1} (g_\theta(\phi_t(\mathbf{x}_i)) - y_{t+1})^2 \tag{3.12}$$

where θ are the learnable parameters of the neural network, and the overall training objective is

$$\mathcal{L}(\theta) = \frac{1}{N_{T_w}} \sum_{i=1}^{N} \mathcal{L}^i(\theta) + \frac{\lambda}{2} ||\theta'||_2^2 \tag{3.13}$$

where $\lambda \in \mathbb{R}$ is a hyperparameter of the L_2 regularisation of the weights θ', i.e., excluding biases. We compute the loss for a batch of training examples following the stochastic gradient descent updates of the Adaptive Moment Estimation (Adam) [124] algorithm.

3.4 Experimental Settings

In this section, we describe the experiments using the proposed model to predict the RUL of turbofan engine degradation data. We describe the datasets used in the experiments and the details about the implementation.

3.4.1 C-MAPPS Datasets

The proposed method is evaluated using the benchmark Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) [191] datasets containing turbofan engine degradation data. The C-MAPPS datasets are composed of four distinct datasets that contain information coming from 21 sensors as well as 3 operational settings. Each of the four datasets possesses several degradation engines split into training and testing data. Moreover, the datasets contain run-to-failure information collected under various operating conditions and fault modes.

Engines in the datasets are considered to start with various degrees of initial wear but are considered healthy at the start of each record. As the number of cycles increases, the engines begin to deteriorate until they can no longer function. Unlike the training datasets, the testing datasets contain temporal data that terminates some time before a system failure.

The original prediction task is to predict the RUL of the testing units using the training units [191]. The details about the four datasets are given in Table 3.1. We

refer to the datasets as FD001, FD002, FD003 and FD004. The operating conditions in the datasets vary between one (sea level) in FD001 and FD003, to six, based on different combinations of altitude (0 - 42000 feet), throttle resolver angle (20 - 100) and Mach (0 - 0.84) in FD002 and FD004. Also, fault modes vary between one (HPC degradation) in FD001 and FD002 and two (HPC degradation and Fan degradation) in FD003 and FD004.

Data	FD001	FD002	FD003	FD004
Engines: Training (N)	100	260	100	249
Engines: Testing	100	259	100	248
Operating Conditions	1	6	1	6
Fault Modes	1	1	2	2

Table 3.1: The C-MAPPS datasets. Each dataset contains a number of training engines (Engines: Training (N)) with *run-to-failure* information and a number of testing engines (Engines: Testing) with information terminating before a failure is observed.

3.4.2 Data Preprocessing

The temporal input data coming from 21 sensor values and 3 operational settings are used across all experiments. We note that for both FD001 and FD003 datasets, 7 sensor values have constant readings and have little impact in predicting target RUL values. We normalise the input data and RUL values by scaling each feature individually such that it is in the $[0, 1]$ range via min-max normalisation as

$$\bar{x}_t^{i,j} = \frac{x_t^{i,j} - \min_{i,t}(x_t^{i,j})}{\max_{i,t}(x_t^{i,j}) - \min_{i,t}(x_t^{i,j})} \tag{3.14}$$

where $x_t^{i,j}$ denotes the scalar input corresponding to the i-th training example and the j-th feature at time t.

RUL targets, i.e., the number of time steps that an engine will remain operational, are only available at the last time step for each engine in the test datasets. In RUL prediction tasks, it is reasonable to estimate the RUL as a constant value when the engines operate under normal conditions [89]. Similar to [137, 144], we propose a piece-wise linear degradation model to define the observed RUL values in the training sets. That is, after an initial period with constant RUL values, we assume that the RUL targets decrease linearly as the number of observed cycles progresses. We denote as R_e the initial period when the engines are still working in their desired conditions.

3.4.3 Performance Metric

Similar to other prognostic studies using the same datasets, we measure the performance of the proposed method using two metrics, the root mean squared error (RMSE) and the scoring function Eq. (3.15), proposed in [191], i.e.,

$$s = \sum_{i=1}^{N_{T_w}} \mathbb{1}(c_i < 0) e^{-\frac{c_i}{a_1}} + \mathbb{1}(c_i \geq 0) e^{\frac{c_i}{a_2}} - 1 \tag{3.15}$$

where $\mathbb{1}(\cdot)$ is an indicator function that takes value 1 when the predicate is true and 0 otherwise, $a_1 = 13$, $a_2 = 10$, $c_i = \hat{y}_i - y_i$, and $\hat{y}_i = g_\theta(\phi_{i-1}(\mathbf{x}))$. That is, c_i is the difference between predicted and observed RUL values. The scoring function penalises positive errors more than negative errors as these have a higher impact on RUL prognostics tasks. That is, predicting a higher RUL value is worse than predicting a lower value due to usually higher costs of corrective maintenance and machine downtime.

3.4.4 Hyperparameter Search

We perform 10-fold cross-validation to estimate the performance of the models and choose the best hyperparameters. To ensure that we do not validate the model on different parts of the same sequence, we split engine units uniformly at random in the original training datasets into training and validation sets containing 90% and 10% of the original engines, for example, following this procedure for FD001 results in 90 engines selected for training and 10 for validation.

Hyperparameter	Range
Learning rate	{0.001, 0.01, 0.1}
Batch size	{256, 512, 1024}
Number of layers (LSTM, Dense)	{1, 2}
Number of units (LSTM)	{20, 32, 64, 100}
Number of units (Dense)	{20, 30}
Number of units (Attention)	{128}
L_2 Regularisation (λ)	{0.0, 0.01, 0.1}
T_w	{15, 20, 30, 40}

Table 3.2: Hyperparameter values evaluated in the proposed methodology.

For each hyperparameter setup, we start from a base network of the form LSTM(32) + Dropout(0.5) + ReLU(Dense(30)) + Dropout(0.1) + ReLU(Dense(20)) + Dense(1). In this notation, a layer in the network is denoted as Activation(Layer(Units)), dropout [201] layers are introduce to control overfitting and are denoted as Dropout(Rate). In all experiments, the base network is trained for

200 epochs with a learning rate of 0.001, batch size of 256, a T_w of 30, for FD001 and FD003, and a T_w of 20 for FD002 and FD004. We point out that these base parameters were chosen based on previous architectures for the same datasets [140, 144]. During each hyperparameter search, we start from the base architecture and we consider the range of values presented in Table 3.2. We discuss the most sensitive hyperparameters, i.e., the number of LSTM neurons, the number of LSTM layers, the size of time window T_w and the batch size, and report the average RMSE on the test C-MAPPS datasets in Figure 3.3.

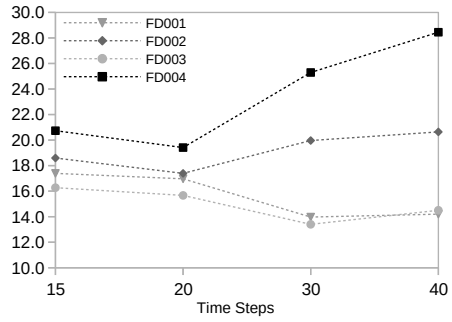
Time Window In our experiments, the time window T_w was the most sensitive hyperparameter. In the tests, we considered values of T_w in {15, 20, 30, 40}. Results in Figure 3.3a show that for FD001 and FD003, a T_w of 30 timesteps yields the best performing scores, i.e., 14.0 and 13.4, respectively. For the remaining two datasets, the lowest RMSE values are found for a T_w of size 20. FD002 has its lowest RMSE at 17.4 and FD004 at 19.4. Moreover, a change in the time window results in RMSE up to 28.4 for FD004 and 20.6 for FD002. We point out that in the test datasets, the shortest T_w length of FD001, FD002, FD003, FD004 are 31, 21, 38 and 19, respectively.

Batch Size The batch size has a small effect on the performance of the model but can still affect the training time of the algorithm. That is, a larger batch size can reduce the number of mini-batches needed to complete a single epoch. The results in Figure 3.3b show that increasing the batch size does not lead to a substantial change in RMSE performance. Across all datasets, a batch size of 256 results in the lowest average RMSE values.

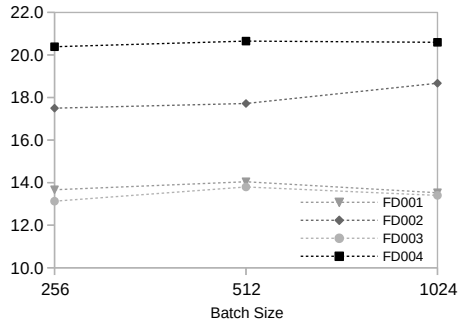
Number of LTSM Units The number of units (neurons) in the LSTM layer impacts datasets differently, as shown in Figure 3.3c. In most cases, increasing the dimension of the LSTM layers results in lower errors. However, in the FD004 dataset, 32 neurons result in the lowest RMSE at 19.5. On the other hand, the performance on FD003 seems to improve as we increase the number of units in the LSTM layer. Since we seek a general architecture with good performance, we select the number of LSTM units based on the overall best result across all four datasets in the final model.

Number of LTSM Layers We report the results of the number of LSTM layers in Figure 3.3d. We can observe that the number of layers has adverse impacts on the performance of the models, with more than one layer typically resulting in larger errors than a single LSTM layer in most cases. We note, however, that this observation is not consistent across all datasets. For instance, FD004 has improved performance with 2 LSTM layers with 100 nodes compared to a single one. Overall, adding more LSTM layers adding more layers added more computational burden

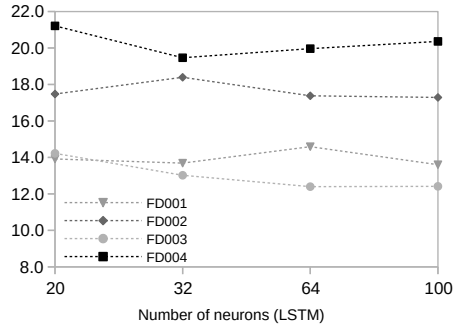
without further performance gains. In general, a single layer containing 100 neurons resulted in the best overall results across all datasets.



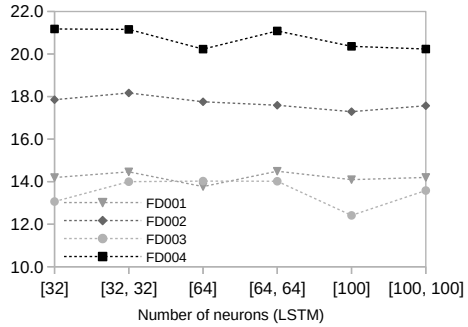
(a) Average RMSE on the test datasets for different values of T_w.



(b) Average RMSE on the test datasets for different values of batch size.



(c) Average RMSE on the test datasets for different number of neurons in the LSTM layer.



(d) Average RMSE on the test datasets for different number of LSTM layers and neurons.

Figure 3.3: Average RMSE on the test datasets for the models' hyperparameters neurons.

3.4.5 Training Parameters

Based on the results obtained from the previous section, we select the following architecture for all the studied datasets, LSTM(100) + Dropout(0.5) + Attention(128) + ReLU(Dense(30)) + Dropout(0.1) + ReLU(Dense(20)) + Dense(1). We point out that our goal is to evaluate a single architecture that can perform well for all datasets. However, better performance can be obtained if we search for specific hyperparameters for each dataset [144]. Similar to our hyperparameter search, we split the data into training and validation datasets containing 90% and 10% engines of the original training dataset. To reduce the effect of randomness in our final results, we report the average and standard deviation of 10 experiments.

The inputs features and RUL outputs are normalised individually according to Eq. (3.14), and dropout is introduced to control overfitting. We train the models for a maximum of 200 epochs and select mini-batches of 256 samples for gradient updates. We stop training if no improvement is seen for 20 epochs in the validation dataset. We train the network for the Adam algorithm and clip gradient norms to 1. We select the learning rate of 0.001 based on grid-search after the remaining parts of the architecture have been defined. Finally, T_w is 30, for FD001 and FD003, and 20 for FD002 and FD004.

We evaluate our models using the C-MAPPS testing datasets, where the goal is to predict the RUL of input sequences seen up to a point before failure. In our results, we use rectified labels based on the value $R_e = 125$ for training, validation and testing. That is, if $y_t > R_e$ then they are replaced with R_e. Since we train on normalised outputs, we multiply the outputs of our LSTM model by R_e to retrieve their original scale.

All our experiments are performed on an Intel Core i5 7th generation processor with 16 GB RAM and a GeForce GTX 1070 graphics processing unit. We implement the models using the Python 3.6 programming language using the Keras [44] library with the TensorFlow [158] backend.

3.5 Results

In this section, we present and compare the results using the proposed architecture against other methods in the literature. We present the performance of our architecture using the RMSE and scoring function. We also present the attention activations to visualise time-related features used for RUL prediction at each time step.

3.5.1 Prediction Performance

We implement two versions of the proposed architecture: one containing the attention layer (LSTM + A) and one without the attention mechanism (LSTM). Additionally, we present a comparison of the models to the state-of-the-art results in the C-MAPPS datasets using the RMSE and scoring performance in Tables 3.3 and 3.4.

We compare our model to other deep learning architectures trained on the same datasets. First, we compare to LSTM methods proposed by [144] (GA + LSTM) and [243] (LSTM) to test whether our LSTM implementation can offer any gains over previously implemented LSTM architectures. Our approach is similar to the one proposed in [243], but we do not learn to output an entire sequence of RUL estimates for each timestep of the input sequence, i.e., we learn from temporal inputs of size T_w, $\phi_t(\mathbf{x})$ to predict the RUL of the next time step y_{t+1}. Moreover, different from the previous LSTM models, we do not perform pre-clustering of operating conditions in datasets FD002 and FD004 nor perform unsupervised pretraining as

	RMSE				
Method	FD001	FD002	FD003	FD004	R_e
LSTM [243]	16.14	24.49	16.18	28.17	130
MODBNE [240]	15.04	25.05	12.51	28.66	-
CNN [140]	12.61	22.36	12.64	23.31	125
GA + LSTM [144]	**12.56**	22.73	**12.10**	22.66	115-135
LSTM (± StDev)	13.64 (± 0.80)	17.76 (± 0.43)	12.49 (± 0.29)	21.30 (± 1.06)	125
LSTM + A (± StDev)	13.95 (± 0.43)	**17.65** (± 0.47)	12.72 (± 0.73)	**20.21** (± 0.63)	125

Table 3.3: RMSE comparison between the proposed LSTM methods and other methods in the literature on the C-MAPPS datasets

	Scoring Function				
Method	FD001	FD002	FD003	FD004	R_e
LSTM [243]	338	4,450	852	5,550	130
MODBNE [240]	334	5,585	422	6,558	-
CNN [140]	274	10,412	284	12,466	125
GA + LSTM [144]	**231**	3,366	251	**2,840**	115-135
LSTM (± StDev)	300 (± 31)	**1,638** (± 192)	267 (± 42)	2,904 (± 979)	125
LSTM + A (± StDev)	320 (± 30)	2,102 (± 250)	**223** (± 17)	3,100 (± 576)	125

Table 3.4: Scoring function comparison between the proposed LSTM methods and other methods in the literature on the C-MAPPS datasets

reported in [144]. We also compare to the CNN method proposed in [140] and the multiobjective deep belief networks ensemble (MODBNE) proposed by [240] all of which reported high-performance results in the studied datasets.

In Table 3.3, we present the results of the proposed model using the RMSE performance. As it can be observed, our model achieves lower RMSE values for datasets FD002 and FD004, i.e., in datasets with more operating conditions. Our model results in 21% (FD002) and 11% (FD004) relative improvement over the best-reported RMSE. For the remaining datasets, our models are worse than the ones reported in [144] but still comparable, with an 8% (FD001) and 3% (FD003) performance reduction. We point out that unlike the GA + LSTM method, we do not use heuristic search to select the best-performing hyperparameters for each dataset and thus expect that the single architecture may not yield the best results without tailoring its hyperparameters.

We compare our models in more detail with LSTM [243] and GA + LSTM [144]. In the first, the architecture proposed is similar to our model; however, several differences are present. In our implementation, we do not pre-cluster the operating conditions on datasets FD002 and FD004. Moreover, we train our model over time windows of the input sequence, as seen in Section 3.4.4, tuning the time-window

size leads to the most improvement. When compared to GA + LSTM, our method differs on the lack of pretraining of the network and in the overall final architecture, including activation functions. Furthermore, we use a much smaller time window for training. Lastly, different from both methods, we normalise both inputs and outputs to the 0-1 range, which helps to stabilise learning. After weight optimisation, the output is multiplied by R_e to recover the original (rectified) values. As shown in Tables 3.3 and 3.4, the rectified RUL values are similar to other others in the literature and match the ones proposed in [140].

Notably, the architecture results in higher performance gains for FD002 and FD004 than the other datasets. We have noted that including the raw operating conditions features in the model, selecting the correct time window for propagating gradients in the network and increasing the number of hidden neurons in the LSTM layers led to the most benefits for these datasets. Different from previous approaches, using non-clustered operational settings leads to better predictions in our proposed architecture. Moreover, increasing the number of hidden neurons increases the neural network's capacity to extract more complex features of the multiple operating conditions of FD002 and FD004. We argue that this modification also improves performance in the attention model as the attention mechanism can attend to a more complex hidden representation of the input space before a prediction.

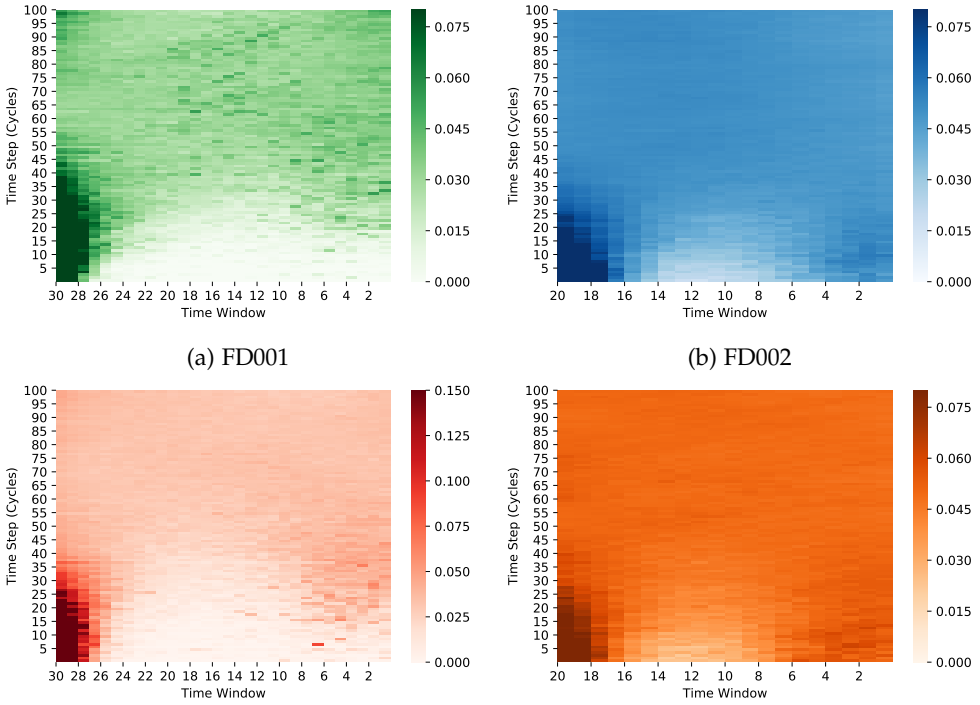
In Table 3.4, we observe that our method can achieve much lower scoring values for dataset FD002 yielding a 52% relative improvement over the best-reported results. In our tests, the average observed rectified RUL of FD002 is 73.69 while the predicted values from our method average at 71.34. These results do not necessarily translate in the same magnitude as the RMSE results because the RMSE weighs both positive and negative errors equally and averages the errors. On the other hand, the scoring function is summed across all test engines, and any positive deviation can cause major changes in its value. We can observe that results present similar performance to previously proposed methods with the attention model, even though we do not optimise for the scoring metric.

3.5.2 Attention Weights

The model combined with an attention layer has achieved similar performance to the models containing just LSTM and fully connected layers. A small improvement in RMSE can be achieved for FD004 and FD002. While for the scoring function, the LSTM + A model only outperforms the original LSTM model for the FD003 dataset. We expect attention to be beneficial when generating long sequences in which outputs may depend on a different part of the input sequence. In our prediction task, the output is the RUL at the next step, and the hidden vectors at time t are expected to learn most of the relevant information for predicting this next step. Nonetheless, adding the attention mechanism benefits the more complex datasets with more op-

erating conditions and fault modes. The data in FD002 and FD004 are more diverse and have different sensor value variations of the multiple operating conditions and fault modes. In this case, we argue that the attention network is given a chance to learn these more diverse temporal relationships and attend to different parts of the input relevant to different engines under multiple conditions.

The attention weights in Eq. (3.9) can be retrieved to observe the importance the networks give to each time step of the context vector. This can help us interpret which timesteps the network focuses on to make the RUL predictions for the next time step. It is important to notice that in our attention mechanism, the network has access to both a context vector and the last hidden state h_t. This architecture choice has the effect of forcing the network to attend to parts different from the last timestep before a prediction, which we expect to be the most relevant hidden state for the prediction at the next timestep.



(a) FD001

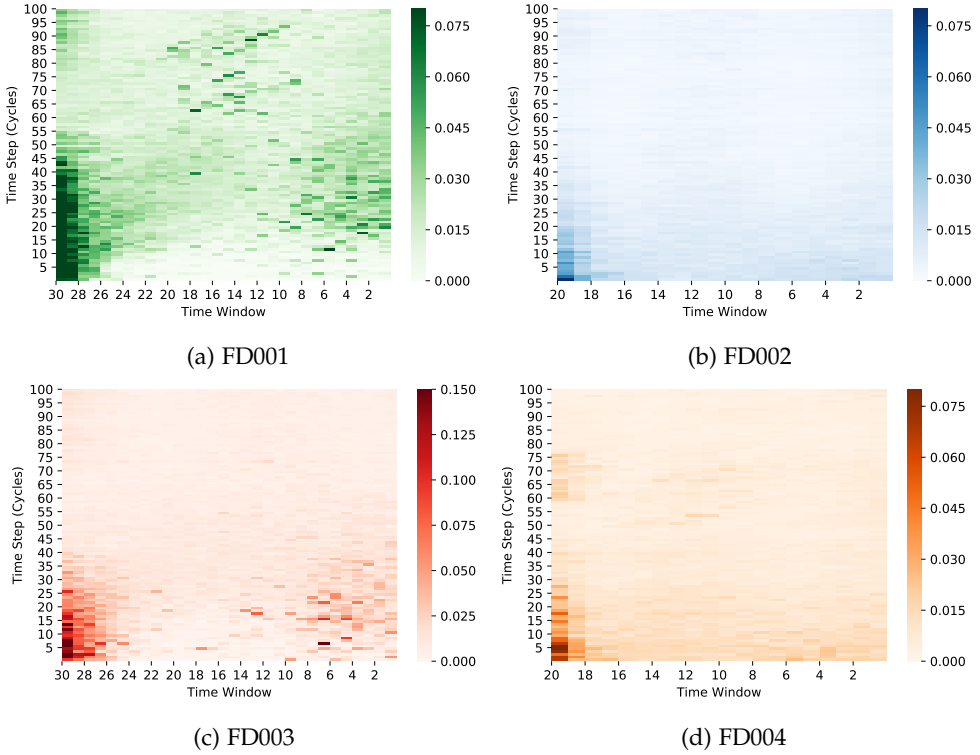
(b) FD002

(c) FD003

(d) FD004

Figure 3.4: Average attention weights starting 100 time steps before a failure for validation examples in each dataset.

In Figures 3.4 and 3.5, we present the average and standard deviations of attention weights over validation examples starting 100 timesteps before a failure. In the figures, the vertical axis corresponds to a timestep, with the topmost value cor-



(a) FD001

(b) FD002

(c) FD003

(d) FD004

Figure 3.5: Standard deviation of attention weights starting 100 time steps before a failure for validation examples in each dataset.

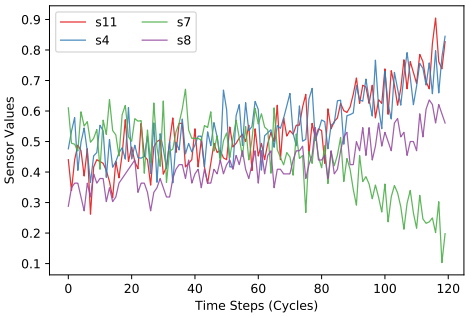
responding to 100 timesteps before a failure and the horizontal axis representing the size of the time window, with the leftmost side representing the farthest time step from the current RUL prediction, i.e., x_{t-T_w+1}. Thus, each row in the graphs corresponds to the average and standard deviation of the attention weights of the validation examples for each timestep in the time window.

We observe that at the start (top), the network does not focus on specific temporal parts of the inputs to make a prediction, assigning similar attention weights to all previous timesteps. In general, Figure 3.4 shows that as time progresses and the predictions approach the end of lifetime, attention is shifted to either the beginning of the time window (farther away from the prediction time on the left) or to the last timesteps (closer to the prediction time on the right). This is can be seen in more details for FD001, in Figures 3.4a and 3.5a, and for FD003 in Figures 3.4c and 3.5c. In these cases, at the end of the lifetime, the average attention weights shift towards the beginning of the time window, i.e., further away in the past. Nonetheless, the standard deviation values show that attention is shifted towards the end of the time window, closer to the current timestep. As it can be observed in Figures 3.4b

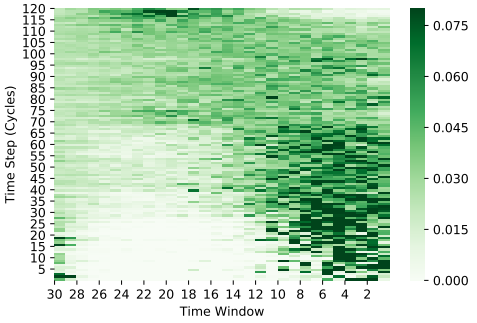
and 3.4d, the attention weights of FD002 and FD004, present a similar pattern with attention shifting towards the beginning of the time window as engines degrade. However, weights are more uniform across the other timesteps in comparison to FD001 and FD003.

We also present two specific examples, one for an engine of FD001 in Figure 3.6 and one for an engine of FD004 in Figure 3.7. In the figures, we present selected sensor values, the learned attention weights and the RUL prediction at each time step. In Figure 3.6, we observe how attention weights shift towards the end of the time window as sensors start to show a degradation trend. This indicates that the network can attend to the changing values in the input. We also notice that the learned attention weights are lower in the central part of the time window as degradation increases. As the trend and slope of the sensor sequences are important for degradation estimation, the network learns to focus both on older and more recent values, indicating that it is focusing on how much the sensors are changing in the given time window. In Figure 3.7, we observe that the sensors in Figure 3.7a are more irregular and tend to show smaller changes of slope as degradation increases. Nonetheless, the learned attention weights in Figure 3.7b show that the network has learned to focus its attention on similar parts of the input as time progresses, mostly focusing at the beginning and end of the time window as RUL approaches zero. However, the weights are more evenly spread across the time window. That is, the weights focus on information from different parts of the time window, possibly due to the spikes observed in the sensor data.

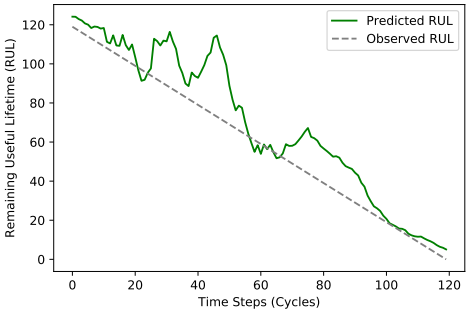
These results offer new insights into how the time-related features are used by the LSTM architecture while making RUL predictions. For example, a network containing attention mechanisms trained to identify faulty behaviour can be used to inspect input sensor values as time progresses. As it is hard to inspect all incoming sensor data visually, attention mechanisms could offer a visualisation method for fault prognostics and identification. In such cases, a temporal visual inspection would offer a representation as to when faulty behaviour starts. Such early warnings could be used to predict future failures in highly complex systems with a multitude of sensors.



(a) Sensor values over time.

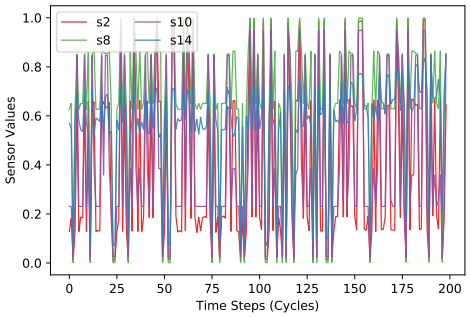


(b) Attention weights at each time step within the time window as degradation increases.

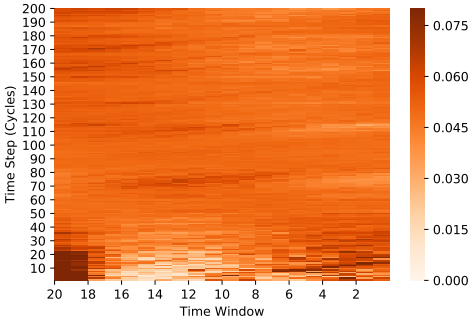


(c) Predicted and RUL targets.

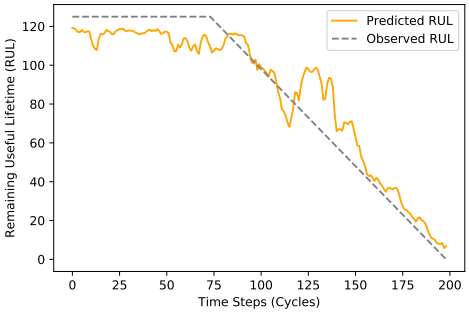
Figure 3.6: Sensor values, attention weights and Remaining Useful Lifetime predictions for example ID 8 in FD001.



(a) Sensor values over time.



(b) Attention weights at each time step within the time window as degradation increases.



(c) Predicted and RUL targets.

Figure 3.7: Sensor values, attention weights and Remaining Useful Lifetime predictions for example ID 32 in FD004.

3.6 Conclusions

In this chapter, we proposed an architecture based on long short-term memory (LSTM) for RUL prediction. Our architecture is enhanced with an attention mechanism that focuses on different parts of the sequential input relevant for the prediction. Moreover, the attention weights can be used to visualise the temporal relationships between inputs and predicted RUL outputs.

Our results show that the proposed methodology is competitive with other proposed methods for RUL predictions. We show the model's effectiveness in comparison to other state-of-the-art deep learning methods on the same task. The learned attention weights show that the LSTM network focuses on different parts of the temporal input while predicting which part of the degradation cycle. Our results show that the learned attention mechanism can be used to visualise the trained weights and offer better insights into the relationship between temporal inputs coming from sensors and the output RUL predictions.

Appendix

3.A Appendix A: C-MAPSS datasets

The Commercial Modular Aero Propulsion System Simulation (C-MAPSS) datasets are engine degradation simulation datasets. Four different datasets are simulated under different combinations of operating conditions and fault modes. Each dataset records several sensor channels and fault evolutions. These datasets are provided by the Prognostics Center of Excellence at the American National Aeronautics and Space Administration's (NASA) website [167].

The C-MAPSS simulates an engine model of the 90,000 lb thrust class (see Figure 3.8) and atmospheric models capable of simulating (i) altitudes from sea level to 40,000 ft, (ii) Mach from 0 to 0.9 and (iii) sea-level temperatures from -60 to 103 °F [191]. The C-MAPSS also allows for a power management system that allows the engine to be operated over a wide range of thrust levels throughout the full range of flight conditions. Moreover, the control system consists of a fan speed controller, regulators and limiters. The limiters include high-limit regulators that prevent the engine from exceeding its designed limits for core speed, engine-pressure ratio, and high-pressure turbine (HPT) exit temperature, and a limit regulator that prevents the static pressure at the high-pressure compressor (HPC) exit from being too low and acceleration limiters of the core speed [191]. The C-MAPSS can produce several outputs of the simulated engines.

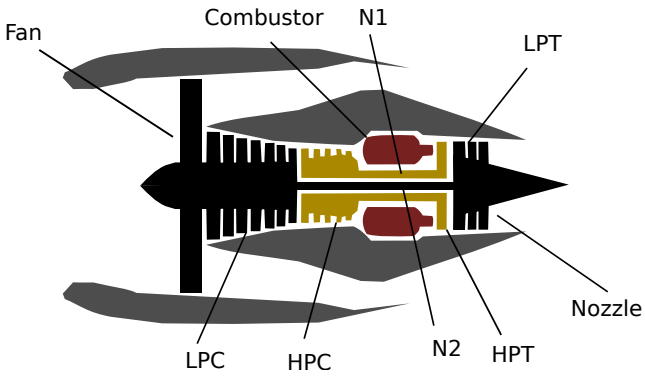


Figure 3.8: A schematic overview of an engine modelled via C-MAPSS. HPC: high-pressure compressor, LPC: low-pressure compressor, HPT: high-pressure turbine, LPT: low-pressure turbine, N1: low-pressure spool speed, N2: high-pressure spool speed. Adapted from [191].

Each of the four datasets (FD001, FD002, FD003 and FD004) has multiple time

series. Each time series corresponds to a different engine, i.e., each dataset has a set of engines of the same type. Each engine starts with different degrees of initial wear and manufacturing variation, which is unknown. This level of wear is considered normal, i.e., not faulty. There are three operational settings that have a substantial effect on engine performance. These settings are also included in the data. The data is also contaminated with sensor noise.

The engines are operating under normal conditions at the start of each time series and start to degrade at some point during the series. In the training sets, the degradation grows in magnitude until a predefined threshold is reached beyond which the machines are considered to have failed (run-to-failure data). In the test set, the time series end some time prior to complete degradation. The C-MAPPS datasets contain information coming from 21 sensors measurements as well as 3 operational settings. A detailed description of the sensor measurements is presented in [191]. The operating conditions in the datasets vary between one (sea level) in FD001 and FD003, to six, based on different combinations of altitude (0 - 42000 feet), throttle resolver angle (20 - 100) and Mach (0 - 0.84) in FD002 and FD004. Also, fault modes vary between one (HPC degradation) in FD001 and FD002 and two (HPC degradation and Fan degradation) in FD003 and FD004.

Chapter 4

Predicting Remaining Useful Lifetime under Varying Operating Conditions

---

In data-driven asset prognostics, sufficient prior observed degradation data is critical for remaining useful lifetime predictions. Most machine learning methods assume that training (source domain) and testing (target domain) condition monitoring data have similar distributions and feature representations. However, due to different operating conditions, fault modes and noise, distribution and feature shift exist across different domains. In the best case, this shift reduces the performance of predictive models when no run-to-failure data is available for re-training in the target domain. In the worst case, it is necessary to wait for run-to-failure information to train new methods. To address these issues, this chapter focuses on a data-driven approach for domain adaptation in prognostics to adapt remaining useful life estimates to a target domain, containing sensor information from remote monitoring devices but no data about the remaining life of assets.

This chapter is based on [51].

4.1 Introduction

Prognostics and health management (PHM) is aimed at increasing reliability, availability and reducing maintenance costs of assets [10]. In PHM, remaining useful lifetime (RUL) relates to the amount of time left before a piece of equipment is considered not to perform its intended function. Therefore, accurate RUL prognostics, i.e., predicting point-estimates of the remaining lifetime of assets, enable the interested parties to assess an equipment's health status and to plan future maintenance actions, e.g., logistics of personnel, spare parts and services [176].

With abundant conditioning monitoring data, machine learning methods can be used to learn degradation models directly from data. Unlike physics-based [47] and statistical-based [199] models, machine learning models in PHM can be applied without prior degradation knowledge [137]. Specifically, neural networks have been receiving much attention given their ability to approximate high dimensional nonlinear functions directly from raw data [70]. Moreover, several deep neural network architectures built to support temporal inputs have been successfully applied to prognostics problems, e.g. recurrent neural networks (RNN), long short-term memory (LSTM) [144, 243] and gated recurrent units (GRU) [232].

However, in classical machine learning, models need enough annotated historical data to achieve a significant performance level [140, 243]. Presumably, interested parties already adopt time-based maintenance policies and observing run-to-failure data is difficult. To overcome this problem, methods have to find ways to handle censored data [98] or use simulated data, which may lead to imperfect models that do not represent real-world scenarios. Even when enough run-to-failure data are available, algorithms trained for one specific task cannot be generalised to different but related tasks. For example, an algorithm trained for prognostics for a particular fault mode often does not generalise well to other fault modes under similar machinery conditions [137].

To address these issues, learning models trained with specific run-to-failure data have to adapt to data with different input features, data distributions and limited fault information, i.e., different *domains*. In machine learning, this situation is referred as *domain adaptation* [174], a sub-problem of transfer learning. Several algorithms were proposed to different flavours of the domain adaptation problem [71, 75, 114, 214]. More recently, adversarial deep neural networks have shown strong performance in domain adaptation in fault diagnosis and other complex tasks, e.g., computer vision. However, many adaptation methods are less suitable for multivariate sensor data as they do not consider temporal dependencies of sequential data. As a result, general deep domain adaptation methods are hardly applicable to common scenarios in RUL prognostics.

In this chapter, we propose to use LSTMs [97] to address the problem of learning from temporal data across related domains with different operating conditions and fault modes. We learn from a *source* domain with run-to-failure annotated data and

a *target* domain containing only sensor data, i.e., we consider an *unsupervised* domain adaptation problem. We perform adversarial learning similar to [75] to learn common domain-invariant feature representations in an RUL regression task. To the best of our knowledge, we are the first to focus on unsupervised domain adaptation for PHM under the given setup. To showcase the efficacy of the proposed algorithms, we use the NASA Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) turbofan degradation datasets [191]. We present the results of the proposed method against other adapted and non-adapted models in predicting the RUL of aircraft engines.

Contributions and Organisation We summarise the main contributions of this chapter as follows:

1. We propose a novel methodology for RUL prediction that can handle feature distribution shift across domains under different operating conditions and fault modes.

2. Unlike most unsupervised domain adaption methods in PHM, we incorporate heterogeneous temporal data from multiple sensors in an RUL estimation task.

3. We show in our experiments that the proposed method improves prognostics predictions on unlabelled target data, i.e., without observed RUL values, compared to non-adapted methods.

The remainder of this chapter is structured as follows. In the next section, we briefly discuss previous works in deep learning for prognostics and domain adaptation. In the subsequent section, we present our model detailing the learning algorithm and architecture. In Section 4.4, we describe the experimental design used in this paper. In Section 4.5, we present the learning procedure and detail the choice of model hyperparameters. Lastly, in Section 4.6, we compare and contrast the performance of proposed methods and provide an analysis of the results.

4.2 Related Literature

Deep Learning for RUL Prediction In PHM, several artificial intelligence methods have been proposed to predict the RUL of engineering assets [26, 88, 210, 244]. In special, feed-forward neural networks (FFNN) have drawn special attention given their ability to approximate complex functions directly from raw data [106, 135]. However, in many PHM applications, multi-dimensional temporal data is present. Architectures such as RNNs are a natural fit as their recurrent structure can handle temporal input data. However, RNNs have limitations when learning long-term dependencies due to vanishing gradient issues [25]. To address these issues LSTM

[97] and GRU [43] networks were introduced. Such networks have internal gates that enable them to preserve their memory state over a longer period and reduce vanishing gradient problems.

In particular, LSTMs have shown strong performance in RUL prognostics. [237] showed that LSTMs could outperform RNNs, GRUs and Adaboost-LSTM for an RUL prediction. [232] showed similar results by training an LSTM after extracting dynamic difference features. [243] showed that a sequence of LSTM layers could outperform other architectures, including convolutional neural networks (CNN). Recently, [144] showed that restricted Boltzmann machines could be used for unsupervised pretraining. In this two-stage method, a parameter optimised LSTM yields high performance in the C-MAPPS datasets outperforming previous architectures [140].

Other deep architectures have been proposed for PHM applications. For instance, [11] proposed a 2D CNN based and showed competitive results when compared to an FFNN, a relevance vector regression and support vector regression. [140] proposed a 1D CNN that show competitive results on the C-MAPPS datasets with lower training times when compared to recurrent models. However, vanilla CNNs do not maintain long-term temporal dependencies and are less suitable for long sequences of varying lengths.

Deep Domain Adaptation Methods Most prognostics studies have assumed access to enough run-to-failure information. Assumptions also included training (source) and future (target) data coming from the same distribution and feature space [11, 140, 144, 237, 243]. However, in real-life PHM scenarios, RUL values may be absent and come from different marginal distributions. Examples include data coming from different devices in varied operating conditions.

In unsupervised domain adaptation, algorithms are built to handle *domain shift* in the distribution and feature spaces [174]. Initial methods in unsupervised adaptation attempted to re-weight source example losses to reflect the target distribution [105, 114]. However, re-weighting often assumes a restricted form of domain shift and selection bias. On the other hand, subspace alignment methods attempt to learn a mapping function that aligns the source and target in a different subspace [71]. Maximum mean discrepancy (MMD) [82] based methods (e.g., transfer component analysis (TCA) [173] and joint distribution adaptation (JDA) [149]) can be interpreted as moment matching methods over first-order statistics using kernel tricks. Similarly, CORrelation ALignment (CORAL) [202] attempts to align the second-order statistics between source and target domains.

In classification, domain-adapted neural networks have outperformed previous methods. In general, methods have attempted to restrain the target error by the source error plus a notion of the discrepancy between the source and the target domains [22]. For example, [148, 215] incorporate MMD metrics in the loss function to reduce domain discrepancy. Similarly, [203] propose to incorporate a CORAL loss

function for the same purpose. Another approach, based on [22], uses a proxy of the $\mathcal{H}$-divergence to minimise the difference between source and target domains to directly find common representations that reduce the difference across domains [75]. Adversarial learning [81] methods have also shown high performance on domain adaptation [75, 180, 214]. For example, [214] proposes to pre-train a classifier in the source domain task and a discriminator to learn a target representation. Domain adversarial neural networks (DANN) [75] aims at finding a common feature space where source and target domain distributions are minimised via a loss inspired by the theory in [22]. More important, DANN can outperform metric minimisation methods while requiring a simple regularisation term based on a classification loss function. Similar to our work, [180] proposed an adversarial method using variational recurrent autoencoders and showed promising results in a classification task using healthcare data.

In regression, the theory in [46] presented point-wise loss guarantees in domain adaptation for a broad class of kernel-based regularisation algorithms. The results are based on the more general discrepancy distance that is tailored to compare distributions in adaptation with arbitrary loss functions [156], building upon the theory for 0-1 loss functions in [23]. Other works proposed to factorise a multivariate density into a product of bi-variate copula functions to identify independent changes between domains [150]. Recently, [169] have introduced a domain-invariant partial-least-squares regression using a domain regulariser to align source and target distributions in a latent space. However, few works have attempted to perform domain adaptation in regression when the input data are composed of temporal multivariate data [9].

Domain Adaptation in Prognostics In PHM, most methods for domain adaptation have focused on classifications tasks. On classic unsupervised domain adaptation, [234] have achieved better cross-domain classification results using TCA for gearbox fault diagnosis. In neural networks, [152] proposed a deep model where the MMD metric is employed to reduce domain discrepancy in fault diagnosis. A CNN approach based on wide first-layer kernels is proposed to perform adaptation in the presence of noisy data [241]. Similarly, [142] have proposed a muti-kernel MMD regularisation CNN. Recently, [141] proposed a deep generative model to generate fault target data using a labelled source domain and an unlabelled target domain resulting in better classification performance.

Relevant to our work, [238] proposed a *supervised* domain adaptation approach by exploiting labelled data from the target domain using the C-MAPPS datasets. The method aims to "fine-tune" a bi-directional LSTM previously trained on the source domain. Unlike our approach, this method requires target labelled RUL data and does not optimise towards common feature representations. In our work, we build upon previous works using LSTMs for RUL prediction [144, 232, 237, 238, 243]. To handle domain shift, we perform *unsupervised* domain adaptation from a

labelled source domain (containing observed RUL values) to an unlabeled target domain. We use a gradient reversal layer (GRL) [75] to learn domain-invariant features via adversarial learning (see Section 4.3.5). We validate our method on the C-MAPPS datasets using source-target pairs under different fault modes and operating conditions.

4.3 LSTM Domain Adaptation Network

In this section, we detail our domain adaptation model to predict the RUL of assets across domains with different fault modes and operating conditions. We first introduce the problem and the notations used in the paper and then further discuss the proposed method and its components.

4.3.1 Problem Definition

We denote a source domain data $D_{\mathcal{S}} = \{(\mathbf{x}^i_{\mathcal{S}}, \mathbf{y}^i_{\mathcal{S}})\}_{i=1}^{N_{\mathcal{S}}}$, containing $N_{\mathcal{S}}$ training examples, where $\mathbf{x}^i_{\mathcal{S}}$ belongs to a feature space $\mathcal{X}_{\mathcal{S}}$ and denotes a multivariate sequential data of length T_i and $q_{\mathcal{S}}$ features, i.e., $\mathbf{x}^i_{\mathcal{S}} = [x_t]_{t=1}^{T_i} \in \mathbb{R}^{q_{\mathcal{S}} \times T_i}$. Moreover, $\mathbf{y}^i_{\mathcal{S}} \in \mathcal{Y}_{\mathcal{S}}$ denotes RUL values of length T_i with $\mathbf{y}^i_{\mathcal{S}} = [y_t]_{t=1}^{T_i} \in \mathbb{R}_{\geq 0}^{1 \times T_i}$. Where for each $t \in \{1, 2, ..., T_i\}$, $x_t \in \mathbb{R}^{q_{\mathcal{S}}}$ and $y_t \in \mathbb{R}_{\geq 0}$ represent the t-th measurement of all variables and RUL labels, respectively. Similarly, we assume a target domain data $D_{\mathcal{T}} = \{\mathbf{x}^i_{\mathcal{T}}\}_{i=1}^{N_{\mathcal{T}}}$, where $\mathbf{x}^i_{\mathcal{T}} \in \mathcal{X}_{\mathcal{T}}$ and $\mathcal{X}_{\mathcal{T}} \in \mathbb{R}^{q_{\mathcal{T}} \times T_i}$ but no labels are available. We assume $D_{\mathcal{S}}$ and $D_{\mathcal{T}}$ are sampled from distinct marginal probability distributions, and have different feature spaces.

Our goal is to learn a function g_θ with learnable parameters θ such that we can approximate the corresponding RUL in the target domain examples at testing time, i.e., $\mathbf{y}^i_{\mathcal{T}} \approx g_\theta(\mathbf{x}^i_{\mathcal{T}})$. Clearly, our assumption is that the true mapping between input-output pairs is somewhat similar across domains for adaptation to be possible. At training time, we have access to source training examples and their real-valued labels $D_{\mathcal{S}}$ and we assume access to training examples from the target domain $D_{\mathcal{T}}$, i.e., an unsupervised domain adaptation setup. We assign a domain label $d^i \in \{0, 1\}$ to each i-th training example to indicate the domain it originates.

4.3.2 Time Windows Processing

To handle different sequence lengths in different domains, we employ a sliding time window approach. Let $\mathbf{x} = [x_t]_{t=1}^{T_i}$, where T_i denotes the size of each sequence length. We define a function ϕ_t that divides each sequence of size T_i in sequential time windows of size T_w, i.e., $\phi_{t-1}(\mathbf{x}) = [x_{t-T_w}, ..., x_{t-1}]$. After the transformation, previous sensor data within the time window $[x_{t-T_w}, ..., x_{t-1}]$ are collected to form an input vector to predict y_t, where $t > T_w$. If $T_i \leq T_w$ we apply zero-padding

(pad the sequence with zeros) on the left side of $\mathbf{x}^i$ until T_i has size $T_w + 1$. This ensures that each original time series will have $n_i = T_i - T_w$ training samples. We define as $\bar{N}_\mathcal{S}$ and $\bar{N}_\mathcal{T}$ the updated number of examples after the transformation, i.e., $\bar{N}_\mathcal{S} = \sum_{i=1}^{N_\mathcal{S}} n_i$ and $\bar{N}_\mathcal{T} = \sum_{i=1}^{N_\mathcal{T}} n_i$. In a slight abuse of notation, we assume that the labels of each complete time series are broadcasted to all inputs after the time window transformation, where $d^i = 0$ for $i \in \{1, \ldots, \bar{N}_\mathcal{S}\}$ represents inputs from the source domain and $d^i = 1$ for $i \in \{1, \ldots, \bar{N}_\mathcal{T}\}$ representing samples from the target domain. We maintain T_w fixed across source and target domains to allow the same number time steps influence RUL predictions.

4.3.3 Long Short-Term Memory Neural Network

LSTMs have been proposed to accommodate temporal relationships between inputs and outputs in PHM when enough training data is available [11, 140, 144, 237, 243]. Such networks offer recurrent connections to encode temporal data and control how information flows within its cells by updating a series of gates. In our proposed

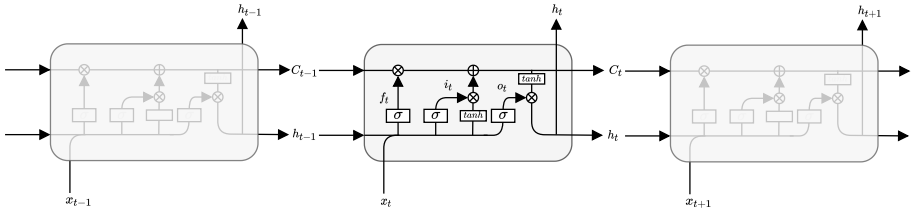


Figure 4.1: LSTM memory cell. Adapted from [170].

model, we use LSTM layers to extract temporal features from the time window of size T_w. In our implementation, the memory cell (Figure 4.1) consists of three non-linear gating units that update a cell state $C_t \in \mathbb{R}^u$, using a hidden state vector $h_{t-1} \in \mathbb{R}^u$ and inputs $x_t \in \mathbb{R}^q$ of the form:

$$f_t = \sigma\left(W_f x_t + R_f h_{t-1} + b_f\right) \tag{4.1}$$

$$i_t = \sigma\left(W_i x_t + R_i h_{t-1} + b_i\right) \tag{4.2}$$

$$o_t = \sigma\left(W_o x_t + R_o h_{t-1} + b_o\right) \tag{4.3}$$

where σ is the sigmoid activation function responsible for squeezing the output to the 0-1 range, $W_s \in \mathbb{R}^{u \times q}$ are the input weight matrices, $R_s \in \mathbb{R}^{u \times u}$ are the recurrent weight matrices, and $b_s \in \mathbb{R}^u$ are bias vectors. The subscript s can either be the forget gate f, input gate i or the output gate o, depending on the gate.

After computing f_t, i_t and $o_t \in \mathbb{R}^u$, the new cell state $\widetilde{C}_t$ candidate is computed as follows:

$$\widetilde{C}_t = \tanh\left(W_C x_t + R_C h_{t-1} + b_C\right) \tag{4.4}$$

Similar to the gate operations: $W_C \in \mathbb{R}^{u \times q}$, $R_C \in \mathbb{R}^{u \times u}$, and $b_C \in \mathbb{R}^u$. The previous cell state C_{t-1} is then updated to the new cell state C_t:

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \tag{4.5}$$

with $\odot$ denoting the element-wise multiplication.

In the previous equations, the forget gate f_t is responsible for deciding which information will be thrown away from the cell state. Next, the input gate i_t decides which states will be updated from a candidate cell state. The input and forget gates are then used to update a new cell state for the next time step. Lastly, the output gate o_t decides which information the cell will output and the new hidden state h_t is computed by applying a tanh (hyperbolic tangent) function to the current cell state times the output gate results.

$$h_t = o_t \odot \tanh(C_t) \tag{4.6}$$

4.3.4 Domain Adversarial Neural Networks

Unlike other domain adaptation methods, domain adversarial neural networks (DANN) [75] aims at combining domain adaptation and feature learning in one training procedure, such that predictions can be made on features that are both discriminative and invariant to the domains. This way, learned weights could be applied directly to the target domain without having its predictive quality degraded by domain shift. DANN achieves this goal by relying on the notion of $\mathcal{H}$-divergence [22, 23] measuring the distance between two domain distributions.

Definition 4.1 ([22, 23]). *Given two domain distributions $D_{\mathcal{S}}^{\mathcal{X}}$ and $D_{\mathcal{T}}^{\mathcal{X}}$, and a hypothesis class $\mathcal{H}$ being the set of binary classifiers $h : \mathcal{X} \to \{0, 1\}$, the* $\mathcal{H}$-divergence *between $D_{\mathcal{S}}^{\mathcal{X}}$ and $D_{\mathcal{T}}^{\mathcal{X}}$ is*

$$d_{\mathcal{H}}(D_{\mathcal{S}}^{\mathcal{X}}, D_{\mathcal{T}}^{\mathcal{X}}) = 2 \sup_{h \in \mathcal{H}} \left| P_{\mathbf{x} \sim D_{\mathcal{S}}^{\mathcal{X}}}\big(h(\mathbf{x}) = 1\big) - P_{\mathbf{x} \sim D_{\mathcal{T}}^{\mathcal{X}}}\big(h(\mathbf{x}) = 1\big) \right|.$$

In words, the $\mathcal{H}$-divergence relies on the capacity of the hypothesis class to distinguish between domains [75]. [22] showed that for a symmetric hypothesis class $\mathcal{H}$, one can calculate the empirical $\mathcal{H}$-divergence $\hat{d}_{\mathcal{H}}$, by measuring the divergence between two samples $D_{\mathcal{S}} \sim D_{\mathcal{S}}^{\mathcal{X}}$ and $D_{\mathcal{T}} \sim D_{\mathcal{T}}^{\mathcal{X}}$ of the marginal distributions as

$$\hat{d}_{\mathcal{H}}\left(D_{\mathcal{S}}, D_{\mathcal{T}}\right) = 2\left(1 - \min_{h \in \mathcal{H}} \left[\frac{1}{N_{\mathcal{S}}} \sum_{i=1}^{N_{\mathcal{S}}} \mathbb{1}[h(\mathbf{x}_{\mathcal{S}}^i) = 0] + \frac{1}{N_{\mathcal{T}}} \sum_{i=1}^{N_{\mathcal{T}}} \mathbb{1}[h(\mathbf{x}_{\mathcal{T}}^i) = 1] \right]\right) \tag{4.7}$$

where $\mathbb{1}[a]$ is the indicator function which is 1 if argument a is true, and 0 otherwise.

[22] shows that even if $\hat{d}_{\mathcal{H}}$ is too hard to compute, we can approximate it by learning a classifier on the problem of discriminating between source and target

examples. Then, the risk of the classifier trained on this problem approximates the minimisation part of Equation (4.7) [23]. Given an empirical risk ϵ on this domain classification problem, the $\mathcal{H}$-divergence is then approximated by the proxy $\mathcal{A}$-distance [23]:

$$\hat{d}_{\mathcal{A}} = 2(1 - 2\epsilon). \tag{4.8}$$

The $\mathcal{A}$-distance, for which the above equation is a proxy, can be expressed as $d_{\mathcal{A}}(D_{\mathcal{S}}^{\mathcal{X}}, D_{\mathcal{T}}^{\mathcal{X}}) = 2 \sup_{a \in \mathcal{A}} |P_{D_{\mathcal{S}}^{\mathcal{X}}}(a) - P_{D_{\mathcal{T}}^{\mathcal{X}}}(a)|$, where $\mathcal{A}$ is a set of subsets of $\mathcal{X}$, is the same as the $\mathcal{H}$-divergence by choosing $\mathcal{A}$ to be the set represented by the binary characteristic function $h(\mathbf{x}) = 1$ [22, 23].

DANN works by estimating the minimisation part of Equation (4.7) by including a regularising term in a neural network loss function. It proposes to learn a trade-off between the source risk (main task) and the empirical $\mathcal{H}$-divergence. It attempts to control the $\mathcal{H}$-divergence by finding representations where both the source and the target domain are as indistinguishable as possible [75]. In other words, it ensures that the representation of the neural network contains less discriminative information about the domain of the inputs while inducing low approximation error on source examples.

To learn features that are at the same time discriminative and domain-invariant, the loss function of DANN is designed to include a regularisation factor that approximates the $\mathcal{H}$-divergence. First, source and target inputs are mapped through feature extraction layers that map inputs into a new space representation parametrised by θ_f. Prediction layers map this new space to an output of the original classification task parametrised by weights θ_y. Training such a neural network using a prediction loss $\mathcal{L}_y^i$ for the i-th example leads to the following optimisation problem:

$$\min_{\theta_f, \theta_y} \left[\frac{1}{N_{\mathcal{S}}} \sum_{i=1}^{N_{\mathcal{S}}} \mathcal{L}_y^i(\theta_f, \theta_y) + \alpha R(\theta_f) \right] \tag{4.9}$$

where $R(\theta_f)$ is a regularisation factor and α is a scalar.

Inspired by the proxy of the $\mathcal{A}$-distance, DANN works by using a domain classifier parameterised by θ_d and learns a logistic regressor $\mathcal{L}_d^i$ (see Equation (4.15)) modelling the probability that a given input belongs to the source or target domain. It then computes an approximation of the $\mathcal{H}$-divergence in Equation (4.7) as $2(1 - R(\theta_f))$, by including a regularisation term:

$$R(\theta_f) = \max_{\theta_d} \left[-\frac{1}{N_{\mathcal{S}}} \sum_{i=1}^{N_{\mathcal{S}}} \mathcal{L}_d^i(\theta_f, \theta_d) - \frac{1}{N_{\mathcal{T}}} \sum_{i=1}^{N_{\mathcal{T}}} \mathcal{L}_d^i(\theta_f, \theta_d) \right] \tag{4.10}$$

Combining Eqs. (4.9) and (4.10) gives rise to a minimax (adversarial) optimisation procedure. DANN proposed to learn this problem by stochastic gradient procedure, in which updates are made in the negative direction of the gradient for minimisation steps and the direction of the gradient for maximisation steps. Important

to training the algorithm is the gradient reversal layer (GRL). During the forward propagation, the GRL acts as an identity transformation. However, during back-propagation, the GRL takes the gradient from the subsequent level and changes its sign. The GRL is inserted between the feature extractor and the domain classifier. Therefore, running stochastic gradient descent (SGD) in the resulting architecture implements the adversarial updates (see Equations (4.18) - (4.20)) optimising towards a saddle point of the DANN loss function.

In our problem, the (empirical) source risk is computed for a regression loss instead of the 0-1 loss of [22, 23]. To approximate the 0-1 source risk with our regression risk we bound the labelling function $f : \mathcal{X} \to [0,1]$, where $\mathbf{y} = f(\mathbf{x})$ are the values that we aim to learn and employ a loss function $\mathcal{L}_y(g,f) = |g(\mathbf{x}) - f(\mathbf{x})|$, where $g : \mathcal{X} \to [0,1]$ [23]. Note that we relate functions g and h, by a shared mapping function z, where $g(\mathbf{x}) = (g \circ z)(\mathbf{x})$ and $h(\mathbf{x}) = (h \circ z)(\mathbf{x})$. In our implementation, we approximate the error of the classifier between the two domains as logistic regression, bound the labels of the source regression risk between $[0,1]$ and ensure that the initial parameters of the neural network module do not output values outside the bounds of the labelling function (see Section 4.3.5).

One can theoretically achieve better bounds on the domain adaptation task for regression by bounding the target risk using the empirical source risk $\hat{\mathcal{R}}_{D_\mathcal{S}}(g) = \frac{1}{N_\mathcal{S}} \sum_{i=1}^{N_\mathcal{S}} \mathcal{L}(g(\mathbf{x}^i), \mathbf{y})$ plus an approximation of the more general *discrepancy metric* [156], defined as $disc_\mathcal{L}(D_\mathcal{S}^\mathcal{X}, D_\mathcal{T}^\mathcal{X}) = max_{g,g' \in \mathcal{F}} |\mathcal{R}_{D_\mathcal{S}^\mathcal{X}}(g,g') - \mathcal{R}_{D_\mathcal{T}^\mathcal{X}}(g,g')|$, where $\mathcal{F}$ is a set of functions mapping $\mathcal{X}$ to $\mathcal{Y}$ and $\mathcal{R}_Q(g,g') = \mathbb{E}_Q[\mathcal{L}(g,g')]$ is the risk under distribution Q over $\mathcal{X}$ considering the loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$. The discrepancy metric includes the $\mathcal{A}$-distance as a special case under the 0-1 loss and $\mathcal{A} = \mathcal{H}\Delta\mathcal{H} = \{|h - h'| : h, h' \in \mathcal{H}\}$, i.e., the disagreement set between the two hypothesis [156]. Note that one can more explicitly minimise the regression objective using the discrepancy metric as

$$\min_{g \in F} \hat{\mathcal{R}}_{D_\mathcal{S}}(g) + disc_\mathcal{L}(D_\mathcal{S}, D_\mathcal{T}) \tag{4.11}$$

where approximating the maximisation in the empirical discrepancy measure, $disc_\mathcal{L}(D_\mathcal{S}, D_\mathcal{T})$ leads to a similar minimax problem as the one solved in DANN. For example, one can solve for g and g' the empirical approximation of the discrepancy measure $\text{disc}_\mathcal{L}$ as

$$\max_{g,g' \in \mathcal{F}} \frac{1}{N_\mathcal{S}} \sum_{i=1}^{N_\mathcal{S}} \mathcal{L}\big((g \circ z)(\mathbf{x}_\mathcal{S}^i), (g' \circ z)(\mathbf{x}_\mathcal{S}^i)\big) - \frac{1}{N_\mathcal{T}} \sum_{i=1}^{N_\mathcal{T}} \mathcal{L}\big((g \circ z)(\mathbf{x}_\mathcal{T}^i), (g' \circ z)(\mathbf{x}_\mathcal{T}^i)\big) \tag{4.12}$$

in combination with the objective in Eq. (4.11). However, even for fixed g, applying the minimax objective to the above equations can lead to numerical instability due to unbounded $g(\mathbf{x})$ in the maximisation. In our implementation, we approximate

the discrepancy metric employing as a proxy the empirical $\mathcal{H}$-distance, which upper bounds the $\mathcal{H}\Delta\mathcal{H}$-distance [22, 156], by learning a classifier to distinguish between domains. In the following sections, we show that this approximation allows us to adapt RUL predictions to the target risk learning from the bounded labelling function of the source domain.

4.3.5 LSTM Deep Adversarial Neural Network

Our model, referred as LSTM-DANN and depicted in Figure 4.2, is trained to predict for each input $\mathbf{x}$, real values $\mathbf{y}$ for the source domain, and domain labels $d \in \{0, 1\}$ for the source and target domains, respectively. Similar to [75], we use a DANN approach and decompose our learning method in three parts. Our feature extractor g_f, first decomposes the temporal inputs by a combination of LSTM layers mapping to a hidden state $h_{t-1} \in \mathbb{R}^u$. Later, it embeds the LSTM outputs via a fully-connected (FC) layer into a feature space $f \in \mathbb{R}^l$. We denote the vector of learnable parameters in this combination of layers as θ_f, i.e., $f = g_f(\phi_{t-1}(\mathbf{x}); \theta_f)$. This new feature space f is first mapped to an otherwise real-valued y_t label via a function $g_y(f; \theta_y)$ composed of FC layers with parameters θ_y. Lastly, the same feature vector f is mapped to a domain label d by a mapping function of FC layers $g_d(f; \theta_d)$ with parameters θ_d.

During training, we aim at minimising a source regression loss $\mathcal{L}_y^i$ using the observed RUL values from the source domain $\mathbf{y}_S^i$. Thus, the parameters of the feature extractor and regressor are optimised towards the same goal, i.e., minimising a regression loss. This ensures that features f are trained towards the main learning task. We also aim at finding features that are domain invariant, i.e., find a feature space f in which $D_S^{\mathcal{X}}$ and $D_T^{\mathcal{X}}$ are similar. To address this contrasting objective, we look at an auxiliary loss $\mathcal{L}_d^i$ over the domain classifier function $g_d(f; \theta_d)$. We want to estimate the dissimilarity between domains by inducing a high adversarial loss in features θ_f when the domain classifier $g_d(f; \theta_d)$ has been trained to discriminate between the two domains [75]. We consider the losses from here onwards to be the loss of each time step of the transformed time series.

We train the model in an adversarial procedure [81]. In the first pass, we learn features by minimising the weights of the feature extractor in the direction of the regression loss $\mathcal{L}_y^i$, between a prediction and a label y. In the second pass, we train the algorithm to maximise the same weights in the direction of a domain-classification loss that is being trained to minimise its overall domain classification error $\mathcal{L}_d^i$. In other terms, we define the model loss functions in terms of the learning function g and parameters $\theta = \{\theta_f, \theta_y, \theta_d\}$ and minimise the combined loss function

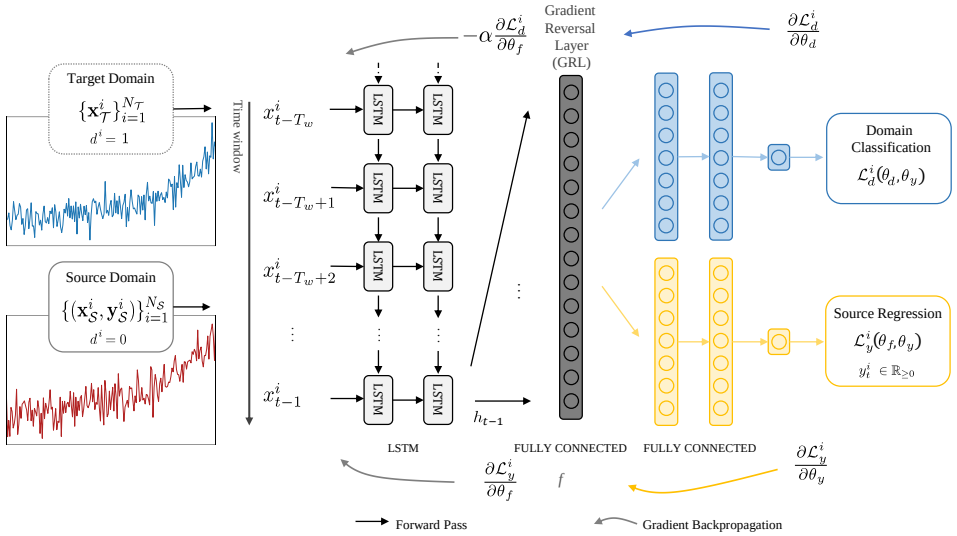


Figure 4.2: The proposed LSTM-DANN architecture (best seen in colours). Source and target temporal inputs are divided in sequential time windows of size T_w. The network's shared weights first extract temporal information from input features via LSTM layers (light grey). After, the last hidden state of the LSTM layer is passed to the feature map f (dark grey). The feature map f is mapped to two functions: a source regression loss on the source examples responsible for learning weights that predict the RUL at time t (yellow), and a domain classification loss on the source and target examples aimed at classifying domains (blue). A GRL is inserted in the connection between features f and the classifier. This layer inverts the sign of the gradient in the backward pass, making the weights of the feature extractor maximise towards the domain classification loss while the specialised weights minimise towards the same loss.

based on the empirical risk $\mathcal{L}$ expressed as:

$$\mathcal{L}(\theta_f, \theta_y, \theta_d) = \frac{1}{\bar{N}_S} \sum_{i=1}^{N_S} \sum_{t=T_w+1}^{T_i} \mathcal{L}_{y_t}^i(\theta_f, \theta_y) - \alpha \left(\frac{1}{\bar{N}_S} \sum_{i=1}^{\bar{N}_S} \mathcal{L}_d^i(\theta_f, \theta_d) + \frac{1}{\bar{N}_T} \sum_{i=1}^{\bar{N}_T} \mathcal{L}_d^i(\theta_f, \theta_d) \right) \tag{4.13}$$

where we write the l.h.s. of the empirical risk explicitly w.r.t. each timestep, and the losses $\mathcal{L}_{y_t}^i$, $\mathcal{L}_d^i$ are expressed as:

$$\mathcal{L}_{y_t}^i(\theta_f, \theta_y) = \left| \hat{y}_t^i - y_t^i \right|^p, \tag{4.14}$$

$$\mathcal{L}_d^i(\theta_d, \theta_y) = -\left(d^i \log \hat{d}^i + (1 - d^i) \log(1 - \hat{d}^i) \right), \tag{4.15}$$

respectively. In the equations, $\hat{y}_t$ is the RUL prediction at time t coming from the source domain, i.e., $\hat{y}_t^i = g_y(g_f(\phi_{t-1}(\mathbf{x}_S^i); \theta_f); \theta_y)$ and $\hat{d}^i$ is the domain prediction from source and target domains, i.e., $\hat{d}_i = g_d(g_f(\mathbf{x}^i; \theta_f); \theta_d)$. $\mathcal{L}_y^i(\theta_f, \theta_y)$ is a regression loss that when averaged takes the form of the mean absolute error (MAE) when $p = 1$ and the mean squared error (MSE) when $p = 2$. $\mathcal{L}_d^i(\theta_d, \theta_y)$ is the binary cross-entropy loss between domain labels d^i and α is a positive hyperparameter that weighs the domain classification loss during training.

We optimise the loss function $\mathcal{L}$ by searching for a saddle point solution $\hat{\theta}_f, \hat{\theta}_y, \hat{\theta}_d$, of the minimax problem below:

$$(\hat{\theta}_f, \hat{\theta}_y) = \arg\min_{\theta_f, \theta_y} \mathcal{L}(\theta_f, \theta_y, \hat{\theta}_d) \tag{4.16}$$

$$\hat{\theta}_d = \arg\max_{\theta_d} \mathcal{L}(\hat{\theta}_f, \hat{\theta}_y, \theta_d) \tag{4.17}$$

and update the learning weights in the network we use gradient updates [75] of the form:

$$\theta_f \leftarrow \theta_f - \lambda \left(\frac{\partial \mathcal{L}_{y_t}^i}{\partial \theta_f} - \alpha \frac{\partial \mathcal{L}_d^i}{\partial \theta_f} \right), \tag{4.18}$$

$$\theta_y \leftarrow \theta_y - \lambda \left(\frac{\partial \mathcal{L}_{y_t}^i}{\partial \theta_y} \right), \tag{4.19}$$

$$\theta_d \leftarrow \theta_d - \lambda \left(\alpha \frac{\partial \mathcal{L}_d^i}{\partial \theta_d} \right). \tag{4.20}$$

We use stochastic estimates of the updates in Equations (4.18) - (4.20) via SGD and its variants. The learning rate λ represents the learning steps taken by the SGD algorithm as training progresses. To achieve the desired updates, we use a GRL [75] alongside the gradient updates. The GRL makes it possible to learn the weights without many transformations of current implementations of the backpropagation algorithm in deep learning libraries.

4.3.6 LSTM-DANN Architecture

The architecture of our proposed network, depicted in Figure 4.3, is composed as follows. We implement two network architectures, one taking source training examples optimising towards the source regression loss and one taking the source and target examples optimising towards the adversarial objective of the domain classifier. The LSTM layers and FC layer are shared across the two networks. This implementation allows for two learning rates (λ_d, λ_y) that can be selected while training the algorithm. The number of LSTM layers and FC layers in the source

regression and domain classification networks are hyperparameters of the model. However, the number of FC layers in the feature extraction portion of the networks is fixed at one. Moreover, we use dropout [201] after each layer in the feature extractor, source regression and domain classification networks. Additionally, we apply L_2 regularisation on the weights θ_d of the domain classifier and θ_y of the source regression in Equation (4.13).

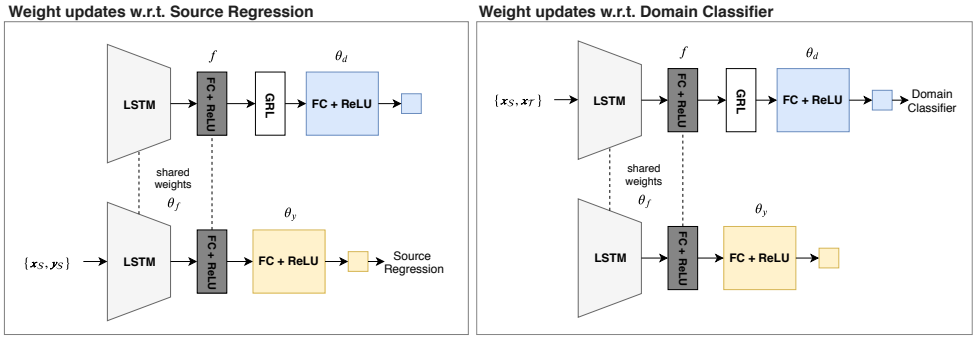


Figure 4.3: The architecture of the LSTM-DANN. LSTM layers and an FC layer are shared among two networks. During training, source examples and RUL values in the first network are passed to the network to update weights θ_f and θ_y to minimise the source regression loss. Source and target examples are passed to a second network aimed at optimising towards a domain classification loss. Weights θ_d are minimised towards the classification loss, while weights θ_f are maximised towards the same loss via the GRL.

In the source regression, both the shared weights θ_f and the source regression weights θ_y are minimised towards the regression loss in Equation (4.14). In the domain classification network, a GRL is inserted between the minimisation and maximisation portion of the network, i.e., between f and FC layers in the domain classifier. This way, feature extraction weights θ_f are minimised towards the regression loss in the first network and maximised over the minimised weights θ_y of the domain classifier network in the second pass. We continue this dual optimisation procedure until no further improvement is seen (see Section 4.5.1). An identity function maps the output of the source regression to the normalised RUL in the 0-1 range at each time step, whereas a sigmoid function squeezes the output of the domain classifier between 0-1. All FC layers, albeit those in the output layers, are followed by a rectified linear unit (ReLU) activation function in our implementation.

4.4 Design of Experiments

In this section, we describe the dataset details, the data prepossessing steps and the performance metrics used in our experiments.

4.4.1 C-MAPPS Datasets

The method is evaluated using the Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) [191] datasets containing turbofan engine degradation data. The C-MAPPS datasets are composed of four distinct datasets, each containing information coming from 21 sensors and three operational settings. Each dataset has several degradation engines split into training and testing examples.

Data	FD001	FD002	FD003	FD004
Engines: Training (N)	100	260	100	249
Engines: Testing	100	259	100	248
Operating Conditions (OC)	1	6	1	6
Fault Modes (FM)	1	1	2	2

Table 4.1: The C-MAPPS datasets. Each dataset contains a number of training engines (Engines: Training (N)) with run-to-failure information and a number of testing engines (Engines: Testing) with information terminating before a failure is observed. Operating Conditions: Each dataset can have one or six operating conditions, based on altitude (0 - 42000 feet), throttle resolver angle (20 - 100) and Mach (0 - 0.84). Fault Modes: each dataset can have and one (HPC degradation) or two (HPC degradation and Fan degradation) fault modes.

Engines in the datasets start with various degrees of initial wear but are considered healthy at the start of each record. As the number of cycles increases, the engines begin to deteriorate until they can no longer function. At this point, engines are considered unhealthy. Training datasets have run-to-failure information collected over the entire life until failure. Unlike training datasets, testing datasets contain temporal data that terminates some time before a system failure. The original task is to predict the RUL of the testing units [191].

In our experiments, we consider the case when enough run-to-failure data is available in a set of fault modes and operating conditions, but only sensor information is available in a different set, i.e., no observed failures. We consider each one of the datasets as source and target domains (only the training sets for training the algorithm) and perform domain adaptation on all source-target pairs. Operating conditions in the datasets vary between one in FD001 and FD003 and six in FD002 and FD004, and fault modes vary between one in FD001 and FD002 and two in FD003 and FD004. The details about the four datasets are presented in Table 4.1.

4.4.2 Data Preprocessing

Sensor values and operational settings data are used as input to our architecture. In our datasets, FD001 and FD003 have seven sensors with constant readings. However, as the constant readings are not consistent across the four datasets, we main-

tain these values in our analysis. Original distributions and feature values across datasets with the same fault modes and operating conditions can be similar. Thus, we need to ensure that enough distribution shift exists so that performing adaptation is desirable. To induce a higher discrepancy between domains and aid SGD updates, we normalise each dataset individually and scale inputs and RUL values to the (0-1) range using min-max normalisation:

$$\tilde{x}_t^{i,j} = \frac{x_t^{i,j} - \min(x^j)}{\max(x^j) - \min(x^j)} \tag{4.21}$$

where $x_t^{i,j}$ denotes the original i-th data point of the j-th feature at time t and x^j is the vector of all input examples and timesteps of the j-th feature.

It is reasonable to estimate RUL as a constant value when the engines operate in normal conditions [89]. Therefore, we use a piece-wise linear degradation model to define the observed RUL values in the training datasets. That is, after an initial period with constant RUL values, we assume that the RUL targets decrease linearly. We denote as R_e the initial period when the engines are still working in their desired conditions. A constant R_e of 125 cycles is selected to based on previous works [137, 144].

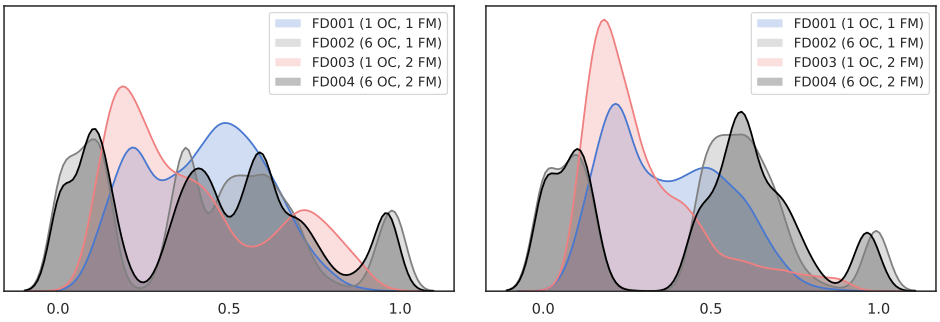
4.4.3 Domain Shift

In Figure 4.4, we present two normalised sensor values between 100 and one time step before a failure occurs for each dataset. We observe a lower distribution shift between dataset pairs FD001-FD003 and FD002-FD004, i.e., pairs that have data simulated under the same operating conditions [191]. However, there is still a significant distribution shift between FD001 and FD003 due to varying fault modes. On the other hand, a smaller distribution shift exists between FD002 and FD004. In practice, the different distributions make models data-specific, i.e., a model trained in one dataset often does not perform well in a different dataset unless the sensor values driving the fault behaviour are similar across source-target pairs.

4.4.4 Performance Metrics

Similar to previous works, we measure the performance of the proposed method using two metrics. We use the root mean squared error (RMSE) as this can be directly related to equations (4.13) and (4.14) and provide an estimation of how well the model is performing in the target prediction task. We also evaluate our model using a *scoring* function shown in equation (4.22) proposed by [191]:

$$s = \sum_{i=1}^{n} \mathbb{1}(c_i < 0) e^{-\frac{c_i}{a_1}} + \mathbb{1}(c_i \geq 0) e^{\frac{c_i}{a_2}} - 1 \tag{4.22}$$



(a) Sensor 15 (b) Sensor 7

Figure 4.4: Distribution of normalised sensor values between 100 and one time step before a failure. Sensor distributions are more similar between FD001 and FD003 and FD002 and FD004 pairs due to identical operating conditions.

where $\mathbb{1}(\cdot)$ is an indicator function that takes value 1 when the predicate is true and 0 otherwise, $a_1 = 13$ and $a_2 = 10$ and $c_i = \hat{y}_i - y_i$ [191]. In other words, c_i is the difference between predicted and observed RUL values and s is summed over all examples. We assume no access to y_i in the target domain during training. However, we utilise the values of y_i in the target datasets for evaluation of the adaptation performance. Note that the scoring metric penalises positive errors more than negative errors as these have a higher impact on maintenance policies.

4.5 Training Procedure and Hyperparameter Selection

In this section, we present the training procedure of the proposed algorithm as well as the hyperparameters selection for each experiment in our tests.

4.5.1 Training Procedure

For training, we select the input sensors, operational settings and labelled RUL values from the source data and only sensors and settings from target datasets. After the C-MAPPS datasets are normalised according to Equation (4.21), a time window of size T_w is used to generate the temporal input data arranged as $(\bar{N}, T_w, q)$, where $\bar{N}$ is the number of training samples and q the number of input features. We separate the original training data into training (seen by the algorithm) and cross-validation (used for stopping criteria) data containing 90% and 10% of the original dataset. For testing, the last sliding window of every trajectory in the testing data is selected as input data.

We split the training set into mini-batches (collection of data samples). During training, we randomly select mini-batches of the same size coming from the

source and target domains. As it can be seen in Table 4.1, the datasets have a different number of training samples. To cope with this difference, we oversample the domain containing the smaller number of samples to match the same number of samples from the larger dataset. After the network has seen all training examples, we consider an epoch finished.

Next, the proposed LSTM-DANN architecture is defined, including the number of LSTM and FC layers, number of units in each layer, learning rate and gradient update algorithm. Weights are initialised using the Xavier initialisation method [78]. We train the models for a maximum of 200 epochs and interrupt training if no cross-validation improvement in the source regression is seen for 20 epochs. In our experiments, the MAE loss ($p = 1$) in Equation (4.14), presents the best performing results and is selected as loss function. Moreover, it better approximates the 0-1 loss function and the hypothesis class of the proxy domain discrepancy metric. We start with fixed learning rates, and after 100 epochs, the learning rate is multiplied by a 0.1 factor to allow for stable convergence. We clip gradient norms to one to avoid exploding values. After training, the data coming from the target domain, including RUL values, are fed to the network to generate RUL estimations and performance measures.

4.5.2 Hyperparameter Selection

We perform grid-search on the gradient update optimiser and learning rates (λ_d, λ_y), after fine-tuning the remaining parameters manually (Table 4.2). To assess performance, we observe source regression and domain classifier error on the source domain. We generally observe better results for lower regression source error while high domain classification loss is induced, with its value controlled by the adversarial classification loss, i.e., no distinction between domains. In our tests, the SGD algorithm results in better performance, and the L_2 regularisation factor is selected as 0.01 for all experiments pairs. The remaining hyperparameters settings are presented in Table 4.3.

Hyperparameter	Range
L_2 Regularisation Weight	{0.0, 0.01, 0.1}
Layers (LSTM, source regression, domain classification)	{1, 2}
Units (LSTM, f, source regression, domain classification)	{16, 20, 30, 32, 64, 100, 128, 512}
Dropout Rate	{0.1, 0.3, 0.5, 0.7}
α	{0.8, 1.0, 2.0, 3.0}
Batch Size	{256, 512, 1024}
Learning Rate (source regression (λ_y), domain classification (λ_d))	{0.001, 0.01, 0.1}
Optimiser	{SGD, RMSProp [211], Adam [124]}
T_w	{30, 20, 15}

Table 4.2: Hyperparameter values evaluated in the proposed methodology.

Source	Target	LSTM Layers, (Units), [Dropout]	f (Units)	Source regression Layers, (Units), [Dropout]	Domain classification Layers, (Units), [Dropout]	α	Batch Size	λ_y, λ_d	T_w
FD001	FD002	1, (128), [0.5]	(64)	1, (32), [0.3]	1, (32), [0.3]	0.8	256	0.01, 0.01	20
FD001	FD003	1, (128), [0.5]	(64)	1, (32), [0.3]	1, (32), [0.3]	0.8	256	0.01, 0.01	30
FD001	FD004	1, (128), [0.7]	(64)	2, (32, 32), [0.3]	1, (32), [0.3]	1.0	256	0.01, 0.1	15
FD002	FD001	1, (64), [0.1]	(64)	1, (32), [0.0]	2, (16, 16), [0.1]	1.0	512	0.01, 0.01	30
FD002	FD003	1, (64), [0.1]	(512)	2, (64, 32), [0.0]	2, (64, 32), [0.1]	2.0	256	0.1, 0.1	30
FD002	FD004	2, (32, 32), [0.1]	(32)	1, (32), [0.0]	1, (16), [0.1]	1.0	256	0.1, 0.1	15
FD003	FD001	2, (64, 32), [0.3]	(128)	2, (32, 32), [0.1]	2, (32, 32), [0.1]	2.0	256	0.01, 0.01	30
FD003	FD002	2, (64, 32), [0.3]	(64)	2, (32, 32), [0.1]	2, (32, 32), [0.1]	2.0	256	0.01, 0.01	20
FD003	FD004	2, (64, 32), [0.3]	(64)	2, (32, 32), [0.1]	2, (32, 32), [0.1]	2.0	256	0.01, 0.01	15
FD004	FD001	1, (100), [0.5]	(30)	1, (20), [0.0]	1, (20), [0.1]	1.0	512	0.01, 0.01	30
FD004	FD002	1, (100), [0.5]	(30)	1, (20), [0.0]	1, (20), [0.1]	1.0	512	0.01, 0.01	20
FD004	FD003	1, (100), [0.5]	(30)	1, (20), [0.0]	1, (20), [0.1]	1.0	512	0.01, 0.01	30

Table 4.3: Selected hyperparameters for each source-target experiment pair.

4.6 Experimental Results

In this section, the prognostic performance of the proposed domain adaptation method is presented. All experiments were performed on an Intel Core i5 7^{th} generation processor with 16 GB RAM and a GeForce GTX 1070 GPU. We implemented our architecture using Python 3.6 and Keras [44] with TensorFlow [158] backend. Experiments consider each of the C-MAPPS datasets as the source domain and the remaining datasets as target domains. In total, we have 12 different experiments and results are averaged over 10 trials to reduce the effect of randomness, i.e., 120 experiments in total. For each experiment, we report the mean and standard deviations of the performance metrics.

We start by comparing the proposed method with baseline *non-adapted* LSTM models trained in the source domain and applied on the target domain (Source-Only), and trained in the target domain using the target domain labels (Target-Only) representing the ideal situation when target RUL values are available. We also run our algorithms with CNN, RNN and FC architectures as feature extractor g_f to showcase the effect of using LSTM for temporal feature learning. Furthermore, we assess our method against other well-known unsupervised domain adaptation methods: transfer component analysis (TCA) and CORrelation ALignment (CORAL) and the supervised domain adaptation method proposed in [238]. Lastly, we compare our proposed architecture to performing standardisation before training in the C-MAPPS datasets.

4.6.1 Non-adapted Models under Domain Shift

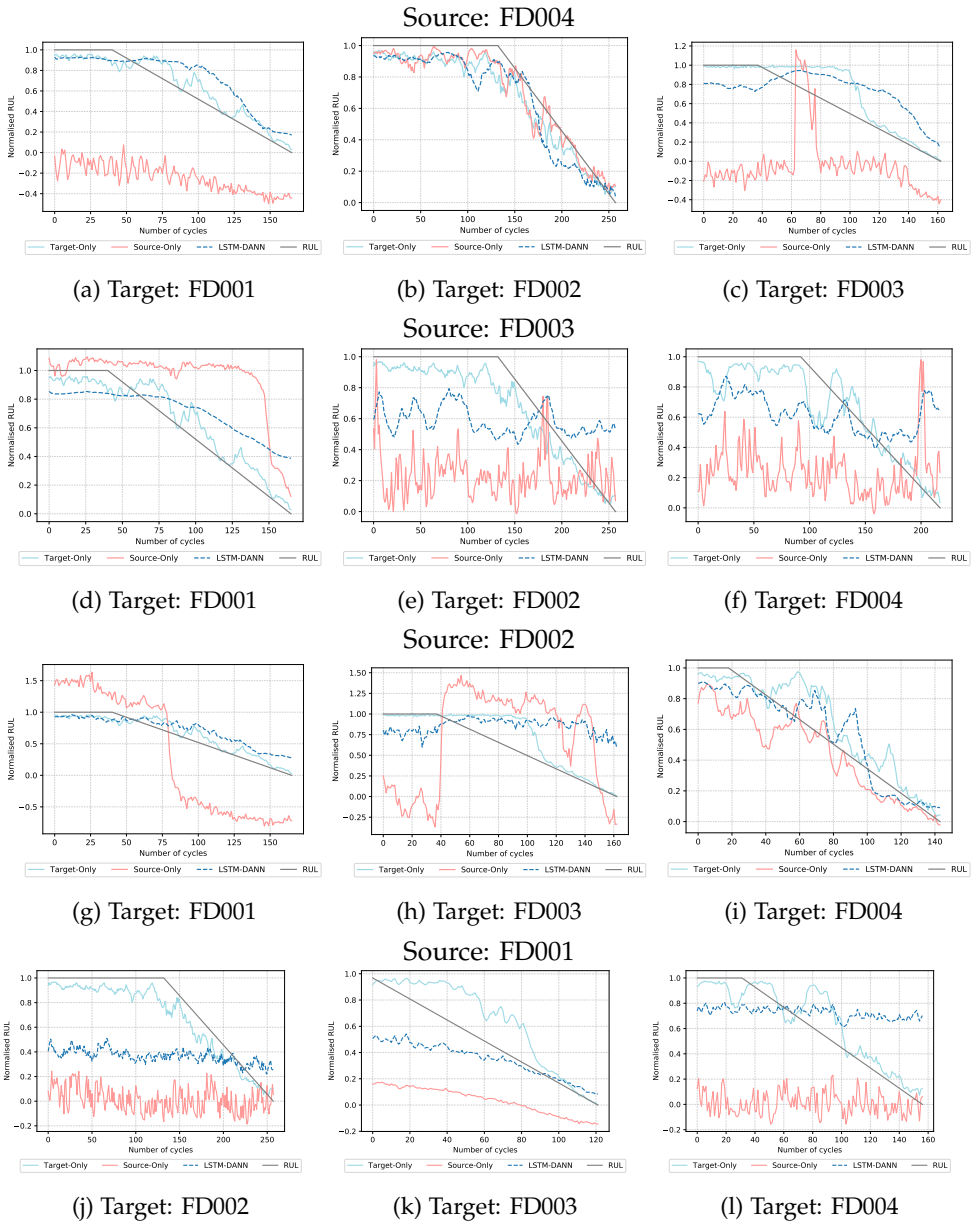
In this section, we compare the proposed model with Source-Only models serving as a baseline and Target-Only models serving as “upper bound" for LSTM-DANN. For the models without adaptation, we train a network of the form: LSTM(100) + Dropout(0.5) + ReLU(FC(30)) + Dropout(0.1) + ReLU(FC(20)) + FC(1), where we denote each learning layer in the network as Activation(Layer(Units)) and

dropout layers as DROPOUT(RATE). We train each model for 100 epochs using the Adam [124] optimiser with a learning rate of 0.001. We use an MSE loss function and T_w equal to 30, 20, 30, 15 for FD001, FD002, FD003, FD004. In Figure 4.5, we present the target RUL values as well as the predictions coming from the LSTM-DANN, Source-Only and Target-Only models for examples in the cross-validation sets. We analyse the results splitting the analysis for each domain, as its selection poses distinct difficulties in the adaptation results.

Source FD004 RUL predictions for FD004 acting as source domain are presented in Figures 4.5a, 4.5b, 4.5c. We note that Source-Only predictions for target domains FD001 (Fig. 4.5b) and FD003 (Fig. 4.5c) do not fit the observed RUL values. On the other hand, LSTM-DANN shows a better fit of the degradation model leading to smaller errors. For target domain FD002, the Source-Only model already provides a good fit for the observed RUL values (Figure 4.5b), due to low discrepancies between the two datasets. In this case, domain adaptation results in predictions similar to Source-Only. We point out that FD004 contains six operating conditions and two fault modes. Therefore, LSTM-DANN can find correspondences between operating conditions and fault modes in each source-target pair.

Source FD003 In the figures, adaptation from FD003 to FD002 (Fig. 4.5e) and FD004 (Fig. 4.5f) show worse results than for FD001 (Fig. 4.5d). FD003 is more similar to FD001, varying only the number of fault modes. As it can be seen in Table 4.4, the LSTM-DANN model yields a lower RMSE value when compared to the Source-Only model. For target domains FD002 and FD004, the differences between domains are more prominent. FD003 has one operating condition, while FD002 and FD004 have six. LSTM-DANN can improve over the Source-Only model, showing a better prediction error in the test sets despite the difficulties in transferring from such domains. However, estimated values are noisy and do not fit the linear degradation model completely.

Source FD002 In Figures 4.5g, 4.5h and 4.5i we present engines from FD001, FD003 and FD004 sets. Similar to the inverted experiment, the similarities between the FD004 and FD002 make Source-Only and LSTM-DANN models fit the target data with high accuracy. Predictions are improved on FD001 and FD003 sets in comparison to Source-Only. FD001 and FD002 share the same fault mode (HPC degradation), but FD002 has more operating conditions. While FD003 has one operating condition and two fault modes, our algorithm can produce lower errors and a better fit than a model without adaptation. In other words, it was possible to transfer from a domain that has more operating conditions than the target domain under the same (FD001) or different (FD003) fault modes.



(a) Target: FD001 (b) Target: FD002 (c) Target: FD003

(d) Target: FD001 (e) Target: FD002 (f) Target: FD004

(g) Target: FD001 (h) Target: FD003 (i) Target: FD004

(j) Target: FD002 (k) Target: FD003 (l) Target: FD004

Figure 4.5: RUL predictions of the Target-Only, Source-Only and LSTM-DANN models for one engine coming from the target domain cross-validation datasets.

Source FD001 FD001 presents the highest errors when functioning as a source domain. When target domains are FD002 and FD004 (Figs. 4.5j and 4.5l) the best solution found is one that yields flattened predictions. FD001 is the dataset containing only one operating condition and fault mode. Similar to other results, transferring from fewer operating conditions results to be a much harder problem. When the target domain is FD003 (Fig. 4.5k), we adapt to a domain with similar operating conditions. For this case, the model results in RUL prediction curves that fit the linear degradation model.

Source	Target	Source-Only	LSTM-DANN (Δ%)	Target-Only
FD001 (1 OC, 1 FM)	FD002 (6 OC, 1 FM)	71.7 ± 3.9	**48.6** (-32%) ± 6.8	17.8 ± 0.4
FD001 (1 OC, 1 FM)	FD003 (1 OC, 2 FM)	51.2 ± 3.4	**45.9** (-10%) ± 3.6	12.5 ± 0.3
FD001 (1 OC, 1 FM)	FD004 (6 OC, 2 FM)	73.9 ± 4.5	**43.8** (-41%) ± 4.1	21.3 ± 1.1
FD002 (6 OC, 1 FM)	FD001 (1 OC, 1 FM)	164.8 ± 23.0	**28.1** (-83%) ± 5.0	13.7 ± 0.8
FD002 (6 OC, 1 FM)	FD003 (1 OC, 2 FM)	154.0 ± 21.8	**37.5** (-76%) ± 1.5	12.7 ± 0.3
FD002 (6 OC, 1 FM)	FD004 (6 OC, 2 FM)	37.8 ± 2.2	**31.8** (-16%) ± 1.6	21.3 ± 1.1
FD003 (1 OC, 2 FM)	FD001 (1 OC, 1 FM)	49.9 ± 7.6	**31.7** (-36%) ± 9.4	13.7 ± 0.8
FD003 (1 OC, 2 FM)	FD002 (6 OC, 1 FM)	70.3 ± 4.0	**44.6** (-36%) ± 1.2	17.8 ± 0.4
FD003 (1 OC, 2 FM)	FD004 (6 OC, 2 FM)	69.3 ± 4.5	**47.9** (-31%) ± 5.8	21.3 ± 1.1
FD004 (6 OC, 2 FM)	FD001 (1 OC, 1 FM)	188.0 ± 25.9	**31.5** (-83%) ± 2.4	13.6 ± 0.8
FD004 (6 OC, 2 FM)	FD002 (6 OC, 1 FM)	**20.9** ± 1.7	24.9 (+19%) ± 1.8	17.8 ± 0.4
FD004 (6 OC, 2 FM)	FD003 (1 OC, 2 FM)	157.3 ± 20.4	**27.8** (-82%) ± 2.7	12.5 ± 0.3

Table 4.4: RMSE ± Standard Deviation - Comparison between Source-Only, Target-Only and LSTM-DANN on the test datasets.

Source	Target	Source-Only	LSTM-DANN	Target-Only
FD001 (1 OC, 1 FM)	FD002 (6 OC, 1 FM)	$> 10^6 \pm > 10^6$	**93,841** ± 55,493	1,638 ± 203
FD001 (1 OC, 1 FM)	FD003 (1 OC, 2 FM)	206,778 ± 59,887	**27,005** ± 12,385	267 ± 49
FD001 (1 OC, 1 FM)	FD004 (6 OC, 2 FM)	$> 10^6 \pm > 10^6$	**57,044** ± 60,160	2,904 ± 49
FD002 (6 OC, 1 FM)	FD001 (1 OC, 1 FM)	$> 10^6 \pm > 10^6$	**8,411** ± 11,855	300 ± 33
FD002 (6 OC, 1 FM)	FD003 (1 OC, 2 FM)	$> 10^6 \pm > 10^6$	**17,406** ± 5,702	267 ± 49
FD002 (6 OC, 1 FM)	FD004 (6 OC, 2 FM)	134,531 ± 54,923	**66,305** ± 14,723	2,904 ± 492
FD003 (1 OC, 2 FM)	FD001 (1 OC, 1 FM)	37,559 ± 19,248	**5,113** ± 4,865	300 ± 33
FD003 (1 OC, 2 FM)	FD002 (6 OC, 1 FM)	$> 10^6 \pm > 10^6$	**37,297** ± 15,578	1,638 ± 203
FD003 (1 OC, 2 FM)	FD004 (6 OC, 2 FM)	$> 10^6 \pm > 10^6$	**141,117** ± 66,218	2,904 ± 492
FD004 (6 OC, 2 FM)	FD001 (1 OC, 1 FM)	$> 10^6 \pm > 10^6$	**7,586** ± 2,735	300 ± 33
FD004 (6 OC, 2 FM)	FD002 (6 OC, 1 FM)	**1,944** ± 446	17,001 ± 12,927	1,638 ± 203
FD004 (6 OC, 2 FM)	FD003 (1 OC, 2 FM)	$> 10^6 \pm > 10^6$	**5,941** ± 1791	267 ± 49

Table 4.5: Scoring ± Standard Deviation - Comparison between Source-Only, Target-Only and LSTM-DANN on the test datasets.

In Tables 4.4 and 4.5, we present the average RMSE followed the percentage

change (Δ%) over Source-Only models dataset as well as the average Scoring performances for all experiment pairs. The proposed method can improve performance over almost all Source-Only models. In particular, adaptation results change depending on the operating conditions in the source domain. We achieve better outcomes for source domain FD004, as it contains all operating conditions and fault modes in the other datasets. Also, when the domain shift in small, Source-Only models already achieve reasonable performance in the target domain. Particularly, while transferring from FD004 to FD002, Source-Only yields lower RMSE than LSTM-DANN. For this experiment, RMSE values of Source-Only, LSTM-DANN and Target-Only are close. That is, adding information from the target domain adds little improvement.

Moreover, among 10 runs, LSTM-DANN was able to find lower RMSE and Scoring values than Source-Only indicating that hyperparameters could be tweaked to achieve better adaptation performance. As a rule of thumb, results show that LSTM-DANN performs whenever the source domains “shares" information with the target domain (either same fault modes or operation conditions). The model achieves the best results when source domain data under more conditions or fault modes are used for adaptation.

4.6.2 Feature Extractor Comparison

In this section, we assess the effect of using an LSTM combined with an FC layer as the feature extractor g_y in the proposed architecture. The following feature extracting architectures are carried out: fully-connected (FC-DANN), convolutional (CNN-DANN) and recurrent (RNN-DANN). In other words, we substitute the LSTM layers in Figure 4.3 by FC, CNN and RNN layers. To allow comparison, the remaining parts of the architecture g_d and g_y remain unchanged in all implementations, including the loss function. We select the hyperparameters of the architectures by changing the learning rates, α, dropout rates and optimisation algorithm according to Table 4.2.

FC-DANN We implement a multi-Layer perceptron (MLP) by first flattening the input vectors to an input of size $T_w \times q$. In effect, the FC network has no information on the temporal dependencies of the input data. We replace the original LSTM and FC layers with a single FC layer containing 100 units followed by a ReLU nonlinearity. We train the network using the Adam algorithm and pass mini-batches of 512 examples from each domain. Therefore, FC-DANN offers a comparison to a simpler architecture that does not consider recurrent connections while having comparable depth and size.

CNN-DANN Our second implementation incorporates a 2D CNN feature extractor based on the effective implementation reported in [140]. For training, our

inputs are shaped as $(T_w, q, 1)$ tensor, i.e., one feature channel. We use four convolutional layers, each containing 10 filters with size 10×1 and one layer containing one filter of size 3×1 to replace the LSTM layers. Later, the convolved features are flattened, and an FC layer with 100 units completes the feature extraction. A dropout layer is inserted between the flattened layer and the FC layer. The extracted features are then passed to the remaining of the architecture, as reported in Table 4.3. The network is trained using the SGD algorithm over batches of 512 training examples from each domain. Since CNN architectures have been successfully applied to RUL prediction using the C-MAPPS datasets, we want to evaluate the performance results in comparison to recurrent architectures.

RNN-DANN We replace the LSTM layers with vanilla RNN layers (no gating) using the same parameters of Table 4.3. We want to test whether having a gated architecture helps to learn domain invariant features. This architecture is trained in the same manner as the proposed LSTM network.

Source	Target	FC-DANN	CNN-DANN	RNN-DANN	LSTM-DANN
FD001 (1 OC, 1 FM)	FD002 (6 OC, 1 FM)	67.1 ± 24.1	50.3 ± 6.4	51.8 ± 8.2	**48.6** ± 6.8
FD001 (1 OC, 1 FM)	FD003 (1 OC, 2 FM)	64.2 ± 28.7	**42.8** ± 3.3	43.3 ± 3.2	45.9 ± 3.6
FD001 (1 OC, 1 FM)	FD004 (6 OC, 2 FM)	74.8 ± 32.5	87.0 ± 60.8	53.1 ± 1.8	**43.8** ± 4.1
FD002 (6 OC, 1 FM)	FD001 (1 OC, 1 FM)	52.1 ± 27.3	47.4 ± 16.3	**25.5** ± 2.9	28.1 ± 5.0
FD002 (6 OC, 1 FM)	FD003 (1 OC, 2 FM)	73.9 ± 46.4	40.8 ± 0.2	41.3 ± 2.7	**37.5** ± 1.5
FD002 (6 OC, 1 FM)	FD004 (6 OC, 2 FM)	**26.3** ± 0.5	38.0 ± 7.3	32.1 ± 0.8	31.8 ± 1.6
FD003 (1 OC, 2 FM)	FD001 (1 OC, 1 FM)	**28.7** ± 10.1	33.8 ± 6.0	31.4 ± 11.8	31.7 ± 9.3
FD003 (1 OC, 2 FM)	FD002 (6 OC, 1 FM)	77.2 ± 16.5	68.2 ± 15.2	54.6 ± 7.0	**44.6** ± 1.2
FD003 (1 OC, 2 FM)	FD004 (6 OC, 2 FM)	92.7 ± 47.2	73.7 ± 67.5	58.5 ± 7.2	**47.9** ± 5.8
FD004 (6 OC, 2 FM)	FD001 (1 OC, 1 FM)	71.3 ± 47.4	67.4 ± 24.3	**27.3** ± 3.6	31.5 ± 2.4
FD004 (6 OC, 2 FM)	FD002 (6 OC, 1 FM)	24.9 ± 1.8	39.1 ± 2.5	26.4 ± 2.5	**24.9** ± 1.8
FD004 (6 OC, 2 FM)	FD003 (1 OC, 2 FM)	47.5 ± 25.1	45.5 ± 11.0	34.3 ± 8.6	**27.8** ± 2.7

Table 4.6: RMSE ± Standard Deviation - Comparion of different architectures for domain adaptation.

In Table 4.6, we present the average RMSE results of each of the tested architectures. Performance results show that LSTM-DANN outperforms other architectures in the majority of the experiments. However, FC-DANN can outperform recurrent and convolution architectures in two experiments. In these cases, source and target domains share the same number of operating conditions, facilitating domain adaptation. For the other instances, the recurrent models RNN-DANN and LSTM-DANN outperform both architectures with FC and CNN layers as feature extractors. These results show that using recurrent layers helps adaptation performance in the studied datasets. In particular, RNN-DANN outperforms LSTM-DANN in two experiments when the target domain is FD001, i.e., FD004 to FD001 and FD002 to FD001. For the remaining experiments, LSTM-DANN outperforms RNN-DANN and the other architectures showing that having a gated architecture helps to learn

better RUL estimates.

As [135] has shown, it is possible to learn simple MLP networks that can result in reasonable RUL performance in the studied datasets. In our experiments, an MLP network coupled with a DANN mechanism could only perform adaptation when the data from the source and target domains were already “similar". That is, although we can train simpler architectures such as FC-DANN, we could only outperform more complex networks in two experiments with the same operational conditions. Adding the complexity of the recurrent connections increases training times in comparison to FC-DANN and CNN-DANN but also helps at learning better temporal features from different domains.

4.6.3 Comparison to Classic Domain Adaptation Methods

In this section, we provide a comparison of the proposed LSTM-DANN architecture to out-of-the-box unsupervised domain adaptation methods, transfer component analysis (TCA) [173], and CORrelation ALignment (CORAL) [202]. We also compare to the supervised adaptation method based on a Bidirectional LSTM architecture [238]. There, our goal is to showcase the effectiveness of our method in comparison to another architecture that can leverage target labels for training.

Out-of-the-box Unsupervised Domain Adaptation

We compare the proposed method to two well-known unsupervised domain adaptation methods TCA and CORAL.

Source	Target	TCA-NN	TCA-DNN	CORAL-NN	CORAL-DNN	LSTM-DANN
FD001 (1 OC, 1 FM)	FD002 (6 OC, 1 FM)	94.1 ± 1.0	90.0 ± 2.9	99.2 ± 3.6	77.5 ± 4.6	**46.4** ± 3.6
FD001 (1 OC, 1 FM)	FD003 (1 OC, 2 FM)	120.0 ± 1.0	116.1 ± 1.0	60.0 ± 0.7	69.6 ± 5.2	**37.3** ± 3.4
FD001 (1 OC, 1 FM)	FD004 (6 OC, 2 FM)	120.1 ± 1.0	113.8 ± 6.9	107.7 ± 2.8	84.6 ± 7.0	**43.5** ± 5.3
FD002 (6 OC, 1 FM)	FD001 (1 OC, 1 FM)	94.7 ± 1.1	85.6 ± 5.5	77.9 ± 19	80.9 ± 9.4	**31.2** ± 5.4
FD002 (6 OC, 1 FM)	FD003 (1 OC, 2 FM)	107.4 ± 3.7	111.5 ± 7.2	60.9 ± 15.1	79.8 ± 10.1	**32.2** ± 3.1
FD002 (6 OC, 1 FM)	FD004 (6 OC, 2 FM)	93.5 ± 2.8	94.4 ± 6.7	37.5 ± 0.5	43.6 ± 3.6	**27.7** ± 1.5
FD003 (1 OC, 2 FM)	FD001 (1 OC, 1 FM)	98.7 ± 0.4	90.5 ± 4.6	26.5 ± 0.5	**26.5** ± 1.9	30.6 ± 6.2
FD003 (1 OC, 2 FM)	FD002 (6 OC, 1 FM)	90.5 ± 0.3	80.8 ± 4.3	113.2 ± 4.5	75.6 ± 9.5	**43.1** ± 1.4
FD003 (1 OC, 2 FM)	FD004 (6 OC, 2 FM)	78.9 ± 5.3	102.6 ± 3.2	113.9 ± 5.5	77.2 ± 9.1	**49.7** ± 9.1
FD004 (6 OC, 2 FM)	FD001 (1 OC, 1 FM)	98.5 ± 0.4	85.6 ± 5.0	119.1 ± 16.7	94.0 ± 8.8	**25.4** ± 4.2
FD004 (6 OC, 2 FM)	FD002 (6 OC, 1 FM)	75.3 ± 1.7	80.8 ± 5.8	37.3 ± 0.6	30.9 ± 1.4	**26.9** ± 3.3
FD004 (6 OC, 2 FM)	FD003 (1 OC, 2 FM)	77.2 ± 6.0	102.9 ± 2.7	68.1 ± 11.1	68.6 ± 11.2	**23.6** ± 5.0

Table 4.7: RMSE ± Standard Deviation - Comparion of unsupervised domain adaptation methods on cross-validation target data.

TCA TCA uses a reproducing kernel Hilbert space (RKHS) and the MMD function to construct a feature space that minimises the difference between the domains. We use TCA to find a feature representation and train an FFNN with one FC layer

and 32 units (TCA-NN) and a Deep FFNN (TCA-DNN) with the same number of layers and units as our Target-Only models. In our tests, we apply the Radial Basis Function (RBF) kernel extracting 20 transfer components with $\lambda = 1$ and $\gamma = 1$.

CORAL Different from TCA, CORAL minimises domain shift by aligning the second-order statistics of source and target distributions. After the alignment, the new space can be used to train a model in the transformed source domain. Similar to TCA, we use the CORAL feature space to learn an FFNN (CORAL-NN) and a Deep FFNN (CORAL-DNN). For all FFNN methods, we use the same input data, ReLU as hidden activation, MSE as loss function and Adam [124] with a learning rate of 0.001.

Both TCA and CORAL are not entirely suitable for temporal data. Thus, we learn on input features from $t-1$ and RUL at time t. Results are summarised for target datasets in Table 4.7. On average, LSTM-DANN yields lower RMSE than the methodologies tested in all but one experiment pair. In other terms, this means that the methodology performs better than the tested out-of-the-box adaptation methods not tailored for temporal sensor data. However, CORAL can achieve low RMSE results when the source domain has more operating conditions (FD004 to FD002) or fault modes (FD003 to FD001) than the target domain.

Supervised Domain Adaptation

We compare our method to the supervised domain adaptation architecture BLSTM-SDA [238]. In BLSTM-SDA, a Bidirectional LSTM architecture is first trained in the source domain and fine-tuned in the target domain using RUL labels. Thus, we expect this methodology to perform better than LSTM-DANN over our domain adaptation experiments. In Table 4.8, we present the results obtained from the original paper for BLSTM-SDA, BLSTM-SDA (Target-Only) and LSTM-DANN.

As expected, results show that leveraging RUL labels in the target domain indeed result in lower RMSE values. However, for source domains FD004 and FD002, our results are surprisingly close to those obtained via BLSTM-SDA, e.g. FD002 to FD004. That corresponds to our previous findings in which we show that transferring from more operating conditions yields a better fit to target domain degradation functions. However, unlike our results, pre-training the BLSTM-SDA on the source domain does not always lead to improved results on the target domain, e.g. FD004 to FD003 and FD002 to FD001. BLSTM-SDA has no domain alignment mechanism, and transferring from different operating conditions results in mixed performance gains. Even though our method cannot achieve the same results as using target labels, it manages to either improve or remain close to all Source-Only (only source labels are available to our model) models in our experiments.

Source	Target	LSTM-DANN	BLSTM-SDA	BLSTM-SDA (Target-Only)
FD001 (1 OC, 1 FM)	FD002 (6 OC, 1 FM)	48.6	**20.8**	21.7
FD001 (1 OC, 1 FM)	FD003 (1 OC, 2 FM)	45.9	**14.3**	16.3
FD001 (1 OC, 1 FM)	FD004 (6 OC, 2 FM)	43.8	-	25.9
FD002 (6 OC, 1 FM)	FD001 (1 OC, 1 FM)	28.1	**18.3**	14.3
FD002 (6 OC, 1 FM)	FD003 (1 OC, 2 FM)	37.5	-	16.3
FD002 (6 OC, 1 FM)	FD004 (6 OC, 2 FM)	31.8	**25.5**	25.9
FD003 (1 OC, 2 FM)	FD001 (1 OC, 1 FM)	31.7	**13.6**	14.2
FD003 (1 OC, 2 FM)	FD002 (6 OC, 1 FM)	44.6	-	21.7
FD003 (1 OC, 2 FM)	FD004 (6 OC, 2 FM)	47.9	**23.4**	25.9
FD004 (6 OC, 2 FM)	FD001 (1 OC, 1 FM)	31.5	-	14.3
FD004 (6 OC, 2 FM)	FD002 (6 OC, 1 FM)	24.9	**20.8**	21.7
FD004 (6 OC, 2 FM)	FD003 (1 OC, 2 FM)	27.8	**24.1**	16.3

Table 4.8: RMSE - Comparison between LSTM-DANN and BLSTM-SDA supervised domain adaptation architecture.

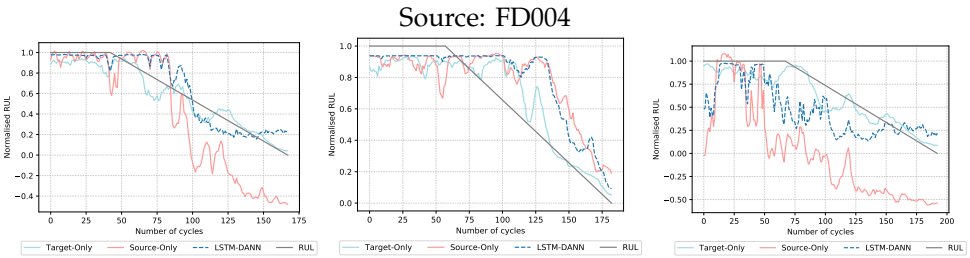
4.6.4 Relationship to Standardisation

A straightforward way to align source and target domains and reduce the difference in input distributions is to perform standardisation. That is local mean centring and division by the standard deviation of each input feature. To test whether such transformation already suffices as an alignment strategy, we standardise the data before feeding it to Source-Only and Target-Only architectures. We select FD004 as the source domain as results have shown that adaptation is possible for all remaining C-MAPPS datasets. We compare the original LSTM-DANN trained on the data with min-max normalisation to a model trained on the standardised data (LSTM-DANN-STD) and reference models Source Target-Only trained on the standardised data. We present the RMSE values for each methodology in the target domains in Table 4.9.

Source	Target	Source-Only-STD	LSTM-DANN	LSTM-DANN-STD	Target-Only-STD
FD004 (6 OC, 2 FM)	FD001 (1 OC, 1 FM)	53.3 ± 5.0	**31.5** ± 2.4	32.6 ± 2.0	14.5 ± 1.5
FD004 (6 OC, 2 FM)	FD002 (6 OC, 1 FM)	23.2 ± 1.0	24.9 ± 1.8	**21.8** ± 1.7	18.4 ± 0.4
FD004 (6 OC, 2 FM)	FD003 (1 OC, 2 FM)	62.8 ± 7.5	**27.8** ± 2.7	40.2 ± 7.0	16.0 ± 0.3

Table 4.9: Test performance (RMSE $\pm$ Standard Deviation) of Source-Only, Target-Only, LSTM-DANN-STD. Models are trained on standardised training data with zero-mean and unit-variance.

The results in Table 4.9 show that, on average, the RMSE performances of Source-Only models are considerably improved in comparison to Table 4.4. However, our proposed methodology (LSTM-DANN-STD) still outperforms the baseline models and provide a better fit to the target data (Figure 4.6). Furthermore, training on standardised data causes the LSTM-DANN-STD models to saturate and



(a) Target: FD001 (b) Target: FD002 (c) Target: FD003

Figure 4.6: RUL predictions of the Target-Only, Source-Only and LSTM-DANN-STD models in the standardised target cross-validation datasets.

start overfitting much faster than in previous experiments. This effect negatively impacted the adaptation performance on FD003 as training progressed for more epochs than necessary. To address this effect, simple changes in the models' hyperparameters could be made, for example, by considering lower values of α and reduced learning rates.

4.7 Conclusions

In this chapter, a deep learning method for domain adaptation in prognostics is proposed based on a long short-term memory and a domain adversarial neural network (LSTM-DANN). Experiments are carried out on the popular Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) datasets to show the effectiveness of the proposed method. The goal of the task is to estimate the remaining useful lifetime of aircraft engines units while transferring from a source domain with observed RUL values to a target domain with only input features.

We propose an architecture aimed at finding weights that can accommodate domain shift via adversarial learning. In general, we can achieve lower performance errors compared to a model with no adaptation (Source-Only). A more effective transfer is achieved when the source domain has more fault modes or operating conditions than the target counterpart. On the other hand, transferring from a domain with fewer operating conditions and fault modes is a much harder task. However, even in the latter scenario, LSTM-DANN can correct the RUL predictions to alleviate performance errors.

Our remaining experiments show that recurrent architectures yield better adaptation performance in comparison to non-temporal methods such as Convolutional and Fully-Connected architectures. Moreover, we compare our proposed approach to a supervised domain adaptation LSTM architecture [238] and show that our model can achieve close performance results to a method that can leverage target labels. Furthermore, we compare our method against out-of-the-box domain adapt-

ation methods. In our tests, the proposed network has shown superior performance in comparison to networks learned on transfer component analysis (TCA) [173], and CORrelation ALignment (CORAL) [202] feature spaces. We point out that no thorough evaluation of domain adaptation methods was performed as most methodologies do not focus on sequential data. Lastly, we show that our proposed model yields better performance results than a simple zero-mean unit-variance alignment between source and target domains.

Chapter 5

Learning 2-opt Heuristics for Routing Problems

In this chapter, we study the problem of learning to solve routing problems. We focus on improvement methods, that is, methods that, given a poor initial solution, can search for an improved solution in a process akin to local search. Our method is tailored to search for good two edge exchanges but can be expanded to a more general number of edge exchanges with minor modification and extra care about the feasibility of the solutions. The main objective is to learn improvement policies from instance information that can approach near-optimal solutions. This objective gives rise to a more general search procedure than local search as we are not concerned with the immediate improvement of a solution but rather the entire chain of changes that results in the optimal solution.

This chapter is based on [49, 52].

5.1 Introduction

The travelling salesman problem (TSP) is a well-known combinatorial optimisation problem. In the TSP, given a set of locations (nodes) in a graph, we need to find the shortest tour that visits each location exactly once and returns to the departing location. The TSP is NP-hard [175] even in its Euclidean formulation, i.e., nodes are points in the 2D space. Classic approaches to solving the TSP can be classified in exact and heuristic methods. The former methods have been extensively studied using integer linear programming [4] which are guaranteed to find an optimal solution but are often too computationally expensive to be used in practice. The latter methods are based on (meta)heuristics, and approximate algorithms [7] that find solutions requiring less computational time, e.g., edge swaps such as k-opt [91]. However, designed heuristics require specialised knowledge, and their performances are often limited by algorithmic design.

Recent works in machine learning and deep learning have focused on learning heuristics for combinatorial optimisation problems [24, 147]. For the TSP, both supervised learning [115, 223] and reinforcement learning [21, 62, 121, 128, 231] methods have been proposed. The idea behind the proposed methods is that a machine learning method could learn better heuristics by extracting useful information directly from data rather than having an explicitly programmed behaviour. Most approaches to the TSP have focused on learning construction heuristics, i.e., methods that can generate a solution sequentially by extending a partial tour. These methods employed sequence representations [21, 223], graph neural networks [115, 121] and attention mechanisms [62, 128, 231] resulting in high-quality solutions. Construction methods still require additional procedures such as beam search, classical improvement heuristics, and sampling to achieve such results. This limitation hinders their applicability as it is required to revert to handcrafted improvement heuristics and search algorithms for state-of-the-art performance. Thus, learning improvement heuristics, i.e., when a solution is improved by local moves that search for better solutions remains relevant. Here, if we can learn a policy to improve a solution, we can use it to get better solutions from a construction heuristic or even random solutions. Recently, a deep reinforcement learning method [231] has been proposed for such a task, achieving near-optional results using node swap and 2-opt moves. However, the architecture has its output fixed by the number of possible moves, making it less favourable to expand to general k-opt [92].

Two natural extensions of the TSP are the multiple TSP (mTSP) and the capacitated vehicle routing problem (CVRP). In the first, we consider the original problem augmented with more salesmen, constrained on the size of tours or number of visits. The CVRP also considers multiple salesmen (vehicles) with a maximum capacity. Customers have certain demand values that need to be fulfilled by vehicles without exceeding their total capacity. These problems are harder to solve than the TSP due to the added constraints and usually require tailored heuristics. Both

problems have also been the subject of the recent interest in combining machine learning and combinatorial optimisation [66, 104, 117, 177]. However, few previously proposed models can be seamlessly used in multiple routing problems [128, 231].

Contributions and Organisation In this chapter, we propose a deep reinforcement learning algorithm trained via policy gradient to learn improvement heuristics based on 2-opt moves. Our architecture is based on a pointer attention mechanism [223] that outputs nodes sequentially for action selection. We introduce a reinforcement learning formulation to learn a stochastic policy of the next promising solutions, incorporating the search's history information by keeping track of the current best-visited solution. Our results show that we can learn policies for the Euclidean TSP that achieve near-optimal solutions even when starting with poor quality solutions. Moreover, our approach can achieve better results than previous deep learning methods based on construction [21, 62, 115, 121, 128, 155, 223] and improvement [231] heuristics. Compared to [231], our method can be easily adapted to general k-opt, and it is more sample efficient. Our method outperforms other effective heuristics such as Google's OR-Tools [178] for simulated instances and are close to optimal solutions. Lastly, it can be easily expanded to the mTSP and CVRP.

5.2 Related Literature

Exact approaches for the TSP, such as linear programming, may require a large amount of computational time to find optimal solutions. For this reason, designing fast heuristics for the TSP is often necessary. However, classical heuristics require specialised knowledge, which may be unavailable or scarce and may have sub-optimal handcrafted design rules. Note that the same can be said about machine learning methods. That is, they also require expert knowledge to be built and tuned. However, machine learning methods are allowed to exploit the search space of the combinatorial problems and can potentially learn to select better heuristics rules using patterns learned from data. Such patterns and information learned from data may not be apparent or lacking for human experts. Moreover, machine learning methods can be retrained to fit the underlying data distribution of problem instances, making them adaptive to instance distribution shifts. Thus, methods that can automatically learn good heuristics have the potential to be more effective and efficient than handcrafted ones, replacing some parts of the human-based design of solutions with learned ones.

In machine learning, early works for the TSP have focused on Hopfield networks [99] and deformable template models [3]. However, the performance of these approaches has not been on par with classical heuristics [132]. Recent deep learning methods have achieved high-performance learning construction heuristics for

the TSP. Pointer networks (PtrNet) [223] learned a sequence model coupled with an attention mechanism trained to output TSP tours using solutions generated by Concorde [4]. In [21], the PtrNet was further extended to learn without supervision using policy gradient, trained to output a distribution over node permutations. Other approaches encoded instances via graph neural networks. A structure2vec (S2V) [121] model was trained to output the ordering of partial tours using deep Q-learning. Later, graph attention was employed to a hybrid approach using 2-opt local search on top of tours trained via policy gradient [62]. Graph attention was extended in [128] using REINFORCE [229] with a greedy rollout baseline, resulting in lower optimality gaps. Recently, the supervised approach was revisited using graph convolution networks (GCN) [115] learning probabilities of edges occurring in a TSP tour. It achieved state-of-the-art results up to 100 nodes whilst also combining with search heuristics.

Recent machine learning approaches specialised for the mTSP include [117], which proposed a neural network architecture trained via supervised learning. Combined with constraint enforcing layers, they can achieve competitive results in comparison to OR-Tools. In [104], multi-agent reinforcement learning is used to learn an allocation of agents to nodes, and regular optimisation is used to solve TSP associated with each agent. The VRP has gained much interest since [168]. In this chapter, a policy gradient algorithm is proposed to generate solutions as a sequence of consecutive actions. Later, [128] extended the attention method to the VRP outperforming [168]. Followed by [231] who also expanded their model to the VRP case obtaining lower gaps. A specialised VRP model combined reinforcement and supervised learning to learn to construct solutions, outperforming [128], but trained on different distributions of node locations [66]. Another VRP method, named neural large neighbourhood search (NLNS) [102] proposed integrating learning methods and classical search. In the method, the policy is trained to reconstruct randomly destroyed solutions. Another approach named Learn to Improve (L2I) [151] considered learning improvements policies by choosing from a pool of operators. Recently, deep policy dynamic programming (DPDP) [127] was proposed with the aim to combine neural heuristics with dynamic programming. The method is trained to predict edges from example solutions and outperforms previous neural approaches solving TSPs and VRPs with 100 nodes.

It is important to previous *end-to-end* methods to have additional procedures such as beam search, classical improvement heuristics, and sampling to achieve good solutions. Thus, in this chapter, we encode edge information using graph convolutions and use classical sequence encoding to learn node orderings. We decode these representations via a pointing attention mechanism to learn a stochastic policy of the action selection task. In the TSP, our approach resembles classical 2-opt heuristics [84] and can outperform previous deep learning methods in solution quality and sample efficiency.

5.3 Background

5.3.1 Travelling Salesman Problem

We focus on the 2D Euclidean TSP. Given an input graph, represented as a sequence of n locations in a two dimensional space $X = \{x_i\}_{i=1}^{n}$, where $x_i \in [0,1]^2$, we are concerned with finding a permutation of the nodes, i.e., a tour $S = (s_1, \ldots, s_n)$, that visits each node once (except the starting node) and has the minimum total length (cost). We define the cost of a tour as the sum of the distances (edges) between consecutive nodes in S as

$$L(S) = \|x_{s_n} - x_{s_1}\|_2 + \sum_{i=1}^{n-1} \left\|x_{s_i} - x_{s_{i+1}}\right\|_2 \tag{5.1}$$

where $\|\cdot\|_2$ denotes the ℓ_2 norm.

5.3.2 k-opt Heuristic for the TSP

Improvement heuristics enhance feasible solutions through a search procedure. A procedure starts at an initial solution S_0 and replaces a previous solution S_t with a better solution S_{t+1}. Local search methods such as the effective Lin-Kernighan-Helsgaun (LKH) [91] heuristic perform well for the TSP. The procedure searches for k edge swaps (k-opt moves) that will be replaced by new edges resulting in a shorter tour. A simpler version [143] considers 2-opt (Figure 5.1) and 3-opt moves alternatives as these balance solution quality and the $O(n^k)$ complexity of the moves. Moreover, sequential pairwise operators such as k-opt moves can be decomposed into simpler l-opt ones, where $l < k$. For instance, sequential 3-opt operations can be decomposed into one, two or three 2-opt operations [91]. However, in local search algorithms, the quality of the initial solution usually affects the quality of the final solution, i.e., local search methods can easily get stuck in local optima [84].

To avoid local optima, different metaheuristics have been proposed, including simulated annealing and tabu search—these work by accepting worse solutions to allow more exploration of the search space. In general, this strategy leads to better solution quality. However, metaheuristics still require expert knowledge and may have sub-optimal rules in their design. To tackle this limitation, we propose to combine machine learning and 2-opt operators with learning a stochastic policy to improve TSP solutions sequentially. A stochastic policy resembles a metaheuristic, sampling solutions in the neighbourhood of a given solution, potentially avoiding local minima. Our policy iterates over feasible solutions, and the minimum cost solution is returned at the end. The main idea of our method is that taking future improvements into account can potentially result in better policies than greedy heuristics. Our approach is detailed in the following sections.

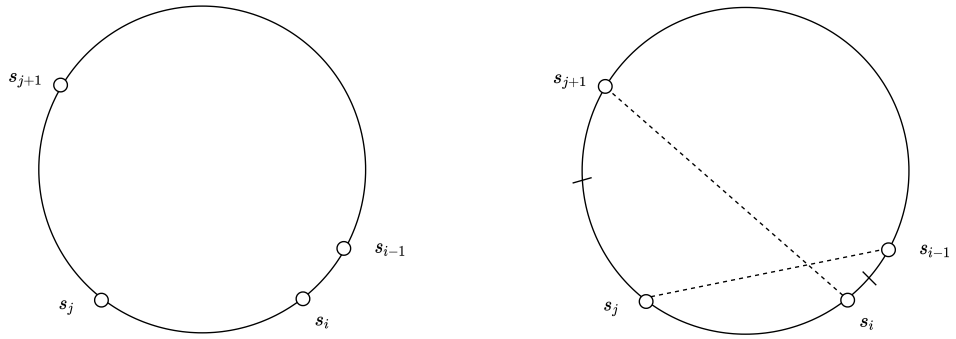


Figure 5.1: TSP solution before a 2-opt move (left), and after a 2-opt move (right). Added edges are represented in dashed lines and removed edges are represented with a strike through. Note that after the change the sequence of nodes $s_i, \ldots, s_j$ is inverted.

5.4 Reinforcement Learning Formulation

Our formulation considers solving the TSP via 2-opt as a Markov decision process (MDP), detailed below. In our MDP, a given state $\bar{S}$ is composed of a tuple of the current solution (tour) S and the lowest-cost solution S' seen in the search. The proposed neural architecture (Section 5.5) approximates the stochastic policy $\pi_\theta(A|\bar{S})$, where θ represents trainable parameters. Each $A = (a_1, a_2)$ corresponds to a 2-opt move where a_1, a_2 are node indices. Our architecture also contains a *value* network that outputs value estimates $V_\phi(\bar{S})$, with ϕ as learnable parameters. We assume TSP samples drawn from the same distribution and use policy gradient to optimise the parameters of the policy and value networks (Section 5.6).

5.4.1 Markov Decision Process

States A state $\bar{S}$ is composed of a tuple $\bar{S} = (S, S')$, where S and S' are the current and lowest-cost solution seen in the search, respectively. That is, given a search trajectory at time t and solution S,

$$S_t = S, \tag{5.2}$$

$$S'_t = S' = \arg\min_{S_{\tilde{t}} \in \{S_0, \ldots, S_t\}} L(S_{\tilde{t}}). \tag{5.3}$$

Actions Actions correspond to 2-opt operations that change a *solution* S to new solution. We model actions as tuples $A = (a_1, a_2)$ where $a_1, a_2 \in \{1, \ldots, n\}$, $a_2 > a_1$ correspond to index positions of solution $S = (s_1, \ldots, s_n)$.

Transitions Given $A = (i, j)$ transitioning to the next state defines a deterministic change to solution $\hat{S} = (\dots, s_i, \dots, s_j, \dots)$, resulting in a new solution $S = (\dots, s_{i-1}, s_j, \dots, s_i, s_{j+1} \dots)$ and state $\bar{S} = (S, S')$. That is, selecting i and j in $\hat{S}$ implies breaking edges at positions $(i-1, i)$ and $(j, j+1)$, inserting edges $(i-1, j)$ and $(i, j+1)$ and inverting the order of nodes between i and j (See Figure 5.1).

Rewards Similar to [231], we attribute rewards to actions that can improve upon the current best-found solution, i.e.,

$$R_t = L(S'_t) - L(S'_{t+1}) \tag{5.4}$$

Environment Our environment runs for $\mathbb{T}$ steps. For each run, we define episodes of length $T \leq \mathbb{T}$, after which a new episode starts from the last state in the previous episode. This ensures access to poor quality solutions at $t = 0$ and high-quality solutions as t grows.

Returns Our objective is to maximise the expected return G_t, which is the cumulative reward starting at time step t and finishing at T at which point no future rewards are available, i.e., $G_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} R_{t'}$ where $\gamma \in (0, 1]$ is a discount factor, introduced to stabilise learning.

5.5 Policy Gradient Neural Architecture

Our neural network, based on an encoder-decoder architecture is depicted in Figure 5.2. Two encoder units map each component of $\bar{S} = (S, S')$ independently. With a slight abuse of notation, we define inputs to each encoder unit as $X = [x_1, \dots, x_n]^\top$ and X' analogously, where x_i are node coordinates of node s_i in S and S'. The encoder then learns representations that embed both graph topology and node ordering. Given these representations, the *policy* decoder samples action indices $a_1, \dots, a_k$ sequentially, where $k = 2$ for 2-opt. The *value* decoder operates on the same encoder outputs but outputs real-valued estimates of state values. We detail the components of the network in the following sections.

5.5.1 Encoder

The purpose of our encoder is to obtain a representation for each node in the input graph given its topological structure and its position in a given solution. We incorporate elements from graph convolution networks (GCN) [125] and sequence embedding via recurrent neural networks (RNN) to accomplish this objective [97] and use edge information to build a more informative encoding of the TSP instance graph.

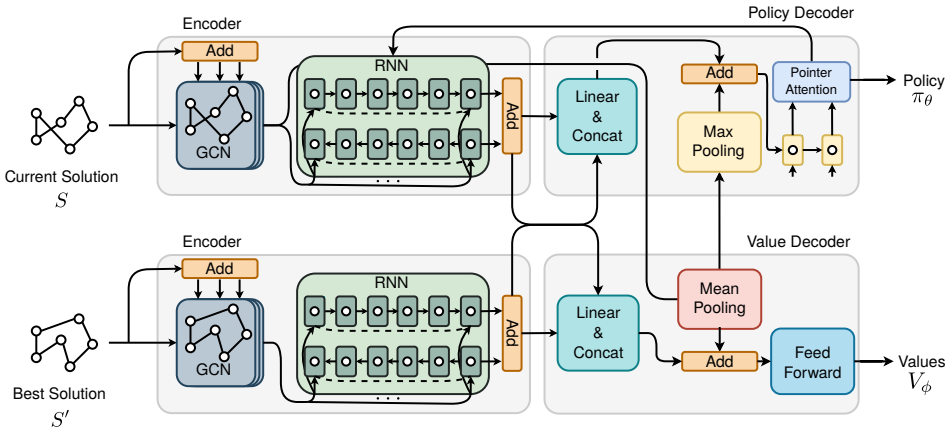


Figure 5.2: In the architecture, a state $\bar{S} = (S, S')$ is passed to a dual encoder where graph and sequence information are extracted. A policy decoder takes encoded inputs to query node indices and output actions. A value decoder takes encoded inputs and outputs state values.

Embedding Layer

We input two dimensional coordinates $x_i \in [0,1]^2$, $\forall i \in 1, \ldots, n$, which are embedded to d-dimensional features as

$$x_i^0 = W_x x_i + b_x \,, \tag{5.5}$$

where $W_x \in \mathbb{R}^{d \times 2}$, $b_x \in \mathbb{R}^d$. We use as input the Euclidean distances $e_{i,j}$ between coordinates x_i and x_j to add edge information and weigh the node feature matrix. To avoid scaling the inputs to different magnitudes we adopt symmetric normalisation [125] as

$$\tilde{e}_{i,j} = \frac{e_{i,j}}{\sqrt{\sum_{j=1}^{n} e_{i,j} \sum_{i=1}^{n} e_{i,j}}} \,. \tag{5.6}$$

Then the normalised edges are used in combination with GCN layers to create richer node representations using their neighbouring topology.

Graph Convolutional Layers

In the GCN layers, we denote as x_i^ℓ the node feature vector at GCN layer ℓ associated with node i. We define the node feature at the subsequent layer combining features from nodes in the neighbourhood $\mathcal{N}(i)$ of node i as

$$x_i^{\ell+1} = x_i^\ell + \sigma_r \Big(\sum_{j \in \mathcal{N}(i)} \tilde{e}_{i,j} (W_g^\ell x_j^\ell + b_g^\ell) \Big) , \tag{5.7}$$

where $W_g^\ell \in \mathbb{R}^{d\times d}$, $b_g^\ell \in \mathbb{R}^d$, σ_r is the rectified linear unit and $\mathcal{N}(i)$ corresponds to the remaining $n-1$ nodes of a complete TSP network. At the input to these layers, we have $\ell = 0$ and after $\mathbb{L}$ layers we arrive at representations $z_i = x_i^{\mathbb{L}}$ leveraging node features with the additional edge feature representation.

Sequence Embedding Layers

Next, we use node embeddings z_i to learn a sequence representation of the input and encode the ordering of nodes. Due to symmetry, a tour from nodes $(1,\ldots,n)$ has the same cost as the tour $(n,\ldots,1)$. Therefore, we read the sequence in both orders to explicitly encode the symmetry of a solution and the order of the nodes. To accomplish this objective, we employ two long short-term memory (LSTM) as our RNN functions, computed using hidden vectors from the previous node in the tour and the current node embedding resulting in

$$(h_i^{\rightarrow}, c_i^{\rightarrow}) = \text{RNN}(z_i^{\rightarrow}, (h_{i-1}^{\rightarrow}, c_{i-1}^{\rightarrow})), \quad \forall i \in \{1,\ldots,n\} \tag{5.8}$$

$$(h_i^{\leftarrow}, c_i^{\leftarrow}) = \text{RNN}(z_i^{\leftarrow}, (h_{i+1}^{\leftarrow}, c_{i+1}^{\leftarrow})), \quad \forall i \in \{n,\ldots,1\} \tag{5.9}$$

where in (5.8) a forward RNN goes over the embedded nodes from left to right, in (5.9) a backward RNN goes over the nodes from right to left, and $h_i, c_i \in \mathbb{R}^d$ are hidden vectors. We point out that the RNN modules are included to impose order in the tour for the policy decoder. That is, the bi-LSTM imposes ordering for the 2-opt operation and aids node (edge swap) selection. With the bidirectional orderings, even if the same tour is observed in one of its circular permutations, the predecessor and successor information of each node is maintained, which helps edge selection, i.e., remove $(i-1,i)$, $(j,j+1)$ and add $(i-1,j)$, $(i,j+1)$. Note that a 2-opt move only requires the difference between the costs of the removed and inserted edges.

Our representation reconnects back to the first node in the tour ensuring we construct a sequential representation of the complete tour, i.e.,

$$(h_0^{\rightarrow}, c_0^{\rightarrow}) = \text{RNN}(z_n, 0) \tag{5.10}$$

and

$$(h_{n+1}^{\leftarrow}, c_{n+1}^{\leftarrow}) = \text{RNN}(z_1, 0) \tag{5.11}$$

Afterwards, we combine forward and backward representations to form unique node representations in a tour as

$$o_i = \tanh\Big((W_f h_i^{\rightarrow} + b_f) + (W_b h_i^{\leftarrow} + b_b)\Big), \tag{5.12}$$

and a tour representation $h_n = h_n^{\rightarrow} + h_n^{\leftarrow}$, where $h_i, o_i \in \mathbb{R}^d$, $W_f, W_b \in \mathbb{R}^{d\times d}$ and $b_f, b_b \in \mathbb{R}^d$.

Dual Encoding

In our formulation, a state $\bar{S} = (S, S')$ is represented as a tuple of the current solution S and the best solution seen so far S'. We encode both S and S' using independent encoding layers (Figure 5.2). Following this notation we define a sequential representation of S' after going through encoding layers as $h'_n \in \mathbb{R}^d$. Note that in the proposed MDP, it is necessary to know the cost of the best solution seen in the search to be able to compute the rewards. Thus, we consider that the agent has complete information about the state space necessary to compute the cost improvement over the best-seen solution.

5.5.2 Policy Decoder

We aim to learn the parameters θ of a stochastic policy $\pi_\theta(A|\bar{S})$ that given a state $\bar{S}$, assigns high probabilities to moves that reduce the cost of a tour. Following [21], our architecture uses the chain rule to factorise the probability of a k-opt move as

$$\pi_\theta(A|\bar{S}) = \prod_{i=1}^{k} p_\theta\left(a_i|a_{<i}, \bar{S}\right), \tag{5.13}$$

and then uses individual softmax functions to represent each term on the RHS of (5.13), where a_i corresponds to node positions in a tour, $a_{<i}$ represents previously sampled nodes and $k = 2$. At each output step i, we map the tour embedding vectors to the following *query* vector

$$q_i = \tanh\Big((W_q q_{i-1} + b_q) + (W_o o_{i-1} + b_o)\Big), \tag{5.14}$$

where $W_q, W_o \in \mathbb{R}^{d\times d}$, $b_q, b_o \in \mathbb{R}^{d\times d}$ are learnable parameters and $o_0 \in \mathbb{R}^d$ is a fixed parameter initialised from a uniform distribution $\mathcal{U}(\frac{-1}{\sqrt{d}}, \frac{1}{\sqrt{d}})$. Our initial query vector q_0 receives the tour representation from S and S' as

$$h_{\bar{s}} = (W_s h_n + b_s) \| (W_{s'} h'_n + b_{s'}) \tag{5.15}$$

and a *max pooling* graph representation $z_g = \max(z_1, \ldots, z_n)$ from S to form

$$q_0 = h_{\bar{s}} + z_g, \tag{5.16}$$

where learnable parameters $W_s, W_{s'} \in \mathbb{R}^{\frac{d}{2}\times d}$, $b_s, b_{s'} \in \mathbb{R}^{\frac{d}{2}}$ and $\cdot\|\cdot$ represents the concatenation operation. Our query vectors q_i interact with a set of n vectors to define a pointing distribution over the action space. As soon as the first node is sampled, the query vector updates its inputs with the previously sampled node using its sequential representation to select the subsequent nodes.

Pointing Mechanism

We use a pointing mechanism to learn a distribution over node outputs given encoded actions (nodes) and a state representation (query vector). Our pointing mechanism is parameterised by two trainable attention matrices $K \in \mathbb{R}^{d\times d}$ and $Q \in \mathbb{R}^{d\times d}$ and vector $v \in \mathbb{R}^d$ as

$$u_j^i = \begin{cases} v^\top \tanh(Ko_j + Qq_i), & \text{if } j > a_{i-1} \\ -\infty, & \text{otherwise}, \end{cases} \tag{5.17}$$

where

$$p_\theta\left(a_i \mid a_{<i}, \bar{S}\right) = \text{softmax}(C\tanh(u^i)) \tag{5.18}$$

represents a probability distribution over n actions, given a query vector q_i with $u^i \in \mathbb{R}^n$. We mask probabilities of nodes prior to the current a_i as we only consider choices of nodes in which $a_i > a_{i-1}$ due to symmetry. This ensures a smaller action space for our model, i.e., $n(n-1)/2$ possible feasible permutations of the input. We clip logits in $[-C, +C]$ [21], where $C \in \mathbb{R}$ is a parameter to control the entropy of u^i.

5.5.3 Value Decoder

Similar to the policy decoder, our value decoder works by reading tour representations from S and S' and a graph representation from S. That is, given embeddings z for each node, the value decoder works by reading the outputs z_i for each node in the tour and the sequence hidden vectors h_n, h'_n to estimate the value of a state as

$$V_\phi(\bar{S}) = W_r\, \sigma_r\Big(W_z\Big(\frac{1}{n}\sum_{i=1}^{n} z_i + h_v\Big) + b_z\Big) + b_r\,, \tag{5.19}$$

with

$$h_v = (W_v h_n + b_v) \| (W_{v'} h'_n + b_{v'}), \tag{5.20}$$

where $W_z \in \mathbb{R}^{d\times d}$, $W_r \in \mathbb{R}^{1\times d}$, $b_z \in \mathbb{R}^d$, $b_r \in \mathbb{R}$ are learned parameters that map the state representation to a real valued output, $W_v, W_{v'} \in \mathbb{R}^{\frac{d}{2}\times d}$, $b_v, b_{v'} \in \mathbb{R}^{\frac{d}{2}}$ map the tours to a combined value representation and ϕ are the combined learnable parameters of the value approximation[2]. We use a *mean pooling* operation in (5.19) to combine node representations z_i in a single graph representation. This vector is then combined with the tour representation h_v to estimate current state values.

[2]Note that parameters θ and ϕ have shared parameters in the form of the encoding layers.

5.6 Policy Gradient Optimisation

In our formulation, we maximise the expected rewards given a state $\bar{S}$ defined as

$$J(\theta \mid \bar{S}) = \mathbb{E}_{\pi_\theta}[G_t \mid \bar{S}]\,. \tag{5.21}$$

Thus, during training, we define the total objective over a distribution $\mathcal{S}$ of uniformly distributed TSP graphs (solutions) in $[0, 1]^2$ as

$$J(\theta) = \mathbb{E}_{\bar{S}\sim\mathcal{S}}[J(\theta \mid \bar{S})]. \tag{5.22}$$

To optimise our policy, we resort to the policy gradient learning rule, which provides an unbiased gradient estimate w.r.t. the model's parameters θ. During training, we draw B i.i.d. transitions and approximate the gradient of $J(\theta)$, indexed at $t = 0$ as

$$\nabla_\theta J(\theta) \approx \frac{1}{B}\frac{1}{T}\Big[\sum_{b=1}^{B}\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t^b \mid \bar{S}_t^b)(G_t^b - V_\phi(\bar{S}_t^b))\Big] \tag{5.23}$$

where the advantage function is defined as $\mathcal{A}_t^b = G_t^b - V_\phi(\bar{S}_t^b)$ and the superscript b represents a transition sample from the the mini-batch of size B, i.e., $b \in \{1, \ldots, B\}$. To avoid premature convergence to a sub-optimal policy [162], we add an entropy bonus

$$H(\theta) = \frac{1}{B}\sum_{b=1}^{B}\sum_{t=0}^{T-1} H(\pi_\theta(\cdot \mid \bar{S}_t^b))\,, \tag{5.24}$$

with $H(\pi_\theta(\cdot \mid \bar{S}_t^b)) = -\mathbb{E}_{\pi_\theta}[\log \pi_\theta(\cdot \mid \bar{S}_t^b)]$, and similarly to (5.23) we normalise values in (5.24) dividing by k, i.e., the number of indices to select ($k = 2$ for 2-opt).

Moreover, we increase the length of an episode after a number of epochs, i.e., at epoch e, T is replaced by T_e. The value network is trained on a mean squared error objective between its predictions and Monte Carlo estimates of the returns, formulated as an additional objective

$$\mathcal{L}(\phi) = \frac{1}{B}\frac{1}{T}\Big[\sum_{b=1}^{B}\sum_{t=0}^{T-1} \Big\|G_t^b - V_\phi(\bar{S}_t^b))\Big\|_2^2\Big]\,. \tag{5.25}$$

Afterwards, we combine the previous objectives and perform gradient updates via adaptive moment estimation (ADAM) [124], with $\beta_H, \beta_V \in \mathbb{R}$ representing weights of (5.24) and (5.25), respectively. Note that our model is close to REINFORCE [229] with periodic episode length updates and added value function approximation. In our case, this is beneficial as, at the start, the agent learns how to behave over small episodes for easier credit assignment, later tweaking its policy over larger horizons. The complete algorithm is depicted in Algorithm 1.

Algorithm 1: Policy gradient training

Input: Policy network π_θ, critic network V_ϕ, number of epochs E, number of batches $\mathbf{N}_B$, batch size B, step limit $\mathbb{T}$, length of episodes T_e, learning rate λ

Initialise policy and critic parameters θ and ϕ;
for $e = 1, \ldots, E$ **do**
  $T \leftarrow T_e$
  for n $= 1, \ldots, \mathbf{N}_B$ **do**
    $t \leftarrow 0$;
    Initialise random $\bar{S}_0^b, \ \forall b \in \{1, \ldots, B\}$;
    while $t < \mathbb{T}$ **do**
      $t' \leftarrow t$;
      while $t - t' < T$ **do**
        $A_t^b \sim \ \pi_\theta(.|\bar{S}_t^b), \ \forall b$;
        Take A_t^b, observe $\bar{S}_{t+1}^b, R_t^b, \ \forall b$;
        $\bar{S}_t^b \leftarrow \bar{S}_{t+1}^b, \ \forall b$;
        $t \leftarrow t + 1$;
      for $i \in \{t', \ldots, t-1\}$ **do**
        $G_i^b \leftarrow \sum_{\tilde{t}=i}^{t'+T-1} \gamma^{\tilde{t}-t'} R_{\tilde{t}}^b \ , \ \forall b$;
      $g_\theta \leftarrow \frac{1}{Bk}\Big[\frac{1}{T}\sum_{b=1}^{B}\sum_{i=t'}^{t-1} \nabla_\theta \log \pi_\theta(A_i^b \mid \bar{S}_i^b)\mathcal{A}_i^b + \beta_H \nabla_\theta H(\pi_\theta(\cdot \mid \bar{S}_i^b))\Big]$;
      $g_\phi \leftarrow \frac{1}{BT}\Big[\beta_V \sum_{b=1}^{B}\sum_{i=t'}^{t-1} \nabla_\phi \Big\| G_t^b - V_\phi(\bar{S}_i^b))\Big\|_2^2\Big]$;
      $\theta, \phi \leftarrow \mathrm{ADAM}(\lambda, -g_\theta, g_\phi)$;

5.7 Experiments and Results

We conduct extensive experiments to investigate the performance of our proposed method. We consider three benchmark tasks, Euclidean TSPs with 20, 50, and 100 nodes, named TSP20, TSP50, and TSP100, respectively. For all tasks, node coordinates are drawn uniformly at random in the unit square $[0, 1]^2$ during training. For validation, a fixed set of TSP instances with their respective optimal solutions is used for hyperparameter optimisation. For a fair comparison, we use the *same* test dataset as reported in [115, 128] containing 10,000 instances for each TSP size. Thus, previous results reported in [128] are comparable to ours in terms of solution quality (optimality gap). Results from [231] are not measured in the same data but use the same data generation process. Thus, we report the optimality gaps reported in the original paper. We note, however, that because the number of instances is large, the difference in average performance is mitigated. Moreover, we report running times reported in [115, 128, 231]. Since time can vary due to implement-

ations and hardware, we rerun the method of [128] in our hardware to assess the running times. Due to provided supervised samples, the method of [115] is not ideal for combinatorial problems. Thus, we compare our results in more detail to [128] (running time and solution quality), and [231] (solution quality and sample efficiency).

5.7.1 Experimental Settings

All our experiments use a similar set of hyperparameters defined manually using the validation performance. We use a batch size $B = 512$ for TSP20 and TSP50 and $B = 256$ for TSP100 due to limited GPU memory. For this reason, we generate 10 random mini-batches for TSP20 and TSP50 and 20 mini-batches for TSP100 in each epoch. TSP20 trains for 200 epochs as convergence is faster for smaller problems, whereas TSP50 and TSP100 train for 300 epochs. We use the same $\gamma = 0.99$, ℓ_2 penalty of 1×10^{-5} and learning rate $\lambda = 0.001$, λ decaying by 0.98 at each epoch. Loss weights are $\beta_V = 0.5$, $\beta_H = 0.0045$ for TSP20 and TSP50, $\beta_H = 0.0018$ for TSP100. β_H decays by 0.9 after every epoch for stable convergence. In all tasks, $d = 128$, $\mathbb{L} = 3$ and we employ one bi-LSTM block. The update in episode lengths are $T_1 = 8, T_{100} = 10, T_{150} = 20$ for TSP 20; $T_1 = 8, T_{100} = 10, T_{200} = 20$ for TSP50; and $T_1 = 4, T_{100} = 8, T_{200} = 10$ for TSP100. $C = 10$ is used during training and testing. v is initialised as $\mathcal{U}(\frac{-1}{\sqrt{d}}, \frac{1}{\sqrt{d}})$ and remaining parameters are initialised according to PyTorch's default parameters.

We train on an RTX 2080Ti GPU, generating random feasible initial solutions on the fly at each epoch. Each epoch takes an average time of 2m 01s, 3m 05s, and 7m 16s for TSP20, TSP50, and TSP100, respectively. We clip rewards to 1 to favour non-greedy actions and stabilise learning. Due to GPU memory, we employ mixed-precision training [113] for TSP50 and TSP100. For comparison with [231], we train for a maximum step limit of 200. Note that our method is more sample efficient than the proposed in [231], using 50% and 75% of the total samples for TSP20 and TSP50/100 during training. During testing, we run our policy for 500, 1,000, and 2,000 steps to compare to previous works. Our implementation is available online[3].

5.7.2 Experimental Results and Analysis

We learn TSP20, TSP50, and TSP100 policies and depict the optimality gap and its exponential moving average in the log scale in Figure 5.3. The optimality gap is averaged over 256 validation instances and 200 steps (same as training) in the figure. The results show that we can learn effective policies that decrease the optimality gap over the training epochs. We also point out that increasing the episode length improved validation performance as we consider longer planning horizons in (5.23). Moreover, it is interesting to note that the optimality gap grows with the instance

[3] `https://github.com/paulorocosta/learning-2opt-drl`

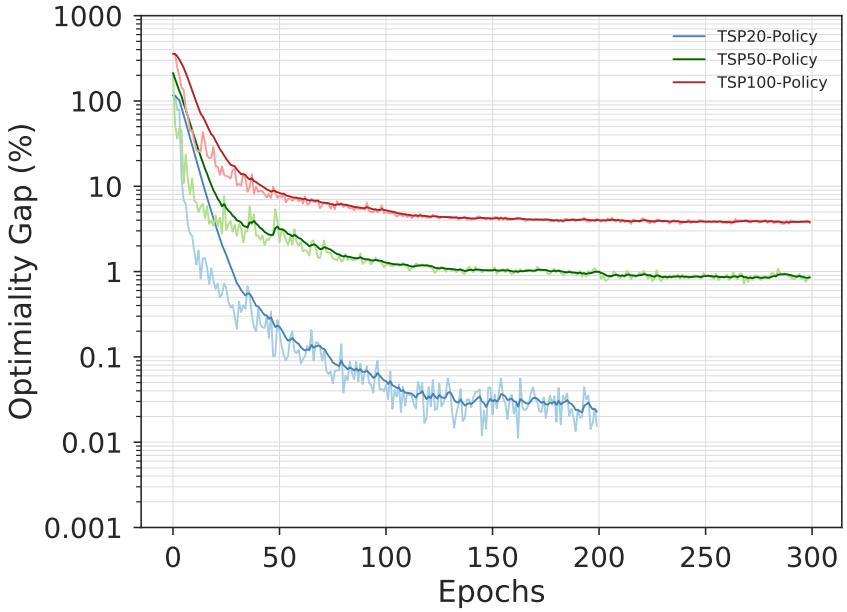


Figure 5.3: Optimality gaps on 256 validation instances in the log scale. Strong colours represent the exponential moving average of the optimality gaps. Light colours show the average optimality gap at each epoch of the validation instances whilst sampling 200 solutions (steps).

size as solving larger TSP instances is harder. Additionally, we report the gaps of the best performing policies in Figure 5.4. In the figure, we show the optimality gap of the best solution for 512 test instances over 2,000 steps. Here, results show that we can quickly reduce the optimality gap initially, and later steps attempt to fine-tune the best tour. In the experiments, we find the optimal solution for TSP20 instances and stay within optimality gaps of 0.1% for TSP50 and 0.7% for TSP100. Overall, our policies can be seen as a solver requiring only random initial solutions and sampling to achieve near-optimal solutions.

To showcase that, we compare the learned policies with classical 2-opt first improvement (FI) and best improvement (BI) heuristics, which select the first and best cost-reducing 2-opt operation, respectively. Since local search methods can get stuck in local optima, we include a version of the heuristics using *restarts*. That is, we restart the search at a random solution as soon as we reach a local optimum. We run all heuristics and learned policies on 512 TSP100 instances for a maximum of 1,000 steps starting from the same solutions. The boxplots in Figure 5.5 depict the results. We observe that our policy (TSP100-Policy) outperforms classical 2-opt heuristics finding tours with lower median and less dispersion. These results support our initial hypothesis that considering future rewards in the choice of 2-opt moves leads to better solutions. Moreover, our method avoids the worst-case $O(n^2)$ complexity

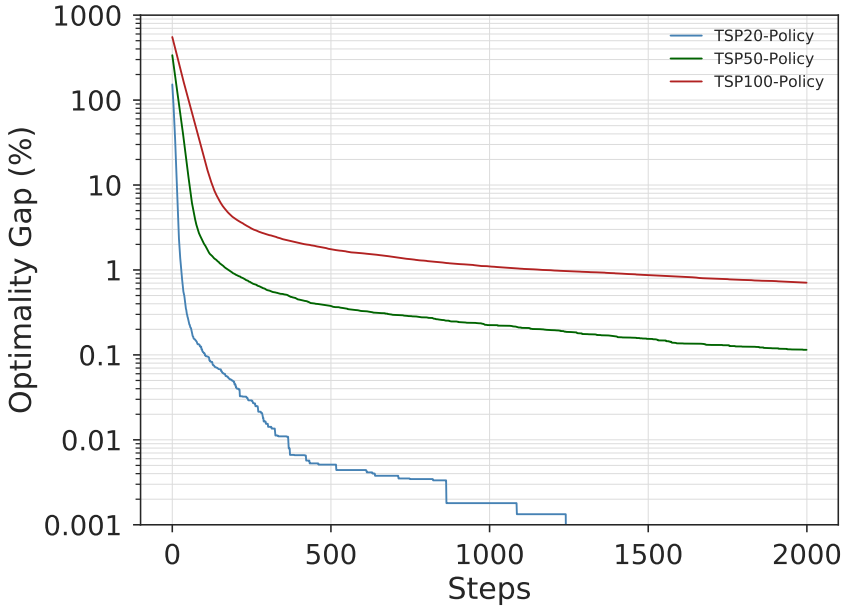


Figure 5.4: Optimality gaps of best-found tours on 512 testing instances over 2,000 sampled solutions (steps) in the log scale.

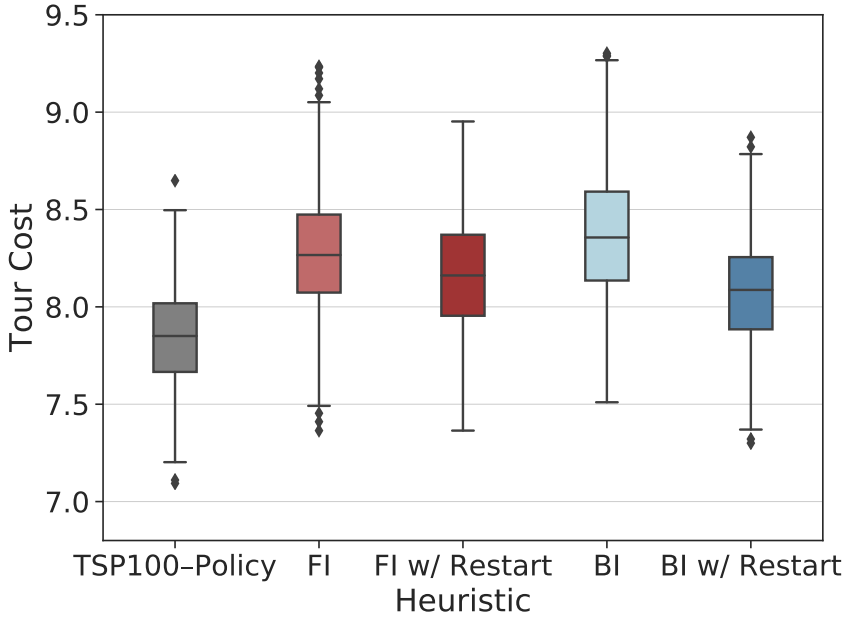


Figure 5.5: Tour costs of learned, first improvement (FI) and best improvement (BI) heuristics with restarts on TSP100 instances after sampling 1,000 solutions (steps).

of selecting the next solution of FI and BI.

Comparison to Classical Heuristics, Exact and Learning Methods

We report results on the same 10,000 instances for each TSP size as in [128] and rerun the optimal results obtained by Concorde to derive optimality gaps. We compare against Nearest, Random and Farthest Insertion constructions heuristics. Moreover, we include the vehicle routing solver of OR-Tools [178] containing 2-opt and LKH as improvement heuristics.

We add to the comparison recent deep learning methods based on construction and improvement heuristics, including supervised [115, 223] and reinforcement [21, 62, 121, 128, 231] learning methods. We note, however, that supervised learning is not ideal for combinatorial problems due to the lack of optimal labels for large problems. Previous works to [128] are presented with their reported running times and optimality gaps as in the original paper. For recent works, we present the optimality gaps and running times as reported in [115, 128, 231]. We report previous results using greedy, sampling and search decoding and refer to the methods by their neural network architecture. We note that the test dataset used in [231] is not the same, but the data generation process and size are identical. For this reason, we report the optimality gaps of the original paper and recalculate the corresponding costs based on the optimal costs in our datasets. We focus our attention on GAT [128], and GAT-T [231] (GAT-Transformer) representing the best construction and improvement heuristic, respectively. Note that we omit LKH for the TSP as it achieves optimal results. Note that for the TSP, new works such as in [127] appeared after the first version of this chapter and are not included in the table with the results.

Our results, in Table 5.1, show that with only 500 steps, our method outperforms traditional construction heuristics, learning methods with greedy decoding and OR-Tools achieving 0.01%, 0.36% and 1.84% optimality gap for TSP20, TSP50, and TSP100, respectively. Moreover, we outperform GAT-T requiring half the number of steps (500 vs 1,000). We note that with 500 steps, our method also outperforms all previous reinforcement learning methods using sampling or search, including GAT [62] applying 2-opt local search on top of generated tours. Our method only falls short of the supervised learning method GCN [115], using beam search and shortest tour heuristic. However, GCN [115], similar to samples in GAT [128], uses a beam width of 1,280, i.e. it samples more solutions. Increasing the number of samples (steps) increases the performance of our method. When sampling 1,000 steps (280 samples short of GCN [115], and GAT [128]) we outperform all previous methods that do not employ further local search improvement and perform on par with GAT-T on TSP50, using 5,000 samples (5 times as many samples). For TSP100, sampling 1,000 steps results in a lower optimality gap (1.26%) than all compared methods. Lastly, increasing the sample size to 2,000 results in even lower gaps,

0.00% (TSP20), 0.12% (TSP50) and 0.87% (TSP100).

Table 5.1: Performance of TSP methods w.r.t. Concorde. *Type*: **SL**: Supervised Learning, **RL**: Reinforcement Learning, **S**: Sampling, **G**: Greedy, **B**: Beam Search, **BS**: **B** and Shortest Tour and **T**: 2-opt Local Search. *Time*: Time to solve 10,000 instances reported in [115, 128, 231] and ours.

	Method	Type	TSP20			TSP50			TSP100		
			Cost	Gap	Time	Cost	Gap	Time	Cost	Gap	Time
	Concorde [4]	Solver	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)
Heuristics	OR-Tools [178]	S	3.85	0.37%		5.80	1.83%		7.99	2.90%	
Heuristics	Nearest Insertion	G	4.33	12.91%	(1s)	6.78	19.03%	(2s)	9.46	21.82%	(6s)
Heuristics	Random Insertion	G	4.00	4.36%	(0s)	6.13	7.65%	(1s)	8.52	9.69%	(3s)
Heuristics	Farthest Insertion	G	3.93	2.36%	(1s)	6.01	5.53%	(2s)	8.35	7.59%	(7s)
Const.+Greedy	PtrNet [223]	SL	3.88	1.15%		7.66	34.48%			-	
Const.+Greedy	GCN [115]	SL	3.86	0.60%	(6s)	5.87	3.10%	(55s)	8.41	8.38%	(6m)
Const.+Greedy	PtrNet [21]	RL	3.89	1.42%		5.95	4.46%		8.30	6.90%	
Const.+Greedy	S2V [121]	RL	3.89	1.42%		5.99	5.16%		8.31	7.03%	
Const.+Greedy	GAT [62]	RL,T	3.85	0.42%	(4m)	5.85	2.77%	(26m)	8.17	5.21%	(3h)
Const.+Greedy	GAT [128]	RL	3.85	0.34%	(0s)	5.80	1.76%	(2s)	8.12	4.53%	(6s)
Const.+Search	GCN [115]	SL,B	3.84	0.10%	(20s)	5.71	0.26%	(2m)	7.92	2.11%	(10m)
Const.+Search	GCN [115]	SL,BS	3.84	0.01%	(12m)	**5.70**	**0.01**%	(18m)	7.87	1.39%	(40m)
Const.+Search	PtrNet [21]	RL,S		-		5.75	0.95%		8.00	3.03%	
Const.+Search	GAT [62]	RL,S	3.84	0.11%	(5m)	5.77	1.28%	(17m)	8.75	12.70%	(56m)
Const.+Search	GAT [62]	RL,S,T	3.84	0.09%	(6m)	5.75	1.00%	(32m)	8.12	4.64%	(5h)
Const.+Search	GAT {1280} [128]	RL,S	3.84	0.08%	(5m)	5.73	0.52%	(24m)	7.94	2.26%	(1h)
Impr.+Sampling	GAT-T {1000} [231]	RL	3.84	0.03%	(12m)	5.75	0.83%	(16m)	8.01	3.24%	(25m)
Impr.+Sampling	GAT-T {3000} [231]	RL	3.84	0.00%	(39m)	5.72	0.34%	(45m)	7.91	1.85%	(1h)
Impr.+Sampling	GAT-T {5000} [231]	RL	3.84	0.00%	(1h)	5.71	0.20%	(1h)	7.87	1.42%	(2h)
Impr.+Sampling	Ours {500}	RL	3.84	0.01%	(5m)	5.72	0.36%	(7m)	7.91	1.84%	(10m)
Impr.+Sampling	Ours {1000}	RL	**3.84**	**0.00**%	(10m)	5.71	0.21%	(13m)	7.86	1.26%	(21m)
Impr.+Sampling	Ours {2000}	RL	**3.84**	**0.00**%	(15m)	5.70	0.12%	(29m)	**7.83**	**0.87**%	(41m)

Generalisation to Larger Instances

Since we are interested in learning general policies that can solve the TSP regardless of its size, we test the performance of our policies when learning on TSP50 instances (TSP50-Policy) and applying it on larger TSP100 instances. Results, in Table 5.2, show that we can extract general enough information to still perform well on 100 nodes. Similar to a TSP100-Policy, our TSP50-Policy can outperform previous reinforcement learning construction approaches and requires fewer samples. With 1,000 samples, TSP50-Policy performs similarly to GAT-T [231] using 3,000 samples, at a 1.86% optimality gap. These results are closer to optimal than previous learning methods without further local search improvement as in GCN [115]. When increasing to 2,000 steps, we outperform all compared methods at a 1.37% optimality gap.

Table 5.2: Performance of policies trained on 50 and 100 nodes on TSP100 instances

	TSP100-Policy		TSP50-Policy	
Steps	Cost	Gap	Cost	Gap
500	7.91	1.84%	7.98	2.78%
1000	7.86	1.26%	7.91	1.86%
2000	7.83	0.87%	7.87	1.37%

Running Times and Sample Efficiency

Comparing running times is difficult due to varying hardware and implementations among different approaches. In Table 5.1, we report the running times to solve 10,000 instances as reported in [115, 128, 231] and ours. We focus on learning methods, as classical heuristics and solvers are efficiently implemented using multi-threaded CPUs. We note that our method cannot compete in speed with greedy methods as we start from poor solutions and require sampling to find improved solutions. This is neither surprising nor discouraging, as one can see these methods as a way to generate initial solutions for an improvement heuristic like ours. We note, however, that while sampling 1,000 steps, our method is faster than GAT-T [231] even though we use a less powerful GPU (RTX 2080Ti vs Tesla V100). Moreover, our method requires fewer samples to achieve superior performance. The comparison to GAT [128] is not so straightforward as they use a GTX 1080Ti and a different number of samples. For this reason, we run GAT [128] using our hardware and report running times sampling the same number of solutions in Table 5.3. Our method is slower for TSP20 and TSP50 sampling 2,000 solutions. However, as we reach TSP100, our method can be computed faster and, overall, requires less time to produce shorter tours.

Table 5.3: Performance of GAT [128] vs our method. Results are compared on the same hardware sampling the same number of solutions.

Method	TSP20		TSP50		TSP100	
	Cost	Time	Cost	Time	Cost	Time
GAT {500}	3.839	**(3m)**	5.727	(10m)	7.955	(27m)
Ours {500}	3.836	(5m)	5.716	**(7m)**	7.907	**(10m)**
GAT {1,000}	3.838	**(4m)**	5.725	(14m)	7.947	(42m)
Ours {1,000}	3.836	(10m)	5.708	**(13m)**	7.861	**(21m)**
GAT {2,000}	3.838	**(5m)**	5.722	**(22m)**	7.939	(1h13m)
Ours {2,000}	3.836	(15m)	5.703	(29m)	7.832	**(41m)**

Ablation Study

In Table 5.4, we present an ablation study of the proposed method. We measure the performance at the beginning and towards the end of the training procedure, i.e., at epochs 10 and 200, rolling out policies for 1,000 steps for 512 TSP50 instances and 10 trials. We point out that our main objective is to find good policies as early as possible. In other words, good policies found earlier are considered better than waiting more time to obtain the same results. We observe that removing the LSTM (a) affects performance the most, leading to a large 134.42% gap at epoch 200. Removing the GCN component (b) has a lower influence but also reduces the overall quality of policies, reaching a 0.30% optimality gap. We then test the bidirectional LSTM (c) effect, replacing it with a single LSTM. In this case, gaps are even higher, at 2.20%, suggesting that encoding the symmetry of the tours is important. We also compare two variants of the proposed model, one that does not take as input the best solution (d) and one that shares the parameters of the encoding units (e). For these cases, we note that the final performance is similar to the proposed method, i.e. 0.22% optimality gap. However, in our experiments, the proposed method can achieve better policies faster, reaching a 3.0% gap at epoch 10, whereas (d) and (e) yield policies at the 4.55% and 5.15% level, respectively.

Table 5.4: Ablation studies on 512 TSP50 instances running policies for 1,000 steps.

	Epoch: 10		Epoch: 200	
	Opt. Gap (%)	Cost	Opt. Gap (%)	Cost
Proposed	**3.00 ± 0.08**	**5.87**	**0.22 ± 0.01**	**5.72**
(a) w/o bi-LSTM	203.87 ± 0.61	17.33	134.42 ± 0.56	13.37
(b) w/o GCN	9.74 ± 0.08	6.26	0.30 ± 0.01	5.72
(c) w/o bidirectional	17.94 ± 0.15	6.73	2.20 ± 0.05	5.82
(d) w/o best solution	4.55 ± 0.04	5.96	**0.22 ± 0.02**	**5.72**
(e) shared encoder	5.15 ± 0.06	6.00	0.23 ± 0.01	5.72

Generalisation to Real-world TSP instances

In Table 5.8 in Appendix 5.A, we study the performance of our method on TSPlib [184] instances. In general, these instances come from different node distributions than those seen during training, and it is unclear whether our learned policies can be reused for these cases. We compare the results of the policy trained on TSP100 sampling actions for 2,000 steps to results obtained from OR-Tools. We note that of 35 instances tested, our method outperforms OR-Tools in 12 instances. These results are encouraging as OR-Tools is a very specialised heuristic solver. When we compare optimality gaps 8.61% (ours) and 3.70%, we see that our learned

policies are not too far from OR-Tools even though our method never trains on instances with more than 100 nodes. The difference in performance increases for large instances, indicating that fine-tuning or training policies for more nodes and different distributions can potentially reduce this difference. However, similar to the results in Table 5.2, our method still can achieve good results on instances with more than 100 nodes, such as ts225 (0.86% gap).

5.8 Expanding to other Routing Problems

In this section, we show how the proposed methodology can be extended to more general routing problems. In particular, we expand the proposed model to search for solutions for the multiple TSP and the VRP. The specific changes to handle these problems are detailed in the following sections.

5.8.1 The Multiple Travelling Salesmen Problem

The multiple TSP (mTSP) [17] is an extension to the original TSP that includes a number of salesmen m starting and ending their tours at a depot location. The goal is to construct tours for the m salesmen such that the total cost of the tours is minimised. In our formulation, we include an extra depot node with index 0 and coordinates $x_0 \in [0, 1]^2$ and the remaining customer nodes $\{1, \dots, n\}$. Since adding more salesmen without any imposed constraint would lead to the same solution as the TSP, we include two additional constraints in the problem formulation, (i) each salesman needs to be utilised in a feasible solution and (ii) in a given salesman tour at least $\nu = 2$ nodes have to be visited, excluding the depot. The latter ensures that a tour cannot be formed by visiting just one node and returning to the depot, reducing the remaining problem to a TSP with $n - 1$ nodes. The remaining constraints are usual TSP constraints.

Instance Generation

We follow the same instance generation procedure as the TSP, i.e., we draw $n + 1$ nodes (including the depot) at random from a uniform distribution in the 0-1 square.

Initial Solution Generation

We represent a solution S to the mTSP, as an ordered list of nodes, $S = (s_1, \dots, s_p)$, where $s_i \in \{0, \dots, n\}$. In our solution, each tour is represented by adding the depot index at the beginning and ending of each tour without repetition. For example, a solution with two tours and $n = 5$ is represented as $S = (0, 1, 2, 0, 4, 3, 5, 0)$, where the first tour visits nodes 0, 1, 2 and 0 and the second tour visits nodes 0, 4, 3, 5 and

0. The size of a solution p depends on n (number of customers) and m (number of salesmen) and it is expressed as $p = n + m + 1$.

We generate initial solutions by first sampling instances and then breaking the canonical ordering of nodes into m tours. We start from a solution containing all the nodes, i.e., $S = (0, 1 \dots, n)$ and find the depot positions of the tours by first computing the number of required splits $\eta = \lfloor \frac{n}{m} \rfloor$, then for $m - 1$ depot positions (the last depot position is always at the end of the solution), we find the indices of the depot by:

$$i_0(i) = (i-1)\eta + \nu + 2 \quad \forall i \in 1, \dots, m-1 \tag{5.26}$$

and we insert each depot at its corresponding index. Lastly, we add a depot to the end of the solution S, ensuring we have short and long tours in a given initial solution.

mTSP Neural Architecture

Encoder We use the same encoding architecture for the mTSP as for the TSP; however, the embedding layer and the $\mathbb{L}$ GCN layers operate only on the $n + 1$ node coordinates of the underlying instance graph, ensuring we only encode the information about the instance. That is,

$$x_i^{\ell+1} = x_i^{\ell} + \sigma_r\Big(\sum_{j \in \mathcal{N}(i)} \tilde{e}_{i,j}(W_g^{\ell} x_j^{\ell} + b_g^{\ell})\Big), \forall i \in \{0, \dots, n\} \tag{5.27}$$

here we define x_i as the coordinates of node $i \in \{0, \dots, n\}$, i.e., respecting the labels of the nodes 0 until n in the instance graph. Note that this is different (although equivalent) from the TSP case where we read the input features from each node in the order defined by the solution, i.e., x_{s_i}. This difference in the mTSP and VRP cases ensures that we embed the features in the GCN layers that correspond to the instance graph and not to the solutions, which repeats the depot multiple times.

To encode the ordering of a solution, the RNN layers take as input the graph embedded node features and proceed to perform the solution encoding, i.e.,

$$(h_i^{\rightarrow}, c_i^{\rightarrow}) = \text{RNN}(z_i^{\rightarrow}, (h_{i-1}^{\rightarrow}, c_{i-1}^{\rightarrow})), \quad \forall i \in \{1, \dots, p\} \tag{5.28}$$

$$(h_i^{\leftarrow}, c_i^{\leftarrow}) = \text{RNN}(z_i^{\leftarrow}, (h_{i+1}^{\leftarrow}, c_{i+1}^{\leftarrow})), \quad \forall i \in \{p, \dots, 1\} \tag{5.29}$$

where z_i corresponds to the node features of node s_i, i.e, $z_i \in \{x_0^{\mathbb{L}}, .., x_n^{\mathbb{L}}\}$, and $z_i = x_{s_i}^{\mathbb{L}}$.

Tour Length Constraints and Masking The first action masks all depot positions and the last customer node at the end of the last tour. Then the second action considers only customer nodes indices that are greater than the index a_1 that, when selected, results in the tour with the minimum length to be greater or equal than

ν. Let $c(S, a_1, j) = \min(c^1(S, a_1, j), \ldots, c^m(S, a_1, j))$, denote the number of customer nodes in the shortest tour in the resulting solution when applying the 2-opt operation defined by (a_1, j) to a solution S, then the masking becomes:

$$u_j^2 = \begin{cases} \tilde{u}_j^2, & \text{if } j > a_1 \wedge c(S, a_1, j) \geq \nu \\ -\infty, & \text{otherwise}. \end{cases} \tag{5.30}$$

where $\tilde{u}_j^i = v^\top \tanh(Ko_j + Qq_i)$. To encode the previous masking, we keep track of an auxiliary indicator $b_i \in \{-1, 0, 1\}$, where $i \in \{1, \ldots, p\}$, represents if a node is right before (-1), after (1) or further away (0) from a depot when traversing the solution from left to right. Thus, checking if $c(S, a_1, j) \geq 2$ can be achieved by

$$u_j^2 = \begin{cases} \tilde{u}_j^2, & \text{if } b_{a_1} = -1 \wedge b_j \neq -1 \wedge j > a_1 \\ \tilde{u}_j^2, & \text{if } b_{a_1} = 1 \wedge b_j \neq 1 \wedge j > a_1 \\ \tilde{u}_j^2, & \text{if } b_{a_1} = 0 \wedge j > a_1 \\ -\infty, & \text{otherwise.} \end{cases} \tag{5.31}$$

Training and Experimental Parameters

We make a few modifications to the training parameters in comparison to the TSP case. We reduce the size of the mini-batches to 64, 128 and 256 for mTSP20, 50, and 100, respectively. This modification allows for faster training when using a more complex masking operation and longer solutions. We train models on instance problems with two values of $m \in \{2, 4\}$. Similarly to the TSP, we sample 10 mini-batches at each epoch and train mTSP20 for 200 epochs and mTSP50 for 300 epochs. To avoid high training times of mTSP100, we use the best-learned policy on mTSP50 as a warm-start for mTSP100 and train for 100 epochs. Our random initial solutions are far from optimality with costs 11.51, 26.98, 52.78 for $m = 2$ and 12.46, 27.94, 53.80 for $m = 4$ over the increasing instance sizes. Each epoch takes on average 2m, 6m, and 10m for mTSP20, 50, and 100, respectively. We run two sets of experiments, one containing 1,000 instances to mitigate the high running times of our baselines and one with 10,000 instances to be comparable with the TSP experiments. The remaining parameters of the model remain the same as for the TSP.

Experimental Results and Analysis

We roll out the learned policies by sampling 2,000 solutions on each of the 1,000 and 10,000 sets of instances to assess the performance of our method. We compare the performance to an integer linear programming (ILP) formulation of the problem running the Gurobi solver [83] for a max of 30s for each instance. We also include the highly effective LKH3 [92] heuristic as a baseline as it balances solution quality

and speed. Moreover, LKH3 is the state-of-the-art algorithm for several routing problems. We implement both baselines in a serialised manner. This is comparable to our results as even though we sample actions in batches taking advantage of batch parallelisation, we perform the 2-opt actions in series.

Table 5.5: mTSP results on 1000 instances compared to the best results obtained using Gurobi (30s) and LKH3.

		mTSP20			mTSP50			mTSP100		
Salesmen	Method	Cost	Gap	Time	Cost	Gap	Time	Cost	Gap	Time
m=2	Gurobi (30s)	4.21	0.00%	(5m)	5.95	0.31%	(5h)	9.62	21.55%	(8h)
	LKH3	4.21	0.00%	(12m)	5.93	0.00%	(25m)	7.91	0.00%	(27m)
	Ours {2000}	4.21	0.02%	(3m)	5.95	0.33%	(5m)	8.05	1.69%	(9m)
m=4	Gurobi (30s)	5.33	0.00%	(3m)	6.58	0.10%	(5h)	9.68	15.84%	(8h)
	LKH3	5.33	0.00%	(25m)	6.58	0.00%	(28m)	8.35	0.00%	(32m)
	Ours {2000}	5.33	0.08%	(3m)	6.60	0.42%	(5m)	8.51	1.91%	(9m)

Table 5.6: mTSP results on 10000 instances compared to the best results obtained using Gurobi (30s) and LKH3. Gurobi is only run to mTSP20 due to high running times when solving mTSP50 and mTSP100 instances.

		mTSP20			mTSP50			mTSP100		
Salesmen	Method	Cost	Gap	Time	Cost	Gap	Time	Cost	Gap	Time
m=2	Gurobi (30s)	4.20	0.00%	(38m)	-	-	-	-	-	-
	LKH3	4.20	0.00%	(2h)	5.92	0.00%	(3h)	7.92	0.00%	(4h)
	Ours {2000}	4.20	0.02%	(25m)	5.94	0.35%	(39m)	8.05	1.65%	(1h)
m=4	Gurobi (30s)	5.31	0.00%	(30m)	-	-	-	-	-	-
	LKH3	5.31	0.00%	(5h)	6.56	0.00%	(5h)	8.35	0.00%	(6h)
	Ours {2000}	5.31	0.06%	(25m)	6.59	0.42%	(40m)	8.51	1.91%	(1h)

Comparison to Exact and Heuristics Baselines The results for the set of 1,000 instances are presented in Table 5.5. We observe that the learned policies are close to the performances of both Gurobi and LKH3 when solving instances with 20 nodes with 0.02%, 0.08% optimality gaps, respectively. Similar to the TSP, the gap increases as we increase the size of the instances. Moreover, as we increase the size of the instances, the performance of Gurobi running for just 30s decreases considerably, taking significantly longer (8h) and yielding results far from LKH3. On the other hand, our learned policies remain much closer (1.69% for 2TSP100, 1.91%for 4TSP100) to the best results found by LKH3 whilst requiring less time.

We also present the results on 10,000 instances as these should provide better estimates of the performance of our policies. We present the results in Table 5.6. Since Gurobi does not scale, we only provide the results from Gurobi for mTSP20.

The results are similar to those obtained in 1,000 instances, with our model finding similar costs to those found by LKH3 whilst requiring less running time than the heuristic.

5.8.2 The Capacitated Vehicle Routing Problem

In the capacitated vehicle routing problem (CVRP) [212], each customer node has an associated demand, and multiple routes should be constructed starting and ending at a depot. The CVRP is a generalisation of the mTSP. It considers that each vehicle (salesman) has a given capacity and that tours have to be formed such that the combined demand of all customers does not exceed the capacity of the vehicles.

Similar to mTSP, we add an extra depot node with index 0 and coordinates $x_0 \in [0,1]^2$ and consider the remaining nodes as customer nodes. We adopt the same formulation as in [128, 168], and define a capacity D for a single vehicle traversing all the routes. We associate each customer node $i \in \{1,\dots,n\}$ with a demand $0 \leq \delta_i \leq D$. Each route should start and end at the depot and should not exceed the vehicle's total capacity. Similar to [128] we assume a normalised capacity $\hat{D} = 1$ and use normalised demands $\hat{\delta}_i = \frac{\delta_i}{D}$, this allows us to learn general policies that can be used with different capacity magnitudes.

Instance Generation

For comparison, we follow [128, 168] and generate node coordinates sampled uniformly at random in the unit square. The unnormalised demands δ_i where $i \in \{1,\dots,n\}$, are sampled following a discrete uniform distribution from $\{1,\dots,9\}$ and the demand of the depot is $\delta_0 = 0$. Each problem size n defines different capacities D, with $D = 30, 40, 50$, for $n = 20, 50, 100$, and remain fixed for all instances.

Initial Solution Generation

Similar to the mTSP, we represent a solution S to the CVRP, as an ordered list of nodes, $S = (s_1,\dots,s_p)$, where $s_i \in \{0,\dots,n\}$. A tour is represented by adding the depot at the start and beginning of each tour. However, unlike the mTSP, where the number of salesmen is fixed, in the CVRP, a solution can have different lengths depending on the number of tours traversed. To allow for batching solutions, we compute the maximum length of a solution p. Let the maximum demand be $\delta^{\max} = \max(\delta_1,\dots,\delta_n)$ and maximum the number of customers served at maximum demand be $\zeta = \left\lfloor \frac{D}{\delta^{\max}} \right\rfloor$. Then we define the maximum number of possible tours $m^{\max} = \left\lceil \frac{n}{\zeta} \right\rceil$, and finally, the length of the tour is given by $p = n + m^{\max} + 1$. With our parameters, p corresponds to 28, 64 and 121 for $n = 20, 50, 100$.

We generate initial solutions by first sampling the node coordinates and demands. We define an initial solution traversing nodes in the sampled order, i.e., we

start with a solution $S = (0, 1, \ldots, n)$. We accumulate the sum of demands whilst traversing the nodes and construct a tour when $\sum_{i'=0}^{i} \hat{\delta}_{i'} > 1$. At which point we add a depot to the solution and start a new tour with the last visited node i. We repeat this procedure until we visit all customer nodes. Since not all solutions have the same length, we pad the solutions with depot nodes at the end. This allows us to batch solutions respecting their maximum sizes p and lets the algorithm add new depot locations to a solution if deemed necessary. For instance, a CVRP solution of the form $S = (0, 1, 2, 0, 3, 6, 5, 4, 0, \ldots, 0)$ represents two tours, one traversing nodes 0, 1, 2, 0 and the other traversing nodes 0, 3, 6, 5, 4, 0. The remaining depots are padded to complete the solution.

CVRP Neural Architecture

Embedding Layer To allow our model to use both node coordinates and demands of the nodes, we provide the normalised demands $\hat{\delta}_i$ of each node to the embedding layer, where each x_i is the coordinate of node $i \in \{0, \ldots, n\}$ and adjust the dimension of the parameter W_x accordingly. The embedding layers then produces node features following

$$x_i^0 = W_x[x_i || \hat{\delta}_i] + b_x \,. \tag{5.32}$$

GCN Layers We compute the Euclidean distances using the node coordinates x_i as in the TSP case and use the normalised edges $\tilde{e}_{i,j}$ to compute the graph node features similar to the mTSP case by applying $\mathbb{L}$ GCN layers following Eq. (5.27).

RNN Layers We adjust the dimensions and follow the same architecture of the mTSP, i.e. Eqs (5.28) and (5.29), in which the node features $x_i^{\mathbb{L}}$, $i \in \{0, \ldots, n\}$ are used to compose nodes in a solution where $S = (s_1, \ldots, s_p)$, $s_i \in \{0, \ldots, n\}$ and $z_i = x_{s_i}^{\mathbb{L}}$.

Capacity Constraints and Masking To allow for only feasible solutions, we need to ensure that a 2-opt action will not create tours that do not respect the capacity constraints. Thus, before the action selection starts, we create a feasibility matrix $P \in \{0, 1\}^{p \times p}$ and go through all possible $p(p-1)/2$ node exchanges and check if it forms a feasible solution where the maximum demand across all tours do not exceed the capacity D. Then for the first element of the action a_1:

$$u_j^1 = \begin{cases} \tilde{u}_j^1, & \text{if } \max_{k \in \{1, \ldots, p\}} P[j, k] = 1 \\ -\infty, & \text{otherwise} \,. \end{cases} \tag{5.33}$$

and for a_2:

$$u_j^2 = \begin{cases} \tilde{u}_j^2, & \text{if } P[a_1, j] = 1 \\ -\infty, & \text{otherwise}. \end{cases} \tag{5.34}$$

Training and Experimental Parameters

We train on CVRP20 and CVRP50 instances with a mini-batch size of 64 and 128. We do not train our policies on CVRP100 due to high training times in our hardware, but we report the performance of the policy trained on CVRP50 instances tested on CVRP100. For the same reason, we warm-start CVRP50 with a policy trained on CVRP20 and train for additional 200 epochs. Our initial solutions have average costs of 12.53, 29.79, 58.19 for $n = 20, 50, 100$. Each epoch takes 1m83s and 7m30s for instances with 20 and 50 nodes. The remaining training parameters remain identical to the TSP.

Experimental Results and Analysis

We compare our results to other end-to-end deep learning methods and CVRP heuristics. We run our policies for 500, 1000 and 2000 steps on the same 10,000 instances as in [128]. This allows us to compare both optimality gaps and costs. We include the LKH3 baseline from the previous paper and rerun both the deep learning model and the baselines to compare running times. We also compare to the improvement method GAT-T [231] and report the objective gaps and times reported in their original paper since no pre-trained model is available. While learning the CVRP, we note that GAT-T starts from the nearest neighbour heuristic, with much lower costs than our initial solutions. This allows for the model to experience a higher number of solutions that are closer to optimality, i.e., when the action selection is harder. We do not employ such a strategy and always start learning from randomised solutions and thus have to learn in a more complex setup. We also include in the comparison the improvement method L2I; however, the reported results are only averaged over 2,000 instances and cannot be compared to the remaining methods. We also include in the comparison the results obtained with NLNS. Lastly, we compare to the recent DPDP, reporting results for the VRP with 100 nodes and DPDP with beam sizes of 10K (10 thousand), 100K (100 thousand) and 1M (one million), for the VRP with 100 nodes.

Comparison to Heuristics and Learned Baselines We present the comparison to previously proposed methods in Table 5.7. Our method outperforms other reported deep reinforcement learning baselines for CVRP20. The best results are found after sampling 2000 solutions resulting in a 0.37% gap to LKH3. Note that our policy performs better than GAT-T, even when sampling 5000 solutions. For CVRP50, our

Table 5.7: CVRP Results on 10,000 instances reported in [128]. *Costs are estimated from the reported gaps and times are presented as reported in [102, 231]. ** Reported costs are averaged only on 2000 instances and not directly comparable. †Trained on CVRP50.

	CVRP20			CVRP50			CVRP100		
Method	Cost	Gap	Time	Cost	Gap	Time	Cost	Gap	Time
LKH3	6.14	0.00%	(2h)	10.38	0.00%	(8h)	15.65	0.00%	(12h)
GAT (greedy) [128]	6.40	4.30%	(1s)	10.98	5.86%	(1s)	16.80	7.34%	(3s)
GAT {1280} [128]	6.25	1.86%	(7m)	10.62	2.40%	(20m)	16.23	3.72%	(2h)
GAT-T {1000} [231]	6.19*	0.90%	(23m)	10.71*	3.16%	(48m)	16.30*	4.16%	(1h)
GAT-T {3000} [231]	6.17*	0.61%	(1h)	10.55*	1.65%	(2h)	16.11*	2.99%	(3h)
GAT-T {5000} [231]	6.16*	0.39%	(2h)	**10.45***	**0.70%**	(4h)	16.03*	2.47%	(5h)
NLNS [102]	6.19*	0.90%	(7m)	10.54*	1.54%	(24m)	15.99*	2.17%	(1h)
L2I [151]	6.12**	-	-	10.35**	-	-	15.57**	-	-
DPDP 10K [127]	-	-	-	-	-	-	15.83	1.18%	(2h)
DPDP 100K [127]	-	-	-	-	-	-	15.69	0.30%	(6h)
DPDP 1M [127]	-	-	-	-	-	-	**15.63**	-0.13%	(48h)
Ours {500}	6.22	1.32%	(10m)	10.92	5.29%	(1h)	17.58†	12.31%	(5h)
Ours {1000}	6.18	0.69%	(19m)	10.76	3.70%	(2h)	17.06†	8.80%	(10h)
Ours {2000}	**6.16**	**0.37%**	(39m)	10.65	2.66%	(4h)	16.72†	6.83%	(20h)

learned policy matches GAT (greedy) after sampling 500 solutions. However, GAT-T can achieve lower optimality gaps when sampling more solutions than both our proposed method and GAT. We report CVRP100 results for completeness, although we do not train on instances with 100 customer nodes. As expected, our evaluated policies are farther from the LKH3 baseline when compared to the other learned methods that train on CVRP100 instances, including DPDP 1M. However, the results show that the learned policies can generalise to instances of different sizes. An important aspect of our results in comparison to a constructive method is that we are required to check feasibility each time a solution is generated. This leads to high running times due to the polynomial growth in the feasibility checks as we increase the size of the instances. This issue can be alleviated by running multiple instance mini-batches in parallel, but it is not implemented in our evaluations.

5.9 Conclusions

In this chapter, we introduced a deep reinforcement learning formulation and algorithms to approximate improvement policies based on 2-opt local search operators for the travelling salesman problem (TSP), the multiple TSP, and the capacitated vehicle routing problem. We proposed a neural architecture with graph and sequence embedding capable of outperforming learned construction and improvement heuristics requiring fewer samples for the TSP. Our learned heuristics also

outperformed classical 2-opt and achieved similar performance to state-of-the-art classical heuristics as well as exact solvers in all problems studied.

Appendix

5.A Appendix A: Results on TSPLib

Table 5.8: Performance of OR-Tools vs our method on TSPlib instances.

Instance	Opt.	Ours {2,000}	OR-Tools
eil51	426	**427**	439
berlin52	7,542	7,974	**7,944**
st70	675	**680**	683
eil76	538	552	**548**
pr76	108,159	111,085	**110,948**
rat99	1,211	1,388	**1,284**
rd100	7,910	**7,944**	8,221
kroA100	21,282	23,751	**21,960**
kroB100	22,141	23,790	**22,945**
kroC100	20,749	22,672	**21,699**
kroD100	21,294	23,334	**22,439**
kroE100	22,068	23,253	**22,551**
eil101	629	**635**	650
lin105	14,379	16,156	**15,363**
pr107	44,303	54,378	**44,573**
pr124	59,030	59,516	**60,413**
bier127	118,282	**121,122**	121,729
ch130	6,110	**6,175**	6,329
pr136	96,772	**98,453**	102,813
pr144	58,537	61,207	**59,286**
ch150	6,528	**6,597**	6,733
kroA150	26,524	30,078	**27,503**
kroB150	26,130	28,169	**26,671**
pr152	73,682	**75,301**	75,832
u159	42,080	**42,716**	43,403
rat195	2,323	2,955	**2,375**
kroA200	29,368	32,522	**29,874**
ts225	126,643	**127,731**	127,763
tsp225	3,919	4,354	**4,117**
pr226	80,369	91,560	**83,113**
gil262	2,378	**2,490**	2,517
pr264	49,135	59,109	**51,495**
a280	2,579	2,898	**2,742**
pr299	48,191	59,422	**50,617**
pr439	107,217	143,590	**117,171**
Avg. Opt. Gap	0.00%	8.61%	3.70%

Chapter 6

Learning 2-opt Heuristics from Expert Demonstrations

This chapter builds upon Chapter 5 and aims at learning improvement policies for routing problems focusing on the travelling salesman problem. Unlike the previous chapter, we study the setting where a training agent has access to previous expert heuristics used to gather demonstration data. The agent's goal is to extract information from these expert heuristics to learn policies that can surpass the quality of the heuristics reducing the sample complexity and accelerating the emergence of good quality policies.

This chapter is based on [54].

6.1 Introduction

The travelling salesman problem (TSP) is a well-known combinatorial optimisation (CO) problem where the aim is to find an optimal tour that visits n locations once and returns to the origin. The TSP is known to be NP-hard [175], and solving it to optimality is usually achieved via integer linear programming and dynamic programming methods. However, solving large TSP instances to optimality can be impractical due to high computational costs. For this reason, several (meta)heuristics have been proposed to solve the problem. Heuristics for the TSP can be classified in constructive and improvement methods. In the first, the goal is to compose a solution by iteratively extending a partial tour. In the latter, a complete solution is improved by certain operators searching for better solutions, such as using k-opt edge swaps [90].

Recently, using machine learning to solve CO problems has gained much interest. For many problems, heuristics exist to make algorithmic decisions that otherwise would be too expensive to compute. This makes machine learning a viable option for making decisions in a more automated and optimised manner [24]. Thanks to the advances in deep learning, reinforcement learning (RL) methods have succeeded in learning effective constructive and improvement policies for the TSP [21, 49, 52, 62, 115, 128, 223, 231]. However, these methods require many steps of poor performance in simulation during training, partly due to their simple exploration rules, such as ϵ-greedy for value learning methods and noise-based exploration for policy learning methods.

Since most previous RL methods attempted to learn either improvement or constructive heuristics, it is natural to consider reusing expert information embedded in handcrafted heuristics to accelerate learning. In imitation learning (IL), the goal is precisely to reproduce the behaviour of an expert policy in a sequential decision-making problem [186]. However, classical IL can only learn policies as good as expert policies [186]. In the case of CO problems, these expert policies are usually not optimal. When a suboptimal expert is available, policies learned with standard IL can be inferior to policies learned via RL with approaches such as policy gradient [40].

Moreover, in the case of expert heuristics policies, it is desirable to learn without online access to the policies, given the high cost of computing expert heuristics rules. Thus, we focus on the case when a suboptimal improvement policy exists but can only be used to gather demonstrations, i.e., expert trajectories. Our objective is to incorporate the information from such demonstrations to learn faster than with pure RL and better than the expert policy for the TSP.

To achieve this goal, we propose to combine RL leveraging online interactions with a Euclidean TSP environment and a small number of demonstrations from 2-opt improvement heuristics. We combine a classical policy gradient objective with a previously trained policy via an offline supervised loss, leading the agent

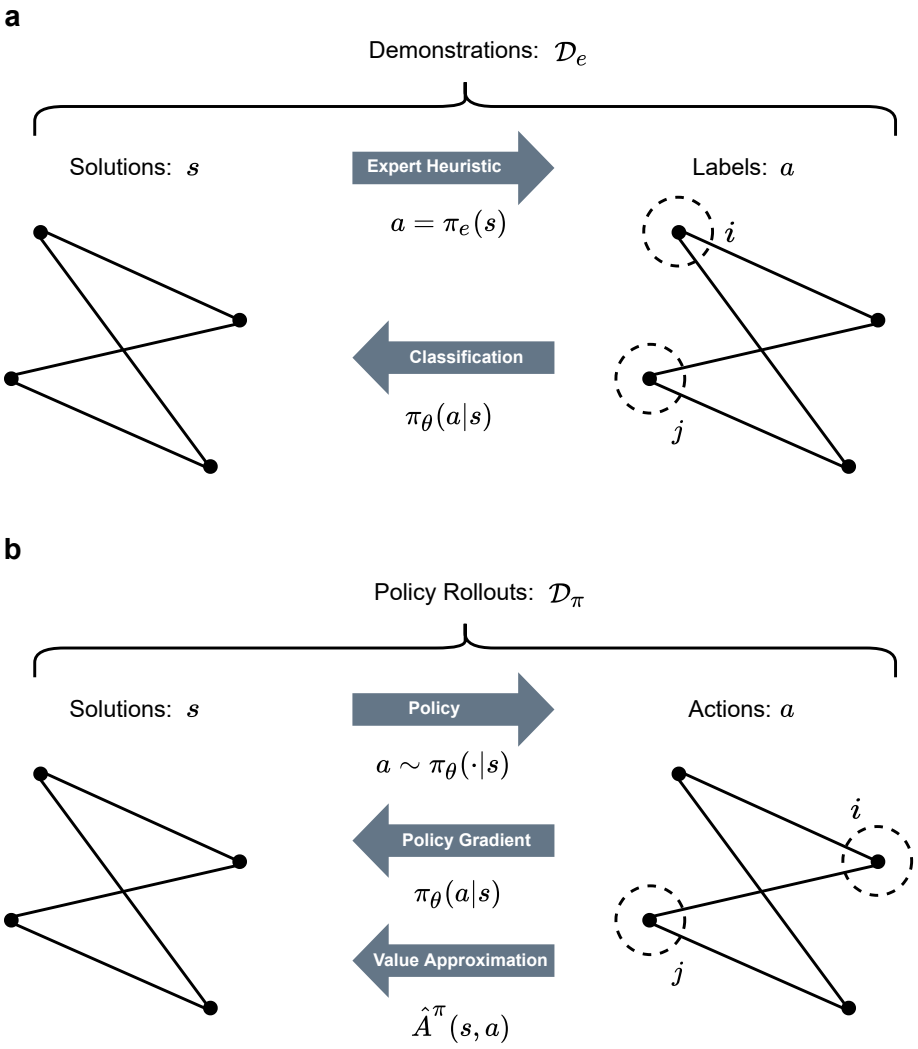


Figure 6.1: In the figures, we present a schematic representation of the proposed method. **a** A behaviour cloning policy is trained on data $\mathcal{D}_e$ made of trajectories from 2-opt local search heuristics acting as experts. **b** The trained policy in **a** is used to warm-start a reinforcement learning policy trained via policy gradient to maximise cost reductions of the next search steps.

to prefer actions experienced in the demonstrations. Then, during its online phase, the method performs updates considering its self-generated data. Our method, depicted in Figure 6.1, is straightforward and can be applied to other CO problems where good heuristics already exist. Our method can outperform IL methods such as behaviour cloning (BC) and pure policy gradient training in our experiments. Moreover, our method performs similar to a recently proposed RL method [49, 52] but requires only 10% of the number of iterations to obtain comparable policies to other effective deep RL methods.

Contributions and Organisation We summarise our main contributions as follows:

- We combine a classical policy gradient objective with a previously trained policy via behaviour cloning on expert demonstrations.
- Our method can learn good policies faster than with pure RL starting from suboptimal 2-opt heuristics as experts.
- We only make use of a small amount of demonstration data, retaining similar sample complexity to RL methods.

The remainder of this chapter is organised as follows. In Section 6.2 we present the related literature. Section 6.3 provides a formal description of the sequential decision-making model of the problem and its background. In Section 6.4 we formalise the proposed algorithm. Section 6.5 presents the experimental results and comparisons to previously proposed methods. Finally, Section 6.6 concludes this chapter and provides some interesting directions for future research.

6.2 Related Literature

The TSP is a challenging problem in CO with applications in many domains [138]. Past successes in solving hard instances have been accredited to heuristics or a combination of heuristics and exact methods. Such heuristics include local search methods such as Lin-Kernighan-Helsgaun (LKH) [91] and metaheuristics such as simulated annealing [217]. In common, these methods aim to reduce expensive computations by exploiting the structure of the problem combined with local and global neighbourhood search.

Recently, deep learning has emerged as a viable option to solve routing problems [24]. Many works have considered approximating a function that attempts to construct a solution or improve a given solution. These approaches resemble heuristics and, as such, can be classified in constructive and improvement methods. These methods have had considerable success, using either supervised or reinforcement learning.

Supervised Learning Supervised learning approaches for the TSP [115, 223] have considered the setting of given offline data containing optimal solutions as outputs. The goal is to learn a function that can reproduce optimal tours directly from node and distance data. Results show that supervised learning can be applied to this setup with fairly reasonable results. However, ensuring optimal labels for larger instances can be too computationally expensive. Note that these methods rely on a given optimal policy. Thus, they can also be seen as a type of Imitation Learning. However, we make a distinction to differentiate between learning from optimal and suboptimal strategies.

Reinforcement Learning Model-free RL breaks the assumption of given optimal solutions [21, 49, 52, 62, 128, 231]. Previous methods used deep function approximation to learn directly from interactions. Most methods have focused on the setting where a solution must be constructed sequentially, starting from a given location (node). In this case, training attempts to find policies that reduce the overall cost of a tour. In practice, to achieve good solutions, these methods have to sample multiple tours to find near-optimal ones [21, 62, 128].

Another approach considers learning over multiple steps, improving a given solution over a series of local operators [39]. For the TSP, these improvement-seeking methods have achieved good results when sampling a similar amount of operators as in construction methods [49, 52, 231]. Note that [49, 52] are this author's previous works detailed in Chapter 5. One major drawback of RL methods is their sample complexity, typically requiring many hours of training. Another aspect of previously proposed methods is that they learn from scratch, i.e., no previous policy is used to aid learning. As it is true that optimal solutions may not be available or expensive to compute, the same is not valid for heuristics. These are cheaper to compute than optimal solutions and can be used as a suboptimal expert to guide policy search. In this chapter, we focus our attention on recent improvement methods that learn over a class of 2-opt policies [49, 52, 231]. In this case, heuristics already exist and can potentially be used to accelerate learning.

Imitation Learning In imitation learning (IL), expert policies can be used as demonstrations of successful behaviour. A simple approach to IL is known as behaviour cloning (BC), which learns a policy through supervised learning on expert demonstrations [186]. Although BC has been used successfully in several instances, it suffers from problems such as distribution shift between expert and behaviour (training) policies [187]. Generative adversarial imitation learning (GAIL) [96] is a more recent approach that obtains performance gains over BC but requires an additional generative adversarial network (GAN) [81] for training. Other approaches in IL considered online access to the expert to surpass the expert policy [36, 40, 204].

However, even when an expert is available for online interactions, as is the case of heuristics, querying expert heuristics can become computationally prohibitive.

Thus, we consider a typical case in CO, i.e., when expert demonstrations exist, but online interaction with the expert is not available or expensive. In CO, IL has been previously applied to learn branching strategies and accelerate branch and bound in the context of integer programming, [77, 122, 157] and predict the objective improvement of cutting planes selection in semidefinite programming problems [15].

Reinforcement Learning from Demonstrations Methods combining RL with demonstrations have shown good results guiding policy learning in robotics and game playing. In [200], demonstration data was used to pre-train the policy network over expert player data. [95] proposed the deep Q-learning from demonstrations (DQfD) and stored the demonstrations in an experience replay buffer. [38, 119] proposed expanding policy gradient with additional GAIL regularisation terms. Like GAIL, these require additional networks for generating expert-like policies. Instead, we focus on a simple BC initialisation term that does not require additional neural networks and can be trained using a pre-existent policy network. Similar to our work, [183] use BC pre-trained policies and natural policy gradient (NPG) [118] with BC augmented loss to learn dexterous manipulation on robotics hands. Similarly, we propose to guide initial exploration via behaviour cloning and combine it with on-policy policy gradients. We note that the general idea of bootstrapping RL with BC while learning TSP heuristics is not yet explored.

6.3 Preliminaries

6.3.1 Traveling Salesman Problem

In the Euclidean TSP, given an input graph, represented as a sequence of n locations in a two-dimensional space $X = \{x_i\}_{i=1}^{n}$, where $x_i \in [0, 1]^2$, we are concerned with finding a permutation of the nodes, i.e. a solution $s = (\bar{s}_1, \ldots, \bar{s}_n)$, that visits each node once (except the starting node) and has the minimum total length (cost). We define the cost of a solution as

$$L(s) = \|x_{\bar{s}_n} - x_{\bar{s}_1}\|_2 + \sum_{i=1}^{n-1} \left\|x_{\bar{s}_i} - x_{\bar{s}_{i+1}}\right\|_2, \tag{6.1}$$

where $\|\cdot\|_2$ denotes the ℓ_2 norm.

6.3.2 First and Best Improvement 2-opt Heuristics

General improvement heuristics enhance feasible solutions through a search procedure. Local search methods start at an initial solution and proceed to replace previous solutions with better solutions. In the effective Lin-Kernighan-Helsgaun

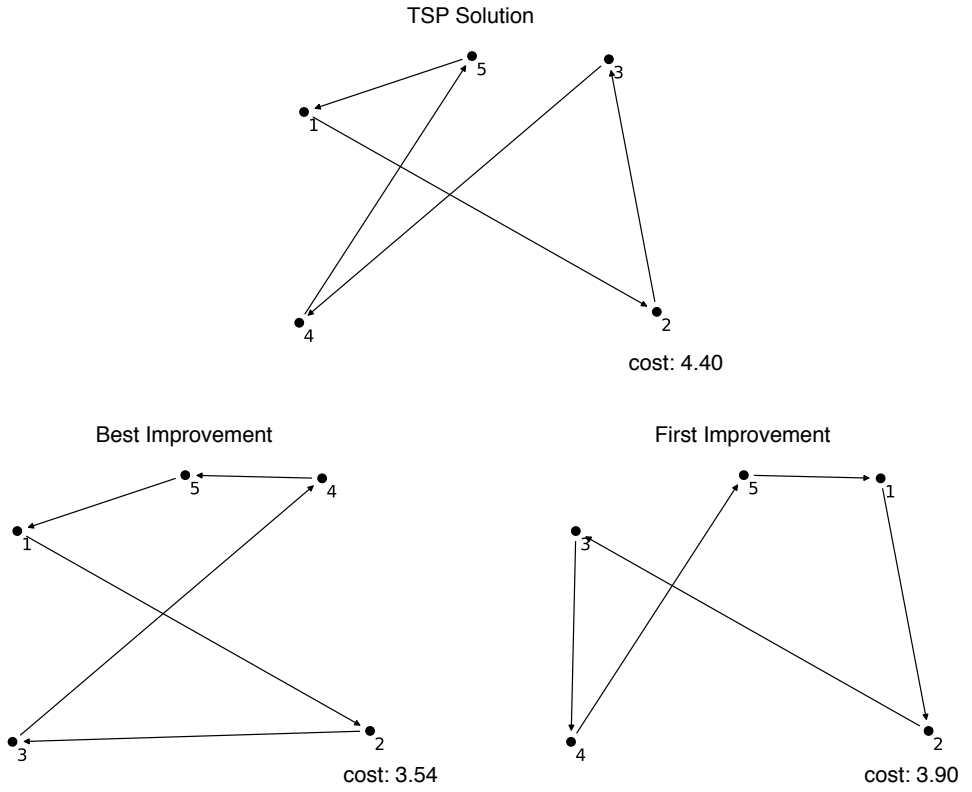


Figure 6.2: TSP solution before a 2-opt move (top), and after a best improvement move $(3, 4)$ (bottom left) and a first improvement move $(1, 3)$ (bottom right). Indices represent the order in which nodes are visited.

(LKH) [91] heuristic for the TSP, the procedure searches for k edge swaps (k-opt moves) that will be replaced by new edges resulting in a shorter tour. A simpler version [143] considers 2-opt (Figure 6.2) and 3-opt moves alternatives as these balance solution quality and the $O(n^k)$ complexity of the moves.

At each iteration of a 2-opt local search, two edges are selected to be deleted and replaced by two edges that result in a better tour. In doing so, 2-opt moves can be expressed by selecting two index positions $(\bar{a}_1, \bar{a}_2) = (i, j)$ of a tour s, i.e., $\bar{a}_1, \bar{a}_2 \in \{1, \ldots, n\}, \bar{a}_1 < \bar{a}_2$, breaking edges between nodes at positions $(i-1, i)$ and $(j, j+1)$, inverting the tour between i and j and adding edges between $(i-1, j)$ and $(i, j+1)$. The selection of the edges is normally done by scanning the current solution for suitable edge pairs.

Since there can be many such pairs, a decision must be made over which move is considered first. Two well-studied choices are selecting the first set of edges at

which an improvement is possible, i.e., first improvement (FI) and the best reducing cost move, i.e., best improvement (BI). On average, selecting BI over FI gives worse results if the initial solution is chosen at random. However, when initialised with a greedy constructive heuristic, BI is better and faster on average [84].

6.3.3 Markov Decision Process

We adopt a standard Markov decision process (MDP) $\mathcal{M}$, defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s'|s,a)$ is the transition distribution after taking action a at state s, $r(s,a)$ is the reward function, $\gamma \in (0,1)$ is a discount factor, detailed below.

States A state s is a solution to the TSP defined by a permutation of n nodes. Note that we only consider a current solution as a state as opposed to both the current and best solution as in [49, 52], and allows us to consider experts that only operate in s when picking the next action.

Actions An action a is a tuple (i,j), where $i,j \in \{1,\dots,n\}$, $i < j$ corresponding to two indices of s.

Transitions Transitions are deterministic and defined by (s,a) pairs. Given action (i,j), transitioning from $s = (s_1,\dots,s_i,\dots,s_j,\dots,s_n)$ is defined by a 2-opt exchange inverting the tour segment between i and j resulting in a next state $s' = (s_1,\dots,s_j,\dots,s_i,\dots,s_n)$.

Objective Given a stochastic policy $\pi(a|s)$[4] : $\mathcal{S} \to \mathcal{P}(\mathcal{A})$ which maps states to action probabilities, its performance is evaluated by the expected discounted sum of rewards (return):

$$J(\pi) = \mathbb{E}_\pi[r(s,a)] = \mathbb{E}_{s_0,a_0,\dots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t)\right], \tag{6.2}$$

where $s_0, a_0, \dots$ is a trajectory induced by policy π, i.e. $s_0 \sim p_0(s_0)$, with $p_0(\cdot)$ being the distribution over initial states, $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim \mathcal{P}(\cdot|s_t,a_t)$. We define standard characterisations of the value function

$$V^\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k},a_{t+k})|s_t = s], \tag{6.3}$$

[4]We refer to deterministic policies as $\pi(s)$.

action value function

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k}, a_{t+k}) | s_t = s, a_t = a] \tag{6.4}$$

and the advantage function as $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$, reflecting the expected additional return of a at state s.

Rewards Rewards are attributed to actions that can improve upon the cost of the best-found state s_t^* in a trajectory, i.e.,

$$s_t^* = \underset{s_{t'} \in \{s_0, \ldots, s_t\}}{\operatorname{argmin}} L(s_{t'}) \tag{6.5}$$

and

$$r(s_t, a_t) = L(s_t^*) - \min\{L(s_t^*), L(s_{t+1})\}. \tag{6.6}$$

In RL, we are interested in finding a policy that maximises $J(\pi)$ by using a set of trajectories $\mathcal{D} = \{\tau_i\}$, where $\tau_i = \{(s_0^i, a_0^i), (s_1^i, a_1^i), \ldots\}$ are sampled from the current policy in on-policy methods or different policies in off-policy methods. We consider policies π_θ with parameters θ; thus, we use π, π_θ and $J(\pi)$, $J(\theta)$ interchangeably.

6.4 Learning 2-opt from Demonstrations

In this chapter, we use a combination of RL and IL to learn 2-opt exchange heuristics. To reduce sample complexity and help exploration, we collect expert demonstrations from FI and BI heuristics, extract a policy from demonstrations and do policy gradient updates over environment interactions. In the following sections, we detail each component of the proposed method.

6.4.1 Policy Gradient

We are mainly concerned with policy gradient methods, which are a class of on-policy model-free RL. In policy gradient, the parameters of the policy are directly optimised towards maximising the main objective defined as

$$J(\theta) = \mathbb{E}_{s \sim d^{\pi}}[\sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^{\pi}(s,a)], \tag{6.7}$$

where $d^{\pi}(s)$ is the stationary distribution of the Markov chain induced by $\pi_\theta(a|s)$. Then, according to the policy gradient theorem [208], the gradient of $J(\theta)$ can be

estimated as:

$$\nabla_\theta J(\theta) \propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) \tag{6.8}$$

where θ can be optimised via gradient ascent. In practice, to reduce variance, $Q^\pi(s,a)$ is replaced by the advantage function $A^\pi(s,a)$ [207]. In this work, we consider the REINFORCE algorithm [229]; thus, during training, we collect on-policy trajectories $\mathcal{D}^\pi$ and replace expectations by empirical samples to compute gradient estimates as:

$$g \propto \sum_{\tau_i \in \mathcal{D}^\pi} \sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) \hat{A}^\pi(s_t^i, a_t^i) \tag{6.9}$$

where $\hat{A}^\pi(s_t, a_t) = G_t - V_\phi^\pi(s_t)$ is an estimate of the advantage function, and

$$G_t = r(s_t, a_t) + \sum_{k=1}^{T} \gamma^k r(s_{t+k}, a_{t+k}) \tag{6.10}$$

are Monte Carlo estimates of returns up until T and $V_\phi^\pi(s_t)$ is an approximation of $V^\pi(s_t)$ with parameters ϕ trained over mean squared errors between G_t and $V_\phi^\pi(s)$.

6.4.2 Learning from Demonstrations

Directly optimising policy gradients with rewards defined in $\mathcal{M}$ can lead to good policies that surpass simple greedy 2-opt heuristics [49, 52, 231]. However, doing so requires a large number of samples and many hours of training. Demonstrations can help alleviate this issue and help to guide exploration to promising reward regions. In this work, we consider demonstrations from deterministic 2-opt policies FI and BI that select actions from states as $a = \pi^e(s)$, where π^e is the expert's policy.

Note that the objective in $\mathcal{M}$ recovers the total improvement over an initial state, albeit the discount factor. Thus, having expert demonstrations from greedy heuristics can help to find regions with more significant improvement. Moreover, the BI heuristic is the optimal policy of a one-step $\mathcal{M}$. Therefore, quickly extracting information from this policy can potentially help to guide policies over larger horizons.

Behaviour Cloning

Exploration in policy gradient methods is done by implicitly using the stochasticity of policies or by explicitly introducing an entropy term to the objective. If the initial policy is poor, learning can be slow, as the algorithm explores states that lead to poor rewards. An effective way to combat this issue is to use expert policies and attempt to mimic their behaviour. Behaviour cloning (BC) is a simple IL method that attempts to learn good policies over expert trajectories $\mathcal{D}^e$ and does not require additional interactions with the expert. The main objective in BC is to train a policy

π_θ on a supervised signal from the distribution of states $d^{\pi^e}(s)$ induced by the expert policy π^e defined as $C(\theta) = \mathbb{E}_{s\sim d^{\pi^e}}[\ell(\pi_\theta(\pi^e(s)|s), \pi^e(s)]$, where $\ell(\cdot)$ is a suitable performance loss. In our case, we approximate the objective above over state-action pairs in $\mathcal{D}^e$ by the log-likelihood loss as

$$C(\theta) \propto \sum_{\tau_i \in \mathcal{D}^e} \sum_t \log \pi_\theta(a_t^i|s_t^i). \tag{6.11}$$

Taking the gradient of this objective resembles the one in (6.9), but here we average over expert and not on-policy trajectories and do not have access to an advantage function. In fact, (6.11) can perform well when states observed by expert policies are similar to states observed by learned policies [186]. The BC policy is then the one that finds parameters θ such that $C(\theta)$ is maximised, and the distribution of actions given a state approaches those on demonstration data. In our study, each trajectory demonstration τ_i^e relates to a TSP instance, and cloning a policy corresponds to cloning different actions in different instances. Since states visited by the expert policy are diverse due to the many instances seen during training, we expect BC to perform reasonably well in the given setting, assuming we can replicate the expert's policy. Moreover, even if learning fails, the BC policy may still perform well by encountering good alternative decisions with a low probability under expert trajectories. Such alternative decisions can result in high rewards under the performance of (6.2) [24], which is ultimately what we care about in the optimisation problem. Nonetheless, in general, we do not expect BC to outperform the expert systematically. Thus, we still need better exploration strategies to discover new policies that can perform better than the expert.

Algorithm 2: Policy Gradient with Behaviour Cloning

Input: Expert demonstrations $\mathcal{D}^e$; parameters θ, ϕ, weight β; maximum steps $\mathbb{T} \geq T$, number of epochs m and number of iterations L,K.

for $l = 1, 2, \ldots L$ **do**
  Update θ with demonstrations for m epochs by:
  $\sum_{(s,a)\in\mathcal{D}^e} \nabla_\theta \log \pi_\theta(a|s)$;
end
for $k = 1, 2, \ldots K$ **do**
  Sample $\mathcal{D}^\pi = \{\tau_i\}_{i=1}^N, \tau_i \sim \pi_\theta$;
  Update θ and ϕ every T steps by:
  $\sum_{(s,a)\in\mathcal{D}^\pi} \nabla_\theta \log \pi_\theta(a|s)\hat{A}^\pi(s,a) + \beta\nabla_\theta H(\theta)$;
  $\frac{1}{2}\nabla_\phi \sum_{(s,a)\in\mathcal{D}^\pi} \|G_t - V_\phi^\pi(s)\|_2^2$;
end

Guiding Exploration with Behaviour Cloning

As demonstrated in [183], we can use demonstration data to both provide good initialisation for RL or use them to guide exploration during RL. In our case, as the number of iterations grow the policy is given a chance to let go of demonstrations and perform policy updates only on on-policy samples aided by an entropy term $H(\theta) \propto \beta \sum_{s \in \mathcal{D}^\pi} \mathbb{E}_{a \sim \pi_\theta}[\log \pi_\theta(\cdot|s)]$, where $\beta \in [0, 1)$. The pseudocode of the procedure is presented in Algorithm 2.

6.4.3 Policy and Value Networks

We adopt the effective neural network architecture reported in [49, 52]. In the paper, an encoder-decoder architecture is proposed to output estimates of both $\pi_\theta(a|s)$ and $V^\pi_\phi(s)$ for a given state s[5]. However, in the original paper, a state is composed of the tuple (s_t, s^*_t), whereas we only consider a solution s_t. Although this modification makes convergence slower, this allows us to have comparable state distributions between expert and online policies since $s_t = s^*_t$ for both FI and BI. In the architecture, depicted in Figure 6.3, the shared encoder is composed of three

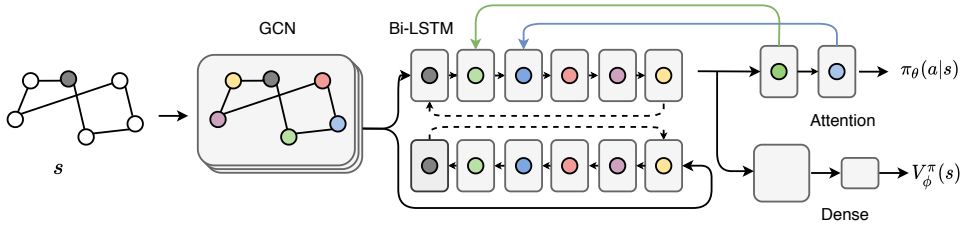


Figure 6.3: A state (solution) s is fed to an encoder where graph and sequence information are extracted. A policy decoder takes encoded inputs to output actions sequentially. A value decoder operates outputs state values estimates.

graph convolutional [125] layers to extract node information and a bi-directional long short-term memory network (LSTM) [97] layer responsible for encoding tour sequence information. The policy decoder then uses both graph and sequence information to generate actions sequentially via a pointing attention mechanism [223]. That is, it generates 2-opt indices sampling over softmax operators $p_\theta(\cdot)$ and uses the chain rule to factorise $\pi_\theta(a|s)$ as

$$\pi_\theta(a|s) = p_\theta(\bar{a}_1|s) p_\theta(\bar{a}_2|\bar{a}_1, s), \tag{6.12}$$

where, $a = (\bar{a}_1, \bar{a}_2)$, $\bar{a}_i \in \{1, \ldots, n\}$. The value decoder employs two dense layers that take combined graph and sequence representations to output value estima-

[5] A portion of parameters θ and ϕ are shared in the encoding layers.

tions. We refer the reader to [49, 52] for details on the original implementation for the TSP.

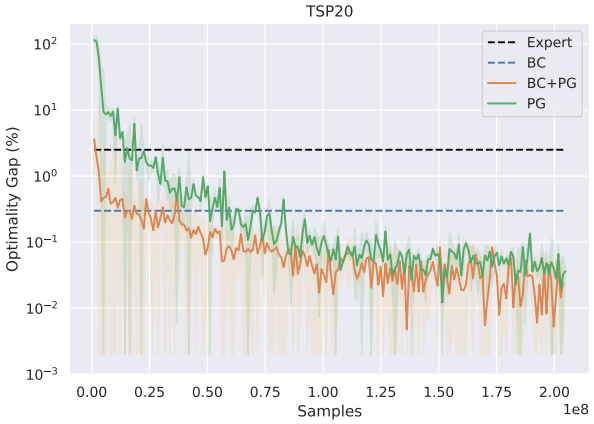
6.5 Experiments

This section aims to investigate whether initialising policy gradient with a BC policy can help with learning 2-opt heuristics faster than a method that uses no previous demonstration data. We conduct extensive experiments to investigate the performance of warm starting policy gradient with BC (BC+PG) to a policy gradient (PG) policy. We consider three tasks in our experiments, Euclidean TSP instances with 20 (TSP20), 50 (TSP50), and 100 nodes (TSP100). For all tasks, node coordinates are drawn uniformly at random in the unit square $[0, 1]^2$. We measure the performance of policies on the optimality gap between the best-found solution, and the optimal solution computed using Concorde [5]. We compare our results on the test dataset as reported in [128] containing 10,000 instances for each task. To benchmark our method, we compare learned policies to other highly specialised RL methods for the TSP. Here, the objective is to achieve near-optimal solutions and compare how achieving a certain performance level depends on the number of samples.

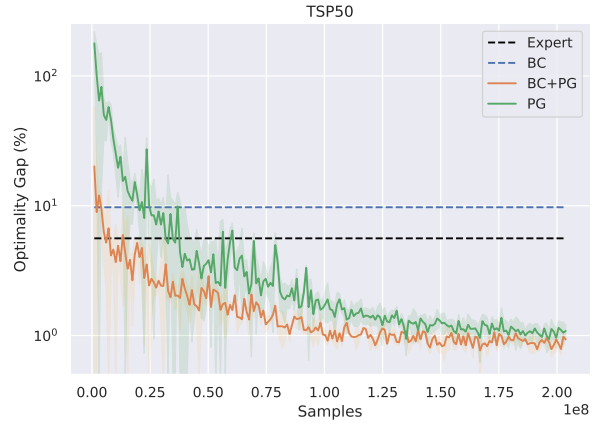
6.5.1 Experimental Settings

In our experiments, a maximum of 100,000 samples of demonstration data is gathered using either FI or BI on 5,120 TSP instances over trajectories of a maximum of 400 steps. When this procedure generated more than 100,000 samples, we undersampled the expert samples uniformly at random. Each experiment block is repeated for FI and BI demonstrations separately. During RL, 5,120 instances of the TSP were generated on the fly and simulated for $\mathbb{T} = 200$ steps. Every T steps policy and value updates were performed, where returns are computed over a maximum of $T = 8$ steps in the future (truncation). In all tasks, we pass twice over the expert demonstrations i.e. $m = 2$ for each iteration $l \in \{1, \ldots, 200\}$ and use a batch size of 512 during BC, and $k \in \{1, \ldots, 200\}$ iterations over a batch size of $512 \times T$ samples during RL. During BC, a teacher forcing [230] ratio of 25% is used to accelerate learning. A fixed validation dataset of 128 instances with their respective optimal solutions was used for manual hyperparameter optimisation rolling out policies for 200 steps.

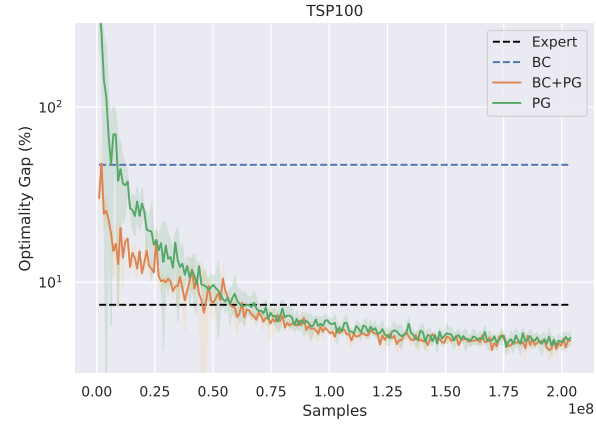
Following the implementation in [49, 52], we employed the same $\gamma = 0.99$, ℓ_2 penalty of 1×10^{-5} and learning rate $\lambda = 0.001$, λ decaying by 0.98 at each k. Loss weights $\beta = 0.0045$ decay by 0.9 after every iteration, and parameter updates were performed via Adam [124]. The remaining neural network hyperparameters were unchanged from the original implementation. We train on an RTX 2080Ti GPU and Ryzen 3950X CPU hardware using PyTorch 1.6. Each iteration takes an average time



(a) TSP instances with 20 nodes.

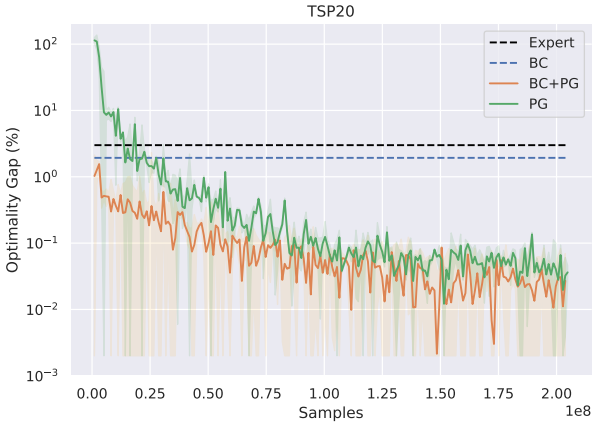


(b) TSP instances with 50 nodes.

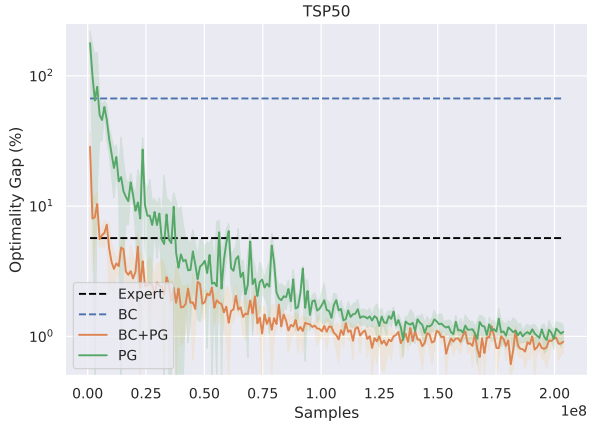


(c) TSP instances with 100 nodes.

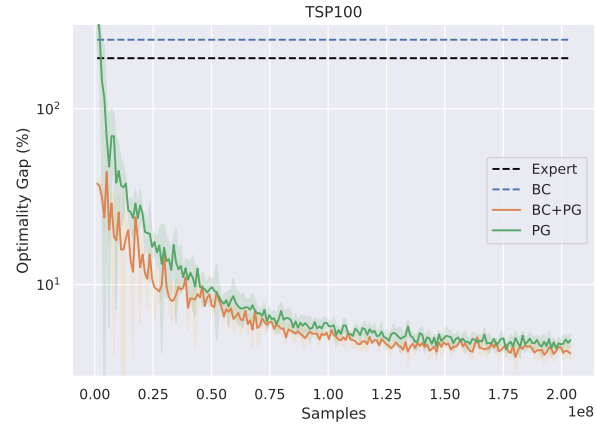
Figure 6.4: Optimality gaps when rolling out policies for 200 steps on 128 instances using best improvement (BI) demonstrations. **Policies**: Expert, behaviour cloning (BC), behaviour cloning followed by policy gradient (BC+PG), and policy gradient (PG).



(a) TSP instances with 20 nodes.



(b) TSP instances with 50 nodes.



(c) TSP instances with 100 nodes.

Figure 6.5: Optimality gaps when rolling out policies for 200 steps on 128 instances using first improvement (FI) demonstrations. **Policies**: Expert, behaviour cloning (BC), behaviour cloning followed by policy gradient (BC+PG), and policy gradient (PG).

of 1m 52s, 2m 23s, and 3m 58s for TSP20, TSP50, and TSP100 during the RL phase. We roll out our policy for 1,000 steps during testing to allow for a fair comparison with previous literature. Our implementation will be made available online.

6.5.2 Experimental Results and Discussion

We present the optimality gaps of the tested policies using BI and FI demonstrations in the log scale in Figures 6.4 and 6.5. In Figure 6.4, we observe that for all tested instances, warm starting PG with a cloned policy using BC yields better solutions (lower optimality gaps). The difference is between the two is more prominent at early iterations due to PG having to sample more exploratory actions. Aided by BC, BC+PG can, in fewer iterations, surpass the performance of the BC policy and the expert. For example, in TSP20 instances, BC+PG can surpass the expert in one iteration, whereas PG requires 10 iterations. To reach a 2% optimality gap in TSP50, PG requires 80 iterations, whereas BC+PG requires 40 iterations, i.e., around 0.42×10^8 samples. Results are similar for TSP 100, where we observe that it takes only 5 iterations (0.06×10^8 samples) for BC+PG to reach a 20% gap, whereas PG requires 20 iterations.

Table 6.1: Performance w.r.t. Concorde. *S*: Number of samples. *Type*: Solver, **SL**: Supervised learning, **BC**: behaviour cloning, **RL**: reinforcement learning.

Method	Type	TSP20			TSP50			TSP100		
		Cost	Gap	S ($\times 10^8$)	Cost	Gap	S ($\times 10^8$)	Cost	Gap	S ($\times 10^8$)
Concorde	Solver	3.84	0.00%	-	5.70	0.00%	-	7.76	0.00%	-
GCN	SL	3.84	0.01%	0.15	**5.70**	**0.01**%	0.15	7.87	1.39%	0.15
GAT-C	RL	3.84	0.09%	**0.05**	5.75	1.00%	**0.05**	8.12	4.64%	**0.05**
AM-C	RL	3.84	0.08%	1.28	5.73	0.52%	1.28	7.94	2.26%	1.28
GAT-I	RL	3.84	0.03%	4.09	5.75	0.83%	4.09	8.01	3.24%	4.09
GCN-I	RL	**3.84**	**0.00**%	2.05	5.71	0.21%	3.07	**7.86**	**1.26**%	3.07
PG@1	RL	7.62	98.72%	0.01	14.08	147.30%	0.01	33.66	333.50%	0.01
PG@5	RL	4.04	5.26%	0.05	7.59	33.30%	0.05	11.06	42.50%	0.05
PG@20	RL	3.85	0.21%	0.21	5.94	4.28%	0.21	8.56	10.29%	0.21
PG@200	RL	3.84	0.01%	2.05	5.71	0.30%	2.05	7.89	1.61%	2.05
BC+PG@1	BC, RL	3.88	1.17%	0.01	6.28	10.29%	0.01	8.86	14.15%	0.01
BC+PG@5	BC, RL	3.84	0.07%	0.05	5.84	2.61%	0.05	8.47	9.02%	0.05
BC+PG@20	BC, RL	3.84	0.02%	0.21	5.74	0.86%	0.21	8.07	3.98%	0.21
BC+PG@200	BC, RL	**3.84**	**0.00**%	2.05	5.71	0.21%	2.05	7.87	1.41%	2.05

In our MDP, increasing the size of instances also corresponds to increasing the action space size by order $O(n^2)$, thus requiring further exploration of good actions before improvements can be made. We note that the performance of BC decreases with the instance size. BC can surpass the expert policy for TSP20 instances due to sampling more actions during validation. For larger instances, BC policies cannot reach the same performance as the expert, although they still aid PG to achieve better performance. Note that BC also needs to be trained via supervised learning

but requires few iterations to converge, and it is much faster to run than PG. On average, a BC policy trained over BI trajectories converged after 0.13×10^8 samples from the 100,000 expert state-action pairs. In terms of running time, learning via BC on TSP100 instances took 28s per iteration, i.e., 12% of the time of the PG counterpart. A similar reduction in time is seen for TSP20 and TSP50. BC convergence is achieved between 50 and 100 iterations, thus requiring 7% of the time to run PG.

In Figure 6.5, we note that as the size of instances increases, the performance of BC decreases considerably. This result is expected as actions become harder to predict given the size of the instance and show that learning directly from FI demonstrations is more challenging than from BI. We argue that this difference comes from BI's more diverse and clear action selection, where actions are selected based on the best improvement move, yielding diverse but consistent action indices each time. FI, on the other hand, selects many moves that swap the first node in the tour, resulting in many actions of the form $a = (1, j)$, where $j > 1$, i.e., at each iteration it starts scanning a solution from the beginning of a tour returning the first improvement encountered. While this is clear from a heuristic perspective, our architecture cannot appropriately learn this action selection rule for large instances. Nevertheless, learning a BC policy from FI demonstrations can still help to converge to better policies than with vanilla PG.

6.5.3 Comparison to Exact and Previous Learning Methods

We report results on the same 10,000 instances for each TSP size as in [128] and optimal costs obtained by Concorde [5]. We include recent state-of-the-art deep learning methods, including supervised learning [115], construction [62, 128] and improvement [49, 52, 231] reinforcement learning methods. We present the best-reported results using sampling for constructive and supervised learning methods, as these yield the lowest costs. We report results after rollouts of 1000 steps for improvement methods, as these are similar to sampling in constructive methods. We note that the test data used in [49, 52, 115, 128] is the same and, therefore, results are directly comparable. In [231], the generation process and size are identical, which decreases the variance of the results.

Since we are mainly interested in obtaining good policies quicker than using only RL, we show the performance of trained policies at different iterations. We refer to our tested policies as PG@k and BC+PG@k, where k is the iteration number to return the best-observed policy on validation instances. We report BC+PG results from BI demonstrations as these resulted in better performance. We refer to the results [115] as GCN in [128], as AM-C, in [62] as GAT-C, in [231] as GAT-I, and in [49, 52] as GCN-I. We point out that our PG uses a simplified version of GCN-I that runs for fewer iterations for TSP50 and TSP100 and leverages just the current solution for action selection. We select GCN-I as a baseline for complexity as it is more sample efficient than [231]. Comparison results, including cost, optimality

gaps and sample complexity[6] are summarised in Table 6.1.

Solution Quality We observe in the results that BC+PG generates better policies than PG over the entire training procedure, i.e., at $k = 200$. For example, in TSP50, BC+PG@200 results in a 0.21% optimality gap, whereas PG@200 results in 0.30%. Moreover, the difference between the two policies is more significant in the early stages of training. This is expected as after BC is over, updates to the policy follow the same gradient updates. When we compare PG with GCN-I, we see the effect of simplifying the neural network, reducing the number of samples, and fixating the planning horizon. The simplification results in larger gaps than the original model. BC+PG brings the optimality gaps much closer to GCN-I, matching it for TSP20 and TSP50. In other words, starting from a cloned policy helps to achieve similar performance as a more expressive model trained over more samples.

Early Policies When comparing early policies, i.e., $k \in \{1, 5, 20\}$, we note the benefit of biasing with BC pretraining at the early stages of RL training. Those policies can achieve much lower gaps than PG at the same k, e.g., BC+PG@1 achieves a 1.17% gap, whereas the same iteration in PG has a 98.72% gap. Compared to previous methods, we expect supervised learning and construction methods to have lower sample complexity than improvement methods, e.g., GCN, GAT-C, and AM-C. Interestingly, early policies learned at PG+BC@20 are competitive with previous RL methods that sample a much larger number of samples. Even if we add the number of samples to convergence during BC ($\sim 0.13 \times 10^8$) to 0.21×10^8, we observe that these policies are not far from results obtained with AM-C and GAT-I sampling 1.28 and 4.09×10^8 data points, respectively. For example, in TSP50, BC+PG@20 achieves a 0.86% gap requiring, whereas GAT-I achieves a 0.83% in the same task.

6.6 Conclusions

In this chapter, we studied learning 2-opt local search heuristics for the Traveling Salesman Problem (TSP) given greedy heuristics as expert demonstration data. We use the demonstration data to provide good initialisation to a reinforcement learning algorithm based on policy gradient. We propose to initialise policy gradient from imitating expert policies using behaviour cloning. Our results show that demonstrations are beneficial at the beginning of RL training leading to good policies quicker than with vanilla policy gradient. Moreover, when training is initialised with behaviour cloning, final policies at the end of learning generate TSP tours with lower optimality gaps. We show that policies learned with few policy

[6] The RL portion of BC+PG.

gradient iterations are competitive with previous deep learning models proposed and can be obtained with a much lower sample complexity than previous models.

Chapter 7

Learning Policies for the Dynamic Travelling Maintainer Problem with Alerts

In this chapter, we study a decision-making problem that considers maintenance and travel decisions, in which the failure times of assets are unknown and stochastic. Moreover, we assume that assets are equipped with real-time monitoring devices and condition-based maintenance prognostics models used to emit alerts, representing the first signs of degradation of the assets. Our goal is to plan maintenance considering the uncertain failure time information received via alerts, asset locations and maintenance costs. We present a modelling framework for the studied problem framework and propose learning policies to minimise the discounted maintenance costs accrued over an infinite time horizon.

This chapter is based on [53], currently under review.

7.1 Introduction

Ensuring availability and reducing operational costs are crucial for industrial assets. Assets such as wind turbines, trains, hospital scanners and aeroplanes are expected to experience minimal downtime. In turn, this expectation can drastically increase the operational and maintenance costs due to frequent maintenance or unexpected downtime. Ideally, assets are maintained just before failure to ensure the highest availability at minimum costs. To this end, two challenges arise: (i) the failure mechanism of an asset can be unknown *a priori* and (ii) assets are often part of a larger network of similar assets. In these cases, it is common that the resources to maintain assets are shared and controlled by a central operator responsible for assets in multiple geographical locations.

To keep assets functioning as intended, one can follow different maintenance policies. Two popular options to choose from are: (i) assets (or components) are maintained at scheduled times, i.e., time-based maintenance (TBM) planning or (ii) assets are regularly inspected to evaluate their degradation, i.e., condition-based maintenance (CBM). With the rise of intelligent monitoring devices such as sensors and predictive models, CBM has become the norm for critical assets. For such assets, predictive models can use real-time sensor readings as inputs to predict future failure times. Finally, this prediction can be communicated to the central operation in the form of *alerts*, which serve as an early indication of failures. In a network of assets, this would result in a list of alerts and predicted failure moments to be reacted by a repair crew. In practice, alerts often carry uncertain information about the failure time due to misreadings, sensor malfunction and prediction errors. Thus, devising effective policies entails considering the uncertainty of alerts in addition to maintenance and travel costs.

An example of asset networks that regularly require maintenance is offshore wind turbine parks. Such parks can contain more than 100 turbines and are often built in a grid-like structure, where pairwise distances between turbines are nearly identical [30]. To maintain wind turbines, field service engineers circulate on a vessel containing equipment for repairs. In this case, the use of remote monitoring devices and failure predictions can represent a significant improvement over TBM policies. For example, a vibration sensor connected to the gearbox of the turbines can be used to detect early degradation as soon as the vibrations pass a certain threshold [120]. Similarly, manufacturers of medical scanners have to perform maintenance on their assets in a network of hospitals. Medical scanners are equipped with an array of sensors that track medical procedures, but this data can also be used to signal degradation. This information can be used to optimise the dispatching of a repair crew in the network.

We introduce a decision-making problem named dynamic travelling maintainer problem with alerts (DTMPA). In this problem, we consider the challenges that arise from the previously mentioned scenarios. Namely, (i) assets are part of an

asset network in which the repairperson has to travel; (ii) on-time maintenance and travelling activities must be performed to ensure maximum availability of assets; (iii) information about future degradation is observed via alerts; (iv) degradation of the assets is stochastic (dynamic) and possibly unknown (model-agnostic). The main objective is to overcome the aforementioned challenges whilst minimising the maintenance costs over a long-term horizon.

To model the DTMPA, we propose a framework employing degradation processes that generate alerts as early indicators of failures. We introduce *information levels* to indicate the amount of information about failure times received by the decision-maker. Our discrete-time model aims to optimise maintenance costs in an asset network served by a single decision-maker responsible for maintenance and travel decisions. Therefore, we combine three major aspects of the DTMPA that arise from the asset network, (i) the existence of an underlying degradation process that governs the failure distribution of the assets, (ii) the capacity limitations that arise from a decision-maker serving a network and (iii) the different cost structures that exist when considering early and late maintenance actions. Following, we detail each component of the proposed framework.

Discrete Time We model the decision epochs in discrete time steps, which represent the decision moments of the application at hand. For example, it can represent an hourly period for wind parks or a weekly decision for hospital equipment. Note that the discrete-time setting also allows us to reduce the state space of the problem by considering states as decision epochs.

Network Layout We consider assets located in a network. Maintenance activities are performed by a single repairperson who can be dispatched around the network. This is motivated by practical applications, where large networks are divided into smaller networks, and each is assigned to one repairperson. Assets are distributed over the network, and travelling between assets takes time. In our formulation, travel times affect maintenance costs as high response times to failures are penalised.

Asset Degradation Every asset can be in several states, denoting the actual health of the asset. We model the transitioning between these states as a stochastic process. In our framework, each asset degradation model may be different. The repairperson is unable to observe the transition to some states and can only take actions based on alerts and their contents.

Observability We assume three states are observable: a *healthy* state, an *alert* state and a *failed* state. However, a machine may have more than these three observable states, denoted as *hidden* states, representing different levels of degradation. When

the degradation model passes through a predefined alert state, this triggers an alert notification. These alerts serve as an early indication of failures for the repairperson.

Alert Information Alerts carry information about the asset's degradation state and may contain additional residual lifetime information. We define four information levels capturing different levels of information available to the repairperson.

Capacity We define stylised networks based on real-world scenarios where larger networks are subdivided into smaller networks served by a single repairperson.

Cost Structure We model three types of costs, preventive maintenance costs, corrective maintenance costs and downtime costs. In our framework, downtime costs represent the cost per time period that an asset is unavailable either due to failure or maintenance activities. We assume that corrective maintenance is more expensive compared to preventive maintenance due to unplanned spare parts demands. The costs for the repairperson to travel between assets are assumed negligible.

Objective Our objective is to devise policies that minimise the expected discounted maintenance costs over an infinite time horizon. Here, we assume that a network of machines runs indefinitely. This is common in industries where machines have to run continuously to ensure uninterrupted operations.

To solve the proposed problem, we propose an algorithm using distributional reinforcement learning [18] trained in a simulated environment optimising the cumulative discounted maintenance costs. We compare this solution with two other methods utilising different information levels, i.e., a class of greedy/reactive heuristics that rank actions based on maintenance costs, travel times and estimated failure times, and a heuristic that leverages the alert information to construct a deterministic approximation of the dynamic problem similar to [1].

Contributions and Organisation The main contributions in this chapter are detailed as follows:

- We introduce the DTMPA on an asset network served by a single repairperson.
- We develop a flexible modeling framework for the DTMPA for maintenance optimisation, which allows for multiple degradation processes.
- Different information levels model the amount and precision of information retrieved from alerts.
- The heuristics and learned policies yield competitive policies for their respective information level compared to the optimal policy under complete information about the degradation process.

- Our results show that the learned policies, i.e., obtained via learning from interaction with the environment, outperform other proposed heuristics, requiring only the minimum level of information from the alerts.

The remainder of this chapter is organised as follows. Section 7.2 introduces the current literature in maintenance optimisation, travelling maintainer problems and reinforcement learning for maintenance decision-making. In section 7.3, we present a detailed description of the framework that models our decision-making problem. Section 7.4 introduces the different solutions methods for the proposed problem. In section 7.5, we discuss and support the numerical parameters used in our evaluations. In section 7.6, we present the numerical results of the proposed solutions and compare them to optimal policies with access to the underlying degradation model. Section 7.7 summarises the managerial insights and section 7.8 concludes and summarises this work.

7.2 Related Literature

There are three streams of literature relevant to this work: single asset maintenance optimisation, travelling maintainer problems, and (deep) reinforcement learning for maintenance optimisation problems. We discuss these three streams in the following sections.

Maintenance Optimisation Maintenance optimisation solutions are most distinguishable between single-asset and multi-asset models [59, 171]. In the multi-asset literature, the focus is on the joint maintenance optimisation of different assets, where asset dependencies can be of four types, namely economic, structural, stochastic, and resource dependency [171]. Previous works modeling the degradation of assets typically employ a discrete and finite number of states, mostly following a Markovian process. Note that models considering a three-state degradation can also be interpreted as a delay-time model. Often, in such models, the second state represents a *defect* phase, which can be determined by inspection [224]. Otherwise, one can replace scheduled inspections with information acquired via sensors. These sensors do not need to measure the degradation of an asset specifically but can give indirect information about its health condition [218], for instance, in the form of alerts [58]. In a multi-asset scenario, practical problems include complex dependencies induced by the geographical layout. Previous works considering centralised management of asset networks take into account traveling times of serving the assets [31, 32]. These introduced maintenance problems considering the travel times between machines and methods to optimise the overall maintenance costs, detailed below.

Travelling Maintainer Problems In the travelling maintainer problem (TMP), the objective is to find a path through a set of assets, defined by the travel times between any two of them, which minimises the sum of the response times (defined as the realised time to reach an asset). Essentially, the TMP is obtained from the travelling salesman problem (TSP) as a mean-flow variant (averaging the sum of response times), assuming that the repair times are insignificant and is thus NP-complete [1]. The TMP complexity increases dramatically if each asset is assigned a deadline, that is, an upper bound on the response time, which the maintainer must not violate. More recently, [213] attempted to use machine learning techniques to estimate time-dependent failure probabilities at asset locations and propose two TMP objectives; solve the learning and the TMP sequentially or simultaneously.

The dynamic TMP (DTMP) is a variant of the TMP that considers the problem where (i.i.d.) service demands arrive uniformly in some regions according to a Poisson process [28]. Service demands must be fulfilled by a single maintainer, such that the average response time is minimal. Another variant [31] studied the objective to minimise the sum of functions of response times to assets. This TMP variant integrates real-time CBM prognostics with the TSP by scheduling maintenance using predicted failure information. Extending on this work, [32] incorporates the travel time between geographically distributed assets. [65] formulated a variant of the DTMP as a Markov decision process (MDP). Dynamic heuristics are proposed for both the dispatching and repositioning of a repair crew using real-time condition information. In [87], the authors investigated a cyclic TMP, formulated as an MDP, combined with condition-based preventive maintenance. Similar to our approach, the proposed control-limit heuristic was compared with the optimal policy and with traditional corrective maintenance policies. When large state spaces are induced by dense asset networks, the exact methods employed to solve these problems can be computationally intractable. Then, one can resort to heuristics or methods that employ function approximation, such as reinforcement learning (RL).

Deep Reinforcement Learning in Maintenance Recently, RL and deep RL (DRL) have emerged as valuable tools to practical decision-making problems [153]. In RL, the goal is to either directly learn policies [208] or value functions [163] measuring the performance of a policy. In general, RL requires access to trajectories sampled from an online environment to estimate the results of decisions (actions) and, in the long term, can converge to optimal policies found via dynamic programming (DP). In practical problems, where large observation and action spaces are intractable for DP and RL, one can resort to policy and value function approximation [207]. When RL is combined with multi-layer (deep) neural networks (NN), this gives rise to DRL.

Recent DRL approaches to maintenance problems have shown promising successes and include learning opportunistic maintenance strategies on parallel machines, resulting in downtime and cost reduction compared to reactive and time-

based strategies [131]. [45] employed DRL to part-replacement management subject to stochastic failures. [2] and [145] considered inspection and maintenance policies to minimise long-term risk and costs in deteriorating engineering assets. This problem included multiple challenges similar to our setting, such as high state cardinality, partial-observability, long-term planning, environmental uncertainties, and constraints. [29] considered policy rollouts and value function approximation to learn policies for a partially observable Markov decision process (POMDP) modelling a robot visiting adjacent machines. However, unlike our case, these works do not consider maintenance activities performed in an asset network with multiple cost structures and degradation mechanisms.

The developed framework in this chapter combines the multiple research streams in maintenance optimisation into a novel framework for sequential decision-making for the maintenance of asset networks under uncertainty. We consider a multi-asset problem that is mainly resource-dependent, as a single repairman maintains the asset network. As in previous works, we assume a countable state space per asset. Additionally, we introduce three observable degradation states per asset, representing a censored observation of assets' actual, hidden degradation states. We assume that alerts received from sensory equipment are directly related to the degradation of assets and contain imperfect residual lifetime predictions from black-box prediction models. As is common in sequential decision-making problems, we adopt the objective to minimise the sum of discounted maintenance costs, i.e., costs incurred due to delays or (unnecessary) maintenance actions. Instead of a hard deadline, we assume that the asset-dependent costs increase if the decision-maker does not perform preventive maintenance before the random deadline triggered by the asset's failure mechanism. We benchmark our method against greedy heuristics based on traditional rankings of alerts tailored to the DTMPA, and a heuristic [53] aimed at optimising the visitation order of a deterministic TMP instance considering the urgency of the alerts and the near-future costs. We propose an algorithmic approach to learn policies for this problem based on the theory of distributional RL [18], i.e., a class of methods that approximates the distribution of long-term costs. In this novel setting, we learn policies that consider both maintenance and travelling decisions in a stochastic environment where transition probabilities are unknown and alerts act as early indicators of failures.

7.3 Dynamic Travelling Maintainer Problem with Alerts

The dynamic travelling maintainer problem with alerts (DMTPA) is defined as follows. There is a single repairman (decision-maker) and a set $\mathcal{M} = \{1, \dots, M\}$ of $M \in \mathbb{N}^+$ *machines*, each at a unique location in a network. Each machine $m \in \mathcal{M}$ is subject to a stochastic degradation process (see Section 7.3.2), as a consequence,

these machines require maintenance to prevent and resolve failures. The decision-maker cannot observe the degradation of machines directly but relies on alerts, denoted as $e_{m,i}$ (i-th alert at machine m), indicating that a machine has shown signs of degradation. Each alert may provide the decision-maker with (partial) information (see Section 7.3.3) about a machine's future degradation, that is, distributional information of the residual lifetime.

The decision-maker is responsible for maintenance and travel activities in the network. Travelling in the network takes time proportional to the distance between machine locations. Performing maintenance requires time during which the machine is assumed to be non-functional. The decision-maker needs to decide to travel to a machine location, start maintenance (resolve an alert) or wait. Performing maintenance is costly; in particular, a higher cost is paid for corrective maintenance (CM) compared to preventive maintenance (PM). Costs are also incurred for each unit of time a machine is non-functional (see Section 7.3.7), here the length of the maintenance period is determined by the maintenance type. The DMTPA's objective is to serve the network of machines while minimising the total maintenance costs. In this work, we model this problem with a discrete-time framework and consider different information levels in the alert. Our goal is to devise online scheduling algorithms that can adapt to this dynamic environment whilst minimising the expected future costs (see Section 7.4). We detail each part of the proposed problem framework in the forthcoming sections.

7.3.1 Network of Machines

In a network layout, depicted in Figure 7.1, each machine is located at a unique location and travelling between machines requires an integer number of time periods. We store the travel time $\theta_{ij} \in \mathbb{N}$ between two machines $i, j \in \mathcal{M}$ in the (symmetric) matrix $\Theta \in \mathbb{N}^{M \times M}$.

7.3.2 Degradation of Machines

Each machine $m \in \mathcal{M}$ is assumed to degrade independently according to an *underlying* discrete degradation model consisting of $\mathcal{N}_m = \{1, \ldots, x_m^{\mathrm{f}}\}$ states, where $|\mathcal{N}_m| \geq 3$. The *observable* states consist of the *healthy state* $x_m^{\mathrm{h}} \equiv 1$, the *down/failed state* x_m^{f} an *alert state* $x_m^{\mathrm{a}} \in \{2, \ldots, x_m^{\mathrm{f}} - 1\}$. We denote the set of observable states with $\mathcal{X}_m = \{x_m^{\mathrm{h}}, x_m^{\mathrm{a}}, x_m^{\mathrm{f}}\}$ where $m \in \mathcal{M}$. The remaining $x_m^{\mathrm{f}} - 3$ underlying states are assumed to be hidden to the decision-maker. In other words, we assume that transitions to the three observable states are perceived by the decision-maker, namely the transitions to the healthy state (after PM or CM), the alert state and the failed state while all other transitions are not observed. To map hidden states to observable

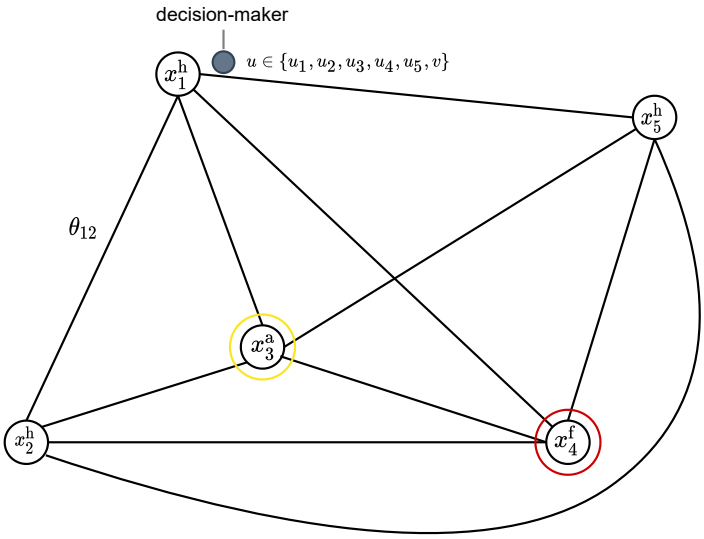


Figure 7.1: Schematic overview of the layout of a network consisting of $M = 5$ machines and one decision-maker. Machines can be in observable states: healthy x_m^{h}, alert x_m^{a} (yellow) and failed x_m^{f} (red). In our modeling framework, the decision-maker can select actions u, referring to travel actions to one of the locations $(u_1, u_2, u_3, u_4, u_5)$ or start maintenance at its current location (v). See Section 7.3.5.

states, we define for each machine $m \in \mathcal{M}$ the mapping $\phi_m : \mathcal{N}_m \to \mathcal{X}_m$ by

$$\phi_m(x_m) = \begin{cases} x_m^{\mathrm{h}} & \text{if } x_m \in \{1, \ldots, x_m^{\mathrm{a}} - 1\}, \\ x_m^{\mathrm{a}} & \text{if } x_m \in \{x_m^{\mathrm{a}}, \ldots, x_m^{\mathrm{f}} - 1\}, \\ x_m^{\mathrm{f}} & \text{if } x_m = x_m^{\mathrm{f}}. \end{cases} \tag{7.1}$$

We model the *underlying* hidden degradation process as follows. For each machine $m \in \mathcal{M}$, we have a degradation process $\{X_m(t), t \in \mathbb{N}\}$ where $X_m(0) = 1$ and $X_m(t) \in \mathcal{N}_m$ for all $t \in \mathbb{N}^+$. Without intervention by the decision-maker, these degradation processes $X_m(t)$ are assumed to be non-decreasing, i.e., $X_m(t+1) \geq X_m(t)$. The transition time between two subsequent states $i, i+1 \in \mathcal{N}_m$ is assumed to be random, given by a random variable $T_m^{i,i+1}$ defined on a positive, integer support.

When a machine m transitions to the alert state x_m^{a}, an *alert* is issued. The time between starting with a healthy machine until such an alert arrives is denoted with T_m^{a} and the time from alert arrival until failure, i.e., the residual lifetime, with T_m^{f}. Without intervention by the decision-maker, both random variables are defined by

first hitting times, namely

$$T_m^{\mathrm{a}} \stackrel{\mathrm{d}}{=} \min_{t\geq 0}\{t : X_m(t) = x_m^{\mathrm{a}} \mid X_m(0) = 1\} \tag{7.2}$$

and

$$T_m^{\mathrm{f}} \stackrel{\mathrm{d}}{=} \min_{t\geq 0}\{t : X_m(T_m^{\mathrm{a}} + t) = x_m^{\mathrm{f}} \mid T_m^{\mathrm{a}}\}. \tag{7.3}$$

Additionally, we assume that when the machine m reaches the failed state x_m^{f}, it resides in this state until the decision-maker performs a maintenance action. Regardless of the machine's state, after maintenance is done, the machine's state $X_m(t)$ is reset to its initial condition $X_m(0) = 1$.

7.3.3 Alerts and Information Levels

The decision-maker receives information through alerts. The i-th alert $e_{m,i}$ at machine m is issued after the i-th transition of $\phi_m(X_m(t))$ from x_m^{h} to the alert state x_m^{a}. Besides the issued alert, the decision-maker also observes when a machine m enters the failed state x_m^{f}.

We assume that the decision-maker receives information from a black-box prediction model, e.g., predictions of the residual lifetime T_m^{f}. We denote the time until an alert and residual lifetime of the i-th replacement, $i \in \mathbb{N}^+$, of a machine m with $T_{m,i}^{\mathrm{a}} \sim T_m^{\mathrm{a}}$ and $T_{m,i}^{\mathrm{f}} \sim T_m^{\mathrm{f}}$, respectively. The information becomes available to the decision-maker at the moment an alert is issued. We capture the uncertainty about the residual lifetime by distinguishing between four information levels:

($\mathbf{L}_0$) The alert $e_{m,i}$ contains no information about $T_{m,i}^{\mathrm{f}}$. The decision-maker has no information about the underlying degradation model.

($\mathbf{L}_1$) The alert $e_{m,i}$ contains an approximation (e.g., an expected failure time estimate or a Normal approximation) of the distribution function of the remaining time to failure. The decision-maker has no information about the underlying model.

($\mathbf{L}_2$) The decision-maker has full information about the underlying model, in particular, knows the distribution of T_m^{f}, i.e., $F_{T_m^{\mathrm{f}}}$. The decision-maker, however, only observes transitions between healthy, alert and failed states.

($\mathbf{L}_3$) The decision-maker has full information about the underlying model and observes each state transition, in particular, meaning that the set of observable states is $\mathcal{N}_m$ for all $m \in \mathcal{M}$.

Given the information level, the objective is to produce a policy that minimises the total expected discounted maintenance costs of a given asset network. The next

section formalises the degradation of the network of assets and the decision maker's actions as a decision process.

The decision-maker influences the environment by selecting actions. However, the exact dynamics and the actual state of the system may be unknown to the decision-maker. We first introduce the relationship between the true, hidden state of the network and the incomplete, observed network state that the decision-maker can use to make decisions. Second, we define the state of the complete network of machines to be a *network state*, and further introduce the costs and the main objective we wish to optimise.

7.3.4 Hidden Network States and Observed Network States

Dependent on the state of the machines and the decision maker's current location $\ell \in \mathcal{M}$, we represent a *hidden network state* as a vector

$$h = (x_1, \ldots, x_M, \ell, \iota, \delta, \hat{t}_1, \ldots, \hat{t}_M) \in \mathcal{H}, \tag{7.4}$$

where $\mathcal{H} = \{\mathcal{N}_1 \times \ldots \times \mathcal{N}_M \times \mathcal{M} \times I \times \Delta \times \mathbb{N}^M\}$. The first M entries of h, $x_1 \in \mathcal{N}_1$ until $x_M \in \mathcal{N}_M$ denote the state of the machines; the entry $\ell \in \mathcal{M}$ contains the current location of the decision-maker; the entry $\iota \in I = \{0, 1\}$ represents whether the decision-maker is travelling or not, respectively; the entry $\delta \in \Delta \subset \mathbb{N}^+$ contains the remaining time units until the decision-maker becomes available again ($\delta = 0$ indicates that the decision-maker is idle); lastly, we introduce entries $\{\hat{t}_m\}_{m=1}^M$ where $\hat{t}_m$ is defined as the *elapsed time* since the last transition of $X_m(t)$ from one of its elements in $\mathcal{N}_m \backslash \{x_m^{\mathrm{f}}\}$, which is defined formally in Section 7.3.8. We exclude the elapsed times since transitioning to the failed state as they do not carry any information.

Let $\{H(t), t \geq 0\}$, $H(t) \in \mathcal{H}$ be a stochastic process on $\mathcal{H}$, which is defined by

$$H(t) = \big(X_1(t), \ldots, X_M(t), \ell(t), \iota(t), \delta(t), \hat{t}_1(t), \ldots, \hat{t}_M(t)\big) \tag{7.5}$$

with initial state defined as $H(0) = (x_1^{\mathrm{h}}, \ldots, x_M^{\mathrm{h}}, 1, 0, 0, 0, \ldots, 0)$. The set of all possible histories of the network up to timestep t is defined as

$$\mathcal{H}_t = \{H_t = (h_0, u_0, \ldots, h_{t-1}, u_{t-1}, h_t) | (h_j, u_j) \in \mathcal{H} \times \mathcal{U}, 0 \leq j \leq t-1, h_t \in \mathcal{H}\}, \tag{7.6}$$

where $\mathcal{U}$ represents the action space of the decision-marker, and it is formally introduced in Section 7.3.5.

The *observed network states* $o \in \Omega = \{\mathcal{X}_1 \times \ldots \times \mathcal{X}_M \times \mathcal{M} \times I \times \Delta \times \mathbb{N}^M\}$ are simply censored hidden network states, i.e., given a network history H_t, the corresponding observed network state $o = (\tilde{x}_1, \ldots, \tilde{x}_M, \tilde{\ell}, \tilde{\iota}, \tilde{\delta}, \tilde{t}_1, \ldots, \tilde{t}_M)$ at time t is achieved through the mapping $\Phi_t : \mathcal{H}_t \to \Omega$ which will be defined in detail in Section 7.3.8. The variables $\{\tilde{t}_m\}_{m=1}^M$ are now defined as the elapsed time since the

last *observable* transition, i.e., the time since the last transition of $\phi_m(X_m(t))$ from one of its elements in $\mathcal{X}_m \backslash \{x_m^{\mathrm{f}}\}$. Note that sets $\mathcal{H}$ and Ω are unbounded.

7.3.5 Actions

At the start of a timestep, the decision-maker can either: (i) choose the action u_m to start travelling immediately to location $m \in \mathcal{M}$, (ii) choose the action v to immediately start a maintenance action at its current location or (iii) continue its previous activity or idling, which is encoded by choosing the action u_ℓ, with ℓ denoting its current location. The complete action space thus consists of $M+1$ actions, specifically, $\mathcal{U} = \{u_m\}_{m=1}^M \cup \{v\}$. The state-dependent action set $\mathcal{U}(h) \subseteq \mathcal{U}$ is given by

$$\mathcal{U}(h) = \begin{cases} \{u_m\}_{m=1}^M \cup \{v\} & \text{if } \delta = 0, \\ \{u_\ell\} & \text{if } \delta > 0. \end{cases}$$

If $\delta > 0$, the decision-maker is already occupied by either performing maintenance or travelling and cannot perform any other action besides u_ℓ. Note that for any hidden state h, regardless of the information level, $\mathcal{U}(h)$ and $\mathcal{U}(o)$, the state-dependent action set corresponding to the observation o induced by h, coincide. A representation of the hidden and observed network states and a transition (see Section 7.3.6) after an action is depicted in Figure 7.2.

7.3.6 Transitions

A network's state transition $h \to h'$ is composed of immediate (deterministic) consequences, i.e.,

$$h \to h^u = (x_1^u, \ldots, x_M^u, \ell^u, \iota^u, \delta^u, \hat{t}_1^u, \ldots, \hat{t}_M^u)$$

of the decision maker's chosen action $u \in \mathcal{U}(h)$ followed by rolling out a unit of time in the network, i.e.,

$$h^u \to h' = (x_1', \ldots, x_M', \ell', \iota', \delta', \hat{t}_1', \ldots, \hat{t}_M'),$$

explained in detail below.

Immediate Action Transitions

$$h^u = \begin{cases} (x_1, \ldots, x_M, \ell, \iota, \delta, \hat{t}_1^u, \ldots, \hat{t}_M^u) & \text{if } u = u_m \wedge m = \ell, \\ (x_1, \ldots, x_M, m, 0, \theta_{\ell,m}, \hat{t}_1^u, \ldots, \hat{t}_M^u) & \text{if } u = u_m \wedge m \neq \ell, \\ (x_1, \ldots, x_{\ell-1}, x_\ell^{\mathrm{f}}, x_{\ell+1}, \ldots, x_M, \ell, 1, t_\ell^{\mathrm{PM}}, & \\ \hat{t}_1^u, \ldots, \hat{t}_{\ell-1}^u, 0, \hat{t}_{\ell+1}^u, \ldots, \hat{t}_M^u) & \text{if } u = v \wedge x_\ell \neq x_\ell^{\mathrm{f}}, \\ (x_1, \ldots, x_M, \ell, 1, t_\ell^{\mathrm{CM}}, \hat{t}_1^u, \ldots, \hat{t}_M^u) & \text{if } u = v \wedge x_\ell = x_\ell^{\mathrm{f}}. \end{cases}$$

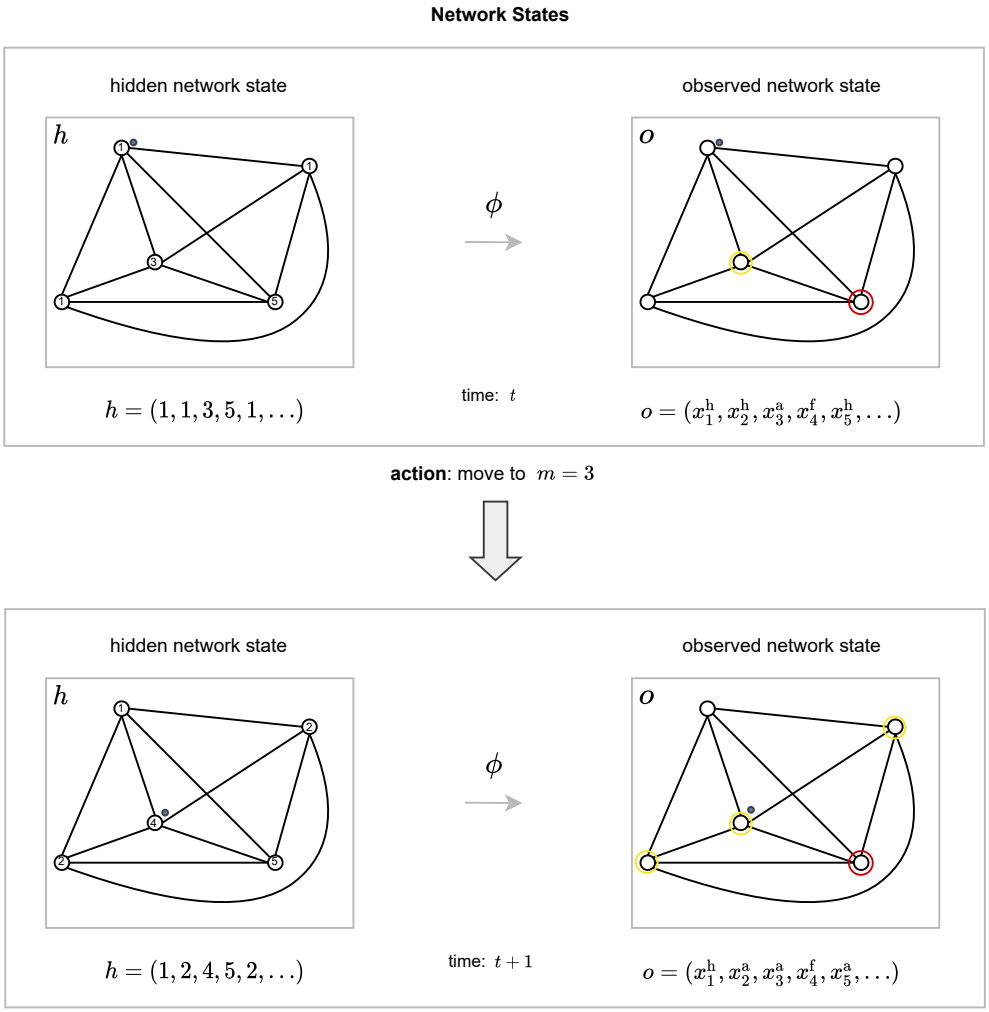


Figure 7.2: A network consisting of $M = 5$ machines. (**Top**) Hidden and observed network states at time t. The decision-maker is at location 1, an alert is present at location 3 (hidden machine state 3), and the machine at location 5 is in a failed state. (**Bottom**) The decision-maker moves to location 3 in the network. Two other machines that were in the healthy state (1) transition to state 2 in the *hidden network state*, the machine at location 3 transitions to state 4. The decision-maker observes two new alerts in machines 2 and 5 in the *observed network states*.

The first case represents the action to move to the decision maker's current location ℓ, which encodes the action to wait. The second case represents the travel action to another machine m. In this case, the remaining unavailability δ is set to $\theta_{\ell,m}$ and subsequently, ℓ is updated. When starting a maintenance period, its length is determined by the type of maintenance, that is, PM (or CM) will take a number of time steps $t_m^{\text{PM}} \in \mathbb{N}^+$ (or $t_m^{\text{CM}} \in \mathbb{N}^+$). Furthermore, initiating maintenance on a machine m advances the degradation state x_m to x_m^{f}, and its elapsed time is reset to zero. The reason to update x_m to x_m^{f} is to model an asset as unavailable during maintenance (regardless of PM or CM). These action transitions are described in the third and fourth cases, respectively.

Time Rollout

1. $\forall m \in \mathcal{M}$:

$$(x'_m, \hat{t}'_m) \leftarrow \begin{cases} (x_m^{\text{h}}, 0) & \text{if } (\ell^u, \iota^u, \delta^u) = (m, 1, 1), \\ (x_m^u, \hat{t}_m^u) & \text{else if } x_m^u = x_m^{\text{f}}, \\ (x_m^u + 1, 0) & \text{else if w.p. } \mathbb{P}(T_m^{x_m^u, x_m^u+1} = \hat{t}_m^u + 1 | T_m^{x_m^u, x_m^u+1} > \hat{t}_m^u), \\ (x_m^u, \hat{t}_m^u + 1) & \text{otherwise,} \end{cases}$$

2. $(\ell', \delta', \iota') \leftarrow (\ell^u, \max(\delta^u - 1, 0), \iota^u)$,

where $\mathbb{P}(T_m^{x_m^u, x_m^u+1} = \hat{t}_m^u + 1 | T_m^{x_m^u, x_m^u+1} > \hat{t}_m^u)$ denotes the conditional probability that machine m advances to the subsequent degradation state. The first line represents the finishing of a maintenance job for each machine, i.e., the machine transitions to a healthy state. The second line represents a machine in a failed state, not requiring any changes. The third line represents the transition to a subsequent degradation state, resetting the elapsed time. Finally, if none of these happens, the elapsed time is increased. After updating the machines, the remaining unavailability δ', due to an ongoing action, is decreased by one.

For each machine, we define the counting process $\{N_m(t), t \geq 0\}$ which counts the repair cycles by

$$N_m(t) = \sum_{t'=0}^{t} \mathbb{1}(X_m(t') = x_m^{\text{h}} \wedge \hat{t}_m(t') = 0),$$

where $\mathbb{1}(\cdot)$ is an indicator function that takes the value 1 if the predicate is true and 0 otherwise. Furthermore, let $\tau_m^n = \inf_{t \geq 0}\{t : N_m(t) = n\}$ be the time until the start of the n-th repair cycle of machine m, where $n \geq 1$. The elapsed times $\hat{t}_m(t)$ and $\tilde{t}_m(t)$ can now be defined as

$$\hat{t}_m(t) = \sup_{t' \in \{0, \ldots, t - \tau_m^{N_m(t)}\}} \{t' : X_m(t - t') = X_m(t)\} \tag{7.7}$$

and

$$\tilde{t}_m(t) = \sup_{t' \in \{0,\ldots,t-\tau_m^{N_m(t)}\}} \{t' : \phi_m(X_m(t-t')) = \phi_m(X_m(t))\}. \tag{7.8}$$

7.3.7 Cost Structure

Maintenance Cost Structure The PM/CM cost structure is defined at machine level. If the decision-maker starts a repair at time t at machine m which is not in a failed state, that is, $X_m(t) \neq x_m^{\mathrm{f}}$, the decision-maker starts preventive maintenance and incurs a cost $c_m^{\mathrm{PM}} \in \mathbb{R}^+$. Otherwise, if the machine has failed prior to the decision-maker choosing the maintenance action v, i.e., $X_m(t) = x_m^{\mathrm{f}}$, a corrective maintenance cost of $c_m^{\mathrm{CM}} \in \mathbb{R}^+$ is incurred. We assume that $c_m^{\mathrm{CM}} \geq c_m^{\mathrm{PM}}$ for all $m \in \mathcal{M}$.

Downtime Cost Structure A machine m at time t is said to be down when it is in a failed state or under repair, i.e., $X_m(t) = x_m^{\mathrm{f}}$. The decision-maker incurs a downtime cost of $c_m^{\mathrm{DT}} \in \mathbb{R}^+$ each time unit the machine is down. The same downtime cost is incurred each time unit of the downtime period, regardless if this period is initiated by a machine failure or a maintenance action. The immediate incurred cost $C_u(h)$ due to choosing action $u \in \mathcal{U}(h)$ in state h, is thus given by

$$C_u(h) = c_\ell^{\mathrm{CM}} \mathbb{1}(u = v, x_\ell = x_\ell^{\mathrm{f}}) + c_\ell^{\mathrm{PM}} \mathbb{1}(u = v, x_\ell \neq x_\ell^{\mathrm{f}}) + \sum_{m \in \mathcal{M}} c_m^{\mathrm{DT}} \mathbb{1}(x_m = x_m^{\mathrm{f}}). \tag{7.9}$$

7.3.8 Objective

Let $\Phi_t^{\mathbf{L}} : \mathcal{H}_t \to \Omega$ be a mapping, dependent on the time and the information level, which maps a history $H_t \in \mathcal{H}_t$ to an observation o by

$$\Phi_t^{\mathbf{L}}(H_t) = \begin{cases} o_t & \text{if } \mathbf{L} \in \{\mathbf{L}_0, \mathbf{L}_1, \mathbf{L}_2\}, \\ h_t & \text{if } \mathbf{L} = \mathbf{L}_3. \end{cases} \tag{7.10}$$

where

$$o_t = (\phi_1(X_1(t)), \ldots, \phi_M(X_M(t)), \ell(t), \iota(t), \delta(t), \tilde{t}_1(t), \ldots, \tilde{t}_M(t)).$$

The entries $\tilde{t}_1(t), \ldots, \tilde{t}_M(t)$, defined in Section 7.3.6, are straightforward to compute from the history H_t. Note that the mapping $\Phi_t^{\mathbf{L}}$ is *consistent* in the sense that it preserves both state-dependent action sets and action-cost pairs, that is, $\mathcal{U}(H(t)) = \mathcal{U}(\Phi_t^{\mathbf{L}}(H_t))$ and $C_u(H(t)) = C_u(\Phi_t^{\mathbf{L}}(H_t))$ for all t and information levels $\mathbf{L}$.

We are interested in a policy $\pi^{\mathbf{L}}$ which minimises the total expected discounted cost of a given network of assets. A policy is defined as $\pi^{\mathbf{L}} = (\pi_1^{\mathbf{L}}, \pi_2^{\mathbf{L}}, \ldots, \pi_t^{\mathbf{L}}, \ldots)$,

where $\pi_t^{\mathbf{L}} : \mathcal{H}_t \times \mathcal{U} \to [0,1]$ is a decision rule which gives the probability of the action to be taken at time t, given the (censored) history of the network H_t. Let $\gamma \in [0,1)$ be the discount factor and $J(\pi^{\mathbf{L}})$ be the total expected discounted cost. Thus, the objective is to find a policy $\pi_*^{\mathbf{L}}$ which satisfies

$$\pi_*^{\mathbf{L}} = \arg\min_{\pi^{\mathbf{L}}} J(\pi^{\mathbf{L}}) = \arg\min_{\pi^{\mathbf{L}}} \mathbb{E}_{\pi^{\mathbf{L}}}\left[\sum_{t=0}^{\infty} \gamma^t C_{u_t}\left(H(t)\right)\right], \tag{7.11}$$

where $u_t \sim \pi_t^{\mathbf{L}}\left(\cdot|\Phi_t^{\mathbf{L}}(H_t)\right)$ is the action at time t sampled from the policy, given the censored observation $\Phi_t^{\mathbf{L}}(H_t)$.

7.4 Solution Approaches

In this section, we present three solution approaches to solve the introduced class of problems.

7.4.1 Greedy and Reactive Heuristics

The simplest class of ($\mathbf{L}_1$) policies that can be used to solve this problem are constructed by ranking subsets of assets based on the network layout, cost structures and alert contents. We employ rankings that prioritise proximity, urgency and economic risk. The greedy/reactive heuristic is obtained by simply selecting the travel/maintenance action on the asset which maximises (or minimises) a criterion of choice at each decision epoch. We define $\mathcal{M}_t^{\mathrm{a}} = \{m \in \mathcal{M} \mid \phi_m(X_m(t)) = x_m^{\mathrm{a}}\}$ and $\mathcal{M}_t^{\mathrm{f}} = \{m \in \mathcal{M} \mid \phi_m(X_m(t)) = x_m^{\mathrm{f}}\}$ the sets of machines which at time t reside in the observed alert state or the failed state, respectively. Additionally, let $\mathcal{M}_t^{\mathrm{u}} = \mathcal{M}_t^{\mathrm{a}} \cup \mathcal{M}_t^{\mathrm{f}}$ be the set of non-healthy machines at time t. Specifically, we construct the greedy heuristic as rankings on subsets $A \subseteq \mathcal{M}_t^{\mathrm{u}}$ and the reactive heuristics as rankings on subsets $A \subseteq \mathcal{M}_t^{\mathrm{f}}$.

To break ties, multiple consecutive rankings (or finally random choices) are considered. For example, if we would rank on proximity and two assets are equally close (in terms of travel time), a secondary (e.g., urgency) ranking is needed to make a difference for the *ex-aequo*. We introduce three such rankings, for which we use a procedure Sort(a, b, c) to sort a collection of tuples a, based on variable b in the tuple, where c lists the sorting direction, increasing ($\uparrow$) or decreasing ($\downarrow$).

Travel Time (T) One can rank the priority of assets based on time needed to reach that location (proximity). For distant assets, the travel times will be long, meaning valuable time will be lost whilst no repairs are done. Hence, one could prioritise based on the closest travel times from the current location of the decision-maker,

i.e., nearest neighbour – a well-known heuristic for the TSP, that is,

$$\mathbf{m}_{\text{ranked}} = \textsc{Sort}\left(\{m, \theta_{\ell,m}\}_{m\in A}, \theta_{\ell,m}, \uparrow\right).$$

Failure Time (F) Given the information of the alert (one would need at least an ($\mathbf{L}_1$) information level), it is possible to rank the machines on the capped expected failure time (urgency). Let $t^{\text{a}}_{m,i}$ be the hitting time of the alert state during the i-th repair cycle of machine m and define

$$\tilde{T}^{\text{f}}_m = \max\left(t, \tau_m^{N_m(t)} + t^{\text{a}}_{m,N_m(t)} + \mathbb{E}\left[T^{\text{f}}_m\right]\right). \tag{7.12}$$

To prevent asset failures, we prioritise machines with the earliest expected failure time. Any asset in the failed state would be prioritised by setting $\tilde{T}^{\text{f}}_m = 0$ if $m \in \mathcal{M}^{\text{f}}_t$. Thus, the ranking is constructed as

$$\mathbf{m}_{\text{ranked}} = \textsc{Sort}\left(\{m, \tilde{T}^{\text{f}}_m\}_{m\in A}, \tilde{T}^{\text{f}}_m, \uparrow\right).$$

Cost Saving (C) One ideally accounts for assets with different cost structures (economic risk). In essence, we must determine if assets with a high benefit of PM over CM should be prioritised over existing failures. Therefore, we propose a ranking constructed as follows. For assets in the alert state, we compute the immediate benefit of PM over CM, specifically $c_m = (c^{\text{CM}}_m - c^{\text{PM}}_m) + (t^{\text{CM}}_m - t^{\text{PM}}_m)c^{\text{DT}}_m$, $m \in \mathcal{M}^{\text{a}}_t$. For assets in the failed state, we compute the total downtime cost when deciding to directly repair the asset, given by $c_m = (\theta_{\ell,m} + t^{\text{CM}}_m)c^{\text{DT}}_m$, $m \in \mathcal{M}^{\text{f}}_t$ and rank as

$$\mathbf{m}_{\text{ranked}} = \textsc{Sort}\left(\{m, c_m\}_{m\in A}, c_m, \downarrow\right)$$

In the remainder of this chapter, we apply the Greedy[F,T,C] heuristic to compare with the other proposed solutions, where [F,T,C] is the order of rankings. This is due to the network design in Section 7.5, where distances are equal between all machines and costs are equal for all machines. Note that the heuristics can be easily adapted to be in the class of ($\mathbf{L}_0$) policies by ignoring the ($\mathbf{L}_1$) urgency ranking. Compared to the reactive heuristic, an important fallback about the greedy heuristic is the haste to solve problems. As soon as a machine has issued the alert, it is regarded as urgent, and hence an idle greedy decision-maker will take immediate action. As such, we expect the greedy heuristic to perform poor in asset networks where the workload is low.

7.4.2 Travelling Maintainer Heuristic

The travelling maintainer heuristic (TMH) [53] policy is inspired by the DTMP, i.e., the TMP, which arises when all the residual lifetimes T^{f}_m are deterministic and

known. These residual lifetimes serve as deadlines, after which the decision-maker starts paying the costs associated with CM instead of PM, i.e., the costs depend on the response times. For $\mathbf{L} \in \{\mathbf{L}_1, \mathbf{L}_2\}$, we can construct (and solve) a TMP instance of the stochastic, dynamic environment using the alert contents. The TMH is obtained by selecting the first action of the solution to the constructed TMP instance if the decision-maker is idling; otherwise, it continues the current activity. The TMH consists of two main steps, detailed below.

First Order Approximation For all assets in the alert state, the TMH approximates the residual lifetime $T^{\mathrm{f}}_{m,N_m(t)}$, $m \in \mathcal{M}^{\mathrm{a}}_t$, by some value $\tilde{T}^{\mathrm{f}}_{m,N_m(t)} \in \mathbb{R}$, e.g., given by a black-box prediction method. Examples of this approximation include the $(\mathbf{L}_1)$-approximation using the expected residual lifetime, i.e., setting $\tilde{T}^{\mathrm{f}}_{m,N_m(t)}$ be the same as Eq. (7.12) or by means of optimal TBM policies under $(\mathbf{L}_2)$ for single machines [53]. The first example is the expected moment of failure, replaced by the current time when this expected moment is passed. The residual lifetime of a failed asset equals 0 by convention, thus setting $\tilde{T}^{\mathrm{f}}_{m,N_m(t)} = t$ for all $m \in \mathcal{M}^{\mathrm{f}}_t$.

Optimisation and Action Selection The TMH solves the constructed TMP instance by optimising the schedule for each possible order in which the assets can be visited. Assuming that any voluntary violation of the maintenance deadline is sub-optimal, the TMH constructs cost-effective maintenance schedules for the machines in the set $\mathcal{M}^{\mathrm{u}}_t$. For each path $\sigma \in \Sigma_{\mathcal{M}^{\mathrm{u}}_t}$ (the set of all permutations on $\mathcal{M}^{\mathrm{u}}_t$), the heuristic constructs an initial *tight* schedule, meaning that travel actions and maintenance actions are executed without any intermediate delay. Given a path σ, the TMH constructs a schedule with two timestamps per asset, the travel time and the beginning of the repair time employing the first-order approximation of failure times as deadlines.

Later, the TMH optimises this initial schedule by first delaying the last repair as much as possible, which allows for the delay of the previous repair and so on. Based on the set of possible schedules, the TMH uses the expected future costs of the trajectories induced by the heuristic policy to select the one(s) with minimum costs and maximum uptime, i.e., delaying maintenance as much as possible. At time t, the policy proceeds to select the first action of this schedule. The TMH policy aims at assessing approximating the dynamic problem with a deterministic one considering the network layout, the cost structure and the information captured in the alerts. Note that it is only necessary to compute a new schedule if new alerts are present in the network. However, the number of possible permutations is combinatorial and applying such an approach can be impractical when the number of machines in set $\mathcal{M}^{\mathrm{u}}_t$ is large. Further details of the heuristic are presented in [53].

7.4.3 n-Step Quantile Regression Double Q-Learning

Reinforcement Learning Formulation We adopt Eq.(7.11) as a DRL objective under $\mathbf{L}_0$. In this formulation, a decision-maker (agent) perceives observable states representing all possible locations, alerts, failures and elapsed times. In general, these observable states differ from the true hidden machine states, which are not accessible. Our asset networks allow us to model the problem as a POMDP, where observable states summarise the available past *history* in the form of elapsed times since the last transition. Lastly, we assume that the proposed environment can be simulated for a number of episodes $E \in \mathbb{N}$. Note that this access is possible when we assume that we learn online by observing past state transitions. Another way of observing such data is to reuse transitions from previous policies or implement a simulation environment that mimics real-world data before deploying the learned agents in a real environment.

We are interested in learning a policy $\pi^{\mathbf{L}_0}$, mapping observable states to action probabilities. Learning is possible by sampling a number of trajectories following a *behaviour* policy, i.e., a policy executing actions in the environment and evaluating its performance with respect to the main objective in Eq.(7.11). To evaluate performance, we define the standard characterisation of the state-action value function measuring the expected sum of costs starting from a given observable state o, action u, at time t as:

$$q^{\pi^{\mathbf{L}_0}}(o,u) = \mathbb{E}_{\pi^{\mathbf{L}_0}}\left[\sum_{k=0}^{\infty} \gamma^k C_{u_{t+k}}\left(H(t+k)\right) | \Phi_t^{\mathbf{L}_0}(H_t) = o, u_t = u\right]. \tag{7.13}$$

To avoid notation clutter, we refer to the class of DRL policies interchangeably as $\pi^{\mathbf{L}_0}$ and π, costs at time t as $c_t := C_{u_t}\left(H(t)\right)$ and observable states as $o_t := \Phi_t^{\mathbf{L}_0}(H_t)$. We assume no access to the environment transition probabilities (model), in which case a policy π may be learned as a mapping from states to actions or extracted from value function approximations. In this work, we propose a value function approximation method, referred to as n-step Quantile Regression Double Q-Learning (nQR-DDQN) with each of its components detailed in the forthcoming sections.

Deep Q-Learning Similar to [12, 94], we extend deep Q-learning (DQN) [163] by *combining* several modifications that improve performance, training times and maintain comparable sampling complexity [55, 94, 216], detailed below. In DQN, off-policy RL is combined with NN approximation to estimate values of state-action pairs. State information is passed to a NN $q_w : \Omega \times \mathcal{U} \to \mathbb{R}$, where w are trainable parameters. Similar to the original DQN, we store a set of *transitions* (o_t, u_t, c_t, o_{t+1}) in a *replay memory* $\mathcal{D}$ (hash table). Transitions are collected following an ϵ-greedy exploration scheme, i.e., picking a random action with probability ϵ, otherwise selecting the action with the lowest estimated q values. For training, we sample

a mini-batch of $N_B \in \mathbb{N}$ transitions from $\mathcal{D}$ uniformly at random to decorrelate past transitions. Based on the mean Bellman optimality operator, the parameters of the NN are trained to minimise the one-step temporal difference (TD[0]) error [207, p. 131], i.e., the error between the current estimate of the value of a state and its one-step update, acting greedily with respect to the approximated q function by means of gradient descent using the mean of the squared error (MSE) loss function

$$\mathcal{L}(w) = (c_t + \gamma \min_{a'} q_{\bar{w}}(o_{t+1}, a') - q_w(o_t, a_t))^2.$$

To stabilise training, DQN introduces a *target* network with parameters $\bar{w}$, i.e., a periodic copy of the *online* q_w that is not directly optimised and used as a target for the future q-value estimates. This copy is introduced to stabilise training and represents a fixed target that does not change at each update of w [163]. In our implementation, this copy is updated every $P \in \mathbb{N}$ episodes. In the remainder of this section, we provide further details on the proposed modifications, namely, double deep Q-Learning, multi-step RL and distributional RL.

Double Deep Q-Learning DQN can be affected by an underestimation bias (overestimation in maximisation) [216]. In our noisy environment, this can lead to overoptimistic actions and prevent learning. Thus, we employ double DQN (DDQN) [216] to decouple the action selection from its evaluation. In this way, the action selection when updating w remains dependent on the online q_w (the current network being optimised), effectively reducing the chance of underestimating action values on the target network $q_{\bar{w}}$. This change reduces harmful estimations in the original DQN objective leading to more stable temporal difference updates. DDQN effectively changes the loss of DQN to

$$\mathcal{L}(w) = (c_t + \gamma q_{\bar{w}}(o_{t+1}, \arg\min_{a'} q_w(o_t, a')) - q_w(o_t, a_t))^2.$$

Multi-Step Reinforcement Learning One-step temporal difference updates of DQN are biased when q_w estimates are far from true q^π values. To obtain better estimates, TD[0] updates can be generalised by bootstrapping over longer horizons. This reduces bias, but it can lead to high variance due to the environment's stochasticity [112]. Nevertheless, using a larger bootstrapping horizon can empirically lead to higher performance and faster learning. We include longer horizons considering the truncated n-step future discounted costs:

$$c^{t:t+n} = c_t + \gamma c_{t+1} + \gamma^2 c_{t+2} + \ldots + \gamma^{n-1} c_{t+n-1}. \tag{7.14}$$

Note that using the costs in Eq.(7.14) makes the updates of DQN on-policy and results in value function updates that rely on old and inaccurate transitions in the replay memory. This is due to the n-step transitions of the behaviour policy being

used for the updates. In the case that the policy that generated the transitions in the replay memory differs substantially from the current policy, the updates based on these estimates can cause inaccurate evaluations of the q function under the current policy. To make updates off-policy again, we need to introduce importance sampling terms [57]. However, these terms can also lead to higher variance. In practice, adding n-step terms even without importance sampling can aid learning, whilst high variance can be controlled by small values of n [93]. Thus, to balance bias (one-step updates) and variance (multi-step updates), we include an additive n-step term weighted by $\alpha \in \mathbb{R}$ in the original DQN training objective. This parameter α is introduced to control how much extra variance we allow when updating the estimated q values.

Distributional Reinforcement Learning In stochastic environments, the transitions, actions and costs can introduce multiple forms of randomness. In our framework, we consider the failure of crucial assets, in which risk-adjusted costs need to be considered when selecting actions. Thus, learning a distribution of future costs allows us to improve policy performance by taking into account the stochasticity of the problem explicitly. Distributional RL [18] aims to learn not a single point estimate of values but a distribution of returns, i.e., the distribution of the random variable

$$z^{\pi}(o,u) = \sum_{k=0}^{\infty} \gamma^k c_{t+k} \tag{7.15}$$

for given o and u, where $q^{\pi}(o,u) := \mathbb{E}_{\pi}[z^{\pi}(o,u)]$. We employ quantile regression DQN (QR-DQN) [55], in which the distribution of returns is modeled via a quantile regression on N data points with fixed uniform weights of the CDF of z^{π} as $\tau_i = \frac{i}{N}$, $i = 1, \ldots, N$.

Thus, we estimate the quantiles by learning a model $\psi_w : \Omega \times \mathcal{U} \to \mathbb{R}^N$, mapping each state-action pair to a probability distribution supported on $\{\psi_w^i(o,u)\}_{i=1}^N$, i.e.,

$$z_{\psi}(o,u) := \frac{1}{N} \sum_{i=1}^{N} \delta_{\psi_w^i(o,u)} \tag{7.16}$$

where δ_z represents a Dirac at $z \in \mathbb{R}$. Moreover, each ψ_w^i represents an estimation of the quantile value corresponding to the CDF quantile weight

$$\hat{\tau}_i = \frac{\tau_{i-1} + \tau_i}{2} \tag{7.17}$$

$\tau_0 = 0$, i.e., the data points that minimise the 1-Wasserstein metric to the true z^{π} [55]. Note that $z_{\psi}(o,u)$ can be used to compute the usual q_w estimates as $q_w(o,u) = \frac{1}{N}\sum_{i=1}^{N} \psi_w^i(o,u)$. To achieve unbiased gradients in QR-DQN, we replace the usual MSE loss of DQN with an asymmetric variant of the Huber loss [107] defined as

$\rho^{\kappa}_{\hat{\tau}_i}(\nu) = |\hat{\tau}_i - \mathbb{1}(\nu < 0)|\mathcal{L}_{\kappa}(\nu)$, where

$$\mathcal{L}_{\kappa}(\nu) = \begin{cases} \frac{1}{2}\nu^2 & \text{for } |\nu| \leq \kappa, \\ \kappa(|\nu| - \frac{1}{2}\kappa), & \text{otherwise,} \end{cases} \tag{7.18}$$

ν corresponds to pairwise TD errors and $\kappa \in \mathbb{R}$. Similar to the mean Bellman optimality operator, the distributional Bellman optimality operator [18], i.e.,

$$z^{\pi}(o,u) = C_u(o) + \gamma z^{\pi}(o', \arg\min_{u'} \mathbb{E}_{\pi}[z^{\pi}(o',u')]), \tag{7.19}$$

where o' is the next observable state, is used to approximate quantiles based on the n-step loss function (with DDQN correction) as:

$$\mathcal{L}^{t:t+n}_{QR}(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{i'=1}^{N} \rho^{\kappa}_{\hat{\tau}_i}(y^{i'}_{t:t+n} - \psi^{i}_{w}(o_t,u_t)), \tag{7.20}$$

where $y^{i'}_{t} = c^{t:t+n} + \gamma^n \psi^{i'}_{\bar{w}}(o_{t+n}, \arg\min_{u'} \sum_{i=1}^{N} \psi^{i}_{w}(o_{t+n},u'))$. Finally, we add the n-step QR loss term to a single-step term to arrive at the objective:

$$w^* = \arg\min_{w} \hat{\mathcal{L}}_{n\text{-QR-DDQN}}(w) = \arg\min_{w}[\mathcal{L}_{QR}(w) + \alpha\mathcal{L}^{t:t+n}_{QR}(w)], \tag{7.21}$$

note that $\mathcal{L}_{QR}(w)$ is the same as $\mathcal{L}^{t:t+1}_{QR}(w)$ and corresponds to TD[0] (one-step) updates. We refer to the above objective as the training objective for nQR-DDQN.

Neural Network Architecture We employ a NN ψ_w, depicted in Figure 7.3, that maps raw observable state vectors to pairs of actions and N quantiles of the distribution of z^{π}, i.e., $\psi_w : \mathbb{R}^d \to \mathbb{R}^{N\times|\mathcal{U}|}$. First, an observable state is flattened into an input vector $\boldsymbol{\zeta} \in \mathbb{R}^d$, where $d = 3M + 2$. Note that the length of the input vector depends linearly on the network size M and the times $\tilde{t}_m$ are truncated to be between $[0,T]$, where T is the maximum allowed simulation time, and later normalised as $\tilde{t}_m/T$. The vector $\boldsymbol{\zeta}$ is then passed through a series of layers $l \in \{1,\ldots,L\}$ of the form

$$\boldsymbol{\zeta}^l = \sigma^l(\mathbf{W}^l\boldsymbol{\zeta}^{l-1} + \boldsymbol{b}^l) \tag{7.22}$$

where $\mathbf{W}^l \in \mathbb{R}^{d^l\times d^{l-1}}$, $\boldsymbol{b}^l \in \mathbb{R}^{d^l}$, $d^l \in \mathbb{N}$, $d^0 = d$, $\boldsymbol{\zeta}^0 = \boldsymbol{\zeta}$ and σ^l is a (non-)linear activation function in every hidden layer, i.e., $l \in \{1,\ldots,L-1\}$. In the output layer L, $d^L = |\mathcal{U}| \times N$, where $|\mathcal{U}|$ is the size of the action space and σ^L is the identity function. The parameters $w = \{\mathbf{W}^l, \boldsymbol{b}^l\}_{l=1}^{L}$ are updated via the gradient descent method, Adaptive Moment Estimation (Adam) [124], computing adaptive learning rates for each parameter, with learning rate $\lambda > 0$ aimed at minimising the loss in Eq.(7.21). Note that λ controls the magnitude of the weight updates. The complete algorithm referred to as nQR-DDQN is presented in Algorithm 3.

Algorithm 3: n-step Quantile Regression Double DQN (nQR-DDQN)

Input: Initialise replay memory $\mathcal{D}$, n, E, N_B, T, N, α, P, λ, ϵ and functions ψ_w, $\psi_{\bar{w}}$ with random weights $w = \bar{w}$

for *episode* $= 1, \ldots, E$ **do**

  Sample initial observable state o_0;

  for $t = 0, \ldots, T-1$ **do**

    Select a random action u_t with probability ϵ, otherwise select $u_t = \arg\min_u \frac{1}{N} \sum_{i=1}^{N} \psi_w^i(o_t, u)$.;

    Execute u_t, observe c_t and o_{t+1};

    Store transition (o_t, u_t, c_t, o_{t+1}) in $\mathcal{D}$;

    if $t \geq n$ **then**

      Sample $\left\{ \left(o_j, u_j, c^{j:j+n}, o_{j+n}\right) \right\}_{j=1}^{N_B}$ from $\mathcal{D}$;

      $$y_j^{i'} = \begin{cases} c^{j:j+n'} & \text{if } n' \leq n \text{ and } o_{j+n'} \text{ terminal} \\ c^{j:j+n} + \gamma^n \psi_{\bar{w}}^{i'}(o_{j+n}, u^*) & \text{otherwise} \end{cases}$$

      where $u^* = \arg\min_{u'} \sum_{i=1}^{N} \psi_w^i(o_{j+n}, u')$;

      Take a gradient step acc. to Eq.(7.21) w.r.t. w;

    end

  end

  Every P episodes, update $\bar{w} = w$;

end

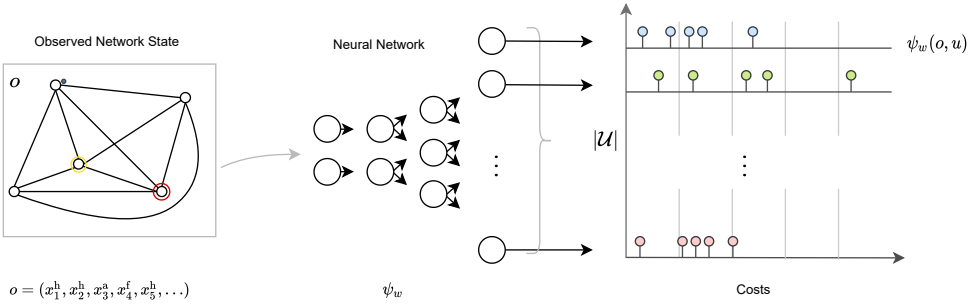


Figure 7.3: A neural network $\psi_w : \mathbb{R}^d \to \mathbb{R}^{N \times |\mathcal{U}|}$, mapping observable state vectors to estimates of the quantile locations of the cost distributions.

7.5 Experimental Settings

To benchmark the solutions in Section 7.4, we construct a set of asset networks where we can explicitly compute the alert information and optimal policies under the ($\mathbf{L}_3$) information level.

7.5.1 Asset Networks

Each machine $m \in \mathcal{M}$ has a similar degradation process defined as follows. The random transition time between two states $i, i+1 \in \mathcal{N}_m$ follows a Geometric distribution with support $\mathbb{N}^+$, i.e., $T_m^{i,i+1} \sim \text{Geo}(p_m^{i,i+1})$, where $p_m^{i,i+1} \in (0,1)$ for all i and m. Note that $T_m^{i,i+1}$ satisfies the Markovian property. The alert state is chosen to be the first non-healthy state, meaning that $x_m^{\text{a}} \equiv x_m^{\text{h}} + 1$. To ease some of the computations, we assume that $p_m^{i,i+1} = p_m^{\text{a}}$ for all $i \in \{x_m^{\text{a}}, \dots, x_m^{\text{f}} - 1\}$, consequently $T_m^{\text{f}} \sim \sum_{k=2}^{x_m^{\text{f}}} \text{Geo}(p_m^{\text{a}})$. Moreover, we reduce the complexity by assuming both unit travel and repair times, that is, $t_m^{\text{PM}} = t_m^{\text{CM}} = \theta_{mm'} \equiv 1$ for all $m, m' \in \mathcal{M}$. We briefly explain the setup per information level.

($\mathbf{L}_0$) The decision-maker (agent) can only interact with the environment. It observes $\mathcal{X}_m$, elapsed times and aims to learn policies for the underlying MDP observing transitions.

($\mathbf{L}_1$) The decision-maker observes $\mathcal{X}_m$, elapsed times and has access to distributional information about T_m^{f} and T_m^{a}. In particular, this information consists of (exact) estimates of $\mathbb{E}[T_m^{\text{f}}]$ and $Var[T_m^{\text{f}}]$.

($\mathbf{L}_2$) The decision-maker only observes transitions between healthy, alert and failed states. The problem can be formulated as a POMDP observing $\mathcal{X}_m$. The decision-maker has access to the distribution of T_m^{f}.

($\mathbf{L}_3$) The decision-maker observes $\mathcal{N}_m$ and has access to the transition probabilities. The problem can be defined and solved as an MDP. Since the transitions are Markovian, we can drop the elapsed times.

We generate experimental settings varying the number of machines $M \in \{1, 2, 4, 6\}$ in a network. The next section introduces three different cost structures considered in the experiments.

Cost Structure For each network size, we define three cost structures, namely C1, C2 and C3. Recall that the combined costs are defined as $\bar{c}_m^{\text{CM}} = c_m^{\text{CM}} + c_m^{\text{DT}} \frac{\gamma^{t_m^{\text{CM}}} - 1}{\gamma - 1}$ and $\bar{c}_m^{\text{PM}} = c_m^{\text{PM}} + c_m^{\text{DT}} \frac{\gamma^{t_m^{\text{PM}}} - 1}{\gamma - 1}$. The three cost structures represent different ratios $\bar{c}^{\text{CM}}/\bar{c}^{\text{PM}}$ of corrective and preventive maintenance costs. We aim to have each cost

structure represent a distinct, realistic cost relationship between preventive and corrective costs. Moreover, each cost structure should induce different optimal policies, i.e., policies that favour more or less frequent maintenance actions. For example, when $\bar{c}^{\text{CM}}/\bar{c}^{\text{PM}}$ is large, i.e., when CM costs greatly surpass PM costs, we expect that preventive maintenance policies outperform reactive policies. Note that the values of $\bar{c}^{\text{CM}}/\bar{c}^{\text{PM}}$ and the respective costs for each cost structure are presented in Table 7.1, we omit the machine index m as we will be reusing the same cost structure for all machines in most experiments.

Table 7.1: Cost structures considered in the experiments.

Cost Structure	c^{CM}	c^{PM}	c^{DT}	$\bar{c}^{\text{CM}}/\bar{c}^{\text{PM}}$
C1	9	0	1	10.09
C2	2	1	10	1.09
C3	4	1	1	2.51

Machine Degradation We consider four types of machine degradation (matrices), namely Q1, Q2, Q3 and Q4 presented in Appendix 7.A. Q1 models the simplest setup of three hidden states, which under the degradation model proposed by [61] is equivalent to a fully-observable MDP. More specifically, due to the three states per machine, this can be seen as a network of delay-time models. This case is specifically introduced to compare the heuristics, and trained agents with the optimal solution obtained using Policy Iteration (PI) under information level $\mathbf{L}_3$. In this case, $\mathcal{N} = \mathcal{X}$ and hence there is no information gap, viz. the optimal solution to the MDP is in the class of $\mathbf{L}_0$ policies. Both machines Q2 and Q3 transition to an alert state with probability 0.2, but degrade differently. Q2 transitions with probability 0.3 as soon as it reaches the alert state, whereas Q3 transitions with probability 0.7, thus Q2 degrades slower than Q3. Machine Q4 is modeled with seven underlying states, having the same transition probabilities of Q2, i.e., *slow* degradation.

7.5.2 Case Study

We introduce 16 *asset networks*, each having a different combination of network size, cost structure and machine degradation matrices. We use a simple shorthand notation to distinguish the various cases: a network with one machine and degradation matrix Q1 is labelled M1-Q1. Note that when we employ multiple distinct degradation matrices, we concatenate the labels in increasing order, e.g., a network with two machines Q2 and Q3 is labelled M2-Q2Q3. For larger network sizes, we use the same degradation matrix for multiple machines by spreading the degradation matrices evenly across the number of machines. Thus, experiment M4-Q2Q3 represents a network with four machines where half is assigned the degradation matrix

Q2, and the other half is assigned the degradation matrix Q3. For each network, we consider the cost structures C1, C2 and C3. This scheme is only changed for the last asset network with $M = 6$, labelled M6-Q2Q3Q4-C. This network increases the complexity by assigning different cost structures to machines. A detailed description per network is presented below.

M1-Q1 The most straightforward setup considers one machine (Q1) such that $\mathcal{X}_m = \mathcal{N}_m$, vanishing the information gap. This edge case is introduced to show that the learned policies and heuristics can match the performance or reproduce the policy found via policy iteration (PI). Depending on the cost structure, we can verify that the optimal policy acts greedily on an alert (under cost structure C1 and C3) or it is reactive with respect to a transition to the failed state (under cost structure C2).

M1-Q4 This setup considers one machine (Q4) where $\mathcal{X}_m \subset \mathcal{N}_m$, i.e., there is partial information about the network state space. We quantify the information gap of having information level $\mathbf{L}_3$ over less information $\mathbf{L} \in \{\mathbf{L}_0, \mathbf{L}_1, \mathbf{L}_2\}$ when presented with a single machine.

M2-Q2Q3 We scale up the model complexity by increasing the network size. We consider two machines Q2 and Q3, which implies that the decision-maker has partial information about the network state space. Note that each machine in this experiment degrades at a different rate, and ideally, a good policy accounts for this by adapting its response time for each machine.

M4-Q2Q3 Essentially, this network is obtained by doubling the M2-Q2Q3 network, i.e., now the decision-maker faces a four machines network. Note that the network now consists of two machines, Q2 and two machines, Q3. This is the maximum network size for which we can still optimise the MDP (under information level $\mathbf{L}_3$) via PI, using our hardware.

M6-Q2Q3Q4 The largest network increases the number of machines to six, and it is obtained by adding two machines Q4 to the network M4-Q2Q3. The result is a much more challenging problem in which each pair of machines will have different lifetime distributions. Moreover, the alerts of Q4 machines are likely to occur very early relative to the failed instance. This induces a challenging time-based maintenance problem where the decision-maker has to consider the risk of postponing maintenance adequately.

M6-Q2Q3Q4-C Lastly, we increase the model complexity by considering multiple cost structures. Network M6-Q2Q3Q4-C is identical to network M6-Q2Q3Q4; how-

ever, we now assign to different machines different cost structures. Q2 machines are assigned cost structure C2, Q3 machines the cost structure C3 and Q4 machines have cost structure C1. This option is more realistic as different assets may have different maintenance cost structures based on type, age, location and utilisation.

7.5.3 Training Parameters

For all experiments, we roll out our simulation for a maximum of $T = 500$ steps[7]. The simulation is restarted with different randomness, i.e., different alert/failure arrivals following identical probability distributions. We train the RL agent for a total of $E = 2000$ episodes and update the target network every $P = 30$ episodes. Initially, we set the exploration ratio $\epsilon = \epsilon_i = 0.1$ and decay linearly over 90% ($e_r = 0.9$) of the training samples, after which the exploration ratio remains constant at $\epsilon = \epsilon_f = 0.005$. We define the discount rate $\gamma = 0.99$ and the learning rate $\lambda = 5 \times 10^{-4}$. We sample mini-batches of size $N_B = 32$ and have a maximum replay memory size $|\mathcal{D}| = 100000$. We use $n = 5$ and weigh the n-step loss by $\alpha = 1$, i.e., it has the same weight as the TD[0] loss.

The NN architecture is identical for all experiments. We employ a Feed-Forward NN, comprised of a linear embedding layer that maps input vectors to a $\mathbb{R}^{64}$ space, i.e., $d^1 = 64$. Following, two layers with the rectified linear unit (ReLU), i.e., $\text{ReLU}(x) = \max(0, x)$ activation function map the previous with dimensions $d^2 = d^3 = 64$. We follow the original implementation of QR-DQN [55], set $\kappa = 1$ and employ $N = 51$ quantiles mapping the inputs to a tensor of size $N \times |\mathcal{U}|$, where $|\mathcal{U}| = M + 1$. We train all models on hardware with a Ryzen 3950X CPU and a RTX 2080Ti GPU. A summary of all the training parameters is presented in 7.B.

7.6 Experimental Results

To test the performance of the proposed solutions, we generated 512 test instances (episodes) for each experimental setting. In our experiments, if the same (deterministic) policy is evaluated twice on an episode, the performance will be identical. Hence, the only difference lies in the information level, i.e., more or less information about the underlying degradation is given to the decision-maker.

We measure the proposed solutions' performance and present the estimated expected discounted cost together with a 95% confidence interval after rolling out the policies for $T = 500$ steps on each episode. We also include the results obtained finding the optimal policy under information level ($\mathbf{L}_3$) following (PI). We were able to solve all cases up to $|\mathcal{M}| = 4$ using Cartesius [35], the Dutch national supercomputer. When solutions yielded identical performance, the policies were

[7] Assuming a maximum error level $\xi(1-\gamma)/c \approx 0.006$ between the infinite horizon costs and truncated horizon costs at time T, i.e., $T > \log(\xi(1-\gamma)c^{-1})/\log(\gamma) - 1$ where $c = 10$.

Table 7.2: Average costs (95%CI) over 512 episodes when rolling out policies for 500 steps. **Bold:** Lowest cost amongst the proposed solution approaches.

Solution (Info. Level)	Asset Network	Cost Structure		
		C1 [95% CI]	C2 [95% CI]	C3 [95% CI]
PI ($\mathbf{L}_3$)	M1-Q1	16.36	123.91	32.72
	M1-Q4	4.730	47.582	9.461
	M2-Q2Q3	21.230	190.275	39.550
	M4-Q2Q3	79.976	432.440	96.166
	M6-Q2Q3Q4	-	-	-
	M6-Q2Q3Q4-C		-	
nQR-DDQN ($\mathbf{L}_0$)	M1-Q1	**16.365** [16.171, 16.56]	**124.96** [123.541, 126.378]	**32.804** [32.443, 33.165]
	M1-Q4	**8.806** [8.608, 9.004]	**47.408** [46.894, 47.922]	**14.513** [14.343, 14.683]
	M2-Q2Q3	**25.139** [24.935, 25.343]	**202.311** [200.675, 203.946]	**46.995** [46.609, 47.381]
	M4-Q2Q3	**92.654** [90.736, 94.571]	**470.625** [467.640, 473.611]	**106.525** [105.717, 107.334]
	M6-Q2Q3Q4	**176.642** [175.234, 178.051]	**711.188** [705.789, 716.586]	**159.527** [158.294, 160.760]
	M6-Q2Q3Q4-C		**347.500** [344.986, 350.014]	
H - Greedy [F,T,C] ($\mathbf{L}_1$)	M1-Q1	**16.365** [16.171, 16.56]	182.233 [180.126, 184.339]	**32.804** [32.443, 33.165]
	M1-Q4	16.61 [16.417, 16.804]	179.75 [177.624, 181.876]	32.818 [32.474, 33.163]
	M2-Q2Q3	30.9 [30.495, 31.305]	306.366 [304.26, 308.472]	56.692 [56.279, 57.105]
	M4-Q2Q3	112.304 [110.395, 114.212]	526.248 [523.62, 528.877]	112.306 [111.444, 113.168]
	M6-Q2Q3Q4	231.498 [228.491, 234.505]	741.568 [735.639, 747.497]	168.064 [166.677, 169.451]
	M6-Q2Q3Q4-C		379.799 [375.934, 383.665]	
H - Reactive [F,T,C] ($\mathbf{L}_1$)	M1-Q1	103.361 [102.217, 104.506]	**124.96** [123.541, 126.378]	51.736 [51.195, 52.278]
	M1-Q4	40.018 [39.595, 40.442]	**47.408** [46.894, 47.922]	20.127 [19.912, 20.342]
	M2-Q2Q3	154.074 [153.033, 155.114]	283.619 [281.469, 285.768]	82.419 [81.845, 82.993]
	M4-Q2Q3	306.278 [304.876, 307.68]	718.158 [713.699, 722.617]	173.682 [172.799, 174.565]
	M6-Q2Q3Q4	396.714 [395.106, 398.321]	1053.663 [1046.581, 1060.745]	231.742 [230.677, 232.806]
	M6-Q2Q3Q4-C		473.647 [470.884, 476.41]	
TMH ($\mathbf{L}_2$)	M1-Q1	**16.365** [16.171, 16.56]	**124.96** [123.541, 126.378]	**32.804** [32.443, 33.166]
	M1-Q4	**8.806** [8.608, 9.004]	**47.408** [46.894, 47.922]	**14.513** [14.343, 14.683]
	M2-Q2Q3	**25.221** [25.037, 25.405]	235.746 [233.683, 237.809]	**46.757** [46.404, 47.111]
	M4-Q2Q3	111.591 [110.188, 112.993]	634.828 [630.347, 639.309]	111.865 [110.988, 112.743]
	M6-Q2Q3Q4	214.435 [212.463, 216.408]	989.006 [982.159, 995.853]	181.077 [179.628, 182.527]
	M6-Q2Q3Q4-C		379.669 [376.796, 382.543]	

also identical. The results are summarised in Table 7.2, where the best performance results among the proposed solutions are represented in bold.

M1-Q1 In the most straightforward experiments, the proposed solutions match the performance and behaviour of the policies found via PI when there is no information gap. Moreover, we see that the optimal policies coincide with the Greedy heuristic (C1 and C3) and the Reactive heuristic (C2).

M1-Q4 Now, the optimal policy of the underlying MDP prescribes to either wait until the last state before failure (C1 and C3) or to delay maintenance to when the asset has failed (C2). Note that the latter policy is reactive, and the performance of the proposed Reactive heuristic matches the performance of PI. The best performing solutions are the TMH and nQR-DDQN. Since both operate under partial information of the failure probabilities, they obtain similar results by either computing an optimal TBM policy [53] or learning via interaction.

M2-Q2Q3 When increasing the number of machines, we expect the performance gap between the ($\mathbf{L}_3$) policy (obtained via PI) and the other policies to grow. In this experiment, the DRL solution achieves the best performance for all cost structures. The TMH policy obtains similar performance to *n*QR-DDQN for both cost structures C1 and C3. The DRL solution obtains a very close result to PI for C2, achieving 202.311 compared to the optimal cost of 190.275. The TMH policy shows poor performance here since it does not account for positive response times.

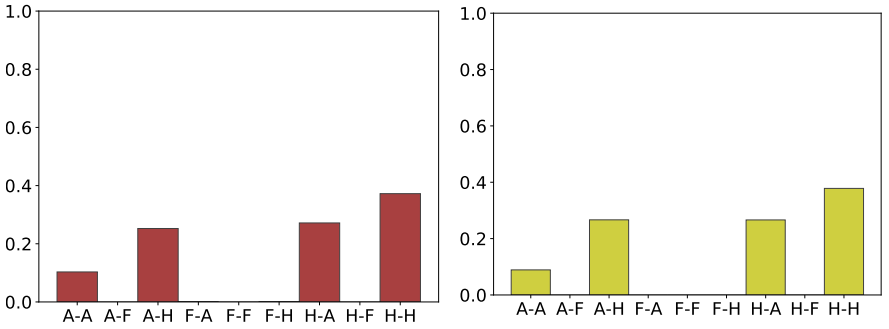
M4-Q2Q3 - M6-Q2Q3Q4 For these experiments, *n*QR-DDQN consistently outperforms the other heuristics. When the network size increases, the heuristics suffer to cope adequately with the increased complexity. The best performing heuristic is the TMH policy, showing that a higher information level yields better policies, even if they are myopic. However, it consistently shows poor performance when machines with the C2 cost structure are part of the asset network. For M4-Q2Q3, *n*QR-DDQN achieves similar performance to the optimal policies under full information.

M6-Q2Q3Q4-C The largest network, containing 6 assets with mixed cost structures, is also the most complex network. In this case, both the Greedy heuristic and the TMH solution yield similar performance. This result is not surprising, as more machines simultaneously reside in the observed alert state; the TMH pushes maintenance decisions to meet as many maintenance deadlines as possible. Thus, the added benefit of having access to a residual lifetime metric is reduced due to the urgency of repairs. Nevertheless, the DRL agent outperforms all heuristics in this scenario, implying that the learned agent is able to delay maintenance decisions more accurately than any investigated heuristic approach.

7.6.1 Comparing Policies

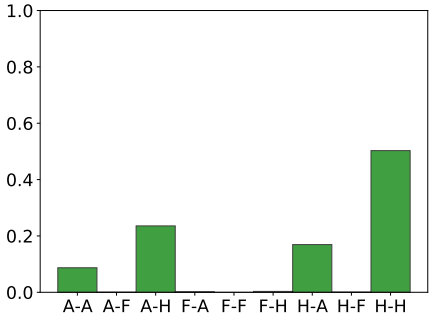
In this section, we focus our attention on the asset network M2-Q2Q3. In this particular experiment, the performance of *n*QR-DDQN, Greedy heuristic and TMH are close, with *n*QR-DDQN obtaining slightly lower costs. We select M2-Q2Q3 with cost structure C1 as a case to provide deeper insights as to why the policies are different. For a case with two machines, we can represent an observable state using the presence of alerts (**H**: Healthy, **A**: Alert, **F**: Failed). To reduce the size of the state space, we collapse the decision maker's location and the elapsed times. Under these simplifications, we can list nine observable network states and examine the *visitation* distribution of each solution. Figure 7.4 shows the visitation distribution when rolling out each policy for 512 episodes.

We first observe how the visitation distribution of the reactive heuristic (Figure 7.4d) differs from the remaining solutions. In the reactive case, the decision-maker visits states where one or both machines are in the failed state (F) more often. Unsurprisingly, given that the C1 cost structure favours preventive maintenance,

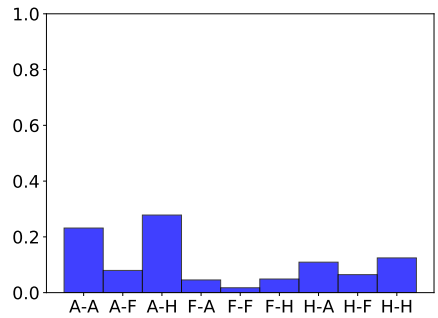


(a) nQR-DDQN $J(\pi)$:25.139

(b) TMH. $J(\pi)$:25.221



(c) H-Greedy. $J(\pi)$:30.9



(d) H-Reactive. $J(\pi)$:154.074

Figure 7.4: Visitation distribution of observable states (M2-Q2Q3) over all episodes (**H**: Healthy, **A**: Alert, **F**: Failed).

this leads to high incurred costs. The greedy heuristic distribution (Figure 7.4c) mostly visits states where both machines are in a healthy state (H), implying that unnecessary costs due to excessive maintenance of the assets are incurred. Since the benefit of PM over CM in C1 is large, this policy yields much lower costs than the Reactive heuristic. Both the TMH (Figure 7.4b) and nQR-DDQN (Figure 7.4a) have similar visitation distributions. In both cases, the policies attempt to delay maintenance to the “right" time. However, the heuristic is more conservative than the learned policy, i.e., the agent visits states where the second machine is an alert state (H-A) more often. This shows that the agent takes more “risk", allowing the second machine to be in the alert state slightly longer than TMH.

7.7 Managerial Insights

For larger networks, characterising the behaviour of the learned policies by the DRL agent becomes challenging as it depends on more complex states containing more machines, alerts and elapsed times. In general, it is not possible to characterise the learned policies via a set of heuristic rules. Nonetheless, we observe that the DRL agent tends to take proactive actions and move to healthy assets which carry the highest economic risk. Moreover, the learned policies tend to take more risk than the other heuristics, learning to delay preventive maintenance as much as possible, sometimes incurring several extra downtime penalties for this reason.

Additionally, for networks with $|\mathcal{M}| = 6$, we observe that in one instance, the DRL agent leaves assets with frequent failures (in terms of the distribution of T_m^{f}) unattended. This can be explained by the single decision-maker reaching its capacity limit. Thus, depending on the network size and degradation processes, a single decision-maker may not be sufficient to maintain the entire network without sacrificing reliability. Decision-makers can employ such insights when determining the size of repair crews, considering the number of assets and their degradation processes. The other considered heuristic policies do not account for such behaviour. Such behavior may be optimal if this reduces the overall maintenance costs.

We note that each of the proposed methods in this work reflects on different assumptions regarding the observed information from the network and relates to different application scenarios: in case no historical information about the degradation of assets or the operational lifetimes is available, greedy or reactive policies can be easily implemented. If methods for estimating the remaining useful lifetime of assets exist, heuristic policies like the TMH can be employed to take into account the trade-off of costs of resolving the alerts in the network. When online observation is possible or a simulation environment exists, policies can be learned via the DRL approach proposed in this work. In this case, an online system can take observations as inputs and generate instructions for a repair crew. As a result, the learned policies can inspire further heuristic developments.

7.8 Conclusion

This chapter introduces the dynamic travelling maintainer problem with alerts (DTMPA) and proposes maintenance strategies for a network of modern industrial assets with stochastic failure times. We consider a real-world scenario where assets are part of a larger asset network, and alerts triggered by degradation monitoring devices are used to make maintenance decisions. We have proposed a general framework to this problem, where independent degradation processes model the alert as an early and imperfect indicator of failures. Therefore, the decision-maker only has access to partial degradation information to help make maintenance and travel actions, leading to the lowest expected discounted cost.

To solve the problem, we have employed three methods utilising different information levels from the alerts. When alerts come with no extra information, a class of greedy and reactive heuristics that rank alerts based on travel times, failure times, or economic value can be employed. When residual lifetime information is available, such as the expected failure time, we can approximate the dynamic problem with a deterministic problem instance similar to the travelling maintainer problem. This method considers the alerts to create and optimise schedules that minimise the expected maintenance costs over the near future. Lastly, when alerts carry only elapsed times since their generation and a simulated environment is present, we can learn policies via deep reinforcement learning trained to approximate the state-action value distributions of long-term costs.

Our results show that we can effectively replicate the performance of optimal policies when no information gap exists between alerts and the real underlying degradation. When an information gap is present, and the number of assets in the network is small (up to four assets), our methods yield effective policies close to optimal under full information. When computing an optimal policy becomes computationally intractable, the learned policies result in the lowest expected discounted cost and can learn policies that balance the failure risk and the maintenance costs better than the competing methods. Lastly, depending on the information level of the alerts, interested parties can select one or more of the proposed methods as viable options to solve the sequential decision-making problem that arises from such scenarios.

Appendix

7.A Appendix A: Degradation Matrices

All the degradation matrices used in the numerical experiments are presented below.

$$\text{Q1} = \begin{bmatrix} 0.8 & 0.2 & 0 \\ 0 & 0.7 & 0.3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{Q2} = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.7 & 0.3 & 0 \\ 0 & 0 & 0 & 0.7 & 0.3 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Q3} = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0.3 & 0.7 & 0 & 0 \\ 0 & 0 & 0.3 & 0.7 & 0 \\ 0 & 0 & 0 & 0.3 & 0.7 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Q4} = \begin{bmatrix} 0.8 & 0.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

7.B Appendix B: Deep Reinforcement Learning Hyperparameters

During training of *n*QR-DDQN we employ a number of hyperparameters listed in Table 7.3.

Table 7.3: *n*QR-DDQN Hyperparameters

Hyperparameter	Description	Value
T	length of an episode	500
$\lvert\mathcal{D}\rvert$	size of the replay memory	100000
E	number of episodes	2000
P	number of episodes to update target NN	30
N_B	mini-batch size	32
γ	discount rate	0.99
e_r	exploration fraction	0.90
ϵ_i	initial exploration probability	0.1
ϵ_f	final exploration probability	0.005
λ	learning rate	5×10^{-4}
n	steps to bootstrap in the n-step loss	5
α	n-step loss weight	1
κ	Huber loss weight	1
L	number of NN layers	4
N	number of quantiles in QR	51
M	number of machines in a network	$\{1, 2, 4, 6\}$

Chapter 8

Conclusions

In this chapter, we provide an overview of the studied problems and the main contributions of this thesis. Next, we discuss the limitations of the methods proposed and present possible directions for future research. Lastly, we relate the contributions of this thesis to the broader application context of using machine learning (ML) in decision-making and discrete optimisation problems. In particular, we discuss the relationships with other methods in machine prognostics and the combination of ML and combinatorial optimisation (CO) methods for industrial problems.

8.1 Thesis Overview

This thesis has studied several problems pertaining to a network of industrial assets with unknown degradation mechanisms that require visitation and maintenance. The overall structure of the thesis follows from breaking down the main problem into three main components. In particular, we first studied prognostics problems, in which the objective is to arrive at accurate asset prognostics from sequential data gathered via remote monitoring devices. Second, we studied routing problems that arise when assets at different locations form an asset network, and the goal is to ensure the lowest visitation costs. Lastly, we studied the combined problem of visiting a network of assets ensuring that maintenance is performed under minimal costs in the presence of censored information about the degradation of assets.

The common theme across the chapters is the focus on data-driven methods and the methodologies employed throughout the thesis. We assume we only have access to sampled data from each problem and aim to learn accurate predictions and decisions directly from this data. The methodologies proposed in this thesis focus on end-to-end, deep learning and deep reinforcement learning methods that can take advantage of the heterogeneous data sources of each problem. The methods proposed comprise algorithmic solutions for the studied problems and can achieve

promising results in comparison to previous works.

In more detail, Chapter 2 provided a background on the ML concepts employed throughout this thesis. In Chapter 3, we studied the problem of predicting the remaining useful life (RUL) of industrial assets. Our objective is to learn to output RUL predictions from sequential data of monitoring devices without requiring expert knowledge about the underlying degradation processes. Chapter 4 considered the RUL prediction problem when run-to-failure labelled data is not available for the asset of interest but exists for assets with different monitoring data distributions and failure modes. This practical problem arises when new equipment versions do not have enough run-to-failure information but still require failure prognostics. We aim to use this data from pre-existing assets to make inferences for other (newer) assets with only sensor information.

In Chapter 5, we moved away from prognostics problems and studied learning policies for routing problems. In this problem, we are mainly concerned with learning good improvement policies to search for near-optimal solutions. We assume access to instance data such as location and distances and seek to solve these routing problems directly from this data. Chapter 6 also studied the problem of learning improvement policies for routing problems. However, we assume that previous expert heuristics exist and can be taken advantage of to improve the speed of learning effective policies. That is, we aim to learn good policies extracting information from sub-optimal policies acting as experts.

Lastly, Chapter 7 combined the first two streams of this thesis in a sequential decision-making problem considering travelling and maintenance decisions. In this problem, named dynamic travelling repairman with alerts (DTMPA), a decision-maker responsible for travelling and maintenance decisions receives alerts relating to the degradation of assets in a network. The alerts represent partial observability of the actual degradation process of assets and are an early indication of failure. In our approach to the DTMPA, we seek to learn policies that optimise preventive (before failure) and corrective (after failure) maintenance costs while serving a network of assets considering their uncertain failure times.

8.2 Main Results

Below we present an overview of the significant results and conclusions for each chapter in this dissertation in relation to each proposed research question (**RQ**).

RQ1 *How to effectively achieve accurate RUL predictions that can retain long term information about degradation data?*

In Chapter 3, we considered an asset prognostics problem learning from condition-based sequential monitoring data of assets. In this problem, we wish to

take advantage of sequential data to learn to output RUL estimates that can later be employed in maintenance policies. Our model can learn from exploiting the relationship between the multiple sensors and the observed RUL of assets. Additionally, the proposed method employs an attention mechanism that allows RUL outputs at the current time to consider the importance of previous time steps within a time window. Numerical experiments show that this attention-based neural network can learn more accurate RUL predictions without requiring pretraining or extensive parameter search of previous deep learning methods. Moreover, analysing the learned weights of the attention mechanism allow us to visualise the temporal relationship between sensor inputs and RUL outputs. This visualisation can help decision-makers better understand the relationships learned by the neural network model, providing more transparency and managerial insights.

RQ2 *How to adapt RUL predictions from previous run-to-failure data to new assets with only sensor information under different operating conditions and fault modes?*

Chapter 4 studied the problem of learning RUL estimates when only sensor information is available for the asset of interest. That is, we assume that labelled RUL information, i.e., the time until failure, does not exist for newly installed assets. We assume that previously labelled data for assets under different operating conditions and fault modes exist and are related to the assets with unlabelled data. We refer to the unlabelled data to which we seek RUL labels as the target domain and the labelled data from previous assets as the source domain. We propose a domain adaptation method that can learn better RUL predictions for the target domain compared to methods without the adaptation mechanism. In particular, the adaptation is most effective when the source contains more operating conditions and failure modes than the target data. Nonetheless, results show that even when adaptation is more challenging (unrelated failure modes and operation), our method can still achieve better RUL estimates than a model that is only trained on the source domain data. Our method can also achieve consistent uplift in performance when compared to previous domain adaptation methods.

RQ3 *How to learn better heuristics to solve routing problems directly from observed data of previous instances?*

In Chapter 5, we considered routing problems. In these problems, our goal is to achieve the lowest visitation costs when considering a network of locations to be visited represented as a graph. In particular, we consider classic NP-hard routing problems such as the travelling salesman problem (TSP), the multiple travelling salesman problem and the vehicle routing problem. In our approach, we propose to learn improvement policies for the problems directly from sampled data. That is, we sample instances of these problems and propose learning

improvement policies based on local search operators employing edge swaps. We propose a neural network architecture that can encode the graph structure and the ordering of nodes for action selection capable of surpassing the quality of other more specialised heuristics and previous ML methods. Starting from the TSP, we show that our method can be generalised to more complex routing problems. Additionally, we can learn policies that achieve competitive performance with previous hand-designed and learned heuristics, requiring fewer samples to achieve near-optimal solutions.

RQ4 *How to take advantage of expert heuristics to learn to solve routing problems?*

Chapter 6 examined a similar objective to Chapter 5 but assumes that we have access to hand-designed (expert) heuristics. Chapter 6 aims to accelerate learning by extracting knowledge in expert heuristics for the travelling salesman problem. A main result shows that our simplified neural network from Chapter 5 can reach good quality policies much sooner than our previous method. In this method, we propose warm-starting policies using previously obtained data from heuristics used as expert demonstrations. Compared to other methods, our proposed network learns to approximate near-optimal solutions to the problems much quicker than previous methods, requiring much lower sample complexity. Moreover, it maintains similar overall performance at the end of training compared to methods that learn from scratch.

RQ5 *How to automatically learn policies to solve a dynamic routing and maintenance problem in an asset network directly from observed data?*

Chapter 7 investigated a combined problem of travelling and maintaining a network of assets, referred to as the dynamic travelling maintainer problem with alerts (DTMPA). In the problem, we assume that a repairperson has access to alert information from assets located in a network. This alert information is an early warning of the degradation of assets coming from condition-based algorithms. Our goal is to learn policies that reduce the overall costs of serving this network, considering the trade-off of performing maintenance too early and incurring a high number of maintenance activities or performing maintenance too soon and incurring less frequent higher corrective maintenance costs. Our proposed method shows that we can learn effective policies under simplified assumptions of travel times and maintenance costs. In this intractable setup for classical CO methods, our solution can surpass the quality of tailored heuristics and approach the quality of an oracle capable of observing the actual degradation of assets.

8.3 Limitations and Future Research

The models and algorithms developed in this thesis can be extended in many different ways. Here we give an overview of their limitations and several interesting directions for future research.

In Chapter 3, we study an RUL prediction problem. Several works have focused on improving the overall prediction performance of RUL point estimates, improving the performance of the work presented in this thesis. An important limitation of the proposed model is that it assumes that, during training, observed temporal sensor data is available for a number of assets. However, practitioners may need RUL predictions without any or very little observed data from systems. Thus, interesting future research directions include devising few-shot learning methods that learn with limited sensor data on assets, e.g., [63]. Moreover, although the proposed attention mechanism provides insights about the temporal relationship between inputs and outputs, it does not provide information on which specific sensor(s) are causing the deterioration of the asset. Future research could focus on extracting information from temporal and individual sensor relationships (similar to spatial-temporal attention in the vision domain [236]) between inputs and RUL outputs. This would allow decision-makers to acquire more valuable insights into which parts or components present faulty behaviour and potentially improve prediction performance. Moreover, self-attention methods [220] can replace the need for recurrent connections in neural networks and pose an interesting line of future development in machine prognostics [146].

Chapter 4 extends the RUL prediction problem for assets under different operating conditions and failure modes, assuming labelled source domain data but no label information on the target domain. This setting has attracted much attention, and many deep learning approaches [64, 181] tackling this problem have appeared recently. One limitation of the results presented in this thesis is the lack of inclusion of more diverse degradation datasets. Thus, future work can explore applying the proposed methods in chapters 3 and 4 to real-world data in different applications of RUL prognostics. Moreover, one of our assumptions is that sensor information exists for the entire run-to-failure time of assets in the target domain. This assumption may be considered too strong, as newly installed assets may not possess access to complete cycles of run-to-failure data. As such, novel methods can take into account the adaptation considering limited run-to-failure cycles in the target domain and measure the adaptation performance depending on the amount of temporal data from the target domain, e.g., [130]. Other directions can consider that in some settings, one can take advantage of a few training examples with RUL labels in the target domain, for instance, coming from assets that failed sooner than expected. In such cases, a semi-supervised learning approach can take advantage of these few labelled examples of the target domain. Another interesting approach is incorporating the physical properties of the actual degradation into neural networks. This

research direction has the potential to yield more accurate prognostics and result in better adaptation performance [37].

In Chapter 5, we propose a model to learn improvement policies for routing problems. In our proposed method, we assume that we are in the canonical cases of each routing problem. In other words, we do not consider many of the possible constraints that may be imposed on such problems, for instance, time windows of nodes, maximum allowed travel time, among others. Recent works, following up on this line of research, have shown promising results in solving problems with more constraints [34], stochastic quantities (dynamic) [242] and other variations of routing problems not considered in this thesis [74]. Another area of further development is in methods that can be applied to instances of larger size. As seen in the methods presented in this thesis, end-to-end learning methods for combinatorial optimisation can be computationally expensive to train for large instances. Thus, methods that can cope with instances with a higher number of nodes can increase the applicability of the machine learning methods for such problems. Other areas of further development include methods that are more sample efficient, i.e., that take fewer samples of training data to converge to a good policy, methods that combine classical search heuristics with machine learning, and methods that can surpass the quality of specialised heuristics and traditional solvers, interesting approaches in these directions include [102, 127].

Chapter 6 builds upon one of the limitations of Chapter 5. That is, the method proposed in Chapter 5 may take significant training time before it converges to a good policy. This high sample complexity is undesirable as we wish to learn good policies under few samples and a limited number of online interactions with the problem's environment. Further improvements to the method proposed in Chapter 6, include learning from demonstration data completely offline without requiring interaction with an online environment. In this offline reinforcement learning setup, the goal is to use demonstrations to achieve optimal generalisation performance for unseen instances. This approach can be interesting when obtaining optimal tours (using a solver) and training via supervised learning [115, 160] becomes impractical. Moreover, under the proposed framework, the method could benefit from more diverse heuristics that better explore the space of possible solutions, including more diversity in the demonstration data. An interesting research direction is thus incorporating randomised algorithms as experts. These algorithms visit random regions of the search space and may allow learning methods to generalise better in the online phase. Other future directions include incorporating information of the expert policies whilst training online, for example, by allowing the online learner to query the expert (if not prohibitively expensive) or (previously learned) rewards-to-go estimates.

In Chapter 7, we define the DTMPA and seek policies that minimise maintenance costs. Several limitations of the currently presented approach are related to the assumptions about the asset network. For instance, in practical applications, a

network can be much larger than those studied in the experiments. Moreover, we only consider the presence of a single decision-maker responsible for travelling and maintenance. In more general scenarios, multiple decision-makers may be responsible for the maintenance of a network. Thus, a more general problem definition can include a repair crew instead of a single repairperson. Other extensions can include multiple assets or multiple components of the same asset present at a single location in the network. In the latter, the degradation of a component could depend on the degradation of other components of the same asset. Another limitation of the proposed method is that we learn policies for fixed network topologies. In reality, we could be interested in finding policies for different network structures under the assumption that they share the same machine types and similar degradation processes. Learning policies for the entire class of problems can be beneficial when decision-makers devise policies for multiple sub-networks with similar assets. Lastly, other recent approaches have also considered employing machine learning solutions for similar dynamic logistics problems that can potentially surpass the solutions of deterministic solvers. Noteworthy examples include [29, 131].

8.4 Discussion

The work presented in this thesis focuses on end-to-end algorithms to prediction and control problems, considering applications to asset prognostics and the combination of ML and CO. This section aims to pose the main results and contributions of this thesis in a broader context and relate them to other research topics.

The deep learning methods for asset prognostics presented in this thesis are related to general ML models for sequential data and prognostics and health management (PHM) applications. In general, sequential prediction problems reoccur in multiple domains such as natural language processing, financial markets, healthcare, among others. This also applies to learning tasks in which one assumption of supervised learning is violated, including the various types of violations such as different feature spaces, learning tasks, and marginal distributions from which the data is sampled. For example, the domain adaptation for sequential data can be easily transported to applications such as healthcare, aimed at adapting prediction models using temporal data of patients across different hospitals in which both the data distributions and feature space may differ [180].

The prognostics methods proposed here are also related to other PHM problems, such as fault detection, fault diagnostics, and other fault prediction methods. In particular, the methods proposed are closely related to the application of deep learning methods to each of these PHM problems [72]. Additionally, the field of assets prognostics has relationships with other disciplines. Many different modelling techniques can be applied to these problems depending on the available information about the systems. In general, other than ML methods, methods fall under the

categories of physics and statistical-based methods. These assume different types of information about the degradation of assets. In the first, the physical properties and equations governing the functioning and degradation of assets can be explicitly modelled. In the latter, there are assumptions about the data distributions typically estimated in a controlled environment or based on expert knowledge. Both assumptions require a different treatment than the models presented in this thesis but can be combined with ML methods to build models with stronger inductive bias [6].

With respect to ML and CO, the methods proposed for routing problems can also be adapted to other well-known routing problems defined over graphs. For example, the proposed methods in chapters 5 and 6 can be easily extended to tackle the orienteering problem [219] and other variations of the travelling salesman problem, such as the prize-collecting one [14]. The methodology can also be extended to consider problems with more constraints, with the extra care of masking infeasible moves to reduce exploration to the feasible space. Note that these problems occur in many application domains, such as drilling problems, order-picking in warehouses, among others [159]. Other applications of the proposed architecture include extending the edge swap operation to consider a dynamic choice of the number of edge swaps at each step, similar to [92]. It is also possible to incorporate other local search operators based on the nodes in the solution graph. Similarly, the proposed model framework in Chapter 7 can be expanded to a multi-agent and multi-component scenario with ease, whilst the proposed algorithm can be employed in other sequential decision-making problems, for example, to the ambulance dispatching problem [194, 209].

Furthermore, we point out that the methods and algorithms proposed in this thesis are just one of the many ways these two disciplines can be combined and benefit each other. For instance, a classical application of ML for CO problems involves acquiring a dataset and using ML models to learn one or more quantities that can later be used as inputs to mixed-integer formulations. Nonetheless, recent interest in the field has extended the combination of CO and ML in different directions. The methods proposed in this thesis make up another way of combining these two fields. Our proposed methods fall under the category of ML for solving CO problems. That is, algorithmic methods that perform learning on an implicit distribution of problem instances aiming to improve the search of solutions directly from experience. Combining ML in this manner allows for replacing heavy computations with fast approximations and sub-optimal algorithmic design decisions by decisions based on data [24]. Many methods fall under this category; for example, ML can be used to learn constraints for mixed integer programming formulations [222], objective functions describing the optimisation criteria [147] or implicit preferences that human planners have when solving optimisation problems [33]. ML methods can also be used in end-to-end learning methods, like the methods presented in this thesis, in which the learning module outputs solutions directly from the input space without requiring a search component [128]. Learning methods can

also be used in combination with classical CO and search algorithms. In this case, a high-level search algorithm can use an ML module to execute parts of its low-level decisions, such as selecting a heuristic to change a solution [39] or the next variable to branch [122, 192] in procedures such as branch-and-bound [136].

ML and CO can also be combined the other way around. CO methods can be used to replace usual optimisation approaches of ML methods to make sure that they are more stable and robust to constraints of the learning task. For instance, mixed integer programming can be employed to learn decision trees [221] and neural networks [109, 245] that are guaranteed to find the optimal training parameters given enough computation time. An important characteristic of this type of model is that it can be easily extended to respect one or several constraints of the problem. For example, these methods can ensure robustness against adversarial examples or minimise the occurrence of false positives. In general, this type of specific expert knowledge is much harder to include in classical ML and deep learning methods.

In summary, the methods proposed in this thesis sit in the much broader application domain of ML for decision-making in industrial applications.

8.5 Final Remarks

In this thesis, we have studied several problems in maintenance and logistics optimisation. We approach the problems studied in this thesis purely from a data-driven perspective. In particular, we focus on deep learning methods that are able to learn directly from the diverse input data in prediction problems in maintenance prognostics and control problems in routing and maintenance applications. We believe that the results and methods proposed in this thesis are attractive to those studying automatic learning methods to maintenance and logistics planning problems. We hope that the methods proposed can represent a small step towards utilising data-driven methods for more general industrial problems.

References

[1] F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou and N. Papakostantinou. 'The complexity of the travelling repairman problem'. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications* 20.1 (1986), pp. 79–87.

[2] C. Andriotis and K. Papakonstantinou. 'Deep reinforcement learning driven inspection and maintenance planning under incomplete information and constraints'. *Reliability Engineering & System Safety* 212 (2021), p. 107551.

[3] B. Angeniol, G.D.L.C. Vaubois and J.-Y. Le Texier. 'Self-organizing feature maps and the travelling salesman problem'. *Neural Networks* 1.4 (1988), pp. 289–293.

[4] D.L. Applegate, R.E. Bixby, V. Chvatal and W.J. Cook. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[5] D. Applegate, R. Bixby, V. Chvatal and W. Cook. 'Concorde TSP solver' (2006).

[6] M. Arias Chao, C. Kulkarni, K. Goebel and O. Fink. 'Hybrid deep fault detection and isolation: Combining deep neural networks and system performance models'. *International Journal of Prognostics and Health Management* 10 (2019), p. 033.

[7] S. Arora. 'Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems'. *Journal of the ACM (JACM)* 45.5 (1998), pp. 753–782.

[8] K. Arulkumaran, M.P. Deisenroth, M. Brundage and A.A. Bharath. 'Deep Reinforcement Learning: A Brief Survey'. *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[9] W. Aswolinskiy and B. Hammer. 'Unsupervised Transfer Learning for Time Series via Self-Predictive Modelling-First Results'. *Proceedings of the Workshop on New Challenges in Neural Computation (NC2)*. Vol. 3. 2017.

[10] V. Atamuradov, K. Medjaher, P. Dersin, B. Lamoureux and N. Zerhouni. 'Prognostics and health management for maintenance practitioners-Review, implementation and tools evaluation'. *International Journal of Prognostics and Health Management* 8.060 (2017), pp. 1–31.

[11] G.S. Babu, P. Zhao and X.-L. Li. 'Deep convolutional neural network based regression approach for estimation of remaining useful life'. *International Conference on Database Systems for Advanced Applications*. Springer. 2016, pp. 214–228.

[12] A.P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z.D. Guo and C. Blundell. 'Agent57: Outperforming the atari human benchmark'. *International Conference on Machine Learning*. PMLR. 2020, pp. 507–517.

[13] D. Bahdanau, K. Cho and Y. Bengio. 'Neural Machine Translation by Jointly Learning to Align and Translate'. *Proceedings of the 3rd International Conference on Learning Representations, (ICLR)*. 2015.

[14] E. Balas. 'The prize collecting traveling salesman problem'. *Networks* 19.6 (1989), pp. 621–636.

[15] R. Baltean-Lugojan, P. Bonami, R. Misener and A. Tramontani. 'Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks' (2019).

[16] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner et al. 'Relational inductive biases, deep learning, and graph networks'. *arXiv preprint arXiv:1806.01261* (2018).

[17] T. Bektas. 'The multiple traveling salesman problem: an overview of formulations and solution procedures'. *Omega* 34.3 (2006), pp. 209–219.

[18] M.G. Bellemare, W. Dabney and R. Munos. 'A distributional perspective on reinforcement learning'. *International Conference on Machine Learning*. PMLR. 2017, pp. 449–458.

[19] R. Bellman. 'A Markovian decision process'. *Journal of mathematics and mechanics* 6.5 (1957), pp. 679–684.

[20] R. Bellman. 'Dynamic programming treatment of the travelling salesman problem'. *Journal of the ACM (JACM)* 9.1 (1962), pp. 61–63.

[21] I. Bello and H. Pham. 'Neural Combinatorial Optimization with Reinforcement Learning'. *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. 2017.

[22] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira and J.W. Vaughan. 'A theory of learning from different domains'. *Machine learning* 79.1-2 (2010), pp. 151–175.

[23] S. Ben-David, J. Blitzer, K. Crammer, F. Pereira et al. 'Analysis of representations for domain adaptation'. *Advances in neural information processing systems* 19 (2007), p. 137.

[24] Y. Bengio, A. Lodi and A. Prouvost. 'Machine learning for combinatorial optimization: a methodological tour d'horizon'. *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[25] Y. Bengio, P. Simard and P. Frasconi. 'Learning long-term dependencies with gradient descent is difficult'. *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[26] T. Benkedjouh, K. Medjaher, N. Zerhouni and S. Rechak. 'Remaining useful life estimation based on nonlinear feature reduction and support vector regression'. *Engineering Applications of Artificial Intelligence* 26.7 (2013), pp. 1751–1760.

[27] D.P. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA, 2019.

[28] D. Bertsimas, G. Van Ryzin et al. 'The dynamic traveling repairman problem'. 1989.

[29] S. Bhattacharya, S. Badyal, T. Wheeler, S. Gil and D. Bertsekas. 'Reinforcement learning for pomdp: partitioned rollout and policy iteration with application to autonomous sequential repair problems'. *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 3967–3974.

[30] S.-P. Breton and G. Moe. 'Status, plans and technologies for offshore wind turbines in Europe and North America'. *Renewable Energy* 34.3 (2009), pp. 646–654.

[31] F. Camci. 'The travelling maintainer problem: integration of condition-based maintenance with the travelling salesman problem'. *Journal of the Operational Research Society* 65.9 (2014), pp. 1423–1436.

[32] F. Camci. 'Maintenance scheduling of geographically distributed assets with prognostics information'. *European Journal of Operational Research* 245.2 (2015), pp. 506–516.

[33] R. Canoy, V. Bucarey, J. Mandi and T. Guns. 'Learn-n-Route: Learning implicit preferences for vehicle routing'. *arXiv preprint arXiv:2101.03936* (2021).

[34] Y. Cao, Z. Sun and G. Sartoretti. 'DAN: Decentralized Attention-based Neural Network to Solve the MinMax Multiple Traveling Salesman Problem'. *arXiv preprint arXiv:2109.04205* (2021).

[35] Cartesius. *Cartesius Supercomputer*. URL: `https://www.surf.nl/en/dutch-national-supercomputer-cartesius`. (accessed: 08.05.2021).

[36] K.-W. Chang, A. Krishnamurthy, A. Agarwal, H. Daume and J. Langford. 'Learning to search better than your teacher'. *International Conference on Machine Learning*. PMLR. 2015, pp. 2058–2066.

[37] M.A. Chao, C. Kulkarni, K. Goebel and O. Fink. 'Fusing physics-based and deep learning models for prognostics'. *Reliability Engineering & System Safety* 217 (2022), p. 107961.

[38] J. Chen and W. Xu. 'Policy Gradient from Demonstration and Curiosity'. *arXiv preprint arXiv:2004.10430* (2020).

[39] X. Chen and Y. Tian. 'Learning to perform local rewriting for combinatorial optimization'. *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 2019, pp. 6281–6292.

[40] C. Cheng, X. Yan, N. Wagener and B. Boots. 'Fast Policy Learning through Imitation and Reinforcement'. *Uncertainty in artificial intelligence*. 2019.

[41] J. Cheng, L. Dong and M. Lapata. 'Long Short-Term Memory-Networks for Machine Reading'. *2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2016, pp. 551–561.

[42] K. Cho, B. van Merriënboer, D. Bahdanau and Y. Bengio. 'On the Properties of Neural Machine Translation: Encoder–Decoder Approaches'. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. 2014, pp. 103–111.

[43] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio. 'Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation'. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1724–1734.

[44] F. Chollet et al. *Keras*. 2015.

[45] M. Compare, L. Bellani, E. Cobelli and E. Zio. 'Reinforcement learning-based flow management of gas turbine parts under stochastic failures'. *The International Journal of Advanced Manufacturing Technology* 99.9-12 (2018), pp. 2981–2992.

[46] C. Cortes and M. Mohri. 'Domain adaptation and sample bias correction theory and algorithm for regression'. *Theoretical Computer Science* 519 (2014), pp. 103–126.

[47] A. Cubillo, S. Perinpanayagam and M. Esperon-Miguez. 'A review of physics-based models in prognostics: Application to gears and bearings of rotating machinery'. *Advances in Mechanical Engineering* 8.8 (2016).

[48] G. Cybenko. 'Approximation by superpositions of a sigmoidal function'. *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[49] P.R.d.O. da Costa, J. Rhuggenaath, Y. Zhang and A. Akcay. 'Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning'. *Asian Conference on Machine Learning*. PMLR. 2020, pp. 465–480.

[50] P.R.d.O. da Costa, A. Akcay, Y. Zhang and U. Kaymak. 'Attention and long short-term memory network for remaining useful lifetime predictions of turbofan engine degradation'. *International Journal of Prognostics and Health Management* 10 (2019), p. 034.

[51] P.R.d.O. da Costa, A. Akçay, Y. Zhang and U. Kaymak. 'Remaining useful lifetime prediction via deep domain adaptation'. *Reliability Engineering & System Safety* 195 (2020), p. 106682.

[52] P. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay and U. Kaymak. 'Learning 2-Opt Heuristics for Routing Problems via Deep Reinforcement Learning'. *Springer Nature Computer Science* 2.5 (July 2021), p. 388.

[53] P. da Costa, P. Verleijsdonk, S. Voorberg, A. Akcay, S. Kapodistria, W. van Jaarsveld and Y. Zhang. 'Policies for the Dynamic Traveling Maintainer Problem with Alerts'. *arXiv preprint arXiv:2105.15119* (2021).

[54] P. da Costa, Y. Zhang, A. Akcay and U. Kaymak. 'Learning 2-opt Local Search from Heuristics as Expert Demonstrations'. *2021 International Joint Conference on Neural Networks (IJCNN)*. 2021.

[55] W. Dabney, M. Rowland, M. Bellemare and R. Munos. 'Distributional reinforcement learning with quantile regression'. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[56] G.B. Dantzig and J.H. Ramser. 'The truck dispatching problem'. *Management science* 6.1 (1959), pp. 80–91.

[57] K. De Asis, J. Hernandez-Garcia, G. Holland and R. Sutton. 'Multi-step reinforcement learning: A unifying algorithm'. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[58] B. de Jonge, W. Klingenberg, R. Teunter and T. Tinga. 'Reducing costs by clustering maintenance activities for multiple critical units'. *Reliability Engineering & System Safety* 145 (2016), pp. 93–103.

[59] B. de Jonge and P.A. Scarf. 'A review on maintenance optimization'. *European Journal of Operational Research* 285.3 (2020), pp. 805–824.

[60] A. de Mathelin, G. Richard, M. Mougeot and N. Vayatis. 'Adversarial weighting for domain adaptation in regression'. *arXiv preprint arXiv:2006.08251* (2020).

[61] C. Derman. 'On optimal replacement rules when changes of state are Markovian'. *Mathematical optimization techniques* 396 (1963), pp. 201–210.

[62] M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak and L.-M. Rousseau. 'Learning heuristics for the tsp by policy gradient'. *Proceedings of the 15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*. 2018, pp. 170–181.

[63] P. Ding, M. Jia and X. Zhao. 'Meta deep learning based rotating machinery health prognostics toward few-shot prognostics'. *Applied Soft Computing* 104 (2021), p. 107211.

[64] Y. Ding, M. Jia, Q. Miao and P. Huang. 'Remaining useful life estimation using deep metric transfer learning for kernel regression'. *Reliability Engineering & System Safety* 212 (2021), p. 107583.

[65] C. Drent, M.O. Keizer and G.-J. van Houtum. 'Dynamic dispatching and repositioning policies for fast-response service networks'. *European Journal of Operational Research* 285.2 (2020), pp. 583–598.

[66] L. Duan, Y. Zhan, H. Hu, Y. Gong, J. Wei, X. Zhang and Y. Xu. 'Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach'. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 3054–3063.

[67] V.P. Dwivedi and X. Bresson. 'A generalization of transformer networks to graphs'. *arXiv preprint arXiv:2012.09699* (2020).

[68] P.G. Eibl, R. Mackenzie and D.B. Kidner. 'Vehicle routeing and scheduling in the brewing industry: a case study'. *International Journal of Physical Distribution & Logistics Management* (1994).

[69] B. Eksioglu, A.V. Vural and A. Reisman. 'The vehicle routing problem: A taxonomic review'. *Computers & Industrial Engineering* 57.4 (2009), pp. 1472–1483.

[70] S. Faghih-Roohi, S. Hajizadeh, A. Nunez, R. Babuska and B. De Schutter. 'Deep convolutional neural networks for detection of rail surface defects'. *Proceedings of the International Joint Conference on Neural Networks*. 2016, pp. 2584–2589.

[71] B. Fernando, A. Habrard, M. Sebban and T. Tuytelaars. 'Unsupervised visual domain adaptation using subspace alignment'. *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 2960–2967.

[72] O. Fink, E. Zio and U. Weidmann. 'Predicting component reliability and level of degradation with complex-valued neural networks'. *Reliability Engineering & System Safety* 121 (2014), pp. 198–206.

[73] A. Galassi, M. Lippi and P. Torroni. 'Attention, please! a critical review of neural attention models in natural language processing. arXiv 2019'. *arXiv preprint arXiv:1902.02181* ().

[74] R. Gama and H.L. Fernandes. 'A reinforcement learning approach to the orienteering problem with time windows'. *Computers & Operations Research* 133 (2021), p. 105357.

[75] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand and V. Lempitsky. 'Domain-adversarial training of neural networks'. *The Journal of Machine Learning Research* 17.1 (2016), pp. 2096–2030.

[76] M.R. Garey and D.S. Johnson. 'Computers and intractability'. *A Guide to the* (1979).

[77] M. Gasse, D. Chételat, N. Ferroni, L. Charlin and A. Lodi. 'Exact combinatorial optimization with graph convolutional neural networks'. *Advances in Neural Information Processing Systems*. 2019, pp. 15580–15592.

[78] X. Glorot and Y. Bengio. 'Understanding the difficulty of training deep feedforward neural networks'. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[79] X. Glorot, A. Bordes and Y. Bengio. 'Deep Sparse Rectifier Neural Networks'. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323.

[80] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[81] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio. 'Generative adversarial nets'. *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[82] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf and A.J. Smola. 'A kernel method for the two-sample-problem'. *Advances in neural information processing systems*. 2007, pp. 513–520.

[83] L. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2021.

[84] P. Hansen and N. Mladenović. 'First vs. best improvement: An empirical study'. *Discrete Applied Mathematics* 154.5 (2006), pp. 802–817.

[85] H. Hasselt. 'Double Q-learning'. *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel and A. Culotta. Vol. 23. Curran Associates, Inc., 2010.

[86] M. Hausknecht and P. Stone. 'Deep recurrent q-learning for partially observable mdps'. *2015 aaai fall symposium series*. 2015.

[87] M.J. Havinga and B. de Jonge. 'Condition-based maintenance in the cyclic patrolling repairman problem'. *International Journal of Production Economics* 222 (2020), p. 107497.

[88] D. He and E. Bechhoefer. 'Development and Validation of Bearing Diagnostic and Prognostic Tools using HUMS Condition Indicators'. *2008 IEEE Aerospace Conference*. 2008, pp. 1–8.

[89] F.O. Heimes. 'Recurrent neural networks for remaining useful life estimation'. *2008 International Conference on Prognostics and Health Management*. 2008, pp. 1–6.

[90] K. Helsgaun. 'An effective implementation of the Lin–Kernighan traveling salesman heuristic'. *European Journal of Operational Research* 126.1 (2000), pp. 106–130.

[91] K. Helsgaun. 'General k-opt submoves for the Lin–Kernighan TSP heuristic'. *Mathematical Programming Computation* 1.2-3 (2009), pp. 119–163.

[92] K. Helsgaun. 'An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems'. *Roskilde: Roskilde University* (2017).

[93] J.F. Hernandez-Garcia and R.S. Sutton. 'Understanding multi-step deep reinforcement learning: a systematic study of the DQN target'. *arXiv preprint arXiv:1901.07510* (2019).

[94] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar and D. Silver. 'Rainbow: Combining improvements in deep reinforcement learning'. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[95] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband et al. 'Deep q-learning from demonstrations'. *Thirty-second AAAI conference on artificial intelligence*. 2018.

[96] J. Ho and S. Ermon. 'Generative adversarial imitation learning'. *Advances in neural information processing systems*. 2016, pp. 4565–4573.

[97] S. Hochreiter and J. Schmidhuber. 'Long short-term memory'. *Neural computation* 9.8 (1997), pp. 1735–1780.

[98] Y. Hong, W.Q. Meeker, J.D. McCalley et al. 'Prediction of remaining life of power transformers based on left truncated and right censored lifetime data'. *The Annals of Applied Statistics* 3.2 (2009), pp. 857–879.

[99] J.J. Hopfield and D.W. Tank. 'Neural computation of decisions in optimization problems'. *Biological cybernetics* 52.3 (1985), pp. 141–152.

[100] K. Hornik, M. Stinchcombe and H. White. 'Multilayer feedforward networks are universal approximators'. *Neural networks* 2.5 (1989), pp. 359–366.

[101] M. Hossain, F. Sohel, M.F. Shiratuddin and H. Laga. 'A comprehensive survey of deep learning for image captioning'. *ACM Computing Surveys (CSUR)* 51.6 (2019), p. 118.

[102] A. Hottung and K. Tierney. 'Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem'. *ECAI 2020*. IOS Press, 2020, pp. 443–450.

[103] Q. Hu and W. Yue. *Markov decision processes with their applications*. Vol. 14. Springer Science & Business Media, 2007.

[104] Y. Hu, Y. Yao and W.S. Lee. 'A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs'. *Knowledge-Based Systems* 204 (2020), p. 106244.

[105] J. Huang, A. Gretton, K. Borgwardt, B. Schölkopf and A.J. Smola. 'Correcting sample selection bias by unlabeled data'. *Advances in neural information processing systems*. 2007, pp. 601–608.

[106] R. Huang, L. Xi, X. Li, C.R. Liu, H. Qiu and J. Lee. 'Residual life predictions for ball bearings based on self-organizing map and back propagation neural network methods'. *Mechanical Systems and Signal Processing* 21.1 (2007), pp. 193–207.

[107] P.J. Huber. 'Robust Estimation of a Location Parameter'. *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101.

[108] A. Hussein, M.M. Gaber, E. Elyan and C. Jayne. 'Imitation Learning: A Survey of Learning Methods'. *ACM Comput. Surv.* 50.2 (Apr. 2017).

[109] R.T. Icarte, L. Illanes, M.P. Castro, A.A. Cire, S.A. McIlraith and J.C. Beck. 'Training binarized neural networks using MIP and CP'. *International Conference on Principles and Practice of Constraint Programming*. Springer. 2019, pp. 401–417.

[110] M. Igl, L. Zintgraf, T.A. Le, F. Wood and S. Whiteson. 'Deep variational reinforcement learning for POMDPs'. *International Conference on Machine Learning*. PMLR. 2018, pp. 2117–2126.

[111] S. Ivanov and A. D'yakonov. 'Modern deep reinforcement learning algorithms'. *arXiv preprint arXiv:1906.10025* (2019).

[112] T. Jaakkola, M.I. Jordan and S.P. Singh. 'On the convergence of stochastic iterative dynamic programming algorithms'. *Neural computation* 6.6 (1994), pp. 1185–1201.

[113] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu et al. 'Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes'. *arXiv preprint arXiv:1807.11205* (2018).

[114] J. Jiang and C. Zhai. 'Instance weighting for domain adaptation in NLP'. *Proceedings of the 45th annual meeting of the association of computational linguistics*. 2007, pp. 264–271.

[115] C.K. Joshi, T. Laurent and X. Bresson. 'An efficient graph convolutional network technique for the travelling salesman problem'. *arXiv:1906.01227* (2019).

[116] L.P. Kaelbling, M.L. Littman and A.W. Moore. 'Reinforcement learning: A survey'. *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[117] Y. Kaempfer and L. Wolf. 'Learning the multiple traveling salesmen problem with permutation invariant pooling networks'. *arXiv preprint arXiv:1803.09621* (2018).

[118] S.M. Kakade. 'A natural policy gradient'. *Advances in neural information processing systems*. 2002, pp. 1531–1538.

[119] B. Kang, Z. Jie and J. Feng. 'Policy Optimization with Demonstrations'. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, July 2018, pp. 2469–2478.

[120] T. Kenbeek, S. Kapodistria and A. Di Bucchianico. 'Data-Driven Online Monitoring of Wind Turbines'. *Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools*. VALUETOOLS 2019. Palma, Spain: Association for Computing Machinery, 2019, pp. 143–150.

[121] E. Khalil, H. Dai, Y. Zhang, B. Dilkina and L. Song. 'Learning combinatorial optimization algorithms over graphs'. *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*. 2017, pp. 6348–6358.

[122] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser and B. Dilkina. 'Learning to branch in mixed integer programming'. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[123] D. Kifer, S. Ben-David and J. Gehrke. 'Detecting change in data streams'. *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 2004, pp. 180–191.

[124] D.P. Kingma and J. Ba. 'Adam: A Method for Stochastic Optimization'. *3rd International Conference on Learning Representations, ICLR 2015*. 2015.

[125] T.N. Kipf and M. Welling. 'Semi-Supervised Classification with Graph Convolutional Networks'. *Proceedings of the 5th International Conference on Learning Representations, (ICLR)*. 2017.

[126] J. Kober, J.A. Bagnell and J. Peters. 'Reinforcement learning in robotics: A survey'. *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[127] W. Kool, H. van Hoof, J. Gromicho and M. Welling. 'Deep Policy Dynamic Programming for Vehicle Routing Problems'. *arXiv preprint arXiv:2102.11756* (2021).

[128] W. Kool, H. van Hoof and M. Welling. 'Attention, Learn to Solve Routing Problems!' *Proceedings of the 7th International Conference on Learning Representations (ICLR)*. 2019.

[129] A. Krizhevsky, I. Sutskever and G.E. Hinton. 'Imagenet classification with deep convolutional neural networks'. *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[130] T. Krokotsch, M. Knaak and C. Gühmann. 'A Novel Evaluation Framework for Unsupervised Domain Adaption on Remaining Useful Lifetime Estimation'. *2020 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 2020, pp. 1–8.

[131] A. Kuhnle, J. Jakubik and G. Lanza. 'Reinforcement learning for opportunistic maintenance optimization'. *Production Engineering* 13.1 (2019), pp. 33–41.

[132] B.F. La Maire and V.M. Mladenov. 'Comparison of neural networks for solving the travelling salesman problem'. *11th Symposium on Neural Network Applications in Electrical Engineering*. IEEE. 2012, pp. 21–24.

[133] R. Lahyani, M. Khemakhem and F. Semet. 'Rich vehicle routing problems: From a taxonomy to a definition'. *European Journal of Operational Research* 241.1 (2015), pp. 1–14.

[134] G. Laporte and I.H. Osman. 'Routing problems: A bibliography'. *Annals of operations research* 61.1 (1995), pp. 227–262.

[135] D. Laredo, Z. Chen, O. Schütze and J.-Q. Sun. 'A neural network-evolutionary computational framework for remaining useful life estimation of mechanical systems'. *Neural networks* 116 (2019), pp. 178–187.

[136] E.L. Lawler and D.E. Wood. 'Branch-and-bound methods: A survey'. *Operations research* 14.4 (1966), pp. 699–719.

[137] Y. Lei, N. Li, L. Guo, N. Li, T. Yan and J. Lin. 'Machinery health prognostics: A systematic review from data acquisition to RUL prediction'. *Mechanical Systems and Signal Processing* 104 (2018), pp. 799–834.

[138] J.K. Lenstra and A.R. Kan. 'Some simple applications of the travelling salesman problem'. *Journal of the Operational Research Society* 26.4 (1975), pp. 717–733.

[139] S. Levine, A. Kumar, G. Tucker and J. Fu. 'Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems'. *arXiv e-prints* (2020), arXiv–2005.

[140] X. Li, Q. Ding and J.Q. Sun. 'Remaining useful life estimation in prognostics using deep convolution neural networks'. *Reliability Engineering and System Safety* 172 (2018), pp. 1–11.

[141] X. Li, W. Zhang and Q. Ding. 'Cross-domain fault diagnosis of rolling element bearings using deep generative neural networks'. *IEEE Transactions on Industrial Electronics* (2018).

[142] X. Li, W. Zhang, Q. Ding and J.-Q. Sun. 'Multi-Layer domain adaptation method for rolling bearing fault diagnosis'. *Signal Processing* 157 (2019), pp. 180–197.

[143] S. Lin and B.W. Kernighan. 'An effective heuristic algorithm for the traveling-salesman problem'. *Operations research* 21.2 (1973), pp. 498–516.

[144] A. Listou Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov and H. Zhang. 'Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture'. *Reliability Engineering & System Safety* 183 (2019), pp. 240–251.

[145] Y. Liu, Y. Chen and T. Jiang. 'Dynamic selective maintenance optimization for multi-state systems over a finite horizon: A deep reinforcement learning approach'. *European Journal of Operational Research* 283.1 (2020), pp. 166–181.

[146] Y. Liu and X. Wang. 'Deep & Attention: A Self-Attention based Neural Network for Remaining Useful Lifetime Predictions'. *2021 7th International Conference on Mechatronics and Robotics Engineering (ICMRE)*. IEEE. 2021, pp. 98–105.

[147] M. Lombardi and M. Milano. 'Boosting combinatorial problem modeling with machine learning'. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2018, pp. 5472–5478.

[148] M. Long, Y. Cao, J. Wang and M.I. Jordan. 'Learning transferable features with deep adaptation networks'. *arXiv preprint arXiv:1502.02791* (2015).

[149] M. Long, J. Wang, G. Ding, J. Sun and P.S. Yu. 'Transfer feature learning with joint distribution adaptation'. *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 2200–2207.

[150] D. Lopez-Paz, J.M. Hernández-lobato and B. Schölkopf. 'Semi-supervised domain adaptation with non-parametric copulas'. *Advances in neural information processing systems*. 2012, pp. 665–673.

[151] H. Lu, X. Zhang and S. Yang. 'A learning-based iterative method for solving vehicle routing problems'. *International Conference on Learning Representations*. 2019.

[152] W. Lu, B. Liang, Y. Cheng, D. Meng, J. Yang and T. Zhang. 'Deep Model Based Domain Adaptation for Fault Diagnosis'. *IEEE Transactions on Industrial Electronics* 64.3 (Mar. 2017), pp. 2296–2305.

[153] N.C. Luong, D.T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang and D.I. Kim. 'Applications of deep reinforcement learning in communications and networking: A survey'. *IEEE Communications Surveys & Tutorials* 21.4 (2019), pp. 3133–3174.

[154] T. Luong, H. Pham and C.D. Manning. 'Effective Approaches to Attention-based Neural Machine Translation'. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Sept. 2015, pp. 1412–1421.

[155] Q. Ma, S. Ge, D. He, D. Thaker and I. Drori. 'Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning'. *AAAI Workshop on Deep Learning on Graphs: Methodologies and Applications*. 2020.

[156] Y. Mansour, M. Mohri and A. Rostamizadeh. 'Domain adaptation: Learning bounds and algorithms'. *22nd Conference on Learning Theory, COLT 2009*. 2009.

[157] A. Marcos Alvarez, L. Wehenkel and Q. Louveaux. 'Online learning for strong branching approximation in branch-and-bound' (2016).

[158] Martıén Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[159] R. Matai, S.P. Singh and M.L. Mittal. 'Traveling salesman problem: an overview of applications, formulations, and solution approaches'. *Traveling salesman problem, theory and applications* 1 (2010).

[160] U.J. Mele, L.M. Gambardella and R. Montemanni. 'A New Constructive Heuristic Driven by Machine Learning for the Traveling Salesman Problem'. *Algorithms* 14.9 (2021), p. 267.

[161] T.M. Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, 1980.

[162] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu. 'Asynchronous methods for deep reinforcement learning'. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*. 2016, pp. 1928–1937.

[163] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski et al. 'Human-level control through deep reinforcement learning'. *Nature* 518.7540 (2015), pp. 529–533.

[164] M. Mohri and A.M. Medina. 'New analysis and algorithm for learning with drifting distributions'. *International Conference on Algorithmic Learning Theory*. Springer. 2012, pp. 124–138.

[165] G.F. Montúfar, R. Pascanu, K. Cho and Y. Bengio. 'On the Number of Linear Regions of Deep Neural Networks'. *NIPS*. 2014.

[166] K.P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[167] *NASA - Prognostics Center of Excellence*. `https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/`. Accessed: 2018-03-01.

[168] M. Nazari, A. Oroojlooy, L.V. Snyder and M. Takác. 'Reinforcement Learning for Solving the Vehicle Routing Problem'. *NeurIPS*. 2018.

[169] R. Nikzad-Langerodi, W. Zellinger, E. Lughofer and S. Saminger-Platz. 'Domain-Invariant Partial-Least-Squares Regression'. *Analytical chemistry* 90.11 (2018), pp. 6693–6701.

[170] C. Olah. *Understanding LSTM Networks*. 2015.

[171] M.C. Olde Keizer, S.D.P. Flapper and R.H. Teunter. 'Condition-based maintenance policies for systems with multiple dependent components: A review'. *European Journal of Operational Research* 261.2 (2017), pp. 405–420.

[172] T. Osa, J. Pajarinen, G. Neumann, J.A. Bagnell, P. Abbeel, J. Peters et al. 'An algorithmic perspective on imitation learning'. *Foundations and Trends in Robotics* 7.1-2 (2018), pp. 1–179.

[173] S.J. Pan, I.W. Tsang, J.T. Kwok and Q. Yang. 'Domain adaptation via transfer component analysis'. *IEEE Transactions on Neural Networks* 22.2 (2011), pp. 199–210.

[174] S.J. Pan and Q. Yang. 'A Survey on Transfer Learning'. *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359.

[175] C.H. Papadimitriou. 'The Euclidean travelling salesman problem is NP-complete'. *Theoretical Computer Science* 4.3 (1977), pp. 237–244.

[176] N. Papakostas, P. Papachatzakis, V. Xanthakis, D. Mourtzis and G. Chryssolouris. 'An approach to operational aircraft maintenance planning'. *Decision Support Systems* 48.4 (2010), pp. 604–612.

[177] B. Peng, J. Wang and Z. Zhang. 'A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems'. *International Symposium on Intelligence Computation and Applications*. Springer. 2019, pp. 636–650.

[178] L. Perron and V. Furnon. *OR-Tools*. Version 7.2. Google, 2019.

[179] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda and Q. Liao. 'Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review'. *International Journal of Automation and Computing* 14.5 (2017), pp. 503–519.

[180] S. Purushotham, W. Carvalho, T. Nilanon and Y. Liu. 'Variational Adversarial Deep Domain Adaptation for Health Care Time Series Analysis'. *29th Conference on Neural Information Processing Systems* Nips (2016).

[181] M. Ragab, Z. Chen, M. Wu, C.S. Foo, C.K. Kwoh, R. Yan and X. Li. 'Contrastive adversarial domain adaptation for machine remaining useful life prediction'. *IEEE Transactions on Industrial Informatics* 17.8 (2020), pp. 5239–5249.

[182] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli and J. Sohl-Dickstein. 'On the expressive power of deep neural networks'. *International Conference on Machine Learning*. PMLR. 2017, pp. 2847–2854.

[183] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov and S. Levine. 'Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations'. *Robotics: Science and Systems XIV* (June 2018).

[184] G. Reinelt. 'TSPLIB—A traveling salesman problem library'. *ORSA journal on computing* 3.4 (1991), pp. 376–384.

[185] F. Rosenblatt. 'The perceptron: a probabilistic model for information storage and organization in the brain.' *Psychological review* 65.6 (1958), p. 386.

[186] S. Ross and D. Bagnell. 'Efficient reductions for imitation learning'. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 661–668.

[187] S. Ross, G. Gordon and D. Bagnell. 'A reduction of imitation learning and structured prediction to no-regret online learning'. *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.

[188] S. Ruder. 'An overview of gradient descent optimization algorithms'. *arXiv preprint arXiv:1609.04747* (2016).

[189] D.E. Rumelhart, G.E. Hinton and R.J. Williams. 'Learning representations by back-propagating errors'. *nature* 323.6088 (1986), pp. 533–536.

[190] G.A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. Citeseer, 1994.

[191] A. Saxena, K. Goebel, D. Simon and N. Eklund. 'Damage propagation modeling for aircraft engine run-to-failure simulation'. *2008 International Conference on Prognostics and Health Management, PHM 2008*. 2008, pp. 1–9.

[192] L. Scavuzzo Montana. 'Learning variable selection rules for the branch-and-bound algorithm using reinforcement learning'. MA thesis. 2020.

[193] T. Schaul, J. Quan, I. Antonoglou and D. Silver. 'Prioritized Experience Replay'. *International Conference on Learning Representations, ICLR*. 2016.

[194] V. Schmid. 'Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming'. *European journal of operational research* 219.3 (2012), pp. 611–621.

[195] J. Schulman, S. Levine, P. Abbeel, M. Jordan and P. Moritz. 'Trust region policy optimization'. *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[196] J. Schulman, P. Moritz, S. Levine, M.I. Jordan and P. Abbeel. 'High-Dimensional Continuous Control Using Generalized Advantage Estimation'. *International Conference on Learning Representations, ICLR*. 2016.

[197] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov. 'Proximal policy optimization algorithms'. *arXiv preprint arXiv:1707.06347* (2017).

[198] D. Sejdinovic, B. Sriperumbudur, A. Gretton and K. Fukumizu. 'Equivalence of distance-based and RKHS-based statistics in hypothesis testing'. *The Annals of Statistics* (2013), pp. 2263–2291.

[199] X.S. Si, W. Wang, C.H. Hu and D.H. Zhou. 'Remaining useful life estimation - A review on the statistical data driven approaches'. *European Journal of Operational Research* 213.1 (2011), pp. 1–14.

[200] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al. 'Mastering the game of Go with deep neural networks and tree search'. *nature* 529.7587 (2016), pp. 484–489.

[201] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. 'Dropout: a simple way to prevent neural networks from overfitting'. *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[202] B. Sun, J. Feng and K. Saenko. 'Return of frustratingly easy domain adaptation.' *AAAI*. Vol. 6. 2016, p. 8.

[203] B. Sun and K. Saenko. 'Deep coral: Correlation alignment for deep domain adaptation'. *European Conference on Computer Vision*. Springer. 2016, pp. 443–450.

[204] W. Sun, J.A. Bagnell and B. Boots. 'Truncated horizon Policy Search: Combining Reinforcement Learning & Imitation Learning'. *International Conference on Learning Representations, ICLR*. 2018.

[205] W. Sun, A. Venkatraman, G.J. Gordon, B. Boots and J.A. Bagnell. 'Deeply aggrevated: Differentiable imitation learning for sequential prediction'. *International Conference on Machine Learning*. PMLR. 2017, pp. 3309–3318.

[206] I. Sutskever, O. Vinyals and Q.V. Le. 'Sequence to sequence learning with neural networks'. *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[207] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[208] R.S. Sutton, D.A. McAllester, S.P. Singh and Y. Mansour. 'Policy gradient methods for reinforcement learning with function approximation'. *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[209] N. Theeuwes, G.-J.v. Houtum and Y. Zhang. 'Improving ambulance dispatching with machine learning and simulation'. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2021, pp. 302–318.

[210] Z. Tian. 'An artificial neural network method for remaining useful life prediction of equipment subject to condition monitoring'. *Journal of Intelligent Manufacturing* 23.2 (2012), pp. 227–237.

[211] T. Tieleman and G. Hinton. 'Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude'. *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[212] P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[213] T. Tulabandhula, C. Rudin and P. Jaillet. 'The Machine Learning and Traveling Repairman Problem'. *Algorithmic Decision Theory*. Ed. by R.I. Brafman, F.S. Roberts and A. Tsoukiàs. 2011, pp. 262–276.

[214] E. Tzeng, J. Hoffman, K. Saenko and T. Darrell. 'Adversarial discriminative domain adaptation'. *Computer Vision and Pattern Recognition (CVPR)*. Vol. 1. 2017, p. 4.

[215] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko and T. Darrell. 'Deep domain confusion: Maximizing for domain invariance'. *arXiv preprint arXiv:1412.3474* (2014).

[216] H. Van Hasselt, A. Guez and D. Silver. 'Deep reinforcement learning with double q-learning'. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[217] P.J. Van Laarhoven and E.H. Aarts. 'Simulated annealing'. *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[218] H.E. van Staden and R.N. Boute. 'The effect of multi-sensor data on condition-based maintenance policies'. *European Journal of Operational Research* (2020).

[219] P. Vansteenwegen, W. Souffriau and D. Van Oudheusden. 'The orienteering problem: A survey'. *European Journal of Operational Research* 209.1 (2011), pp. 1–10.

[220] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser and I. Polosukhin. 'Attention is all you need'. *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*. 2017, pp. 5998–6008.

[221] S. Verwer and Y. Zhang. 'Learning optimal classification trees using a binary linear program formulation'. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 1625–1632.

[222] S. Verwer, Y. Zhang and Q.C. Ye. 'Auction optimization using regression trees and linear models as integer programs'. *Artificial Intelligence* 244 (2017), pp. 368–395.

[223] O. Vinyals, M. Fortunato and N. Jaitly. 'Pointer networks'. *Proceedings of the 29th Conference on Neural Information Processing Systems (NIPS)*. 2015, pp. 2692–2700.

[224] W. Wang. 'An overview of the recent advances in delay-time-based maintenance modelling'. *Reliability Engineering & System Safety* 106 (2012), pp. 165–178.

[225] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot and N. Freitas. 'Dueling network architectures for deep reinforcement learning'. *International conference on machine learning*. PMLR. 2016, pp. 1995–2003.

[226] C.J. Watkins and P. Dayan. 'Q-learning'. *Machine learning* 8.3-4 (1992), pp. 279–292.

[227] K. Weiss, T.M. Khoshgoftaar and D. Wang. 'A survey of transfer learning'. *Journal of Big data* 3.1 (2016), pp. 1–40.

[228] D.J. White. 'A survey of applications of Markov decision processes'. *Journal of the operational research society* 44.11 (1993), pp. 1073–1096.

[229] R.J. Williams. 'Simple statistical gradient-following algorithms for connectionist reinforcement learning'. *Machine learning* 8.3-4 (1992), pp. 229–256.

[230] R.J. Williams and D. Zipser. 'A learning algorithm for continually running fully recurrent neural networks'. *Neural computation* 1.2 (1989), pp. 270–280.

[231] Y. Wu, W. Song, Z. Cao, J. Zhang and A. Lim. 'Learning improvement heuristics for solving routing problem'. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[232] Y. Wu, M. Yuan, S. Dong, L. Lin and Y. Liu. 'Remaining useful life estimation of engineered systems using vanilla LSTM neural networks'. *Neurocomputing* 275 (2018), pp. 167–179.

[233] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and S.Y. Philip. 'A comprehensive survey on graph neural networks'. *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[234] J. Xie, L. Zhang, L. Duan and J. Wang. 'On cross-domain feature fusion in gearbox fault diagnosis under various operating conditions based on Transfer Component Analysis'. *2016 IEEE International Conference on Prognostics and Health Management (ICPHM)*. June 2016, pp. 1–6.

[235] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel and Y. Bengio. 'Show, attend and tell: Neural image caption generation with visual attention'. *International Conference on Machine Learning*. 2015, pp. 2048–2057.

[236] C. Yan, Y. Tu, X. Wang, Y. Zhang, X. Hao, Y. Zhang and Q. Dai. 'STAT: Spatial-temporal attention mechanism for video captioning'. *IEEE transactions on multimedia* 22.1 (2019), pp. 229–241.

[237] M. Yuan, Y. Wu and L. Lin. 'Fault diagnosis and remaining useful life estimation of aero engine using LSTM neural network'. *2016 IEEE International Conference on Aircraft Utility Systems (AUS)*. 2016, pp. 135–140.

[238] A. Zhang, H. Wang, S. Li, Y. Cui, Z. Liu, G. Yang and J. Hu. 'Transfer Learning with Deep Recurrent Neural Networks for Remaining Useful Life Estimation'. *Applied Sciences* 8.12 (2018), p. 2416.

[239] A. Zhang, Z.C. Lipton, M. Li and A.J. Smola. 'Dive into Deep Learning'. *arXiv preprint arXiv:2106.11342* (2021).

[240] C. Zhang, P. Lim, A. Qin and K.C. Tan. 'Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics'. *IEEE Transactions on Neural Networks and Learning systems* 28.10 (2017), pp. 2306–2318.

[241] W. Zhang, G. Peng, C. Li, Y. Chen and Z. Zhang. 'A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals'. *Sensors* 17.2 (2017), p. 425.

[242] Z. Zhang, H. Liu, M. Zhou and J. Wang. 'Solving Dynamic Traveling Salesman Problems With Deep Reinforcement Learning'. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[243] S. Zheng, K. Ristovski, A. Farahat and C. Gupta. 'Long Short-Term Memory Network for Remaining Useful Life estimation'. *2017 IEEE International Conference on Prognostics and Health Management, ICPHM 2017*. 2017, pp. 88–95.

[244] E. Zio and F. Di Maio. 'A data-driven fuzzy approach for predicting the remaining useful life in dynamic failure scenarios of a nuclear system'. *Reliability Engineering & System Safety* 95.1 (2010), pp. 49–57.

[245] T. Þorbjarnarson and N. Yorke-Smith. 'On Training Neural Networks with Mixed Integer Programming'. *IJCAI-PRICAI'20 Workshop on Data Science Meets Optimisation*. 2021.

Summary

The increasingly available data provided by connected devices has created the demand to efficiently store, process, and analyse this data. Due to algorithmic advances and improved computing power, data-driven methods based on deep learning have been leveraged to automatically analyse and interpret patterns enabling improved decision-making directly from gathered data without human intervention. When solving real-world decision-making and optimisation problems, these advances can improve the accuracy, efficiency and effectiveness of previously handcrafted solutions. Thus, to automate the costly and time-intensive design of tailored solutions for prediction and optimisation problems, this dissertation studies deep learning algorithms to learn predictions and solution strategies in the context of maintenance and logistics problems.

Within this context, we consider the availability of data sources, including geographical locations, remote monitoring sensors and personnel availability, and study relevant problems pertaining to a network of industrial assets. First, motivated by prognostics problems when devising maintenance plans, we study the remaining useful lifetime (RUL) prediction of assets directly from remote monitoring device readings, including scenarios in which previously labelled run-to-failure information is not available. Second, we focus on learning strategies for solving transportation problems that arise when locations in a network need to be visited, ensuring minimum visitation costs. In particular, we study well-known routing problems such as the travelling salesman problem and the vehicle routing problem. Third, we combine the first two streams of this dissertation and study a joint transportation and maintenance problem in which a decision-maker faces uncertain RUL predictions in the form of alerts. In this problem, the objective is to minimise preventive and corrective maintenance costs while serving a network of assets at multiple geographical locations.

This dissertation explores methods and problems common to machine learning and operational research. We propose data-driven methods based on the literature on deep learning, reinforcement learning, stochastic processes, and Markov decision processes to address the problems above. Our main contribution relates to novel algorithmic methods that leverage only the available data to learn better

predictions and policies that decision-makers can use in maintenance and routing problems. In particular, we study deep neural network architectures capable of extracting information from sequential data to arrive at accurate RUL predictions. For routing problems, we leverage both the graph structure and sequential information to learn policies via deep reinforcement learning that automatically learn to search for near-optimal solutions. In the combined maintenance and travelling problems, we take as input the information of the locations and uncertain RUL predictions to learn policies capable of surpassing high-quality heuristics and approaching optimal solutions that have complete information about the degradation of assets.

About the Author

Paulo Roberto de Oliveira da Costa was born on the 20th of August, 1987, in Belém, PA, Brazil. After finishing his secondary education at the Primary and Secondary School Tenênte Rêgo Barros in Belém, he started his tertiary education in Computational and Applied Mathematics at the University of Campinas (UNICAMP) in Campinas, SP, Brazil. After obtaining his Bachelor's degree in 2010, he worked for several years in the financial sector in São Paulo, SP, Brazil. Later, in 2015 he started the Master of Science in Business Analytics at University College Dublin (UCD) in Dublin, Ireland, where he obtained his degree with First Class honours (1.1).

Paulo joined the Information Systems (IS) group of the Department of Industrial Engineering and Innovation Sciences at the Eindhoven University of Technology (TU/e) in 2018 as a PhD student under the supervision of prof.dr.ir. Uzay Kaymak, dr. Yingqian Zhang and dr. Alp Akçay. His research interests include machine learning, deep learning, reinforcement learning and combinatorial optimisation. In his research, he developed models and algorithms for maintenance and optimisation problems detailed in this thesis. His research has been carried out as part of the Real-time data-driven maintenance logistics project supported by the Dutch Research Council (NWO) and under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems. In addition, Paulo has taken part in the Graduate Program Operations Management & Logistics (GP-OML) and the Dutch Network on Mathematics and Operations Research (LNMB) teaching programmes. During his PhD, Paulo has been involved in several teaching activities in the disciplines of machine learning, deep learning and algorithmic programming, offered by the IS group at TU/e. Paulo has also taken part in the review of papers for ML venues, organised an international competition hosted at the International Joint Conference on Artificial Intelligence (IJCAI-21), supervised Bachelor's end projects and assisted the supervision of Master's theses during his PhD term. Additionally, Paulo has collaborated with the companies participating in his project, namely, the Dutch Railways (NS), Fokker Services and Philips.

SIKS Dissertation Series

2011 01 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models

02 Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

03 Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems

04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference

05 Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

06 Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage

07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction

08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues

09 Tim de Jong (OU), Contextualised Mobile Media for Learning

10 Bart Bogaert (UvT), Cloud Content Contention

11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective

12 Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining

13 Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling

14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets

15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval

16 Maarten Schadd (UM), Selective Search in Games of Different Complexity

17 Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness

18 Mark Ponsen (UM), Strategic Decision-Making in complex games

19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles

20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach
21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems
22 Junte Zhang (UVA), System Evaluation of Archival Description and Access
23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media
24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior
25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics
26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns
28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure
29 Faisal Kamiran (TUE), Discrimination-aware Classification
30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions
31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science
33 Tom van der Weide (UU), Arguing to Motivate Decisions
34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training
36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach
37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization
39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games
40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development
41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control

42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution
43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge
44 Boris Reuderink (UT), Robust Brain-Computer Interfaces
45 Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection
46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work
47 Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression
48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent
49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012 01 Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda
02 Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models
03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories
04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications
05 Marijn Plomp (UU), Maturing Interorganisational Information Systems
06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks
07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions
08 Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories
09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms
10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment
11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems
13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems

15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.
16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment
17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance
18 Eltjo Poort (VU), Improving Solution Architecting Practices
19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution
20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval
22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval
25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
26 Emile de Maat (UVA), Making Sense of Legal Text
27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
28 Nancy Pascall (UvT), Engendering Technology Empowering Women
29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval
30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making
31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning
33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)
34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications
35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics
36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes

37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation
38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms
39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks
40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia
41 Sebastian Kelle (OU), Game Design Patterns for Learning
42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning
43 Withdrawn
44 Anna Tordai (VU), On Combining Alignment Techniques
45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions
46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior
48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data
49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering
51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013 01 Viorel Milea (EUR), News Analytics for Financial Decision Support
02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics
04 Chetan Yadati (TUD), Coordinating autonomous planning and scheduling
05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns
06 Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
07 Giel van Lankveld (UvT), Quantifying Individual Player Differences
08 Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications

10 Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.
11 Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services
12 Marian Razavian (VU), Knowledge-driven Migration to Services
13 Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
14 Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning
15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications
16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation
17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid
18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification
19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling
20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval
21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation
22 Tom Claassen (RUN), Causal Discovery and Logic
23 Patricio de Alencar Silva (UvT), Value Activity Monitoring
24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning
25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System
26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning
27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance
28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience
29 Iwan de Kok (UT), Listening Heads
30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support
31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications

32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development
33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere
34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search
35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction
36 Than Lam Hoang (TUe), Pattern Mining in Data Streams
37 Dirk Börner (OUN), Ambient Learning Displays
38 Eelco den Heijer (VU), Autonomous Evolutionary Art
39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems
40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games
41 Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
42 Léon Planken (TUD), Algorithms for Simple Temporal Reasoning
43 Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts

2014 01 Nicola Barile (UU), Studies in Learning Monotone Models from Data
02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method
03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions
04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability
06 Damian Tamburri (VU), Supporting Networked Software Development
07 Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior
08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints
09 Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
10 Ivan Salvador Razo Zapata (VU), Service Value Networks
11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support
12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control
13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

14 Yangyang Shi (TUD), Language Models With Meta-information
15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare
16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria
17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations
19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link
21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments
22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training
23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era
24 Davide Ceolin (VU), Trusting Semi-structured Web Data
25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction
26 Tim Baarslag (TUD), What to Bid and When to Stop
27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
28 Anna Chmielowiec (VU), Decentralized k-Clique Matching
29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software
30 Peter de Cock (UvT), Anticipating Criminal Behaviour
31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support
32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data
33 Tesfa Tegegne (RUN), Service Discovery in eHealth
34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.
35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach
36 Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models
37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying

38 Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.
39 Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital
40 Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education
41 Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text
42 Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models
43 Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments
44 Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.
45 Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach
46 Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity
47 Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval

2015 01 Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response
02 Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls
03 Twan van Laarhoven (RUN), Machine learning for network data
04 Howard Spoelstra (OUN), Collaborations in Open Learning Environments
05 Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding
06 Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
07 Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis
08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems
10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning
11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins

12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13 Giuseppe Procaccianti (VU), Energy-Efficient Software
14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations
15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation
16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork
17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners
20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination
21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning
22 Zhemin Zhu (UT), Co-occurrence Rate Networks
23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage
24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection
26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure
27 Sándor Héman (CWI), Updating compressed colomn stores
28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO
29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning
31 Yakup Koç (TUD), On the robustness of Power Grids
32 Jerome Gard (UL), Corporate Venture Management in SMEs
33 Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources
34 Victor de Graaf (UT), Gesocial Recommender Systems
35 Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines

02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VU), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UVA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web
19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UVA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UVA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior

26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (UvT), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis

	50	Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains
2017	01	Jan-Jaap Oerlemans (UL), Investigating Cybercrime
	02	Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
	03	Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
	04	Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
	05	Mahdieh Shadi (UVA), Collaboration Behavior
	06	Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
	07	Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
	08	Rob Konijn (VU) , Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
	09	Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
	10	Robby van Delden (UT), (Steering) Interactive Play Behavior
	11	Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
	12	Sander Leemans (TUE), Robust Process Mining with Guarantees
	13	Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
	14	Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
	15	Peter Berck (RUN), Memory-Based Text Correction
	16	Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines
	17	Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
	18	Ridho Reinanda (UVA), Entity Associations for Search
	19	Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
	20	Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
	21	Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
	22	Sara Magliacane (VU), Logics for causal inference under uncertainty
	23	David Graus (UVA), Entities of Interest — Discovery in Digital Traces
	24	Chang Wang (TUD), Use of Affordances for Efficient Robot Learning

25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VU), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (UvT), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (UvT), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VU), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration

	48	Angel Suarez (OU), Collaborative inquiry-based learning
2018	01	Han van der Aa (VUA), Comparing and Aligning Process Representations
	02	Felix Mannhardt (TUE), Multi-perspective Process Mining
	03	Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
	04	Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
	05	Hugo Huurdeman (UVA), Supporting the Complex Dynamics of the Information Seeking Process
	06	Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
	07	Jieting Luo (UU), A formal account of opportunism in multi-agent systems
	08	Rick Smetsers (RUN), Advances in Model Learning for Software Systems
	09	Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
	10	Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
	11	Mahdi Sargolzaei (UVA), Enabling Framework for Service-oriented Collaborative Networks
	12	Xixi Lu (TUE), Using behavioral context in process mining
	13	Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
	14	Bart Joosten (UVT), Detecting Social Signals with Spatiotemporal Gabor Filters
	15	Naser Davarzani (UM), Biomarker discovery in heart failure
	16	Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
	17	Jianpeng Zhang (TUE), On Graph Sample Clustering
	18	Henriette Nakad (UL), De Notaris en Private Rechtspraak
	19	Minh Duc Pham (VUA), Emergent relational schemas for RDF
	20	Manxia Liu (RUN), Time and Bayesian Networks
	21	Aad Slootmaker (OUN), EMERGO: a generic platform for authoring and playing scenario-based serious games
	22	Eric Fernandes de Mello Araujo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
	23	Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
	24	Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
	25	Riste Gligorov (VUA), Serious Games in Audio-Visual Collections

	26	Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
	27	Maikel Leemans (TUE), Hierarchical Process Mining for Scalable Software Analysis
	28	Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
	29	Yu Gu (UVT), Emotion Recognition from Mandarin Speech
	30	Wouter Beek, The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
2019	01	Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
	02	Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
	03	Eduardo Gonzalez Lopez de Murillas (TUE), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
	04	Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
	05	Sebastiaan van Zelst (TUE), Process Mining with Streaming Data
	06	Chris Dijkshoorn (VU), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
	07	Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
	08	Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
	09	Fahimeh Alizadeh Moghaddam (UVA), Self-adaptation for energy efficiency in software systems
	10	Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
	11	Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
	12	Jacqueline Heinerman (VU), Better Together
	13	Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
	14	Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
	15	Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
	16	Guangming Li (TUE), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models

17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TUE), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VU),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VU), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VU), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OUN), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VU), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VU), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TUE), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TUE), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OUN), Gamification with digital badges in learning programming
36 Kevin Ackermans (OUN), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OUN), Learning visually grounded and multilingual representations

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour

02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TUE), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OUN), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TUE), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
12 Ward van Breda (VU), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OUN), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OUN), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VU), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VU), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging

	24	Lenin da Nobrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
	25	Xin Du (TUE), The Uncertainty in Exceptional Model Mining
	26	Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
	27	Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
	28	Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
	29	Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
	30	Bob Zadok Blok (UL), Creatief, Creatieve, Creatiefst
	31	Gongjin Lan (VU), Learning better – From Baby to Better
	32	Jason Rhuggenaath (TUE), Revenue management in online markets: pricing and online advertising
	33	Rick Gilsing (TUE), Supporting service-dominant business model evaluation in the context of business model innovation
	34	Anna Bon (MU), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines:Integrating Social Practice Theory in Agent-Based Models
	03	Seyyed Hadi Hashemi (UVA), Modeling Users Interacting with Smart Devices
	04	Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
	05	Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
	06	Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
	07	Armel Lefebvre (UU), Research data management for open science
	08	Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
	09	Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
	10	Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
	11	Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision

12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
16 Esam A. H. Ghaleb (UM), BIMODAL EMOTION RECOGNITION FROM AUDIO-VISUAL CUES
17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
19 Roberto Verdecchia (VU), Architectural Technical Debt: Identification and Management
20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
21 Pedro Thiago Timbo Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proenca (LIACS), Robust rules for prediction and description
24 Kaijie Zhu (TUE), On Efficient Temporal Subgraph Query Processing
25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
26 Benno Kruit (CWI & VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs