A hyperheuristic approach towards the training of artificial neural networks

by

Gerrit Stephanus Nel

Dissertation presented for the degree of
Doctor of Philosophy
in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof JH van Vuuren

March 2021

Declaration

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

Date: March 2021

Abstract

In 2015, approximately 2.5×10^{18} bytes of data were generated on a daily basis. The enormity and nature of these data have laid bare the inadequacies of standard data analytic approaches. Researchers and practitioners have for long been unequipped with the necessary means to extract insight from the vast amounts of data at their disposal — until now, that is. Recent advances within the domain of *artificial intelligence* have ushered in a new era, providing the essential connective tissue between data and analysis. These advances can be attributed to instrumental research conducted within the field of *machine learning*, research that has provided algorithms with the inherent ability to learn. A groundbreaking algorithm at the forefront of the current machine learning impetus is the *artificial neural network*. Artificial neural networks are computational models inspired by biological neural networks. This process of neurological emulation enables artificial neural networks to gain an ability intrinsic to their muse — *i.e.* to learn from experience. A characteristic that distinguishes this algorithm from other machine learning algorithms is the efficiency and effectiveness with which it can recognise complex patterns and abstractions within data.

The process according to which this algorithm recognises patterns from data is called *training* and is arguably its most intriguing facet. Conventionally, the method of gradient descent (or steepest ascent) is employed to find good network parameter values. A limitation is, however, imposed on the level of abstraction at which optimisation can thus transpire. A gradient-free approach offers a good alternative. More specifically, the research field of metaheuristics provides powerful optimisation techniques that are applicable in the context of training artificial neural networks. A metaheuristic optimisation approach allows for far greater freedom during artificial neural network training — the network weights, its structure, and its activation functions can be optimised concurrently. This versatility of metaheuristics, as well as their proven capability in many optimisation contexts, serves as justification for why they feature centrally in this dissertation. A challenge to all optimisation approaches, however, relates to the decision of which algorithm to employ for this purpose. Fortunately, the relatively new and promising field of hyperheuristics provides the necessary means to circumvent this challenge — a hyperheuristic is essentially a heuristic that chooses heuristics.

The hyperheuristic considered in this dissertation is called the AMALGAM method. AMALGAM is a powerful and robust optimisation approach that delivers significant performance improvements (approaching a factor of ten), whilst enhancing the level of general applicability over various benchmark problems. This hyperheuristic has not been applied in the literature to the optimisation problem of training artificial neural networks in respect of their network weights, network structure, and activation functions concurrently. An AMALGAM-based hyperheuristic training algorithm is therefore proposed in this dissertation. The novelty of the problem under investigation, however, necessitates a new mathematical learning model. In addition, novel modifications in respect of AMALGAM are made so as to enable its use in neural network train-

ing. A bi-objective hyperheuristic training algorithm is designed, in which the main objective represents a novel network performance measure while a secondary so-called helper objective is incorporated to guide the search process. A test suite, comprising several data sets, is created in order to evaluate the efficacy of the proposed training algorithm.

Three extensive parameter evaluations are performed so as to gain insight into algorithmic performance under different conditions. An in-depth algorithmic performance comparison is also performed during which the performance achieved by the proposed hyperheuristic training algorithm is compared with those of its constituent sub-algorithms. The robustness of the proposed approach is also validated by means of a meta-generalisation analysis. A comparison between the hyperheuristic training algorithm and powerful gradient-based training algorithms is performed which is supplemented by an investigation into the potential consolidation of the hyperheuristic approach with the best gradient-based algorithm. An in-depth investigation is launched into the temporal dynamics of the hyperheuristic's sub-algorithms with a view to gain new insight into this novel approach towards training artificial neural networks and to predict algorithmic performance. A demonstration of how the working of the hyperheuristic can be improved by means of the prediction model is also provided. The structural attributes related to favourable networks produced by the hyperheuristic are analysed with a view to gain new insight into the working of the hyperheuristic.

Opsomming

In 2015 is daar ongeveer 2.5×10^{18} grepe data op 'n daaglikse basis gegenereer. Die omvang en aard van hierdie data het die tekortkominge van standaard data-analitiese benaderings blootgelê. Navorsers en praktisyns het lank nie oor die nodige middele beskik om insig uit die groot hoeveelhede data tot hulle beskikking, te verkry nie — tot nou toe. As gevolg van onlangse vordering binne die vakgebied van *kunsmatige intelligensie* het 'n nuwe era aanbreek wat die nodige bindweefsel tussen data en analise verskaf. Hierdie vordering kan toegeskryf word aan instrumentele navorsing in die gebied van *masjienleer*, navorsing waarin algoritmes wat die inherente vermoë het om te leer, die lig gesien het. 'n Baanbrekende algoritme aan die voorpunt van die huidige masjienleer-momentum is die *kunsmatige neurale netwerk*. Kunsmatige neurale netwerke is berekeningsmodelle wat deur biologiese neurale netwerke geïnspireer is. Hierdie proses van neurologiese nabootsing stel kunsmatige neurale netwerke in staat om 'n vermoë te ontwikkel wat eie is aan hul muse — naamlik om uit ervaring te leer. 'n Eienskap wat hierdie algoritme van ander masjienleeralgoritmes onderskei, is die doeltreffendheid en effektiwiteit waarmee dit komplekse patrone en abstraksies binne data kan herken.

Die proses waarvolgens hierdie algoritme patrone uit data herken, word *leer* genoem en is waarskynlik die mees interessante faset daarvan. Gewoonlik word die gradient-dalingsmetode (of die steilste-hellingmetode) gebruik om goeie netwerkparameterwaardes te vind. Die vlak van abstraksie waarby optimering sodoende kan plaansind, is egter beperk. 'n Gradient-vrye benadering, darenteen, bied 'n goeie alternatief. Meer spesifiek verskaf die navorsingsveld van metaheuristieke kragtige optimeringstegnieke wat in die konteks van kunsmatige neurale netwerkleer toepaslik is. 'n Metaheuristiese optimeringsbenadering maak voorsiening vir veel groter vryheid tydens kunsmatige neurale netwerk-leer — die netwerkgewigte, die netwerkstruktuur en die aktiveringsfunksies van die netwerk kan gelyktydig só geoptimeer word. Hierdie veelsydigheid van metaheuristieke, sowel as hul bewese vermoë in verskeie optimeringskontekste, dien as motivering vir hul kern-oorweging in hierdie proefskrif. 'n Uitdaging vir alle optimeringsbenaderings het egter betrekking op die besluit oor watter metaheuristiek om vir hierdie doel in te span. Gelukkig bied die relatiewe nuwe en belowende studieveld van hiperheuristieke die nodige middele om hierdie uitdaging te oorkom — 'n hiperheuristiek is in wese 'n heuristiek wat heuristieke kies.

Die hiperheuristiek wat in hierdie proefskrif oorweeg word, word die AMALGAM-metode genoem. AMALGAM is 'n kragtige en robuuste optimeringsbenadering wat beduidende prestasieverbeterings (met 'n faktor van tot tien) bied, terwyl die vlak van algemene toepaslikheid oor verskeie toetsprobleme verbeter. Hierdie hiperheuristiek is nog nie in die literatuur op die optimeringsprobleem van kunsmatige neurale netwerk-leer toegepas waarin netwerkgewigte, netwerkstruktuur en aktiveringsfunksies gelyktydig bepaal word nie. 'n AMALGAM-gebaseerde hiperheuristiese leeralgoritme word dus in hierdie proefskrif daargestel. Die oorspronklikheid van die probleem wat ondersoek word, vereis egter dat 'n nuwe wiskundige leermodel geformuleer word.

Daarbenewens word nuwe veranderinge aan AMALGAM voorgestel sodat die algoritme vir neurale netwerk-leer ingespan kan word. 'n Tweedoelige hiperheuristiese leeralgoritme word ontwerp waarin die hoofdoel 'n netwerkprestasiemaatstaf verteenwoordig terwyl 'n sekondêre, sogenaamde hulpdoel daarop gemik is om die optimeringsoekproses te lei. 'n Versameling toetsprobleme, bestaande uit verskeie datastelle, word geskep om die doeltreffendheid van die voorgestelde leeralgoritme te evalueer.

Drie omvattende parameterevaluerings word uitgevoer om sodoende insig te verkry in algoritmiese prestasie onder verskillende omstandighede. Daar word ook 'n diepgaande algoritmiese prestasievergelyking uitgevoer waartydens die prestasie wat deur die voorgestelde hiperheuristiese leeralgoritme bereik word, vergelyk word met dié van sy deelalgoritmes. Die robuustheid van die voorgestelde benadering word ook deur middel van 'n meta-veralgemeningsanalise gevalideer. 'n Vergelyking tussen die hiperheuristiese leeralgoritme en kragtige gradiëntgebaseerde leeralgoritmes word verder uitgevoer en aangevul deur 'n ondersoek na die moontlike konsolidering van die hiperteuristiese benadering met die beste gradiëntgebaseerde algoritme. 'n In-diepte ondersoek na die temporale dinamika van die hiperheuristiek se deelalgoritmes word geloots om insig in hierdie nuwe benadering tot kunsmatige neurale netwerk-leer te verkry en om algoritmiese prestasie te voorspel. 'n Demonstrasie van hoe die werking van die hiperheuristiek deur middel van 'n voorspellingsmodel verbeter kan word, word ook gelewer. Die strukturele kenmerke wat verband hou met gunstige netwerke wat deur die hiperheuristiek gegenereer word, word geanaliseer met die oog op nuwe insig in die werking van die hiperheuristiek.

Acknowledgements

The author wishes to acknowledge the following people and institutions for their various contributions towards the completion of this work:

- My supervisor, Professor JH van Vuuren, for the profound impact he has had on me, both academically and personally. He has instilled a pursuit of excellence in me, in all facets of life, that I will never abandon and I am sincerely grateful for his friendship, guidance, and support throughout the trials and tribulations of this dissertation. Undoubtedly, I will always look up to him and aspire to those tremendous heights only he treads.
- My family, for the continuous support and unwavering belief in me throughout the duration of my studies.
- My friends, for their encouragement and allround support during the completion of work towards this dissertation.
- My fellow *Stellenbosch Unit for Operations Research in Engineering* (SUnORE) colleagues, for their support and the memorable experiences we shared during the past four years.
- The Department of Industrial Engineering at Stellenbosch University, and SUnORE in particular, for the use of its great office space and computing facilities.
- Finally, SUnORE for the financial support that made this research possible.

Table of Contents

List of Acronyms

AI: Artificial Intelligence

AMALGAM: A Multi-algorithm, Genetically Adaptive Multi-objective

AMS: Adaptive Metropolis Search

ANN: Artificial Neural Network

BOHTA: Bi-Objective Hyperheuristic Training Algorithm

CA: Classification Accuracy

CART: Classification and Regression Trees

DE: Differential Evolution

DNN: Deep Neural Network

EA: Evolutionary Algorithm

FNN: Feed-Forward Neural Network

FNSA: Fast Non-dominated Sorting Algorithm

GA: Genetic Algorithm

MAE: Mean Absolute Error

MFNN: Multi-layer Feedforward Neural Network

ML: Machine Learning

MOEA: Multi-Objective Evolutionary Algorithm

MOO: Multi-Objective Optimisation

MOP: Multi-objective Optimisation Problem

MSE: Mean Squared Error

NAG: Nesterov's Accelerated Gradient

NFL: No Free Lunch

NSDE: Non-dominated Sorting Differential Evolution

NSGA-II: Non-dominated Sorting Genetic Algorithm II

OMOPSO: Optimised Multi-Objective Particle Swarm Optimisation

PReLU: Parametric Rectified Linear Unit

PSO: Particle Swarm Optimisation

ReLU: Rectified Linear Unit

RMSE: Root Mean of Squared Error

SFNN: Single-layer Feedforward Neural Network

SGD: Stochastic Gradient Descent

SOTA: State-Of-The-Art

SSE: Sum of Squared Errors

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Contents

1.1 Background

In 1955, the American computer and cognitive scientist John McCarthy defined *Artificial Intelligence* (AI) as "the science and engineering of making intelligent machines, especially intelligent computer programs" [19]. He further claimed that AI is related to the task of utilising computers to "understand human intelligence," but stressed that AI is not necessarily limited to "methods that are biologically observable." The term *intelligence* is somewhat ambiguous in this specific context. Fortunately, a useful test devised by the mathematician Alan Turing may be employed to ascertain whether a machine exhibits intelligence [142]. In the Turing Test (depicted in Figure 1.1), a human evaluator (entity A) is presented with both a machine (entity B) and another human being (entity C). The evaluator does not know the true nature of the two respective entities with whom he or she is conversing. If the evaluator cannot reliably distinguish the machine from the human, based on a conversation *via* a text-only channel (such as a computer keyboard and screen), the machine is said to have passed the test and, subsequently, exhibits intelligence. Thus, according to Russel and Norvig [142], an intelligent machine must satisfy the following criteria:

(1) *Natural language processing* — Communicate in one (or more) languages adequately.

(2) *Knowledge representation* — Store attained input (*e.g.* knowledge and sensory inputs).

(3) *Automated reasoning* — Utilise stored information to draw conclusions.

(4) *Machine learning* — Extrapolate detected patterns and adjust to new situations.

Furthermore, the more comprehensive *Total Turing Test*, proposed by cognitive scientist Stevan Harnad [148], includes the following additional perceptual and physical requirements:

(5) *Computer vision* — Perceive or discern objects.

(6) *Robotics* — Manoeuvre objects.

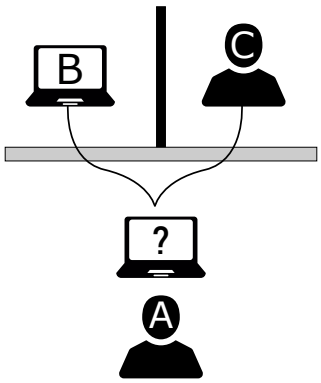


FIGURE 1.1: *A depiction of the Turing Test. A human evaluator (entity A) is presented with both a machine or computer (entity B) and another human being (entity C).*

These six criteria are also associated with the main research areas within AI and one can classify many (if not all) AI-related work as residing within one (or a combination) of these areas. Enabling a machine (or computer) to communicate, memorise, make informed decisions, adapt, perceive, and manipulate objects, is indicative of why AI, even 70 years after its inception, is still a very active field of research [142]. Some of the many industries in which AI is currently utilised are healthcare, business, education, finance, law, and manufacturing, making it a robust utility which is almost universally in high demand [117, 139]. The benefits of AI are tangible and, as a result, many seek to reap these rewards by probing, scrutinising, and improving this ever-advancing research field.

Machine Learning (ML), defined in 1959 by Arthur Samuel as the "field of study that gives computers the ability to learn without being explicitly programmed," is regarded as a general approach towards achieving AI and, ultimately, realising its accompanying benefits [22, 111]. Samuel disproved the misconception that "computers can only do what they are told to" by writing a computer program that played checkers better than its creator [142]. Parsing data, learning from it, and drawing a subsequent conclusion (*e.g.* making a decision or prediction), are deemed by Copeland [22] to form the foundation of ML. According to Russel and Norvig [142], the ability to extrapolate patterns (detectable in data) and to adapt to new problems are two key attributes that contribute to the usefulness of ML within the larger field of AI. One of ML's drawbacks, however, is its data-hungry nature as it usually requires a vast amount of data to enable algorithmic learning [144]. Four main learning paradigms prevail within ML. These are *supervised learning*, *unsupervised learning*, *semi-supervised learning*, and *reinforcement learning*.

In the supervised learning paradigm, *labelled* data are presented, which comprise *independent* variables (*i.e.* input features) and *dependent* variables (*i.e.* output/target variables), the aim being to approximate the underlying function that best represents the input-to-output mapping [58]. *Classification* and *regression* are the two main supervised learning problem classes. The former deals with the assignment of a vector of input features to one of a finite number of discrete categories. Handwriting classification, speech recognition, computer vision, and spam detection are all examples of classification problems [24, 48, 73, 92, 147]. On the other hand, regression deals with problems having output of a continuous nature. Examples include weather, energy load, financial, and manufacturing process yield forecasting [7, 65, 165, 173]. In an unsupervised learning paradigm, the model is presented with input data only (*i.e. unlabelled* or unstructured data), but with no corresponding output or target variable(s). The task is to find some underlying structure (or pattern) within the data. Data clustering, dimensionality reduction, and density estimation are the main problem classes within this learning paradigm [7].

Semi-supervised learning utilises a combination of labelled and unlabelled data. In this paradigm, both supervised and unsupervised learning tasks may be performed [186]. When performing a supervised learning task, *inductive* learning may be employed to improve the input-to-output mapping by incorporating unlabelled data. Alternatively, in *transductive* learning, the focus is on inferring the correct labels for the unlabelled data. When performing an unsupervised learning task, performance may be enhanced by incorporating labelled data. According to Zhu [186], natural language processing, computer vision, and bioinformatics are all application fields in which semi-supervised learning flourishes.

Lastly, within the paradigm of reinforcement learning, the machine is exposed to some dynamic world abstraction and is tasked with determining a course of action (*i.e.* policy) aimed at maximising (or minimising) some world-specific reward (or punishment), through a trial-and-error process. Essentially, this learning paradigm may be defined as a *goal-directed* learning-based computational learning approach [160]. Control theory, inventory management, and the finance industry are all fields to which this paradigm may be applied.

One of the most effective ML techniques for solving a large variety of problems within the field of AI is *Artificial Neural Networks* (ANNs) [53, 85, 142]. A generic representation of an ANN is provided in Figure 1.2. ANNs may be thought of as computational models that emulate the brain's neurological design and information processing capability in order to achieve *learning from experience* [142]. Furthermore, according to the definition proposed by Fausett [41], "an artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks." Based on these definitions it is clear that the brain (and its approach to solving problems) serve as the main muse for ANNs. Given a data set, an ANN is capable of inferring complex relationships (linear or non-linear) that are intrinsic to the properties of the data. Essentially, an underlying functional representation is approximated, which allows computers to perform numerous tasks, such as translating a spoken language, distinguishing an image of a cat from that of a dog, or classifying handwritten digits, to name but a few.

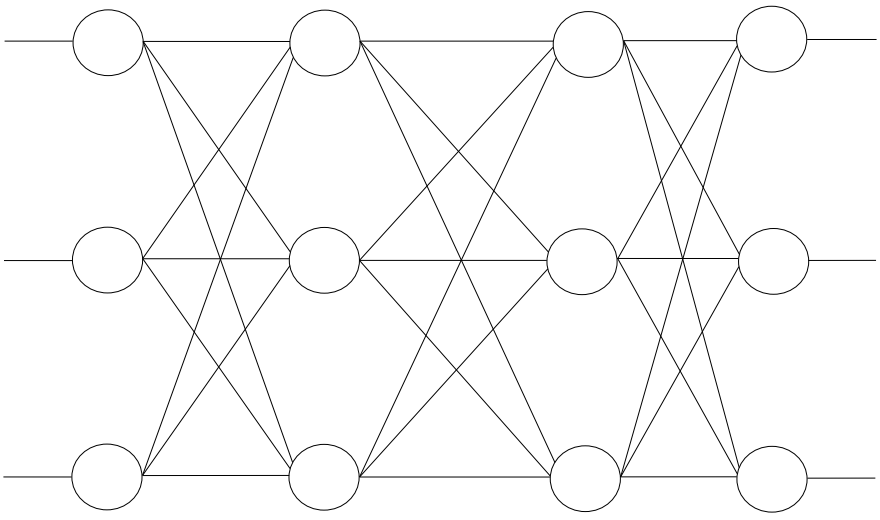


FIGURE 1.2: *A generic representation of an ANN.*

The inception of ANNs in 1943 (before the official conception of AI) may be regarded as the first work now recognised as AI [58, 142, 153]. The research area of ANNs and the larger field of AI were born when the physiologists Warren McCulloch and Walter Pitts proposed a model of

artificial neurons (having two input neurons and one output), with each neuron possessing the characteristic of being either "on" or "off," depending on stimulation by neighbouring neurons within the network. This simple, but effective, network of connected neurons was able to compute any "computable function" and is known today as a *logic circuit*. McCulloch and Pitts also suggested the concept of a network being able to *learn* and, in 1949, Donald Hebb proposed a procedure (now called *Hebbian learning*) for adjusting interconnection strengths within a network. A modified and enhanced version of Hebbian learning is still employed today [53, 142]. One of the main takeaways from Hebb's work was that the brain, and specifically its connectivity, is in a continual state of flux — different functional tasks are continually being learnt, with evolving "neural assemblies" being the result of this process [58]. Another revelatory finding by Hebb was that the strength of synaptic connections between adjacent neurons is increased by repeated activation of one neuron by another.

In 1950, the first official ANN was built by two undergraduate students at Harvard University. This network, called SNARC, consisted of 40 neurons and utilised vacuum tubes and a pilot mechanism. In 1962, further work in the field of ANNs resulted in improved robustness and parallelism, attributable to the work by Winograd and Cowan [142, 177]. Widrow and Rosenblatt made significant contributions during this time, including the establishment of the notions of the *adaline* (adaptive linear unit) and of the *perceptron* (a linear classifier) [58, 142, 153]. Widrow's contributions, which also include the well-known *Least Mean Squares Method*, resulted in a noteworthy decrease in computational time, whereas Rosenblatt proposed a random *trial-and-error* weight updating method so as to achieve (to a certain extent) learning. *Learning Machines*, an influential book written by Nils Nilsson and published in 1965, was another milestone within the field of neural networks, and is regarded as a "classic among neural networks" [58, 142] due to the fundamental groundwork it laid for future work within the field.

From the mid 1960s to the mid 1980s, the momentum of AI (including that of ANNs) was somewhat dampened by a variety of factors. First, AI researchers overstated the capabilities of AI techniques (including neural networks) as well as their own capabilities. This resulted in a general failure to deliver on what was promised within the stated time frame [142]. For example, it was predicted that, within ten years, a computer would be able to beat a chess champion and prove notable mathematical theorems, but these predictions were only realised forty years later [142]. Furthermore, the phenomenon of the *combinatorial explosion* contributed notably to the lapse of AI, as stated in the famous *Lighthill report* by Sir James Lighthill in 1974 [142]. According to Russel and Norvig [142], another contributing factor that had an adverse effect on mainstream AI adoption was "fundamental limitations on the basic structures being used to generate intelligent behaviour." The perceptron neural network exemplified this — although it could *learn* anything it could proficiently represent, a perceptron could not represent much [142]. A damning book by Minsky and Papert, titled *Perceptrons*, declared that this particular network of neurons could only solve *linearly separable problems* [107]. These setbacks resulted in a cessation of funding for and a mass *exodus* from the field of ANN by numerous stakeholders.

Following these setbacks, *competitive learning* (employed in *self organising maps*), *adaptive resonance theory*, *Hopfield* networks, *Boltzmann machines*, and *radial basis function* networks, to name but a few, all made significant contributions to the resurgence of neural networks [58]. During the 1980s, a notable breakthrough came about when researchers started modifying the *backpropagation* algorithm (originally conceived by Bryson and Ho in 1969 and first implemented by Werbos in 1974), subsequently applying it in the context of many learning problems — including in the guise of ANNs [93, 141, 142, 153]. The specific version of the backpropagation algorithm developed by Rumelhart, Hinton, and Williams in 1986 arguably had the most profound impact in the field of ANNs [58]. Smith [153] stated that "the modern era of neural

networks was ushered in" by the introduction of this algorithm. The use of the *sigmoid* function (instead of the *signum* function) in conjunction with the backpropagation algorithm further improved upon the performance of ANNs.

Important strides with regard to the methodologies, theoretical frameworks, network training methods, and hardware-technologies have contributed significantly to the increased application and adoption of ANNs in modern times (*i.e.* 1990s – present) [53, 142]. Another key catalyst for the growth of AI was the availability of significantly larger data sets, which accompanied advances in computing technology. According to Russel and Norvig [142], "a mediocre algorithm with 100 million words of unlabelled training data outperforms the best known algorithm with 1 million words." This statement emphasises the true value of data and it explains why the general availability of data spurred an algorithmic revolution. Companies in many industries started investing considerably in AI research and development, and more specifically ANNs [22]. Due to the important strides made within this area, (which, in turn, led to increased interest and research outputs), ANNs have become comparable to corresponding techniques within the respected fields of statistics, pattern recognition, and ML (other than that of ANNs) [142]. An example of the impact made is the relatively new (and currently sprawling) field of *data analytics* which came into existence as a result of the notable progress made in ML and ANNs.

In 1997, IBM's *Deep Blue* chess-playing computer beat the reigning world chess champion at that time, Garry Kasparov — an unprecedented feat, and one that led to a change in the general mindset as to what machines are truly capable of and how AI could be applied to complex problems. Compared to future work within AI, Deep Blue was in hindsight a crude and inelegant approach towards achieving AI, as it utilised brute force tactics (*i.e.* searching through millions of possible moves within seconds) to achieve its goal — not exactly what was envisioned by the founding fathers of AI. Another landmark breakthrough within the field of AI took place in 2011 and is, once again, attributed to an IBM *supercomputer*, this time called *Watson*. Watson was able to beat two of the all-time best players in the game show *Jeopardy!* [163]. Compared with the accomplishment of Deep Blue, Watson was far superior as it showcased five of the six machine intelligence criteria of the Total Turing Test mentioned earlier [148], whereas Deep Blue only satisfied two of these criteria. Perhaps the most impressive feat of Watson was its ability to process and reason based on *natural language*. Described as the most advanced "question answering" machine, Watson could comprehend a question presented in "everyday human elocution" (just like a human-being), and provide a factually correct answer, using ML techniques (including ANNs) [148]. *AlphaGo*, developed by Alphabet Inc.'s DeepMind, was responsible for the next ground breaking innovation within AI. It was the first computer program to become a champion in the game of *Go* in 2016 [172]. The immense increase in complexity (Go has an upper bound of the order of 10^{170} possible board configurations for a 19×19 board) forced a very different approach by researchers, scientists, and engineers, as opposed to the brute force tactics of Deep Blue [172]. In the case of AlphaGO, *deep learning* (ANNs comprising many layers) was employed in the ML context of reinforcement learning, which greatly contributed to the success of the endeavour. Conquering the game of Go has widely been thought of as the *holy grail* of AI, as it is considered to be one of the (if not *the* most) complex board game, involving not only a high level of strategy, but also human intuition. This unprecedented feat enhanced the already impressive acclaim of ANNs.

Currently, deep learning receives considerable attention from the AI research community, attributable to its effectiveness in lucrative application fields such as natural language processing and machine vision [22, 146]. In the case of the latter, a deep learning endorser, Andrew Ng, achieved a notable breakthrough in 2012 while working for Google, which served as a catalyst for a remarkable period of exploration in the field. Ng's breakthrough was the result of utilising

significantly larger (or deep) ANNs and providing them with vast amounts of data — in Ng's deep neural network, images from ten million YouTube videos were used. The resulting ANNs were capable of surpassing human beings in the task of image classification. The field of *machine vision* has grown rapidly as a result of the utility and power demonstrated by ANNs [53].

The most important facet of ANNs is their *training* — *i.e. learning* the network's parameters in respect of a presented data set (comprising input and/or output data). ANNs typically consist of many processing elements, called neurons (represented by the nodes or circles in Figure 1.2), which are connected *via synaptic* connections (represented by the links in Figure 1.2). Each connection has an accompanying weight, which can be either excitatory or inhibitory, depending on the intrinsic relationships or patterns present within the data [41, 85, 101]. A training algorithm governs the process of learning these weights — the aim being to find the "best" functional approximation (*i.e.* input-output mapping) of the problem at hand by methodically adjusting the weights until a relevant performance measure is maximised (or minimised). Training an ANN may, thus, be expressed in terms of an optimisation problem. Consequently, many traditional approaches within the field of optimisation are applicable.

A gradient-based approach (more specifically, gradient descent) is the most prolific training approach adopted in the academic literature, with *state-of-the-art* (SOTA) results serving as ample justification for its widespread use [53]. The remarkable success of gradient descent may be ascribed to the method of backpropagation, which enables the computationally efficient application of a gradient-based approach to the optimisation problem of training ANNs. A gradient-based approach, however, imposes a key constraint — the objective function (to be minimised or maximised) *must* be continuous and differentiable. This limits the level of abstraction at which optimisation of the network can transpire. Consequently, various restrictions are placed on salient aspects of the network, such as the nature of the functions employed by the neurons (called *activation functions*), the network structure, and the objective function. When formulating the corresponding mathematical model (to be solved by the gradient-based approach), the objective function is required to be continuous and differentiable, whereas the decision variables are predominantly restricted to the network weights — a different formulation would violate the differentiability requirement. The parameters pertaining to the activation functions and network structure are typically optimised separately (after the weights have been adjusted).

In contrast, when adopting a metaheuristic (gradient-free) optimisation approach, it is possible to circumvent many (if not all) of these drawbacks. Due to metaheuristics' applicability to complex, non-linear, and non-differentiable problems, training can transpire on a notably lower level of abstraction, especially compared with that of a gradient-based approach. When formulating the learning problem (as an optimisation problem) to be solved by a metaheuristic, far greater freedom is allowed as few restrictions are imposed by this type of solution methodology. It is this freedom that allows one, for example, to incorporate parameters related to the network structure and the activation functions, when formulating the mathematical model. Therefore, all the network parameters (including the weights, of course) can be optimised concurrently. That is, only one optimisation run is required, as opposed to the multiple runs required when adopting a gradient-based approach. The added flexibility of metaheuristics, as well as their proven capability in many optimisation contexts, ought to be sufficient justification for why they are at the centre of consideration in this dissertation. A metaheuristic training approach naturally also has its disadvantages — the main drawback being its *slow* nature. Many metaheuristics operate concurrently on a number of solutions, potentially hindering the training algorithm's efficiency in terms of computation time. One may, however, argue that this drawback is amply mitigated by the greater flexibility afforded by the inherent nature of metaheuristics and the resulting lower level of abstraction at which optimisation takes place. In the context of training

ANNs, the popularity of metaheuristic optimisation approaches has, to an extent, diminished in recent years, especially so when considering the copious number of breakthroughs accredited to gradient-based approaches. The accomplished repute of metaheuristics should not, however, be disregarded, even in the context of training ANNs.

According to Goodfellow *et al.* [53], discrepancies (in respect of performance) are prevalent when comparing various training algorithms[1], and the choice of which training algorithm to use is surrounded by conjecture and debate. The so-called *No Free Lunch* (NFL) theorem lends vindication to these discrepancies; it essentially states that "an optimisation algorithm that performs particularly well on one set of objective functions, will also perform correspondingly poorly on the other, remaining objective functions" [178]. Today, ANNs are applied to an ever-increasing number of problems, each with their own intricacies and nuances. This contributes to a growing need to devise versatile algorithms capable of adapting to many different problems, algorithms that are not required to be engineered explicitly for each individual problem, but are applicable to a wide variety of problems. When devising such an algorithm it is perhaps apt to adopt an AI-inspired approach — one that exhibits, to an extent, intelligence — so as ultimately to improve the training process of ANNs. A relatively new and promising field of research, called *hyperheuristic* solution techniques, warrants consideration with respect to improving the quality of solutions, whilst enhancing the level of general applicability in the context of ANN training algorithms. According to Burke [16], a hyperheuristic may be regarded as a "heuristic which chooses heuristics" — *i.e.* at any given time, the hyperheuristic manages the selection of low-level heuristics (from a pre-defined set) to be applied to the optimisation problem at hand. One of the main advantages associated with a hyperheuristic approach is greater search effectiveness, attributable to its operation on both a heuristic search space and a solution search space. Hyperheuristics are, therefore, ideal candidates for mitigating the cumbersome nature of choosing a suitable training algorithm for the problem at hand.

The hyperheuristic at the centre of this dissertation is called the AMALGAM[2] method [171]. According to Vrugt and Robinson [171], AMALGAM is a powerful and robust optimisation approach that delivers significant performance improvements (reportedly approaching a factor of ten), whilst enhancing the level of general applicability, in respect of various benchmark problems. AMALGAM's powerful and robust nature is ascribed to the intelligent mechanisms by which it manages its sub-ordinate metaheuristics. To the best of the author's knowledge, AMALGAM has not been applied to the problem of training ANNs at the level of abstraction considered in this dissertation — *i.e.* to optimise the network weights, the network structure, and the activation functions concurrently. This finding, together with the reported success of AMALGAM in various conventional optimisation contexts, raises the question whether it will provide utility within the optimisation context of training ANNs.

1.2 Problem statement

The problem considered in this dissertation is to investigate to what extent a hyperheuristic solution methodology, inspired by AMALGAM, is able to train and obtain high-quality ANNs, whilst exhibiting enhanced levels of general applicability. Networks are to be trained on a notably lower level of abstraction — the aim being to find good network weights, network structure, and activation functions concurrently, which stands in contrast to the traditional paradigm in which only the weights are trained after having decided upon a suitable network structure and set of

[1] Algorithms that deliver SOTA performance in respect of one data set, are not guaranteed to deliver SOTA performance in respect of all other data sets.

[2] AMALGAM is an acronym for *a multi-algorithm, genetically adaptive multi-objective.*

activation functions. The novelty of the problem under investigation necessitates the mathematical formulation of a new learning model. In addition, novel modifications of AMALGAM have to be made so as to facilitate its use in solving the learning model. To this end, a *bi-objective hyperheuristic training algorithm* (BOHTA) is to be designed, in which the main objective represents a suitable network performance measure and a secondary, so-called helper objective is incorporated to guide the search process appropriately. A test suite, comprising several data sets, is further to be created in order to evaluate the efficacy of the BOHTA. To this end an extensive algorithmic performance comparison is to be performed in which the performance of the BOHTA is compared with those of its sub-algorithms and powerful gradient-based approaches. The novelty of the proposed approach necessitates an in-depth investigation into the inner working of the BOHTA, *i.e.* the dynamics of its constituent sub-algorithms. This analysis is to be facilitated by the nature of both the learning model and the proposed solution methodology.

1.3 Research scope

In order to narrow down the scope of the problem considered in this dissertation, the following delimitations are adopted:

***Feed-forward neural networks* (FNNs)** are selected to form the basis of discourse in this dissertation. FNNs are one of the most prominent types of ANNs and are sufficiently representative of the general operation of ANNs. The fundamental premise of training remains, by and large, the same amongst the different ANN types. Popular ANN types such as *convolutional neural networks* and *recurrent neural networks* are therefore excluded from consideration in this study. Hardware constraints provide further motivation for this exclusion. The typical data sets to which convolutional and recurrent neural networks are applied (*i.e.* image classification and natural language processing, respectively) are rendered intractable when using traditional hardware (due to the size and nature of these data sets).

Supervised learning is the ML paradigm that forms the basis of discourse in this dissertation. Similar to the choice of network type, the learning paradigm ought to have little bearing on the findings — the fundamental premise remains, by and large, the same. Within the paradigm of supervised learning, classification problems are, furthermore, considered exclusively because of their considerable popularity.

AMALGAM is the principal research muse of the proposed BOHTA approach, because of its powerful and robust nature as well as its novelty in the given optimisation context. The only metaheuristics considered for inclusion in the BOHTA are therefore the constituent sub-algorithms of AMALGAM — *i.e.* a *genetic algorithm* (GA) [63], *differential evolution* (DE) [157], *particle swarm optimisation* (PSO) [39], and *adaptive Metropolis search* (AMS) [55].

Algorithmic comparisons are limited to the BOHTA, its constituent sub-algorithms, and the most popular gradient-based training algorithms. The principal aim of this research, as per the problem statement in §1.2, is to gain pertinent insight into the working of the BOHTA in the optimisation context of training FNNs. The repute of gradient-based techniques is, however, a matter that cannot be neglected — hence the inclusion of *stochastic gradient descent* (SGD) [136], RMSProp [164], and Adam [79] as part of the algorithmic performance comparison.

1.4 Dissertation objectives

The following nine objectives are pursued in this dissertation:

I To *conduct* a review of the pertinent literature related to:

(a) Relevant mathematical and statistical prerequisites,

(b) FNNs and their application in a supervised learning paradigm,

(c) evolutionary-based metaheuristic solution techniques for multi-objective optimisation,

(d) and hyperheuristic optimisation techniques, with the focus on AMALGAM.

II To *formulate* an appropriate bi-objective mathematical model for the task of training FNNs in respect of their network weights, network structure, and activation functions, for performing supervised learning.

III To *design* an AMALGAM-inspired BOHTA approach capable of providing high-quality solutions to the mathematical learning model formulated in pursuit of Objective II.

IV To *propose* an appropriate encoding scheme capable of representing networks in a versatile format to facilitate their effective training by the BOHTA sub-algorithms.

V To *establish* a test suite of several data sets, each of which represents a different problem instance of the learning model formulated in pursuit of Objective II, in order to demonstrate the capabilities of the BOHTA and draw pertinent insight from its implementation.

VI To *apply* data pre-processing techniques in respect of the test suite of Objective V.

VII To *perform* extensive algorithmic parameter evaluations in a structured and statistically sound manner in order to ascertain good parameter values for the BOHTA, its constituent sub-algorithms, and the gradient-based training algorithms.

VIII To *implement* the BOHTA in the context of the test suite of Objective V in a structured and statistically sound manner. More specifically, to:

(a) *compare* the performance of the BOHTA with those of its individual subordinate algorithms and gradient-based training algorithms,

(b) *investigate* the temporal dynamics of the BOHTA in order to infer pertinent insight into its working,

(c) *determine* the extent to which algorithmic performance can be predicted based on the high-level characteristics (*i.e.* meta-features) of the data sets,

(d) *analyse* the characteristics of favourable network structures produced by the BOHTA, and

(e) *evaluate* the meta-generalisation capabilities in respect of unseen data sets.

IX To *recommend* sensible follow-up work related to the contributions of this dissertation, which may be pursued in future.

1.5 Dissertation organisation

Apart from this introductory chapter, this dissertation contains a further nine chapters (partitioned into four distinct parts), a bibliography, and two appendices. The first part, comprising Chapters 2–4, contains a literature review of material relevant to the work in this dissertation. More specifically, Chapter 2 is devoted to a review of the academic literature on a number of mathematical and statistical prerequisites pertaining to the topic of this dissertation. In Chapter 3, the focus shifts to a review of the literature pertaining to ANNs in particular, in which FNNs form the basis of discussion. The first part is concluded in Chapter 4 with a review of the literature pertaining to metaheuristic and hyperheuristic optimisation techniques. AMALGAM and its constituent sub-algorithms are the focal point of that chapter.

The second part of the dissertation, comprising Chapters 5 and 6, is concerned with a delineation of the modelling approach proposed in this dissertation. More specifically, Chapter 5 is devoted to the formulation of an appropriate mathematical model of the learning problem at hand — *i.e.* the training of an FNN in respect of its network weights, network structure, and activation functions concurrently. The mathematical model is elucidated in respect of its decision variables, constraints, and two objective functions. Chapter 6 contains a detailed description of the solution methodology proposed in this dissertation for solving the bi-objective mathematical model of Chapter 5 — a methodology called the BOHTA approach. The aim of this hyperheuristic optimisation approach is to provide high-quality solutions to instances of the bi-objective mathematical model, whilst demonstrating an enhanced level of general applicability. These instances are represented by the data sets in the test suite selected in pursuit of Objective V and are described along with the necessary data pre-processing techniques.

The third part of the dissertation, comprising Chapters 7 and 8, is devoted to an algorithmic evaluation of the BOHTA, its constituent sub-algorithms, and gradient-based training algorithms. In Chapter 7, extensive algorithmic parameter evaluations are performed. Necessary precursors to the parameter evaluations are a discussion on the method by which the performance of a training algorithm is assessed and a description of the experimental setup for deciding upon desirable algorithmic parameter values. Three algorithmic parameter evaluations are performed, the first of which focusses on establishing good parameter values for the sub-algorithms, the second focusses on establishing good parameter values for the BOHTA itself, and the third focusses on establishing good parameter values for the gradient-based training algorithms. Chapter 8 is devoted to a detailed investigation into the implementation of the BOHTA in the context of solving the learning problem instances induced by the test suite. The relative algorithmic performances of the sub-algorithms are compared with those of different versions of the BOHTA, which is supplemented by a comparison with the performances of the gradient-based training algorithms. The temporal dynamics of the BOHTA are also analysed so as to gain deeper insight into the performance of its sub-algorithms under different circumstances. The extent to which the performance of the sub-algorithms can be predicted — based on the meta-features of the data sets — forms part of this analysis. An investigation into the structural attributes of favourable networks (produced by the BOHTA) is also conducted. In order to gain insight into the robustness of the BOHTA, an evaluation in respect of unseen data sets is performed so as to probe its meta-generalisation capabilities.

The fourth part of the dissertation, comprising Chapters 9 and 10, concludes the dissertation. A summary and critical appraisal of the contributions of the dissertation are provided in Chapter 9, and recommendations with respect to possible follow-up work which may be pursued in future follow in Chapter 10.

Part I

Literature Review

CHAPTER 2

Mathematical and Statistical Prerequisites

Contents

The purpose of this chapter is to present the reader with the necessary mathematical and statistical prerequisites to facilitate a general understanding of the analyses performed later in this dissertation. The chapter comprises two main sections. The first section is devoted to the preliminaries related to multi-objective optimisation. Important concepts pertaining to the notion of Pareto dominance are included in this discussion. Two prominent algorithms, the *fast non-dominated sorting algorithm* (FNSA) and the *crowding distance assignment algorithm*, are introduced and discussed in this section. The next section is devoted to a discussion on the preliminaries associated with the statistical analyses proffered later in this dissertation. Prominent procedures, of a *non-parametric* nature, for comparing the medians of a number of statistical samples are discussed first and include the non-parametric *Friedman test* in conjunction with the *Nemenyi post hoc procedure*. Important *tree-based* statistical learning algorithms are reviewed next. Two popular decision tree algorithms, namely *Classification and Regression Trees* (CART) [12] and C4.5 [126], form the basis of the discussion. A concise summary concludes the chapter.

2.1 Multi-objective optimisation preliminaries

Within the realm of optimisation, there are two main classes of problems: *Single-objective optimisation problems*, where the goal is to optimise a single objective function, and *multi-objective optimisation problems* (MOPs), where several objective functions must be optimised simultaneously. The focus in this dissertation, however, falls on the latter class. When discussing the respective fields of metaheuristic and hyperheuristic optimisation later in this dissertation, a basic understanding of the preliminaries of *multi-objective optimisation* (MOO) is presupposed. For MOPs, there typically exists a preferential set of trade-off solutions, called *Pareto optimal solutions*. It is uncommon in MOO to find a single solution that is optimal with respect to *all* objective functions — hence the prevalence of a set of trade-off solutions. The identified set of trade-off solutions is typically presented to a decision maker for a *post hoc* analysis with the aim of ascertaining a final or definitive solution to the problem at hand in a subjective manner. Due

to MOO forming an integral part of the work contained in this dissertation, a more in-depth elucidation of the notions of *Pareto optimality* and of *Pareto rank* is provided in this section. This elucidation aims to facilitate a greater understanding of the MOO solution techniques employed later in this dissertation.

Both the definitions presented and notational convention adopted in this section closely emulate the work of Miettinen [105]. A necessary precursor to any discussion on Pareto optimality and Pareto ranking is a general model for an MOP, comprising $e \geq 2$ real-valued objective functions, denoted by $h_1(\boldsymbol{d}), h_2(\boldsymbol{d}), \ldots, h_e(\boldsymbol{d})$, where $\boldsymbol{d} = [d_1, \ldots, d_n]$ denotes a real-valued n-dimensional decision vector. It is henceforth assumed, without loss of generality, that all of these objective functions are to be maximised. A general MOP formulation may be presented in the form

$$\left.\begin{array}{ll} \text{maximise} & \boldsymbol{h}(\boldsymbol{d}) = [h_1(\boldsymbol{d}), h_2(\boldsymbol{d}), \ldots, h_e(\boldsymbol{d})] \\ \text{subject to} & \boldsymbol{d} \in \mathcal{S}, \end{array}\right\} \tag{2.1}$$

where $\mathcal{S}$ is the *feasible region* of the problem. The feasible region is assumed to be a subset of the *decision space*, *i.e.* $\mathcal{S} \subseteq \mathbb{R}^n$ for an MOP with continuous decision variables. The *objective vector* is denoted by $\boldsymbol{h}(\boldsymbol{d}) = [h_1(\boldsymbol{d}), h_2(\boldsymbol{d}), \ldots, h_e(\boldsymbol{d})]$, but for simplicity, the denotation $\boldsymbol{z} = [z_1, z_2, \ldots, z_e]$ is henceforth adopted instead, where $z_i = h_i(\boldsymbol{d})$ for all $i \in \{1, \ldots, e\}$. The *feasible objective space* is a subset of the *objective space* $\mathbb{R}^e$, *i.e.* $\mathcal{Z} = \{\boldsymbol{h}(\boldsymbol{d}) \mid \boldsymbol{d} \in \mathcal{S}\}$, and may be thought of as the image of the feasible region under the functions $h_1, h_2, \ldots, h_e$. This notion is illustrated graphically for $n = e = 2$ in Figure 2.1. Depicted is an example of a two-dimensional continuous decision space (the feasible region is represented by the light-grey enclosure) which is projected to its corresponding two-dimensional objective space.

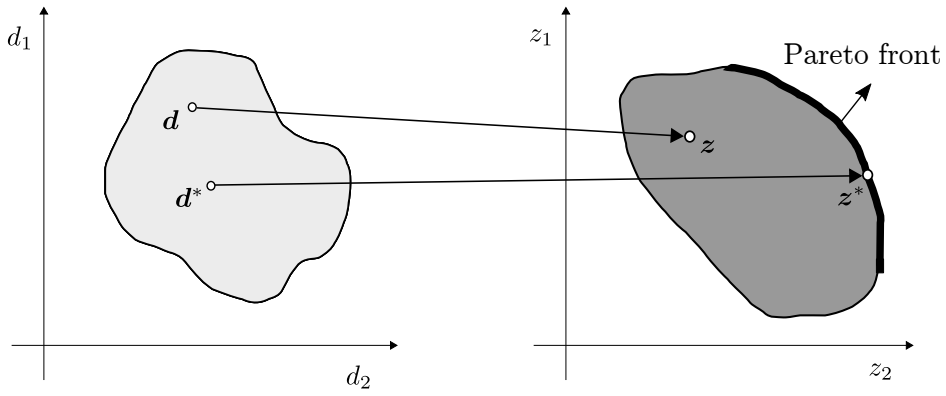


FIGURE 2.1: *An example of a two-dimensional continuous decision space (left) and its corresponding objective space (right) in which both objective functions z_1 and z_2 have to be maximised. The decision vector $\boldsymbol{d}^*$ represents a non-dominated solution (hence its inclusion in the Pareto front), whereas the decision vector $\boldsymbol{d}$ represents a dominated solution. The corresponding objective vectors, denoted by $\boldsymbol{z}^*$ and $\boldsymbol{z}$, are also illustrated.*

Identifying a set of high-quality trade-off solutions for the MOP (2.1) is facilitated by the notion of *Pareto dominance*. A feasible solution to (2.1) is classified as *dominated* if another feasible solution exists that is strictly better than the original solution with respect to at least one objective function, yet no worse with respect to all of the remaining objective functions. Whenever no other solution dominates a given solution, that solution is consequently classified as *non-dominated*.

Stated mathematically, a decision vector $\boldsymbol{d}^* \in \mathcal{S}$ dominates the decision vector $\boldsymbol{d} \in \mathcal{S}$ if $h_i(\boldsymbol{d}^*) \geq h_i(\boldsymbol{d})$ for all $i \in \{1, \ldots, e\}$ and $h_j(\boldsymbol{d}^*) > h_j(\boldsymbol{d})$ for at least one $j \in \{1, \ldots, e\}$. This dominance relation is denoted by $\boldsymbol{d}^* \succ \boldsymbol{d}$. On the other hand, a decision vector $\boldsymbol{d}^*$ is non-dominated within some subset $\mathcal{Q} \subseteq \mathcal{S}$ if no other decision vector $\boldsymbol{d} \in \mathcal{Q}$ dominates $\boldsymbol{d}^*$. The definition

of *Pareto optimality* refers to the special case in which $\mathcal{Q} = \mathcal{S}$, *i.e.* the subset $\mathcal{Q}$ and feasible region $\mathcal{S}$ are equal. A decision vector $\boldsymbol{d}^* \in \mathcal{S}$ is therefore *Pareto optimal* if it is non-dominated in $\mathcal{S}$. The *Pareto set*, denoted by $\mathcal{P}_S$, contains all Pareto optimal decision vectors, whereas the corresponding Pareto optimal objective vectors are contained within the corresponding *Pareto front*, denoted by $\mathcal{P}_F$. Referring, once again, to Figure 2.1, the non-dominated solution is represented by the decision vector $\boldsymbol{d}^*$ (with objective vector $\boldsymbol{z}^*$), whereas the dominated solution is represented by the decision vector $\boldsymbol{d}$ (with objective vector $\boldsymbol{z}$).

Another important concept that relates to Pareto dominance is that of ϵ-dominance, first introduced by Laumanns [89] in 2002. An ϵ-non-dominated set of solutions comprises strictly non-dominated solutions. A decision vector $\boldsymbol{d}^*$ is said to ϵ-dominate another decision vector $\boldsymbol{d}$ if, for some real number $\epsilon > 0$,

$$(1+\epsilon)\, h_i(\boldsymbol{d}^*) \geq h_i(\boldsymbol{d}), \qquad i \in \{1, \ldots, e\}$$

and

$$(1+\epsilon)\, h_j(\boldsymbol{d}^*) > h_j(\boldsymbol{d})$$

for at least one $j \in \{1, \ldots, e\}$ (in the context of a maximisation problem). The size of the archive is thus determined implicitly by ϵ, a user-defined parameter. According to Laumanns, an ϵ-non-dominated set (or archive) tends to exhibit greater diversity and exhibits, as a result, improved convergence during the optimisation process. Another benefit associated with the concept of ϵ-dominance involves improved control over the size of the non-dominated set.

The *non-dominated rank*, also known as the *Pareto rank*, of a feasible solution $\boldsymbol{d}$ to (2.1), denoted by $\rho_{\boldsymbol{d}}$, provides a measure of the quality of the decision vector $\boldsymbol{d}$ and is determined by the following procedure: Given a set of decision vectors, a rank of zero is assigned to all non-dominated decision vectors of the set. Note the convention of starting the Pareto rank numbering at zero; this is done without loss of generality. The initial assignment is followed by the exclusion of all non-dominated decision vectors (those for which $\rho = 0$) from further consideration. A rank of 1 is subsequently assigned to the "new" non-dominated decision vectors in the reduced set. This procedure is iterated until all decision vectors in the original set have been assigned ranks. Each subsequent set of non-dominated decision vectors may be associated with a so-called *front depth*, which is simply the non-dominated rank. Figure 2.2 contains a graphical illustration of the Pareto rank assignment procedure. Depicted graphically is an MOP instance, of the form (2.1) with $e = 2$ (*i.e.* there are two objective functions). Assuming that the corresponding decision vectors are feasible, the objective vectors $\boldsymbol{z} = [z_1, z_2]$ may be plotted as points in a Cartesian plane. The Pareto front, as well as the non-dominated sub-fronts, are depicted together with the vectors' Pareto rank assignments in the figure.

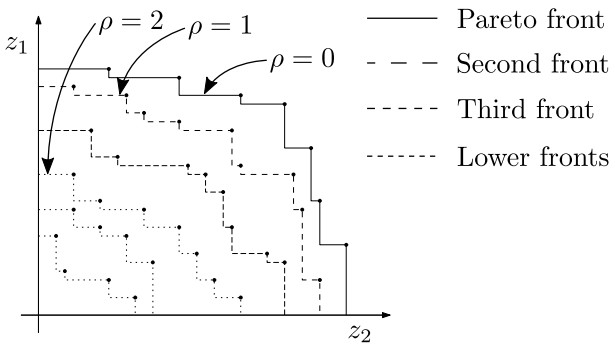


FIGURE 2.2: *Example of a Pareto front, non-dominated sub-fronts, and the Pareto rank assignment for an MOP in which $e = 2$ objectives have to be maximised [145].*

2.1.1 The FNSA

The well-known FNSA, developed by Deb *et al.* [28] as an integral part of the celebrated NSGA-II, provides a computationally efficient non-dominated sorting procedure for assigning a Pareto rank to solutions within a population. This is one of two attributes that often constitute a solution's fitness (the other attribute is discussed later). With respect to fitness, solutions achieving a lower Pareto rank are preferred. A detailed discussion of the inner working of the FNSA is provided in this section. Specific reference is made to the pseudocode description presented in Algorithm 2.1 so as to facilitate the discussion. This description includes minor adjustments in respect of the original version of the algorithm by Deb *et al.* [28].

Algorithm 2.1: FNSA [28, 145]

Input : A population of solutions $\mathcal{P}$.

Output: The population $\mathcal{P}$ partitioned into k successive non-dominated fronts $\mathcal{F}_1, \ldots, \mathcal{F}_k$ and assigned Pareto ranks for each solution.

```
1  F_1 ← ∅;
2  foreach a ∈ P do
3      S_a ← ∅;
4      u_a ← 0;
5      foreach b ∈ P do
6          if a ≻ b then
7              S_a ← S_a ∪ {b};
8          else if b ≻ a then
9              u_a ← u_a + 1 ;
10     if u_a = 0 then
11         ρ_a ← 0 ;
12         F_1 ← F_1 ∪ {a};
13 j ← 1;
14 while F_j ≠ ∅ do
15     Q ← ∅;
16     foreach a ∈ F_j do
17         foreach b ∈ S_a do
18             u_b ← u_b − 1 ;
19             if u_b = 0 then
20                 ρ_b ← j;
21                 Q ← Q ∪ {b};
22     j ← j + 1;
23     F_j ← Q;
```

Execution of the algorithm comprises two stages. The first stage (spanning steps 1–12) involves the following procedure: For each solution $\boldsymbol{a} \in \mathcal{P}$, the algorithm first determines its *domination count*, *i.e.* the number of solutions that dominate $\boldsymbol{a}$, denoted by $u_{\boldsymbol{a}}$. In addition, a set containing all the solutions that dominate $\boldsymbol{a}$, denoted by $\mathcal{S}_{\boldsymbol{a}}$, is also created. Solutions in the first non-dominated front are assigned, by definition, a domination count of zero and a Pareto rank of zero, *i.e.* $u_{\boldsymbol{a}} = 0$ and $\rho_{\boldsymbol{a}} = 0$. A separate set (or front) is then populated by those solutions for which $u_{\boldsymbol{a}} = 0$, and is denoted by $\mathcal{F}_1$. The second stage of the FNSA's execution (spanning steps

13–23) proceeds as follows: Each solution $\boldsymbol{a} \in \mathcal{F}_1$ is visited and the domination count $u_{\boldsymbol{b}}$ of each solution $\boldsymbol{b} \in \mathcal{S}_{\boldsymbol{a}}$ is decremented by one. By discounting the contribution of the solution $\boldsymbol{a}$, the domination count for solutions that are part of the second non-dominated front will now be zero. These solutions are subsequently assigned a Pareto rank of one and then placed in set $\mathcal{F}_2$. This procedure is repeated until all the fronts have been established and solutions ranked accordingly.

2.1.2 The notion of crowding distance

The second attribute contributing to a solution's fitness is its so-called *crowding distance*, which, given some non-dominated front, provides a measure of the density of solutions surrounding it within the front. The process of assigning a crowding distance to each solution in a non-dominated front $\mathcal{T}$ is presented in pseudocode form in Algorithm 2.2. This algorithmic description again contains minor adjustments in respect of the original version by Deb *et al.* [28]. The algorithm starts by initialising each solution's crowding distance to zero, after which $\mathcal{T}$ is sorted in ascending order of magnitude with respect to each objective function $s \in \{1, \ldots, e\}$ separately. The crowding distance of the j-th solution (in the sorted population) is denoted by $\mathcal{T}[j]_{\text{dist}}$. A crowding distance of infinity is next assigned to solutions at the two endpoints of the front, *i.e.* $\mathcal{T}[1]_{\text{dist}} = \mathcal{T}[\ell]_{\text{dist}} = \infty$, where $\ell = |\mathcal{T}|$. In order to calculate $\mathcal{T}[2]_{\text{dist}}, \mathcal{T}[3]_{\text{dist}}, \ldots, \mathcal{T}[\ell-1]_{\text{dist}}$, each intermediate solution's crowding distance is increased by the normalised distance between function values of neighbouring solutions $j-1$ and $j+1$ for the s-th objective. The normalisation is performed in respect of the current range of objective s. This is done for each of the e objectives. In effect, an accumulation of crowding distances over the e objectives is performed. In Algorithm 2.2, $\mathcal{T}[j].h_s$ denotes the function value of objective s for the j-th solution in $\mathcal{T}$. When comparing the fitness of solutions within the same non-dominated front, solutions with a greater crowding distance are preferred as they have fewer neighbouring solutions that are close by. Such solutions therefore exhibit a greater degree of diversity [28].

Algorithm 2.2: Crowding distance assignment algorithm [28, 145]

Input : A non-dominated front $\mathcal{T}$ of cardinality ℓ.
Output: The crowding distances $\mathcal{T}[1]_{\text{dist}}, \ldots, \mathcal{T}[\ell]_{\text{dist}}$.

```
ℓ ← |T|;
foreach j ∈ T do
    T[j]_dist ← 0;
foreach objective s ∈ {1, ..., e} do
    T ← sort(T, s);
    T[1]_dist ← ∞;
    T[ℓ]_dist ← ∞;
    for i ← 2 to (ℓ − 1) do
        T[i]_dist ← T[i]_dist + (T[i + 1].h_s − T[i − 1].h_s)/(h_s^max − h_s^min);
```

A method for distinguishing between two solutions, with respect to their Pareto ranks and crowding distances, is discussed in closing. Given a solution $\boldsymbol{a}$ with Pareto rank $\rho_{\boldsymbol{a}}$ and crowding distance $\boldsymbol{a}_{dist}$, as well as another solution $\boldsymbol{b}$ with Pareto rank $\rho_{\boldsymbol{b}}$ and crowding distance $\boldsymbol{b}_{dist}$, a so-called *crowded comparison operator*, denoted by $\succ_{cc}$, may be used to compare these two solutions' fitness levels. According to this operator, if $\boldsymbol{a}$ has a smaller Pareto rank than $\boldsymbol{b}$, *i.e.* $\rho_{\boldsymbol{a}} < \rho_{\boldsymbol{b}}$, then $\boldsymbol{a}$ is preferred (fitter for selection in an evolutionary optimisation context). If,

however, these two solutions have equal Pareto ranks, *i.e.* $\rho_{\boldsymbol{a}} = \rho_{\boldsymbol{b}}$, then the solution with the larger crowding distance is preferred. The operator may therefore be described mathematically as

$$\boldsymbol{a} \succ_{cc} \boldsymbol{b} \quad \text{if} \quad \begin{cases} \rho_{\boldsymbol{a}} < \rho_{\boldsymbol{b}}, \text{ or} \\ \rho_{\boldsymbol{a}} = \rho_{\boldsymbol{b}} \text{ and } \boldsymbol{a}_{dist} > \boldsymbol{b}_{dist}. \end{cases} \tag{2.2}$$

2.2 Statistical analysis preliminaries

The statistical analyses performed in this dissertation are twofold: First, a comparative study of algorithmic performances is performed between different metaheuristic and hyperheuristic optimisation approaches using relevant statistical tests and procedures. Secondly, an investigation into the extent to which algorithmic performance can be predicted using relevant predictive algorithms is carried out. In §2.2.1, the former is addressed by means of an elucidation of hypothesis testing together with the relevant non-parametric tests, whereas in the case of the latter, a discussion on tree-based statistical learning algorithms transpires in §2.2.2.

2.2.1 Inferential statistical testing

Branke *et al.* [11] argued that it is good scientific practice to employ structured and statistically sound procedures when comparing the performance of metaheuristics. Extensive statistical analyses are performed in this dissertation for this purpose. Therefore, an appropriate and necessary precursor to these analyses is an elucidation of the relevant statistical tests and procedures employed. In order to effectively analyse results and draw the required inferences at a desired level of confidence, an hypothesis testing approach from the field of inferential statistics is adopted in this dissertation. Accordingly, two hypotheses are considered, they are:

- The *null hypothesis*, denoted by H_0, which is assumed to be true and typically represents the assertion of no effect (or no difference in performance quality between at least two algorithms).
- The *alternative hypothesis*, denoted by H_1, which represents the assertion of an effect (or difference in performance quality between at least two algorithms).

In order to determine whether H_0 (assumed to be true) should be rejected in favour of H_1, a statistical test is performed on samples of output data produced by a set of algorithms, called experimental *observations*. An important parameter called the *significance level*, denoted by $\tilde{\alpha}$, facilitates a level of confidence in respect of the decision of whether or not to reject H_0. The value of $\tilde{\alpha}$ is compared with the outcome of the statistical test — represented by a so-called p-value. This value can be interpreted as the probability of obtaining an effect at least as extreme as that obtained in the data samples, under the assumption that H_0 is true [31]. Accordingly, if $p < \tilde{\alpha}$, then H_0 is rejected in favour of H_1 at a significance level of $\tilde{\alpha}$.

There are many statistical tests in the literature which may be used for hypothesis testing. Certain guidelines, however, exist for facilitating the process of choosing a most appropriate test. Some useful guidelines in this respect involve the type of experiment performed and the properties of the observed data. Non-parametric tests that do not make prior assumptions about the underlying distribution of the data are generally recommended when metaheuristics are being compared [31, 64, 82]. The non-parametric Friedman test together with the Nemenyi *post hoc* procedure is such a combination. This combination is employed for the purposes of algorithmic parameter evaluation and algorithmic performance comparison later in this dissertation [61, 64].

The Friedman test is an example of an omnibus test and is used to compare a set of at least two matched samples (*i.e.* tuples of corresponding data points) so as to assess whether there is a significant difference between at least two sample medians at a significance level of $\tilde{\alpha}$ [31]. Accordingly, the null hypothesis H_0 asserts that all sample medians are equal, whereas the alternative hypothesis H_1 asserts that at least two sample medians are not equal. If H_0 is rejected in favour of H_1, then the conclusion is therefore that a significant difference is present between at least two sample medians. The Friedman test involves the transformation of data points into so-called *Friedman ranks*, which are established by sorting the data points in ascending order (separately for each matching) and then assigning corresponding rank values to the samples [64].

The Nemenyi *post hoc* procedure is performed if the preceding Friedman test detects a statistically significant difference between at least two sample medians, *i.e.* if H_0 is rejected in favour of H_1 at a significance level of $\tilde{\alpha}$. This *post hoc* multiple-comparisons procedure enables the identification of individual differences between pairs of samples. The Nemenyi procedure does so by employing the Friedman ranks in order to perform two-tailed pairwise significant tests in respect of all sample pairs, whilst correcting for the multiple inferences it makes [61, 64]. These corrections are essential with a view to adhere to the experiment-wide significance level of $\tilde{\alpha}$. Hochberg and Tamhane [61] claimed that the Nemenyi procedure is a conservative *post hoc* procedure, attributable to the method by which it controls the *Type I error* (*i.e.* the incorrect rejection of a true null hypothesis).

The *Statistical Tests for Algorithms Comparison* [133] platform within the paradigm of the Python 3.7 programming language is utilised in this dissertation to perform the non-parametric tests discussed above. A significance level of $\tilde{\alpha} = 0.05$ is adopted for all statistical analyses in this dissertation.

2.2.2 Statistical learning algorithms

A considerable volume of algorithmic performance data are presented and analysed in this dissertation. In §2.2.1, the relevant statistical tests and procedures used to compare and evaluate the performances of metaheuristic and hyperheuristic optimisation approaches were discussed. In this section, the focus is on the statistical learning algorithms employed later in this dissertation to predict algorithmic performance. So-called *white-box* algorithms form the basis of algorithmic performance prediction performed in this dissertation — attributable to the explicit perusal of the prediction model enabled by this class of statistical algorithms. *Black-box* predictive algorithms, on the other hand, do not provide insight into the manner according to which predictions are made. Tree-based (white-box) algorithms are simple yet powerful prediction algorithms and therefore form the basis of the subsequent discussion

Tree-based algorithms

The selection of tree-based statistical learning algorithms can be substantiated by and ascribed to the context within which these algorithms are used — *i.e.* first, to predict algorithmic performance based on certain high-level characteristics (*i.e.* meta-features) of the data sets and, secondly, to scrutinise the most contributing characteristics in respect of the prediction task. The fundamental working of tree-based algorithms, and more specifically the underlying *learning* process, produces so-called rule formulations which elucidate the predictions made by the algorithms — holistically, these formulated rules constitute the resulting *decision tree*. A delineation of the fundamental concepts of decision trees follows.

Tree-based algorithms segment the so-called *predictor space*[1] into a number of distinct regions. In the case of classification problems, the *median* value of the region to which a new observation belongs is its predicted output value. An assimilation of the rules according to which the segmentation of the predictor space is performed represents the decision tree. An example of such as a decision tree is depicted graphically in Figure 2.3. Passenger records from the cruise liner, *The Titanic*, constitute the data set on which the decision tree in the figure is based. The prediction task on hand is whether or not a passenger would have survived the ship's demise, based on features such as age, gender, and number of siblings on board.

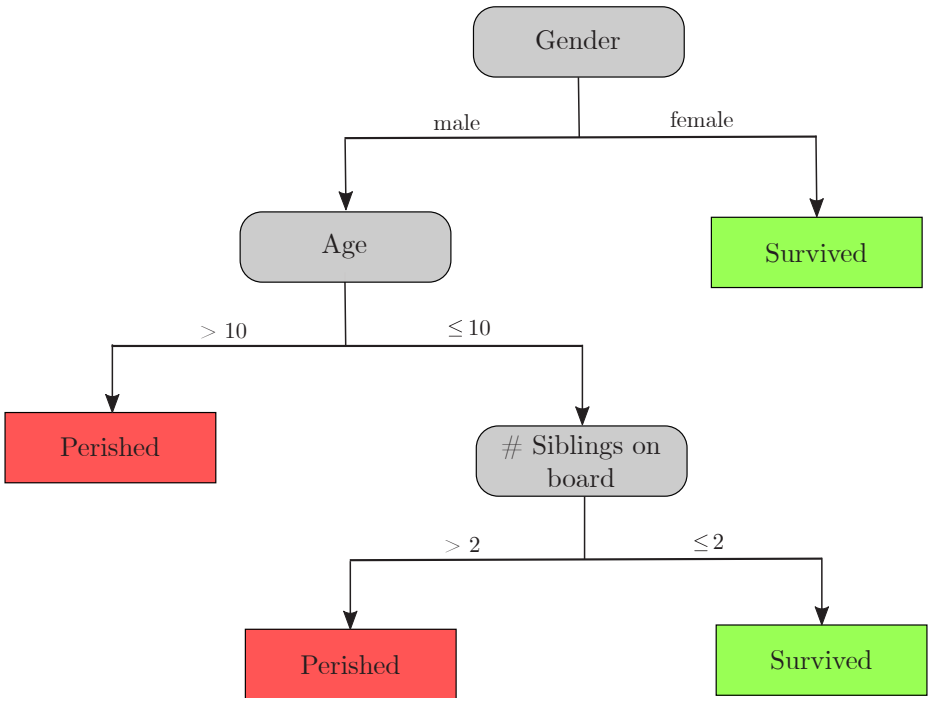


FIGURE 2.3: *A decision tree based on a data set pertaining to the survival of passengers travelling on the Titanic [124].*

The rule formulations depicted in Figure 2.3 state the following: Any male passenger older than ten years was likely to perish; in addition, any male passenger no older than ten years with more than two siblings was also likely to perish. Conversely, any female passenger was likely to survive; furthermore, any male passenger no older than 10 years with no more than two siblings was also likely to survive. A summary of the segmented predictor space (*i.e.* into different regions) is presented in Table 2.1. There are three different nodes in a decision tree, namely: *Terminal nodes* which correspond to the different regions in the tree, *internal nodes* which correspond to the points along the tree where the segmentation (of the predictor space) occurs, and, lastly, the *root node* which corresponds to the first split (*i.e.* gender). The connections between nodes are called *branches* [71, 95].

CART and C4.5

The construction of a decision tree is governed by an algorithmic procedure, of which there are several popular and powerful exemplars in the literature. The CART and C4.5 algorithms represent the two most prolific decision tree algorithmic procedures in the literature. The fundamental

[1] The set of all possible values that can be assumed by the input variables.

working of these two algorithms is by and large the same — consequently, they are described together in this section. Furthermore, the subsequent discussion is limited to classification trees, ascribed to the fact that this is the paradigm in which the algorithms are employed in this dissertation.

TABLE 2.1: *The different segmented regions for the Titanic data set. The corresponding rule formulations are also included.*

	Rule formulation			
Region	Gender	Age	# Siblings	Description
1	Female	—	—	Female passengers
2	Male	> 10	—	Male passengers older than ten years
3	Male	≤ 10	> 2	Male passengers ten years or younger with more than two siblings on board
4	Male	≤ 10	≤ 2	Male passengers ten years or younger with up to two siblings on board

The technique by which decision trees are generated is called *recursive binary splitting* and represents the first step of the CART and C4.5 decision tree algorithms. When *splitting* on the input variables that constitute the (entire) predictor space, *i.e.* $x_1, \ldots, x_i, \ldots, x_n$, each input variable x_i is considered along with all possible manners in which the split may be performed on this variable. In the case of the *age* node in Figure 2.3, for example, different threshold values, denoted by α, would be evaluated in respect of each of the possible values for the passenger age — accordingly, the predictor space is separated into regions for which $age \leq \alpha$ or $age > \alpha$. In the case of qualitative variables, on the other hand, such as *gender* or *country of birth*, a binary split may, for example, be performed according to which one class (or category) is assigned to the first branch whereas the remaining classes are assigned to the second branch. The CART algorithm employs this specific binary splitting technique in the context of qualitative variables whereas the C4.5 algorithm adopts a different approach in which the same number of branches as classes are created [71].

The so-called *purity* of a node represents the quality of a particular split and is, appropriately, employed to determine the best split. A classification problem comprising o classes forms the basis of the following discourse. An important measure, called the *Gini index*, is first reviewed which can be described mathematically as

$$G = \sum_{k=1}^{o} \hat{p}_{uk}(1 - \hat{p}_{uk}),$$

where $\hat{p}_{uk}$ represents the proportion of observations in the u^{th} region originating from the k^{th} class. Small Gini index values correspond to values of $\hat{p}_{uk}$ that are close to one or zero. Observations in region u may be classified into one of the o classes with a high level of confidence — corresponding to a high node purity. Furthermore, James *et al.* [71] state that high-quality splits also coincide with small values for the so-called *cross-entropy* measure, which is given by

$$H = -\sum_{k=1}^{o} \hat{p}_{uk} \log \hat{p}_{uk}. \tag{2.3}$$

The CART algorithm employs the Gini index so as to determine the node with the highest node purity, whereas in the case of the C4.5 algorithm, the cross-entropy measure is used [95].

Assuming the former measure, the predictor space is optimally segmented when the *information gain*

$$Gain(\text{S,D}) = G(S) - \sum_{V \in D} \frac{|V|}{|S|} G(V)$$

is maximised, where S and D represent the original predictor space and a partition thereof, respectively, whereas each V is a subset of S [124]. The process of node splitting is performed repeatedly until an appropriate stopping criterion is met. During each iteration, only one of the existing regions is split, rather than segmenting the entire predictor space. The maximum number of observations that ought to constitute each terminal node is a popular stopping criterion.

Poor predictive performance can be a result of a decision tree that is excessively large and is addressed by a so-called *pruning* process according to which the tree is shrunk to an appropriately sized *subtree* [71]. The estimated *classification* error rate[2], expressed mathematically as

$$E = 1 - \max_k \hat{p}_{uk},$$

is used to select an appropriately sized subtree — *i.e.* the subtree corresponding to the smallest classification error rate is selected. *Cost complexity pruning* and *reduced error pruning* represent two alternative pruning techniques which are applicable when the explicit evaluation of each of the possible subtrees becomes impractical.

The reduced error pruning approach is a simple method according to which all internal nodes (of the tree) are traversed in a *bottom-up* fashion. During the traversal, the replacement of each node with its most frequent class is evaluated in terms of an improvement in predictive performance. This procedure (of node replacement) is iterated until performance starts to decrease [134].

In the case of cost complexity pruning, on the other hand, a smaller pool of candidate subtrees is created by generating a sequence of good subtrees *via* the minimisation of the function

$$E(T) + \psi |T|,$$

where $E(T)$ denotes the error rate for a tree T and $|T|$ denotes the number of terminal nodes in T. Minimisation of the complexity of a tree and maximisation of its predictive performance represent two conflicting goals towards constructing good decision trees. The trade-off between these goals is controlled by a *tuning parameter* ψ. In the case where $\psi = 0$, the resulting tree is the original tree T_0 without pruning, whereas as ψ increases, larger trees are less likely to minimise the function — the size of the tree is therefore penalised. The value of ψ, together with its resulting subtree, is determined empirically based on its performance evaluation in respect of a separate partition of the original data set [71, 134].

2.3 Chapter summary

In this chapter, a detailed description of relevant mathematical and statistical preliminaries was provided in order to facilitate an understanding of the material presented in the remainder of the dissertation. In §2.1, important notions related to MOO were addressed, which included Pareto optimality, Pareto dominance, and Pareto rank. The FNSA and the crowding distance assignment algorithm, which form an integral part of the solution methodology proposed later in the dissertation, were reviewed in §2.1.1 and §2.1.2, respectively.

[2] Typically, the classification error rate is evaluated in respect of a separate partition of the original data set, *i.e.* a validation data set.

This was followed in §2.2 by a discussion on relevant statistical prerequisites. In §2.2.1, a discussion on two non-parametric statistical procedures employed later in this dissertation was presented. The Friedman test, along with the Nemenyi *post hoc* procedure, was reviewed briefly. These statistical procedures facilitate algorithmic parameter evaluations and relative algorithmic performance evaluations performed later in the dissertation. Finally, tree-based statistical learning algorithms were discussed in §2.2.2, which included a unified description of the most prolific decision tree algorithms, *i.e.* CART and C4.5, within the context of a classification prediction problem. Furthermore, prominent pruning techniques for improving predictive performance of the decision trees were discussed.

CHAPTER 3

Artificial Neural Networks

Contents

The aim in this chapter is to review the pertinent literature related to ANNs — more specifically, FNNs. The reader is introduced to the principal concepts and terminology found in the ANN literature so as to facilitate an understanding of the work presented in the remainder of this dissertation. The chapter opens with a discussion on ANN fundamentals, and this is followed by a more in-depth discourse on the foundational components of ANNs. A concise summary serves as the chapter's conclusion.

3.1 Fundamentals of ANNs

An ANN may be regarded as a mathematical or computational model inspired by the neurological physiology of the human brain and its approach towards solving everyday problems [41]. The working of an ANN is classified as a machine learning process, which resides within the realm of AI [22]. Problems deemed intellectually challenging for humans, but trivial for computers, were the main focus of AI during its early years [53]. Attention, however, gradually shifted towards the application of AI with respect to solving problems that are trivial for human beings, but are exceedingly difficult for computers. ANNs have, subsequently, enabled computers to solve these problems with relative ease through the basic principle of *learning from experience* [41]. Given a data set (input values and, if applicable, corresponding output values), a neural network is able to infer rules and relationships that are intrinsic to the features of the data. These rules and relationships would often otherwise be imperceptible to human beings, even when using standard analytical tools. Ultimately, these advanced inferencing capabilities allow ANNs to perform computationally complex tasks such as speech recognition and image classification [53].

In order to comprehend the inner workings of an ANN, one must first refer to its muse — the *biological neural network*. To do so, it is required to delve slightly into the field of *neuroscience*, *i.e.* the study of the nervous system, with the brain being the focal point. A key building block of a biological neural network is the *biological neuron*, illustrated graphically in Figure 3.1.

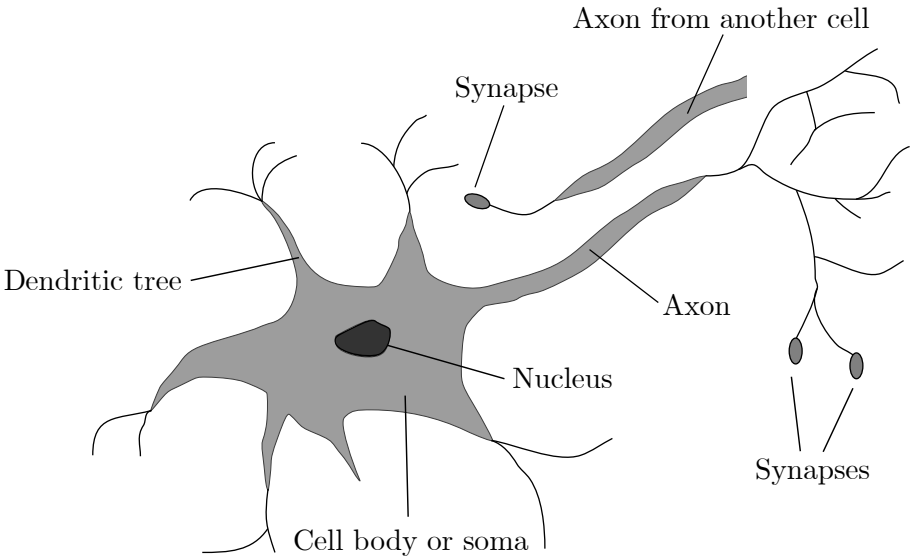


FIGURE 3.1: *A graphical representation of a generic biological neuron [142].*

It has been estimated that the human brain contains, on average, 100 billion of these biological neurons [99]. Biological neurons are linked to other, neighbouring biological neurons *via* synaptic junctions which connect the axon of the neuron to the dendrites (or cell bodies) of other neurons, as indicated in Figure 3.1. These synaptic junctions facilitate the propagation of signals (*i.e.* information) throughout the neural network, controlling brain activity in the short term and also being responsible for long-term changes in neuron connectivity [142]. The electrochemical reactions, which are responsible for signal propagation between neurons, modify incoming neuron signals by typically scaling the signals' frequencies [41, 142]. The input signals to a biological neuron, which vary in terms of strength, are aggregated to form a corresponding output signal. Whether the aggregated input is sent on for further processing depends on the neuron's inherent threshold. When the input is sufficient, the neuron *fires* and the signal is transmitted *via* the axon. It is thought that this operation forms the basis for learning within the brain [142].

Analogously, an ANN contains processing elements, called *artificial neurons*, which loosely mimic the aforementioned working of biological neurons [41]. It is somewhat apparent that the purpose of this *imitation* is, essentially, to attain the inherent quality of being able to learn from information (or data). A common mathematical model of the artificial neuron is illustrated graphically in Figure 3.2.

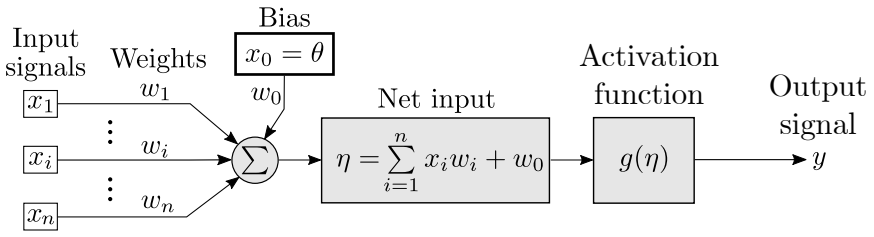


FIGURE 3.2: *A common mathematical model of an artificial neuron.*

For the sake of brevity, artificial neurons are henceforth merely referred to as *neurons* (unless otherwise stated). Within an ANN, neurons are connected by means of directed communication links which allow *input signals* $[x_1 \cdots x_i \cdots x_n]^T$ to pass through the network and, consequently, transmit information. A *weight* w_i accompanies the i-th communication link, with the purpose of scaling the input (as in the case of the biological neuron). Weights may be regarded as adjustable parameters of the network and control the input signal's influence, which may be either excitatory or inhibitory (with positive or negative weight values, respectively), depending on the intrinsic rules, relationships, and patterns present within the data. The ANN is able to *learn* these weights during a so-called *training* process so as to determine the underlying mathematical representation of the data [41]. The training process, governed by a training algorithm, is discussed later in greater detail.

In order to generate the neuron's output value y, the aggregated input (*i.e.* the weighted sum of the input values), denoted by η, is passed through an *activation function* $g(\cdot)$. Similar to the biological neuron, the artificial neuron fires and relays information when the aggregated input exceeds some inherent threshold — *i.e.* if some input is closely associated with some corresponding output, the weights and threshold of the neuron (or network) will reflect this. The activation function mathematically models this firing process. The exact timing of neuron firing is assumed to be inconsequential — it is only the frequency of firing that conveys valuable information [85]. The properties of activation functions are elaborated upon in a later section of this chapter. The resulting output value y is referred to as the neuron's *activation*. A *bias* value ($x_0 = \theta$) is also associated with each neuron. The purpose of incorporating this bias value is to shift the activation function either to the left or right, thus adjusting the neuron's inherent threshold. One of two approaches is typically followed when modelling the ANN's bias mathematically. The first is to set $w_0 = 1$ and only adjust the values of θ for each neuron individually, whereas the second method involves setting $\theta = 1$ and then adjusting the weighted connection, denoted by w_0, for each neuron.

The pattern according to which neurons are interconnected and arranged within an ANN is referred to as the network *architecture* or *structure* [41]. Typically, neurons are partitioned into separate network layers (or subdivisions), with the same activation function and weighted connection pattern being present at each neuron within a specific layer. In most ANNs, neurons within the same layer are either fully interconnected, or not connected at all. Figure 3.3 contains a generic graphical representation of an ANN, with neurons being represented by the nodes or circles, and weighted connections being represented by the links or arrows between the neurons. The graphical illustration in Figure 3.3 is that of the structure of a class of ANNs, called FNNs. The FNN is a sufficiently generic representation of ANNs as it contains the two most important

elements, namely neurons and weights. A more in-depth discussion on the different neural network topologies available in the literature, however, transpires later in the chapter. The first layer of an ANN is called the *input layer*, and the last, the *output layer*. The layers between the input and output layers are called *hidden layers*. Furthermore, neurons within these hidden layers are referred to as *hidden neurons*. According to Fausett [41], "the type of problem influences the choice of architecture, but does not uniquely determine it." Ascertaining an appropriate network architecture for solving a given problem is regarded by many as an inexact science, with varying approaches and heuristics being applied for this purpose.

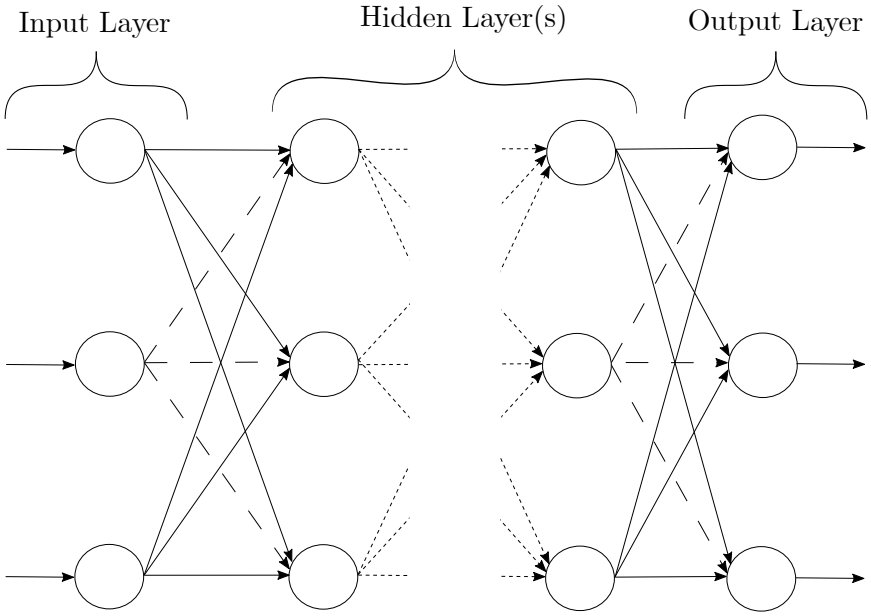


FIGURE 3.3: *A graphical representation of the general architecture of an FNN [145].*

In order for an ANN to be a sufficiently accurate computational model of the problem at hand, a suitable number of neurons in the *hidden layers* is required [53, 184]. A trade-off between *memorisation* and *generalisation* has to be taken into account when deciding upon a suitable number of neurons to include in an ANN [145]. Memorisation refers to the network being able to respond adequately to the input data during the training stage (the stage during which good network parameter values are learnt), whereas generalisation refers to the network's ability to respond correctly to input data during the testing stage (the stage during which the model is exposed to unseen data). An ANN's performance may therefore be expressed in terms of memorisation capability (performance in respect of the *training data set*) and generalisation capability (performance in respect of the *testing data set*) [162]. The different data set types typically considered in ANN tasks (*i.e.* training, validation, and testing) are discussed in greater depth later in this chapter.

Too few neurons may result in the network not being able to *learn* the dynamics/behaviour of the problem accurately, consequently resulting in poor memorisation and generalisation performance. Too many neurons, on the other hand, may result in good memorisation, but poor generalisation, with an unnecessary increase in computational requirements. Good memorisation and poor generalisation are collectively referred to as the network *over-learning* the problem. *Overfitting* is the term typically used to describe this undesired phenomenon. Kwok and Yeung [88] elucidated the aforementioned problem by means of a simple analogy in the context of polynomial curve fitting. As illustrated in Figure 3.4, too few non-zero polynomial coefficients will result in the polynomial not being able to capture the underlying functional representation (*i.e.* too simple a

model), whereas too many non-zero coefficients will result in the polynomial fitting the *noise* in the data (*i.e.* too complicated a model). In the context of ANNs, the coefficients are represented by the neurons (and their accompanying weights). Determining a suitable number of neurons for the problem at hand is a challenge faced by all neural network designers and users.

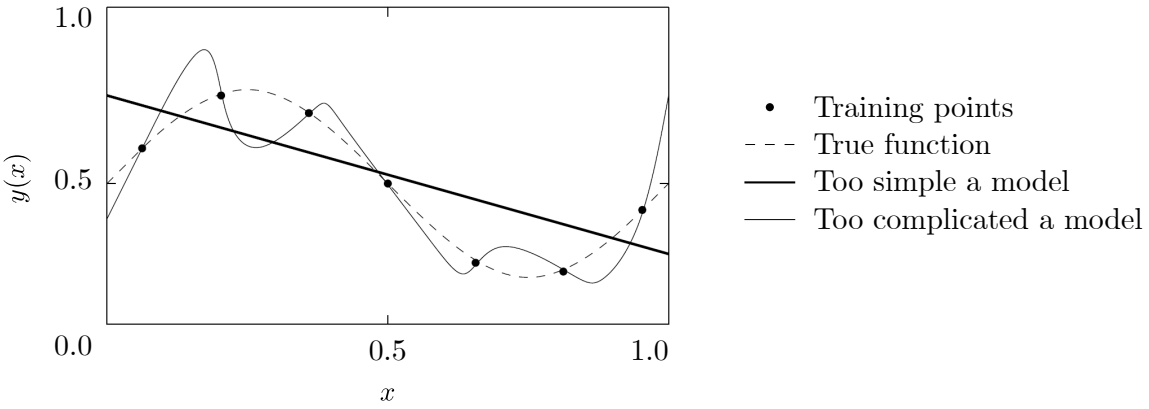


FIGURE 3.4: *An analogous curve fitting problem used to explain the problem of including too few or too many neurons in an ANN. The inputs and outputs are represented by x and y, respectively [145].*

Factors such as the degree of non-linearity and dimensionality of the problem at hand, the size of the available data set, and inherent noise present within the data all provide an indication of the number of hidden neurons required [6, 53, 184]. The following rule of thumb may be adopted in this respect: More neurons are required for highly non-linear problems, whereas *smoother* problems require fewer neurons. Determining exactly (and efficiently) the number of hidden neurons required, is regarded as an "open question" — attributable to the black-box nature of ANNs. There are, however, three approaches towards determining a suitable number of neurons for inclusion in an ANN so as to achieve good generalisation performance. The first is a trial-and-error (or empirical) approach in which different numbers of neurons are experimented with so as to find the best performer [184]. Based on empirical findings, a number of heuristics have been proposed for estimating a suitable number of hidden neurons and, according to Heaton [60], the most commonly adopted heuristic is the following: "The optimal size of the hidden layer is usually between the size of the input and the size of the output layers." The second approach is an *adaptive* process — or *optimisation* process — whereby the number of neurons are treated as decision variables and the objective function is represented by some network performance measure, such as network error or accuracy, for example [32, 88]. Towards this end, researchers have also proposed *structural optimisation algorithms* that attempt to find an optimal number of hidden layers in an ANN for a given learning problem [67, 179]. The third approach involves matching the ANN's model complexity with the problem's complexity, and employs techniques such as constructive algorithms, network pruning, and dropout [88, 129, 155]. Addressing the problem of overfitting (without directly altering the network architecture) may be accomplished by means of the following techniques: Training with noise, performing cross-validation, employing ensemble methods, using parameter *regularisation* methods, performing data set augmentation, and enforcing early stopping, to name but a few [53]. Some of the more popular techniques, mentioned above, are discussed in greater depth later in this chapter.

Another important consideration, in terms of the network architecture (or structure), is deciding upon a suitable number of hidden layers in the ANN. According to Zhang and Gupta [184], approximating arbitrary non-linear functions necessitates at least one hidden layer, whereas non-linear problems that repeatedly exhibit certain localised behavioural components in different regions of the problem space, require networks with one or two hidden layers. For many problems,

a three-layer network (one input layer, one output layer, and one hidden layer) has the same processing capability as a four-layer network, although the latter requires more hidden neurons and thus increases the associated computation time. Bebis and Georgiopoulos [3] further state that, for classification problems, a network with two hidden layers is capable of approximating an arbitrary non-linear function and generating any complex decision region. With arbitrary accuracy, any continuous function can be approximated with only one hidden layer. A more general theoretical finding by Hornik and Stinchombe [66] is that an ANN with one hidden layer in conjunction with activation functions that are "appropriately smooth" is capable of approximating an arbitrary function as well as its derivatives, with "arbitrary" accuracy.

3.2 Activation functions

As mentioned in §3.1, an activation function (sometimes also referred to as a *transfer function*) is applied to the weighted sum of the neuron's input signals (together with its bias) in order to calculate the corresponding output signal y. Within the context of Figure 3.2, the relevant computation is

$$y = g\left(\sum_{i=1}^{n} x_i w_i + w_0\right).$$

The purpose of the activation function $g(\cdot)$ is to model the *firing* process of neurons mathematically. Either linear or non-linear activation functions may be used in ANNs. Non-linear functions are, however, preferred for the following reason: A single-layer network with non-linear activation functions exhibits the same level of functionality as a multi-layer network with linear activation functions [41]. Furthermore, neural networks are capable of approximating any non-linear function, which is partly attributed to the use of activation functions exhibiting non-linearities [72, 90]. Non-linear functions do, however, impose an increased computational burden. Typically, the activation function is some increasing function of the total input to the neuron [85]. For simplicity, it is assumed that the bias is equal to zero (*i.e.* $\theta = 0$) for all activation function plots in this section — thus w_0 is omitted. The net input to each neuron, denoted by η, is therefore

$$\sum_{i=1}^{n} x_i w_i,$$

which represents the weighted sum of the synaptic inputs.

The first noteworthy activation function is the *identity* function. This function is given by

$$g(\eta) = \eta \qquad \text{for all } \eta, \tag{3.1}$$

and is typically employed by neurons in the input layer (*i.e.* input neurons) [145], where the input signal is simply transmitted onwards unaltered — illustrated graphically in Figure 3.5(a). This function may also be employed by neurons within the output layer (*i.e.* output neurons) when the target (or output) values are continuous [41].

Another activation function commonly employed is the *Heaviside* or *binary step function*, which may be seen in Figure 3.5(b). This function is given by

$$g(\eta) = \begin{cases} 0, & \eta < 0, \\ 1, & \eta \geq 0 \end{cases} \tag{3.2}$$

and is usually employed in single-layer networks used for *classification* problems [41], with the function's purpose being to convert the net input (*e.g.* a continuous variable) to a binary output

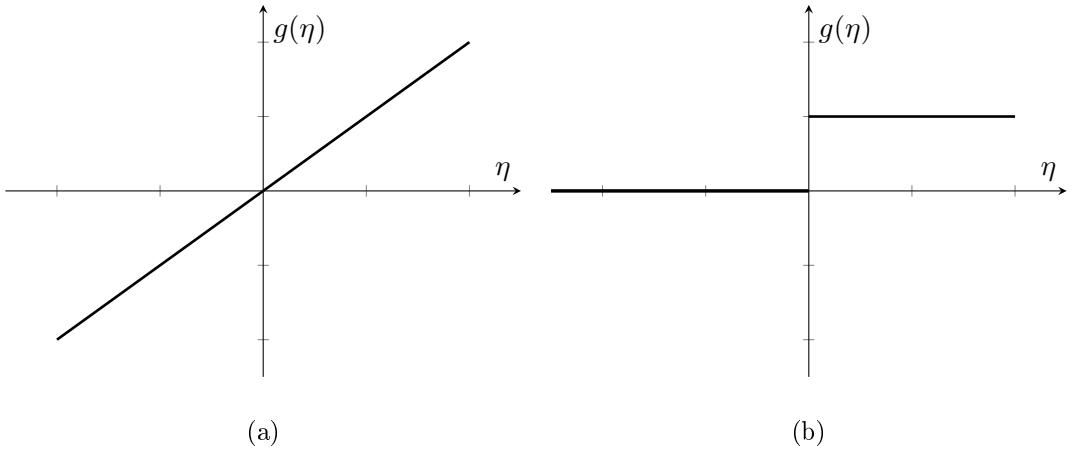


FIGURE 3.5: *Two of the simplest activation functions: (a) The identity function* (3.1) *and (b) the Heaviside function* (3.2).

value. Depending on the net input value a (and a certain threshold value θ), the function produces a corresponding output value. The binary step function is analogous to an *on-or-off* process where the neuron is either fully *stimulated* by the input signal(s) (an excitatory relationship), or not stimulated at all (an inhibitory relationship), due to some intrinsic relation among the features of the data. One of the very first and simplest neuron models, the *McCulloch-Pitts* model, employs this activation function [58]. In this model (identical to the one presented in Figure 3.2, but with the step function as the activation function), the neuron produces an output of $+1$, if the weighted sum of the inputs is non-negative, or a value of 0 otherwise. A key drawback associated with the use of this function is its lack of differentiability, due to its discontinuity at $\eta = 0$. A necessity when employing certain training algorithms, such as the standard *backpropagation algorithm* in conjunction with gradient descent, is the requirement that the activation function be differentiable [41].

The *two-breakpoint piecewise linear* function is another popular activation function. This function is presented graphically in Figure 3.6 and is given by

$$g(\eta) = \begin{cases} 0, & \eta \leq 0 - \frac{1}{2}, \\ \eta, & 0 - \frac{1}{2} < \eta < 0 + \frac{1}{2}, \\ 1, & \eta \geq 0 + \frac{1}{2}. \end{cases} \tag{3.3}$$

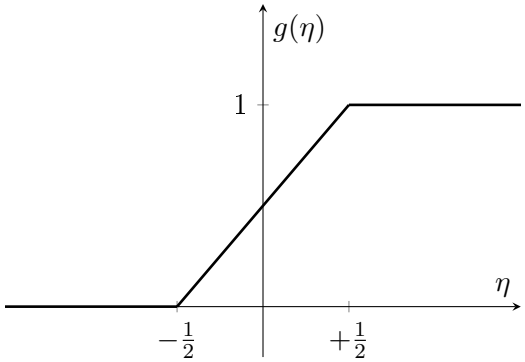


FIGURE 3.6: *The two-breakpoint piecewise linear function* (3.3).

It is normally assumed that the gradient (*i.e.* amplification factor) within the region of linear increase is unity, but this may vary depending on the problem at hand [58]. The step function

is the special case of the two-breakpoint piecewise linear function obtained when the gradient is infinitely large within the region of linear increase.

One of the most common activation functions is the s-shaped *sigmoid* function [58]. This non-linear, strictly increasing function is given by

$$g(\eta) = \frac{1}{1 + e^{(-s\eta)}} \tag{3.4}$$

and is illustrated graphically in Figure 3.7. The output of the sigmoid activation function may be interpreted as a probability, which is convenient when employed by the output neurons of an ANN to solve classification problems. A different approach is also possible. Suppose the input to a layer of n neurons is $\boldsymbol{\eta} = [\eta_1 \cdots \eta_k \cdots \eta_o]^T$. The *softmax* activation function can be used to calculate the categorical probability distribution — *i.e.* the probability that class $k \in \{1, \ldots, o\}$ is applicable, given the n inputs. The softmax-layer activation function, of the form

$$g(\boldsymbol{\eta})_k = \frac{e^{\eta_k}}{\sum_{r=1}^{o} e^{\eta_r}}, \quad k \in \{1, \ldots, o\}, \tag{3.5}$$

is applied to determine these categorical probabilities, with the output being a vector of probabilities with the constituent values adding up to 1.

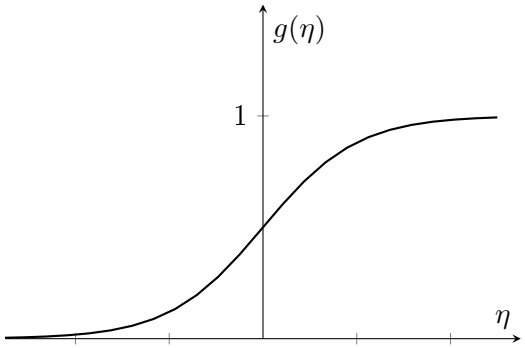


FIGURE 3.7: *The logistic sigmoid function* (3.4).

The sigmoid (or softmax function) effectively maps a real-valued number into the real interval $(0, 1)$. Large negative input values result in an output value of approximately 0, whereas large positive input values result in an output value of approximately 1. Given significantly large/small input values, the phenomenon of *saturation* may occur as the gradients are extremely small at the left- and right-most ends of the function. There is a wide variety of sigmoid functions, with the aforementioned sigmoid function often being called the *logistic* sigmoid function. The parameter s in (3.4) represents the *slope* of the function. When this parameter is infinitely large, the function again simply approximates a binary step function. A key difference between the step function and the sigmoid function, however, is that the former assumes a value of 0 or 1, whereas the latter assumes a range of continuous values in the real interval $(0, 1)$. What contributes to the function's popularity is its somewhat trivial derivative, given by

$$g'(\eta) = sg(\eta)(1 - g(\eta)),$$

which is simply expressible in terms of the function itself. This property renders the sigmoid function an attractive function when used in conjunction with certain training algorithms, as the computational cost of evaluating the derivative is relatively small.

Sigmoid functions have been employed widely in the literature due to their simple and effective emulation of the neuron firing process and, more importantly, the simple nature of evaluating

their derivatives [41, 53, 58, 75, 101, 142]. This function, however, exhibits the significant disadvantage of saturation — derivatives are approximately zero for significantly large/small inputs [75]. This is problematic due to signals becoming unnecessarily suppressed, but may be avoided by means of correctly scaling the inputs and initialising the weights, as discussed later in this chapter.

The binary step, piecewise linear, and logistic sigmoid functions all assume values within the real interval $[0, 1]$. Occasionally, an activation function that ranges from -1 to $+1$ is required. The *signum function*, given by

$$g(\eta) = \begin{cases} -1, & \eta < 0, \\ 0, & \eta = 0, \\ +1, & \eta > 0, \end{cases}$$

achieves this requirement and is very similar to the binary step function, except that it assumes values within the interval $[-1, 1]$.

Another important activation function is the *hyperbolic tangent function*, which is a member of the class of sigmoid functions, and, as a result, has a similar shape to that shown in Figure 3.7. The function is given by

$$g(\eta) = \tanh(\eta)$$

and assumes a continuous range of values within the real interval $(-1, 1)$. The derivative of this function is

$$g'(\eta) = (1 + g(\eta))(1 - g(\eta)),$$

which is, again, expressible in terms of the function itself — an important characteristic when applying certain training algorithms (as previously mentioned). The sigmoid and hyperbolic tangent functions are two of the most popular activation functions used in neural networks [50]. The latter suffers from the same drawback as the former — saturation at both ends of the function. The hyperbolic tangent function is superior to the logistic sigmoid function due to it being zero-centred, which is an important characteristic during training as it prevents undesirable dynamics during gradient updates of the weights [50, 75].

An important factor to consider when selecting an activation function is the nature of the input and output data. For binary input data, both the logistic sigmoid function and the hyperbolic tangent function are typically employed [41]. For a specific problem it may be necessary that the range of the activation function coincides with the range of the target variable. The popular logistic sigmoid function may be modified so as to assume any range $[a, b]$ of output values. This may be achieved by defining the parameters $\gamma = b - a$ and $\eta = -a$. The rescaled sigmoid function can, therefore, be defined as

$$v(\eta) = \gamma g(\eta) - \eta,$$

and the derivative of $v(\eta)$ is given by

$$v'(\eta) = \frac{1}{\gamma}\Big[\eta + v(\eta)\Big]\Big[\gamma - \eta - v(\eta)\Big].$$

Translating an activation function to the right or left is also a necessity, and is accomplished by the neuron's bias (as mentioned earlier). The logistic sigmoid function can be further modified by adjusting its steepness (or slope), which is dictated by the parameter s, as indicated in Figure 3.8. This slope parameter — referred to as a hyper-parameter — may be adjusted so as to further improve the network's performance. The slope is determined by means of various heuristics, such as trial-and-error or treating it as a decision variable when following an optimisation approach. In the latter case, this hyper-parameter is, effectively, *learnt*.

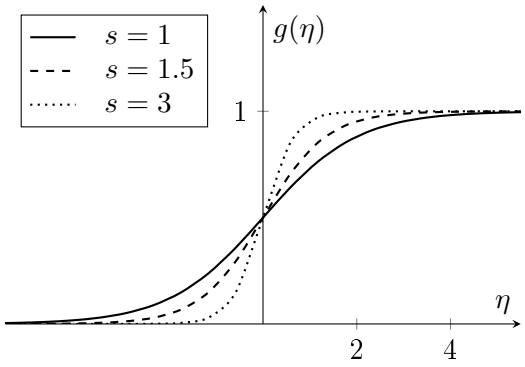


FIGURE 3.8: *The logistic sigmoid function* (3.4) *with different slope values.*

According to Karpathy [59, 75], the *Parametric Rectified Linear Unit* (PReLU) activation function,

$$g(\eta) = \begin{cases} \alpha\eta, & \eta < 0, \\ \eta, & \eta \geq 0, \end{cases} \tag{3.6}$$

has gained noteworthy popularity in recent years. Figure 3.9 contains a graphical representation of this activation function. When compared to the logistic sigmoid and hyperbolic tangent functions, the PReLU function is found to speed up convergence of the gradient descent training algorithm substantially [75]. Additionally, this function is computationally inexpensive to evaluate, which is a desirable quality in larger neural networks. The *Rectified Linear Unit* (ReLU) activation function is the special case of the PReLU function, in the case where $\alpha = 0$, and is also used regularly. A notable disadvantage of this special case, however, is its tendency to cause neurons to *die* prematurely when employed in conjunction with certain training algorithms. In essence, neurons stop firing across the entire data set due to undesirable weight updates rendering η less than 0 (or negative). The problem of dying neurons may be mitigated by introducing a gradient for input values less than 0. The gradient parameter value α may be universal for all neurons, or specific to each neuron. Furthermore, this hyper-parameter is learnable, in the sense that it can be optimised (together with the network weights) during the training process so as to further improve the network performance. According to Glorot *et al.* [50], the use of the ReLU activation function in large ANNs delivers promising results, although, the PReLU function delivers a significant performance advantage over the ReLU function with negligible added computational cost [59].

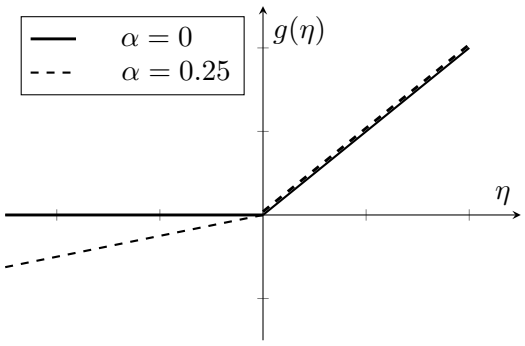


FIGURE 3.9: *The PReLU activation function* (3.6).

3.3 ANN types

In the previous section, specific reference was made to one class of ANNs — feedforward networks. There are, however, other significant network types exhibiting some noteworthy differences. A brief description of FNNs as well as the other network types, deemed influential and consequential in the current ANN literature follows.

In terms of *feedforward* networks, the flow of data within this type of network is transmitted strictly in a forward direction. The key or defining feature is that no *feedback* connections or loops are present, *i.e.* no neuron's output contributes towards a neuron preceding it within the network structure. Another distinguishing factor is the implicit assumption that the features of an input-output pair are independent, which may hold true for some learning problems, but not all. An FNN with a structure comprising only one hidden layer, is classified as a *single-layer feedforward neural network* (SFNN). An FNN with more than one hidden layer is considered a *multi-layer feedforward neural network* (MFNN) or *multi-layer perceptron*. An MFNN is capable of solving more complicated problems than single-layer networks, but with added difficulty during training as a result of its increased complexity [53].

Another notable network type is *recurrent* networks. This type of network is able to process and analyse sequential data (*e.g.* time-series data) with exceptional performance [181]. Feedback connections or loops are present in recurrent neural networks, which remain similar to FNNs on a fundamental level (*i.e.* how signals are propagated throughout the network). These feedback connections allow additional signals to be fed from any hidden layer or output layer, back to preceding layers in the network — *i.e. directed cycles* form between neurons. Consequently, the network is able to respond *temporally* and dynamically to its input values. *Memory* is an inherent trait or feature of these networks. As opposed to FNNs, learning problems with input-output pairs exhibiting direct dependence are the focus area of recurrent neural networks.

The last major network type is *convolutional* networks. Problems involving spatially dependent data, such as images (neighbouring pixels usually exhibit dependence), are efficiently addressed by convolutional neural networks [46]. An assumption is explicitly made that the inputs to the network are images. This, in turn, allows one to encode specific features into the network architecture. The *inter-dependence* present within the data allows the network to exert its computational effort where the profit is the highest — resulting in considerably fewer weights than in FNNs. Convolutional networks excel at image classification or processing tasks.

The term *deep neural network* (DNN) refers to an ANN with *many* hidden layers.

As discussed in some detail later in this chapter, the fundamental premise of training remains largely the same among all of the major network types mentioned above. For the sake of simplicity (and as a result of hardware constraints), however, FNNs are chosen as the basis for discussion in this chapter.

3.4 FNNs

A relatively simple SFNN architecture is assumed as the basis of the discussion in the remainder of this chapter, which aims to elucidate the inner workings of an FNN as well as define the accompanying notation. Figure 3.10 contains a graphical illustration of an SFNN with n input neurons, o output neurons, m hidden neurons, and $h = 1$ hidden layers. The bracketed superscripts in the figure denote the relevant layer, adopting the convention that $f = 0$ for the input layer, $f = 1$ for the hidden layer, and $f = 2$ for the output layer. The network is presented with input vector (or training example) $\boldsymbol{x} = [x_1 \cdots x_i \cdots x_n]$. Typically, the number of input and output

neurons correspond to the number of independent and dependent variables in the problem data set, respectively. As mentioned in §3.1, one of two approaches is usually followed when modelling the biases of a network. In the subsequent discussion, each of the biases in the input layer (or hidden layer) are represented by one layer-bias set to 1, with the outgoing weighted connections being treated as adjustable parameters, as opposed to omitting the weighted connections and adjusting the bias for each neuron individually (this choice is a matter of personal preference).

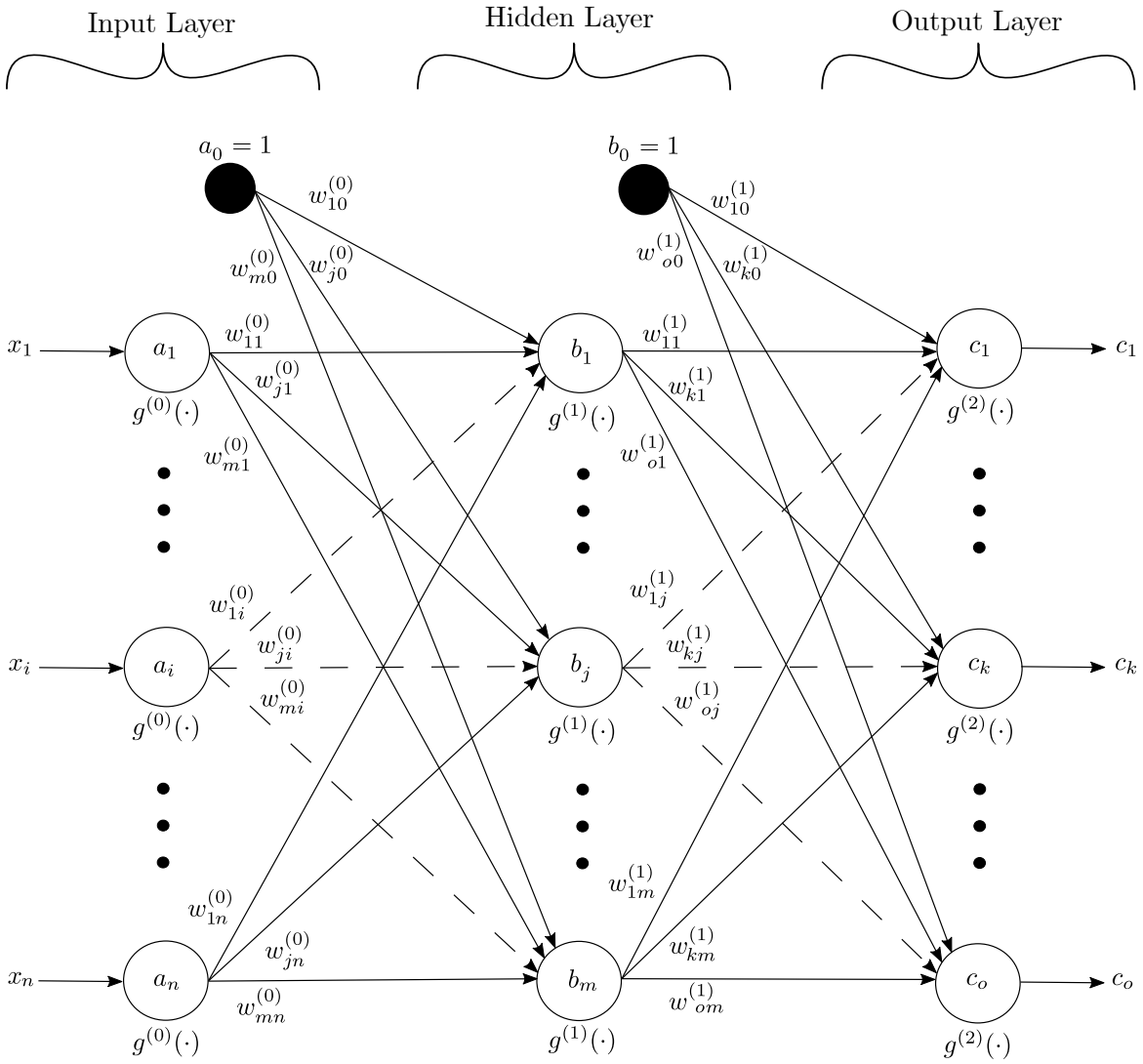


FIGURE 3.10: *Graphical representation of an SFNN with one hidden layer.*

The main components of an SFNN (together with the relevant notation) are:

- Input layer activations, denoted by $\boldsymbol{a} = [a_1 \cdots a_i \cdots a_n]^T$ (excluding bias) or $\tilde{\boldsymbol{a}} = [a_0\ a_1 \cdots a_i \cdots a_n]^T$ (including bias) where $a_0 = 1$,
- hidden layer activations, denoted by $\boldsymbol{b} = [b_1 \cdots b_j \cdots b_m]^T$ (excluding bias) or $\tilde{\boldsymbol{b}} = [b_0\ b_1 \cdots b_j \cdots b_m]^T$ (including bias) where $b_0 = 1$,
- output layer activations, denoted by $\boldsymbol{c} = [c_1 \cdots c_k \cdots c_o]^T$,

- weights corresponding to the connection between input neuron $i \in \{0, \ldots, n\}$ and hidden neuron $j \in \{1, \ldots, m\}$, denoted by $w_{ji}^{(0)}$, and contained within a weight-matrix $\boldsymbol{W}^{(0)}$,
- weights corresponding to the connection between hidden neuron $j \in \{0, \ldots, m\}$ and output neuron $k \in \{1, \ldots, o\}$, denoted by $w_{kj}^{(1)}$, and contained within a weight-matrix $\boldsymbol{W}^{(1)}$ with the entire network's weights being contained within an ordered list $\boldsymbol{w} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}\}$,
- input layer activation functions, denoted by $g^{(0)}(\cdot)$,
- hidden layer activation functions, denoted by $g^{(1)}(\cdot)$, and
- output layer activation functions, denoted by $g^{(2)}(\cdot)$.

Given the aforementioned information, it is possible to express an SFNN analytically in terms of a mathematical function. An identity function is employed in the first layer of the network, resulting in the activation of each input neuron being equal to the input itself, *i.e.* $a_i = x_i$ due to the fact that $x_i = g_i^{(0)}(x_i)$ for $i \in \{1, \ldots, n\}$. These input layer activations are subsequently used to calculate the *net input* to hidden neuron j, denoted by $\eta_j^{(1)}$ for $j \in \{1, \ldots, m\}$. This net input is given by

$$\eta_j^{(1)} = \sum_{i=1}^{n} w_{ji}^{(0)} a_i + w_{j0}^{(0)} = \sum_{i=0}^{n} w_{ji}^{(0)} a_i \tag{3.7}$$

and is used to calculate the j-th hidden neuron activation, denoted by b_j, by employing the associated activation function $g^{(1)}(\cdot)$. In particular,

$$b_j = g^{(1)}\left(\eta_j^{(1)}\right). \tag{3.8}$$

Alternatively, the hidden layer's activations $\boldsymbol{b}$ may be expressed as the product of the weights between the input layer and the hidden layer, denoted by $\boldsymbol{W}^{(0)}$, and the input activations $\tilde{\boldsymbol{a}}$ (including bias). The activation functions employed in this layer, denoted by $g^{(1)}(\cdot)$, is subsequently applied. This mathematical operation is given by

$$\begin{bmatrix} b_1 \\ \vdots \\ b_j \\ \vdots \\ b_m \end{bmatrix} = g^{(1)}\left(\begin{bmatrix} w_{10}^{(0)} & w_{11}^{(0)} & \cdots & w_{1i}^{(0)} & \cdots & w_{1n}^{(0)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j0}^{(0)} & w_{j1}^{(0)} & \cdots & w_{ji}^{(0)} & \cdots & w_{jn}^{(0)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{m0}^{(0)} & w_{m1}^{(0)} & \cdots & w_{mi}^{(0)} & \cdots & w_{mn}^{(0)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_i \\ \vdots \\ a_n \end{bmatrix} \right),$$

or more compactly by

$$\boldsymbol{b} = \boldsymbol{g}^{(1)}\left(\boldsymbol{W}^{(0)} \tilde{\boldsymbol{a}}\right),$$

where $\boldsymbol{g}^{(1)}$ represents the vector of activation functions, of length m. Although all the activation functions contained within this vector are the same, the notation is necessitated for future analysis.

The net input to output neuron k, denoted by $\eta_k^{(2)}$ for $k \in \{1, \ldots, o\}$, is calculated similarly, yielding

$$\eta_k^{(2)} = \sum_{j=1}^{m} w_{kj}^{(1)} b_j + w_{k0}^{(1)} = \sum_{j=0}^{m} w_{kj}^{(1)} b_j. \tag{3.9}$$

The activation of the k-th output neuron, denoted by c_k, is obtained by applying the corresponding activation function $g^{(2)}(\cdot)$ to the net input calculated in (3.9). This calculation yields

$$c_k = g^{(2)}\left(\eta_k^{(2)}\right). \tag{3.10}$$

By combining (3.7)–(3.10), the mathematical function that analytically represents an SFNN, as depicted in Figure 3.10, may therefore be expressed as

$$c_k = g^{(2)}\left(\sum_{j=0}^{m} w_{kj}^{(1)} g^{(1)}\left(\sum_{i=0}^{n} w_{ji}^{(0)} a_i\right)\right), \tag{3.11}$$

Similar to the hidden layer activations, the output layer activations may be expressed as

$$\begin{bmatrix} c_1 \\ \vdots \\ c_k \\ \vdots \\ c_o \end{bmatrix} = g^{(2)}\left(\begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & \cdots & w_{1j}^{(1)} & \cdots & w_{1m}^{(1)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{k0}^{(1)} & w_{k1}^{(1)} & \cdots & w_{kj}^{(1)} & \cdots & w_{km}^{(1)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{o0}^{(1)} & w_{o1}^{(1)} & \cdots & w_{oj}^{(1)} & \cdots & w_{om}^{(1)} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_j \\ \vdots \\ b_m \end{bmatrix}\right),$$

or more compactly as

$$\boldsymbol{c} = \boldsymbol{g}^{(2)}\left(\boldsymbol{W}^{(1)}\tilde{\boldsymbol{b}}\right). \tag{3.12}$$

Evidently, an FNN may be expressed in terms of a non-linear function of the adjustable network weights. Adjusting the weights of the network — referred to as *learning* — is governed by a so-called *training algorithm*, as discussed later in this chapter.

3.5 Network learning and training

Arguably, the most fundamental part of an ANN is *learning* — *i.e.* the process during which the network is trained. The aim in this section is to elucidate the principal concepts and notions that pertain to this fundamental process. The main learning paradigms of ML (and ANNs) are briefly discussed. This is followed by a mathematical representation of the process of supervised learning. An important step, before training transpires, is that of *weight initialisation* and is subsequently touched upon, before providing a discourse on the different data set types. Finally, a discussion on the different paradigms pertaining to when weights are updated, conclude this section.

3.5.1 Fundamentals of learning

The ability to *learn* from experience (or by example), as well as to *improve* performance throughout this learning process, are two significant attributes that render ANNs an especially powerful utility in the field of ML [41, 58]. According to a definition adapted by Haykin [58] (which is based on the seminal work of Mendel and McLaren [102]), learning is defined as "a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded." These *free parameters* primarily consist of the weights and biases, but may also include other hyper-parameters such as the number of

hidden neurons, the number of hidden layers, or parameters inherent to the activation functions, *e.g.* the gradient parameter α in the PReLU function (3.9). The process of stimulation adopts an interactive and iterative nature, with the desired consequence being that the network attains knowledge about its environment (represented by the data) after each iteration of learning. Figure 3.11 serves as a simple visual representation of this learning process. Adjusting the parameters during the learning process is governed by a so-called *training algorithm*[1]. The advocacy of efficient training algorithms are, by and large, due to the fact that real-world ANN applications require the adjustment of several thousand weights — rendering a brute-force approach intractable [109]. A discussion on the most pertinent literature related to ANN training algorithms follows later in this chapter.

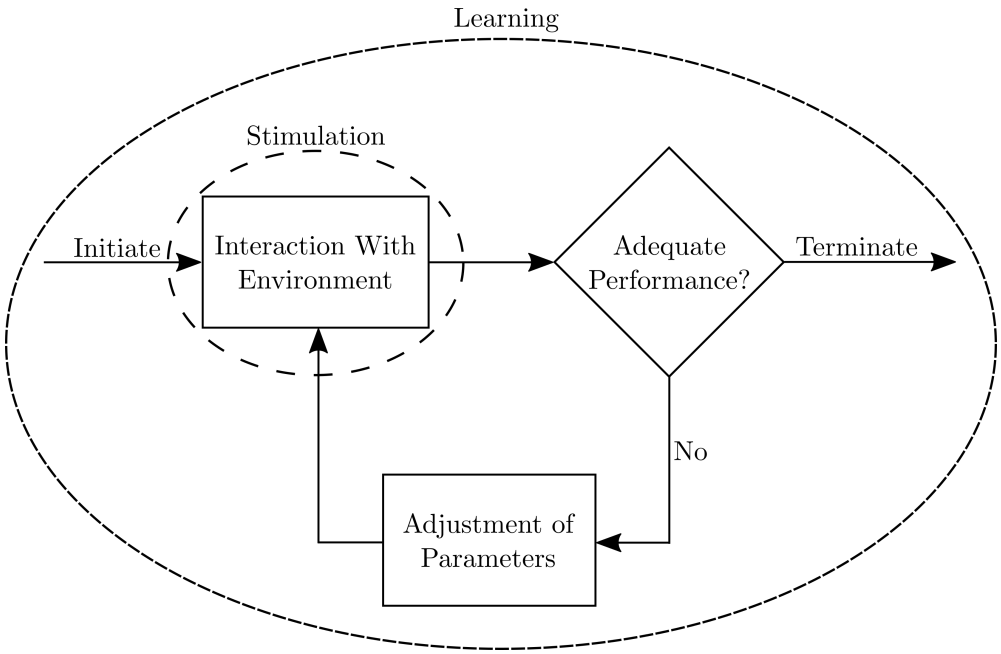


FIGURE 3.11: *The process of learning.*

Fundamentally speaking, an ANN's learning process comprises the following three components:

1. *Stimulation* by the environment,
2. Subsequent *adjustment* of synaptic weights (if necessary), and
3. Modified/new *interaction* with the environment due to changes in the network's internal structure.

3.5.2 Learning paradigms

The learning paradigm to which the ANN is subjected influences the approach adopted towards selecting and implementing the training algorithm [41, 85, 142]. There are four main learning paradigms, namely *supervised learning*, *unsupervised learning*, *semi-supervised learning*, and *reinforcement learning*.

[1] Also referred to as a *learning algorithm* or *learning rule*, depending on the literature consulted.

In the supervised learning paradigm, *labelled* data are presented to the network, which comprise *independent* variables (*i.e.* input features) and corresponding *dependent* variables (*i.e.* output/target variables), the aim being to approximate the underlying function that best represents the input-to-output mapping [58]. *Classification* and *regression* are the two main supervised learning problem classes. The former deals with the assignment of a vector of input features to one of a finite number of discrete categories. Handwriting classification, speech recognition, computer vision, and spam detection are all examples of classification problems [24, 48, 73, 92, 147]. On the other hand, regression deals with problems having output of a continuous nature. Examples include weather, energy load, financial, and manufacturing process yield forecasting [7, 65, 165, 173].

In an unsupervised learning paradigm, the ANN is presented with input data only (*i.e. unlabelled* or unstructured data) consisting of input vectors and no corresponding output/target variables. The training algorithm adjusts the weights such that similar inputs are grouped together and, subsequently, are assigned to the same output [145]. The objective is to find some underlying structure (or pattern) within the data. According to Bishop [7], data clustering, dimensionality reduction, and density estimation are the main problem classes of application within this learning paradigm.

Semi-supervised learning utilises a combination of labelled and unlabelled data. In this paradigm, both supervised learning tasks and unsupervised learning tasks may be performed [186]. When performing a supervised learning task, *inductive* learning may be employed to improve the input-to-output mapping by incorporating unlabelled data. Alternatively, in *transductive* learning, the focus lies on inferring the correct labels for the unlabelled data. When performing an unsupervised learning task, performance may be enhanced by incorporating labelled data. According to Zhu [186], natural language processing, computer vision, and bio-informatics are all application fields in which semi-supervised learning flourishes.

Lastly, within the paradigm of reinforcement learning, the network is exposed to some dynamic environment and is tasked with determining a course of action (*i.e.* policy) aimed at maximising (or minimising) some world-specific reward (or punishment), through a trial-and-error process. Essentially, this learning paradigm may be defined as a *goal-directed* computational learning approach [160]. Control theory, inventory management, and the finance industry are all fields to which this paradigm may be applied.

As mentioned in §1.3, the scope of this dissertation is restricted to supervised learning only. The remainder of the work contained in this dissertation should therefore be viewed within a supervised learning context. A more in-depth description of the process of supervised learning follows.

3.5.3 Mathematical representation of the process of supervised learning

Figure 3.10 is, once again, used as a basis for facilitating the discussion in this section. Recall that in the case of an SFNN (as depicted in Figure 3.10), the list $\boldsymbol{w}$ comprises the weight-matrices $\boldsymbol{W}^{(0)}$ and $\boldsymbol{W}^{(1)}$. A set of inputs, consisting of Q input vectors, is usually presented to the network, and is denoted here by $\mathcal{X} = \{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^q, \ldots, \boldsymbol{x}^Q\}$. The multivariate function f that represents the SFNN is, simply, given by

$$\boldsymbol{c}^q = f(\boldsymbol{x}^q; \boldsymbol{w}), \qquad q \in \{1, \ldots, Q\}, \tag{3.13}$$

which is the more general form of the analytical representation in (3.12). In a typical training process, both an input vector $\boldsymbol{x}^q$ (*i.e.* set of input features representing the independent variables) as well as the corresponding target vector $\boldsymbol{y}^q$ (*i.e.* set of target variables representing the

dependent variables) are known *a priori*, and the weights in $\boldsymbol{w}$ are methodically adjusted so as to approximate an appropriate functional mapping from the input $\boldsymbol{x}^q$ to the desired output $\boldsymbol{y}^q$. Accordingly, denote the set of target vectors by $\mathcal{Y} = \{\boldsymbol{y}^1, \ldots, \boldsymbol{y}^q, \ldots, \boldsymbol{y}^Q\}$. Then the network output $\boldsymbol{c}$ may be regarded as a prediction $\hat{\boldsymbol{y}}^q$ of $\boldsymbol{y}^q$. Subsequently,

$$\hat{\boldsymbol{y}}^q \approx \boldsymbol{c}^q,$$

with the functional mapping being

$$f : \mathcal{X} \mapsto \mathcal{Y}.$$

The process of adjusting the network weights in $\boldsymbol{w}$ is referred to as training — the network is informed what the desired output is that corresponds to an input presented to it, and it is *trained* to methodologically learn the relationship between the input and output. This relationship is then captured in the network's adjustable parameters, *i.e.* learnt weights.

The training of an FNN may, thus, be regarded as an optimisation problem over a high-dimensional parameter space (*i.e.* vector space spanned by the weight vectors), with the goal of achieving the best value for the network's *performance measure* [131]. The performance measure represents the objective function — in this case the *network error*[2] or *network accuracy*, to be minimised or maximised, respectively. The function representing the performance measure has the form

$$Z(\boldsymbol{y}^q, f(\boldsymbol{x}^q; \boldsymbol{w})) \tag{3.14}$$

if *online learning* is employed, or

$$\sum_{q=1}^{Q} Z(\boldsymbol{y}^q, f(\boldsymbol{x}^q; \boldsymbol{w})), \qquad \boldsymbol{y}^q \in \mathcal{Y},\ \boldsymbol{x}^q \in \mathcal{X}, \tag{3.15}$$

if *batch learning* is employed. The difference between batch and online learning is elucidated in the following section. The performance measure determines the shape of the solution space and, if following a gradient-based approach, influences the choice of which algorithmic approach to follow. The optimisation problem of training an FNN with many hidden layers is a highly non-linear and non-convex objective function. As a result, a copious number of local minima and maxima (of varying depths or heights) typically exist [53, 91].

3.5.4 Batch and online learning

Another important facet to learning/training is the choice of when to adjust the network weights [145]. The two main methods are as follows:

(1) *Online learning*: Weights are updated after each (training) example $\boldsymbol{x}^q \in \mathcal{X}$ and $\boldsymbol{y}^q \in \mathcal{Y}$ has been processed by the network, as expressed in (3.14). Advantages of applying this method are significantly faster training, the ability to track changes, and good solutions (in most cases) [91].

(2) *Batch learning*: Weights are only updated after all (or a subset of) the (training) examples have been processed by the network, as expressed in (3.15). The conditions of convergence, when batch learning is employed, are well known, thus making it an attractive choice. Another advantage is that the theoretical analysis of weight behaviour and the convergence rate is simpler [91].

[2] The terms *cost* or *loss* are sometimes used instead.

The number of iterations and epochs is another important aspect related to training. An iteration passes whenever weights are updated, be it after processing a single training example or after processing a batch of training examples. An epoch passes whenever the entire training set has been evaluated by the algorithm. A simple example follows to elucidate these two definitions: If the training data set has 1 000 examples, *i.e.* $Q = 1\,000$, and online learning is employed, then 1 000 iterations will need to have been completed before an epoch passes. When batch learning is employed, on the other hand, with a batch size of *e.g.* 200, an epoch passes when five iterations have been completed. It is of course assumed that training examples are sampled without replacement.

3.5.5 Data set types

A performance measure function represents an indication of how well an ANN is performing with respect to the data set provided. A data set is usually partitioned into a *training* set, a *validation* set, and a *testing* set, with various conventions being followed in terms of choosing the sizes of the respective sets. One common approach in the literature is a 60%:20%:20% split between training, validation, and testing, respectively. The definitions of these sets differ among numerous authors. Fortunately, Brownlee [15] has synthesised the definitions proposed in various key academic studies [71, 87, 132, 142]. The assimilated purpose of each set is defined as follows:

- The training set is used to adjust the model parameters, *i.e.* network weights and biases.
- The validation set is used to carry out an unbiased performance evaluation of the model (fitted to the training set). The validation set is also typically used to adjust the hyper-parameters of the network, *e.g.* the number of hidden layers and/or neurons per layer. Improving performance by means of hyper-parameter adjustment, however, renders the evaluation more biased. In addition, some regularisation techniques employ the validation set in order to improve generalisation performance — a matter that is elucidated later in this chapter.
- The testing set is used to carry out a final unbiased evaluation of the model fitted to the training set and, if applicable, the validation set.

When a model exhibits good performance in respect of the training set, but poor performance in respect of the testing set, it is said to *overfit* the data (as mentioned in §3.1) — *i.e.* exhibits good memorisation, but poor generalisation. Methods for the prevention of overfitting are discussed later in this chapter.

3.5.6 Weight initialisation

Before training commences, the network weights are to be *initialised* — *i.e.* a starting point for the training algorithm has to be generated. According to LeCun *et al.* [91], the initial weight values have a considerable effect on the training process in terms of convergence speed and quality of solutions. The basic convention is to draw *small* weight values randomly from some probability distribution (*e.g.* a uniform or a Gaussian distribution). There are, however, a few modifications to consider when attempting to further improve training performance [91]. One of these considerations is the activation function (and its associated derivative), which is of particular concern when employing gradient-based training algorithms, such as *gradient descent*. Take the sigmoid function in (3.4) as an example: Very small/large values of the parameter s

will cause the function to saturate, producing small gradients and, as a result, slow learning. The sigmoid function's linear region (shown in Figure 3.7) ought to dictate the range of values from which samples are drawn when initialising the weights. Working within this intermediate range of values should result in sufficiently large gradients for effective learning to transpire. This sampling approach can overcome the significant disadvantage associated with the sigmoid activation function, as mentioned in §3.2. LeCun *et al.* [91] suggested the following weight initialisation procedure when utilising sigmoid activation functions: The weights, drawn from some probability distribution, should have a mean of zero and standard deviation of $\sqrt{1/\tilde{m}}$, where $\tilde{m}$ is the so-called *fan-in* number — the number of connections entering the neuron. When the PReLU activation function (3.6) is employed, however, He *et al.* [59] recommended that the randomly sampled values have a standard deviation of $\sqrt{2/\tilde{m}}$.

The purpose of *scaling* the weights is to normalise the variance of each neuron's input, thus ensuring that all neurons follow approximately the same output distribution. This results in an empirical improvement of the convergence rate [59, 91]. Only the sigmoid and PReLU activation functions are considered here due to their prolific use in the ANN literature. The aforementioned scaling procedures presuppose the standardisation of the data set. Standardisation is another key consideration when initialising weights. It is a typical data-preprocessing step and ensures that the mean of each input variable is close to zero, resulting in faster convergence [91].

3.6 Network performance measures

Measures for assessing the performance of ANNs are discussed in this section. This discussion transpires within the context of the two main types of supervised learning problems — classification and regression. The main distinguishing factor between classification and regression problems is that in the latter case, the dependent variable(s) are of a continuous nature, whereas in the former case, the dependent variables are of a discrete/categorical nature (either binary or multi-class). A data set, consisting of Q input-output observations, of the form $\{(\boldsymbol{x}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{x}^q, \boldsymbol{y}^q), \ldots, (\boldsymbol{x}^Q, \boldsymbol{y}^Q)\}$, abbreviated as $(\mathcal{X}, \mathcal{Y})$, is considered as the basis for the following discussion. Each individual input vector has the form $\boldsymbol{x}^q = [x_1^q \cdots x_i^q \cdots x_n^q]$, whereas each output vector has the form $\boldsymbol{y}^q = [y_1^q \cdots y_k^q \cdots y_o^q]$, and the network output vector has the form $\boldsymbol{c}^q = [c_1^q \cdots c_k^q \cdots c_o^q]^T$ for each $q \in \{1, \ldots, Q\}$.

In a paper by Rosasco *et al.* [135], which deals with an investigation of different performance measuring functions from the perspective of statistical learning theory, the following three loss functions are proposed for performance assessment in the context of regression problems (having continuous dependent variables):

1. the *absolute value* loss $\sum_{k=1}^{o} |y_k^q - c_k^q|$,
2. the *square* loss $\sum_{k=1}^{o} (y_k^q - c_k^q)^2$, and
3. the ϵ-*insensitive* loss $\sum_{k=1}^{o} \max\left\{0, |y_k^q - c_k^q| - \epsilon\right\}$.

For classification problems (having discrete or categorical dependent variables), on the other hand, a different approach was proposed by Rosasco *et al.* In addition to the aforementioned absolute and square loss functions, the following additional functions were suggested:

1. the *hinge* loss $\sum_{k=1}^{o} \max\left\{0, 1 - y_k^q \cdot c_k^q\right\}$, and
2. the *logistic* loss $\sum_{k=1}^{o} (\ln 2)^{-1} \ln(1 + e^{-y_k^q \cdot c_k^q})$.

The ϵ-insensitive and hinge loss functions are, to the best of the author's knowledge, not as widely adopted in the ANN literature. Performance assessment functions pertaining to the other functions are therefore reviewed in greater detail. A list of the most prominent absolute value and square loss functions are:

- the *mean absolute error* (MAE)

$$\frac{1}{Q}\sum_{q=1}^{Q}\sum_{k=1}^{o}|y_k^q - c_k^q|, \tag{3.16}$$

- the *sum of squared errors* (SSE)

$$\sum_{q=1}^{Q}\sum_{k=1}^{o}\left(y_k^q - c_k^q\right)^2, \tag{3.17}$$

- the *mean squared errors* (MSE)

$$\frac{1}{Q}\sum_{q=1}^{Q}\sum_{k=1}^{o}\left(y_k^q - c_k^q\right)^2, \text{ and} \tag{3.18}$$

- the *root mean of squared errors* (RMSE)

$$\sqrt{\frac{1}{Q}\sum_{q=1}^{Q}\sum_{k=1}^{o}\left(y_k^q - c_k^q\right)^2}. \tag{3.19}$$

Each of these measures provides an indication of the extent of the error made by the model's estimates or approximations. This error may be interpreted as the cost one is willing to incur by predicting the values c_k^q, instead of y_k^q. In each case, the error across the entire data set is calculated — hence the summation across all Q input-output observations in each case. A batch learning paradigm is therefore followed. If, on the other hand, an online paradigm were to be followed, the summation would be omitted. Typical factors that possibly account for the presence of errors are random noise, the data set not containing enough problem-specific features, or inadequate model parameter and/or hyper-parameter values [123].

The consequences of making incorrect predictions in the context of a specific problem influences which measure to use [123]. One noteworthy difference between these measures is their handling of extreme errors, also known as *outliers* [113]. For example, MAE attaches a relatively small *penalty* to outliers, whereas SSE and MSE square the error, subsequently weighting outliers heavily. The SSE and MSE penalties may sometimes be too excessive, causing poor convergence in the context of certain problems. Square loss functions are also deemed *sufficiently smooth* objective functions — an especially attractive attribute for gradient-based optimisation methods [56]. The nature of the error surface (the objective function mapped out in the parameter space) when employing MSE as the performance measure, is another desirable trait, as will be discussed later in this chapter. The purpose of adopting RMSE is to transform the units into a somewhat more understandable format, as the square root of the error results in it having the same unit as the quantity being estimated. RMSE is especially used to draw comparisons between data sets or models [2].

Berger [5] claimed that most researchers use squared-error measures, also referred to as *quadratic* cost functions, for the following two reasons: First, due to their relationship (or association) with

classic least squares theory and, more importantly, due to their relative simple *mathematical nature* (being easy differentiable). The most widely adopted training algorithms necessitate a differentiable objective function, hence the importance associated with the choice of both the performance measure (and the activation functions of the network). To reiterate, the aforementioned performance measures are not limited to regression problems only, but may also be used in the context of classification problems. They are, however, routinely categorised as applicable to regression problems by convention.

According to Nielsen [116], a very important ANN performance measure (omitted by Rosasco *et al.*), is *cross-entropy*, and is typically employed in conjunction with an output layer employing sigmoid activation functions. For a classification problem, cross-entropy is defined as

$$-\frac{1}{Q}\sum_{k=1}^{o}\sum_{q=1}^{Q}\left(y_k^q \ln\left(c_k^q\right)+\left(1-y_k^q\right)\ln\left(1-c_k^q\right)\right). \tag{3.20}$$

Cross-entropy is the negative log-likelihood of the *Bernoulli* distribution. Another key performance measure, according to Nielsen, is that of the *log-likelihood* function, which is the negative log-likelihood of the multinomial distribution (a multi-class version of the Bernoulli distribution). It is used in conjunction with an output layer employing softmax activation functions, and takes the form

$$-\frac{1}{Q}\sum_{k=1}^{o}\sum_{q=1}^{Q}\ln\left(y_k^q c_k^q\right). \tag{3.21}$$

The functions in (3.20)–(3.21) are said to work "much better" in the context of many supervised learning tasks, provided that sigmoid/softmax activation functions are employed in the output neurons [49]. According to Nielsen [116], these negative log-likelihood functions overcome the problem of *learning slowdown* associated with the quadratic cost functions — *i.e.* the phenomenon where learning tends to slow down as the error diminishes. Figure 3.12 contains a graphical illustration of the solution space (or *error surface*) of an SFNN with one hidden neuron (and therefore two weights) in the case of employing a hyperbolic tangent activation function for the hidden neuron. Two error functions, cross-entropy (the black, top-most surface) and quadratic cost (the red, bottom surface) are plotted in the figure. The advantage of cross-entropy over quadratic cost is reflected by less severe plateaus (*i.e.* flat parts).

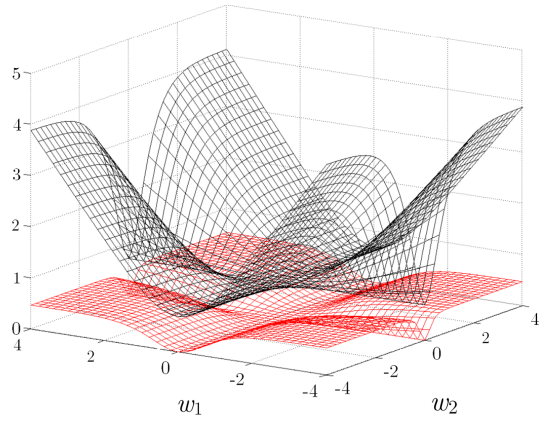


FIGURE 3.12: *Fitness landscape of an SFNN with a hyperbolic tangent activation function, one hidden neuron, weights w_1 and w_2, with cross-entropy (black, top surface) and quadratic cost (red, bottom surface) as performance measures [49].*

One of the more traditional and better known performance measures for classification problems is *classification accuracy* (CA). It is simply the number of correctly classified examples divided by

the total number of examples. This measure's popularity is perhaps attributable to its simplicity and ease of comprehension. It is, however, susceptible to a notable pitfall. To illustrate this disadvantage, consider a binary classification problem with 90% of the data set being zeros and 10% of it being ones. This data set exhibits *class imbalance* — a problem encountered in many ML problems where each class is not represented equally frequently. A common realistic scenario in which this phenomenon is encountered is fraud detection where the data set contains only a small number of fraudulent-cases. If the fitted model always predicts zero, then its classification accuracy is 90%. Without any context, this model would be deemed a relatively good model, but, evidently, it should, in actual fact, be regarded as performing poorly. This measure succumbs to the so-called *accuracy paradox*, which states that a model with a high accuracy may not necessarily be a model with good predictive power [167]. False positives and false negatives, in particular, are excluded from evaluation according to this measure. To explicate the matter, consider the confusion matrix

$$\begin{array}{c} \\ \text{Actual Positive} \\ \text{Actual Negative} \end{array} \begin{array}{c} \begin{array}{cc} \text{Predict Positive} & \text{Predict Negative} \end{array} \\ \begin{bmatrix} \text{TP} & \text{FN} \\ \text{FP} & \text{TN} \end{bmatrix} \end{array}.$$

The notion of *positives* and *negatives* are, henceforth, used interchangeably with ones and zeros, respectively. A false negative is therefore predicting a zero when, in fact, the correct value is one, whereas a false positive is predicting a one when, in fact, the correct value is zero. The classification accuracy is calculated as the sum of the true positives (*i.e.* correctly classified ones) and true negatives (*i.e.* correctly classified zeros), divided by the total number of predictions, mathematically expressed as

$$A = \frac{TP + TN}{TP + TN + FP + FN}.$$

For certain practical ML problems, predicting false negatives has far greater consequences or cost than predicting false positives. Whenever this is not the case (*i.e.* the cost is equal for the latter and the former), classification accuracy is deemed sufficient, although not ideal. A realistic application in which this situation prevails is the classification of a tumour as malignant (positive) or benign (negative). A model with a reasonably high classification accuracy may still predict a relatively large number of false negatives, and considering the sensitive nature of the application, incorrectly classifying a tumour as benign is far more worse than incorrectly classifying it as malignant. Many researchers disregard this consideration when conducting ANN-related studies. There are, however, numerous alternatives to the traditional measure of classification accuracy that avoids its susceptibility to the aforementioned paradox. A widely-used performance measure that accounts for what classification accuracy omits, is the F_1-*score* [14]. In order to calculate this score, a brief overview of the two main components of the F_1-score, namely *precision* and *recall*, is presupposed.

Precision, given by

$$P = \frac{TP}{TP + FP}, \tag{3.22}$$

reflects the *exactness* of the model, with a small value suggesting a large number of false positives [14]. Recall, on the other hand, is given by

$$R = \frac{TP}{TP + FN} \tag{3.23}$$

and reflects the *completeness* of the model, with a small value suggesting a large number of false negatives. Using (3.22) and (3.23) in conjunction with one another, the F_1-score may now be

expressed as

$$F_1 = 2\frac{1}{\frac{1}{P}+\frac{1}{R}} = \frac{2PR}{P+R}. \tag{3.24}$$

A high F_1-score indicates a prediction comprising few false positives and few false negatives — the goal is therefore to maximise this performance measure. A binary classification problem was taken as the basis for the above discussion, although many classification problems comprise more than two classes. If the problem at hand is of a multi-class nature, the weighted average of the F_1-score, calculated for each class, is used instead. Accordingly, the simple arithmetic mean of the class scores represents one popular approach according to which each *class* is given equal weighting. Alternatively, equal weighting can be given to each *observation*, although less prevalent classes exhibiting poor performance can be masked by this approach. Consequently, the former approach is preferred and adopted in this dissertation. Other measures addressing the accuracy paradox also exist, but the F_1-score is deemed satisfactory due to its widespread use in the literature [14].

3.7 Learning procedures

As was stated in §3.5, efficient training algorithms are a necessity when attempting to find a suitable set of network parameter values, so as to approximate a sufficiently adequate functional mapping of the input variables to the output variables. Training algorithms attempt to optimise the chosen network performance measure, which represents an objective function. This function, expressed in terms of the network weights, is typically a highly non-linear and non-convex function exhibiting super-abundant local optima [53]. The well-known *curse of dimensionality* plays a significant role in this optimisation problem as the number of dimensions (represented by the network parameters) is typically very large in most DNN approaches. Rather than following a brute-force approach, efficient computational methods are employed to tackle these problems. When employing performance measures and activation functions of a specific nature (*e.g.* a squared error term and the sigmoid function), the training of the ANN is equivalent to that of minimising a continuous, differentiable function (of many variables) — a well-developed field of study [6]. Logically, many of the conventional algorithmic approaches toward solving problems in the field of vector calculus may be adopted when attempting to train ANNs.

There are three main approaches, the first being *first-order optimisation algorithms*, which use only the gradient of the objective function. According to Goodfellow [53], *gradient descent* is by far the most popular algorithm in this paradigm. The second approach is *second-order optimisation algorithms*, which employ the Hessian matrix. The most powerful of these second-order optimisation algorithms are:

1. Newton's method,
2. conjugate gradient methods, such as the *Polak-Ribiére* method [130] and the *Fletcher-Reeves* method [45],
3. the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) algorithm [44] and its improved version, *Limited Memory* BFGS (or L-BFGS) [94], and
4. the *Levenberg-Marquardt* algorithm [97].

These second-order algorithms have been found to perform exceptionally well in respect of smaller problem instances [53, 100], but as soon as problem complexity increases and the network becomes large with many accompanying parameters, these second-order approaches are rendered

near-intractable. The computational complexity, in some cases, increases from $\mathcal{O}(W)$ to $\mathcal{O}(W^3)$, where W represents the number of network weights, as the Hessian matrix needs to be evaluated by the second-order training algorithm (instead of merely using first-order gradient information). Ordinary computer hardware falters under the resulting heavy computational burden. The use of advanced *Graphical Processing Units* and computer clusters (so as to facilitate parallelisation), make it possible to use these second-order methods in some cases [140], but they are extremely expensive and, as a result, not as widespread. The third approach involves the use of meta-heuristics, which will be discussed in greater depth in Chapter 4.

This section focuses on the most popular of the first-order algorithmic approaches. Specific focus is afforded to gradient descent's most powerful variations, namely *stochastic gradient descent* (SGD), RMSProp, and Adam. A brief overview of the objective function shape in parameter or weight space (*i.e.* the *error surface*, within a minimisation context) is also included as it introduces key concepts that facilitate the ensuing discussion.

3.7.1 Backpropagation

The official arrival of backpropagation during the 1980s heralded a new era for ANNs [142]. The criticism against and limitations of ANNs, laid bare in the damning report by Minsky and Papert [107] in 1969, were subsequently eliminated, and the capabilities of ANNs envisioned by their founding fathers were finally rendered computationally feasible. Up to the arrival of this influential learning procedure, the notion of an ANN was severely limited in terms of its application to complex real-life and theoretical problems — an efficient way of updating/adjusting the synaptic weights for large networks eluded researchers. Backpropagation introduced a computationally efficient method for expressing the contribution that each neuron (or weight) makes towards the network error (or accuracy) — *i.e.* it became possible to evaluate the derivatives of the network performance with respect to the weights. Well-developed mathematical optimisation techniques could subsequently be applied to train ANNs.

The SFNN presented in Figure 3.10 serves as the context in which the derivation of the backpropagation training algorithm is reviewed in this section. This derivation is a reproduced modification of the general derivation proposed by Bishop [6]. As was stated in §3.5.1, the network learning performance measure may be expressed in terms of either a network error or network accuracy (to be minimised or maximised, respectively). The following discussion transpires within a minimisation-of-error context, with the network error function denoted by E.

Suppose that for a data set containing Q training examples, the individual error E^q for the q-th input vector, with $q \in \{1, \ldots, Q\}$, may be found. Then E can be expressed as the sum over all Q of these individual errors. That is,

$$E = \sum_{q=1}^{Q} E^q. \tag{3.25}$$

Furthermore, suppose that

$$E^q = E^q(c_1, \ldots, c_o) \tag{3.26}$$

can be expressed as a differentiable function of the network outputs. Backpropagation provides a procedure for evaluating the derivatives of the error function E with respect to the weights. Using (3.25), these derivatives may then be expressed as the sum of the individual error derivatives (for each input vector) over the training set.

The so-called process of *forward propagation* emanates from the presentation of an input vector to the SFNN's input layer, which is then transmitted (by means of the weighted connections)

over the hidden and output layers. Information thus flows in a *forward* direction through the network. The activations of neurons within these hidden and output layers are computed using (3.7)–(3.12).

Consider, for the weight $w_{kj}^{(1)}$ (between the hidden layer and the output layer) for some $k \in \{1,\ldots,o\}$ and some $j \in \{0,\ldots,m\}$, the evaluation of the derivative of E^q. Although the activations of the neurons depend on the input vector q, clutter is avoided in the following derivation by omitting the corresponding superscript from the respective denotations. The net input $\eta_k^{(2)}$ to output neuron k represents the only intermediate dependency of E^q on $w_{kj}^{(1)}$. It therefore follows from the chain rule for differentiation that

$$\frac{\partial E^q}{\partial w_{kj}^{(1)}} = \frac{\partial E^q}{\partial \eta_k^{(2)}} \cdot \frac{\partial \eta_k^{(2)}}{\partial w_{kj}^{(1)}}. \tag{3.27}$$

In order to simplify the derivation, the notations

$$\delta_k^{(2)} = \frac{\partial E^q}{\partial \eta_k^{(2)}} \tag{3.28}$$

and

$$\delta_j^{(1)} = \frac{\partial E^q}{\partial \eta_j^{(1)}} \tag{3.29}$$

are introduced, which are called the *delta-errors*. It follows from (3.9) that

$$\frac{\partial \eta_k^{(2)}}{\partial w_{kj}^{(1)}} = b_j. \tag{3.30}$$

By substituting (3.28) and (3.30) into (3.27), the derivative

$$\frac{\partial E^q}{\partial w_{kj}^{(1)}} = \delta_k^{(2)} b_j \tag{3.31}$$

is found. Evidently, the only values required to evaluate the derivative of E^q with respect to the weights between the hidden layer and the output layer of the SFNN, are those of $\delta^{(2)}$. Combining the definition of c_k in (3.10) with the fact that E^q depends on $\eta_k^{(2)}$ only through the output neuron activation c_k, the useful expression

$$\begin{aligned}
\delta_k^{(2)} &= \frac{\partial E^q}{\partial c_k}\frac{\partial c_k}{\partial \eta_k^{(2)}} \\
&= \frac{\partial E^q}{\partial c_k}\frac{\partial}{\partial \eta_k^{(2)}}\left(g^{(2)}\left(\eta_k^{(2)}\right)\right) \\
&= g'_{(2)}\left(\eta_k^{(2)}\right)\frac{\partial E^q}{\partial c_k}
\end{aligned} \tag{3.32}$$

may be derived by application of the chain rule to (3.28). The same procedure may be applied to evaluate the derivative of E^q with respect to the weight $w_{ji}^{(0)}$ (between the input layer and the hidden layer) for some $j \in \{1,\ldots,m\}$ and some $i \in \{0,\ldots,n\}$. This yields

$$\frac{\partial E^q}{\partial w_{ji}^{(0)}} = \frac{\partial E^q}{\partial \eta_j^{(1)}} = \delta_j^{(1)} a_i, \tag{3.33}$$

which has the same general form as (3.31). A similar conclusion is reached — the only values required to evaluate the derivative of E^q with respect to weights between the first layer and the hidden layer of the SFNN, are those of $\delta^{(1)}$.

By now letting E^q be a function of the net input to each of the o output neurons, (3.29) may be rewritten as

$$\delta_j^{(1)} = \frac{\partial E^q(\eta_1^{(2)}, \ldots, \eta_o^{(2)})}{\partial \eta_j^{(1)}}. \tag{3.34}$$

Using (3.34) and applying the chain rule twice, followed by the substitution of (3.28) into the resulting expression, it is found that

$$\begin{aligned}\delta_j^{(1)} &= \frac{\partial E^q}{\partial \eta_1^{(2)}}\frac{\partial \eta_1^{(2)}}{\partial \eta_j^{(1)}} + \ldots + \frac{\partial E^q}{\partial \eta_o^{(2)}}\frac{\partial \eta_o^{(2)}}{\partial \eta_j^{(1)}} \\ &= \sum_{k=1}^{o} \frac{\partial E^q}{\partial \eta_k^{(2)}}\frac{\partial \eta_k^{(2)}}{\partial \eta_j^{(1)}} \\ &= \sum_{k=1}^{o} \delta_k^{(2)}\frac{\partial \eta_k^{(2)}}{\partial b_j}\frac{\partial b_j}{\partial \eta_j^{(1)}}.\end{aligned} \tag{3.35}$$

By employing (3.8) and (3.9), the final expression for $\delta_j^{(1)}$ may be written as

$$\delta_j^{(1)} = g'^{(1)}\left(\eta_j^{(1)}\right)\sum_{k=1}^{o} w_{kj}^{(1)}\delta_k^{(2)}, \tag{3.36}$$

which is referred to as the so-called *backpropagation formula*. The above derivation was performed in the context of a SFNN, but a generalisation to FNNs with more than one hidden layer is possible. By propagating the δ-values backward (from neurons in successive layers in the FNN), the δ-value for any hidden neuron may be computed [6].

The backpropagation algorithm may be summarised as follows:

1. Present the q-th input vector to the FNN, where $q \in \{1, \ldots, Q\}$.
2. Perform forward propagation through the network by means of (3.7)–(3.12) — *i.e.* calculate all neuron activations.
3. Use (3.32) to evaluate $\delta_k^{(2)}$ for each neuron in the output layer.
4. Backpropagate the $\delta_k^{(2)}$-values and evaluate $\delta_j^{(1)}$ by means of (3.36).
5. Finally, use (3.31) and (3.33) to evaluate the derivatives of the error E^q.

Based on (3.25), repeated implementation of the five-step algorithm above to each input vector results in an evaluation of the derivative of the total error E with respect to the FNN's weights, yielding

$$\frac{\partial E}{\partial \boldsymbol{w}} = \sum_{q=1}^{Q} \frac{\partial E^q}{\partial \boldsymbol{w}}. \tag{3.37}$$

According to Bishop [6], the computational complexity of the backpropagation algorithm is $\mathcal{O}(W)$, where W is the total number of network weights. The other, more rudimentary approach

of explicitly deriving formulae for the derivatives and then numerically evaluating them by means of forward propagation, results in a computational complexity of $\mathcal{O}(W^2)$. Consequently, backpropagation reduces the computational complexity by one order of magnitude from $\mathcal{O}(W^2)$ to $\mathcal{O}(W)$. To emphasise the importance of this algorithmic breakthrough, a parallel is drawn to the *fast Fourier transform* algorithm, which reduces the computational complexity of evaluating an L-point Fourier transform from $\mathcal{O}(L^2)$ to $\mathcal{O}(L \log L)$ [6]. Widespread adoption of Fourier transforms in many practical applications was the result of this breakthrough. One can argue that the advent of the backpropagation algorithm had a similar impact on the field of ANNs.

3.7.2 The error surface

Before discussing the different gradient descent-based training algorithms, a brief overview of the properties of the error surface is presented. The error surface is the objective function mapped out in parameter/weight space. Depicted graphically in Figure 3.13, a general multi-dimensional hyper-parabolic form is exhibited by the error surface of a simple SFNN, having linear activation functions. The *global minima* of this surface may be found by solving a set of coupled linear equations. More complex networks (*i.e.* comprising many layers of weights), however, exhibit highly non-linear error functions and, consequently, many local minima and maxima. Each of these satisfy the equation

$$\nabla E^q = 0, \tag{3.38}$$

where ∇E^q denotes the gradient of the error function (in weight space) for training example $q \in \{1, \ldots, Q\}$. This condition is applicable when an online learning paradigm is followed. When training transpires within a batch learning paradigm, however, the derivative of E, *i.e.* the sum of the derivatives across all Q training examples (as indicated in (3.37)) is used instead of the individual error term E^q. Gradient information may thus be used to find the "best" network weights for the problem at hand. Any minimum corresponding to a globally smallest error function value is regarded as a global minimum, whereas the remaining minima are regarded as *local minima* (and similarly for global maxima and local maxima). Condition (3.38) is, however, also met by *saddle points*. That is, by points that correspond to a relative maximum and a relative minimum when analysed from two orthogonal directions in parameter space. All of these *points of interest* which satisfy (3.38) are classified as stationary points, as illustrated in Figure 3.14. Goodfellow *et al.* [53] claimed that a global minimum will result in an overfitted model (exhibiting poor generalisation) as the training set is essentially memorised and fitted *perfectly* by the model, without learning the true underlying functional representation of the problem (as discussed in §3.1).

It is possible to work on an even lower level of abstraction — *i.e.* utilising second-order derivatives. The second-order derivatives, captured by the Hessian matrix, provide a measure of the surface's *curvature* and makes it possible to ascertain the behaviour of the gradient as the network weights change. As mentioned earlier, this information allows certain algorithms to perform well when applied to small problem instances, but in practice (where problem instances tend to be very complicated) these methods are computationally infeasible for the most part [140] and, as a result, are excluded from the current discussion.

Goodfellow *et al.* [53] determined the *weight space symmetry* for an FNN with h hidden layers and m hidden neurons in each layer as $(m!)^h$. That is, any point in the weight space will have $(m!)^h$ equivalent points achieving the same error function value. Thus, large networks will exhibit many local or global minima as a result of this high degree of symmetry. According to Bishop [6], the symmetry exhibited by FNNs is not necessarily influenced by the specific activation functions used, as a wide range of activation functions produce similarly shaped weight spaces.

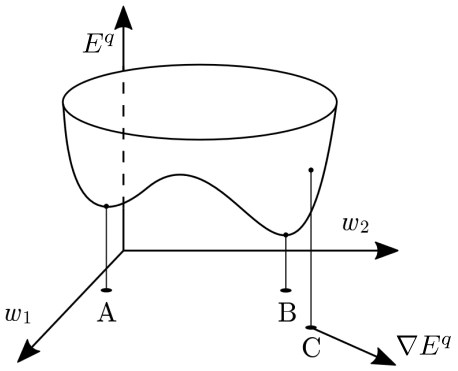


FIGURE 3.13: *A hypothetical error function E^q resulting from a network with weights w_1 and w_2. The minima of E^q are represented by points A and B. The local gradient of the error surface, at any point C, is given by vector ∇E^q [6].*

If local minima corresponding to large error values (compared with the global minimum) are common, gradient-based training algorithms have been reported to face a considerable challenge [53]. For "sufficiently" large networks, most local minima achieve small error function values and so finding a true global minimum ought not to be the priority. Deciding which of the many minima to use is an irrelevant matter for practical problems. Although the training algorithms discussed hereafter are based on a local stepwise search process through the weight space, the presence of multiple equivalent minimum points is therefore not of particular interest.

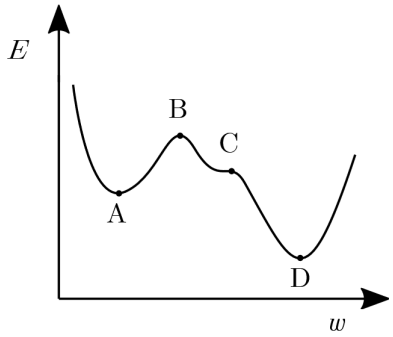


FIGURE 3.14: *A graphical illustration of an error curve as a function of a single weight w and four stationary points A, B, C, and D, representing a local minimum, a local maximum, a saddle point, and a global minimum, respectively [6].*

3.7.3 The method of gradient descent

The backpropagation algorithm provides a computationally efficient method for evaluating the derivatives of the error function with respect to the adjustable weights of the network. Having access to the network's gradient information allows for the application of a wide range of algorithms that are able to train ANNs adequately for large-scale problems [6, 140]. In this section, specific emphasis is placed on a specific first-order optimisation technique — the gradient descent method for minimisation. Its prolific use in the literature and its simple nature renders it an obvious candidate for inclusion in this discussion.

Generally speaking, finding *closed-form* expressions for the error surface minima is not always possible — this is attributable to the non-linear nature of the error function. As a result, efficient algorithms are employed to search *intelligently* for minima in the weight space. Gradient descent

(also known as *steepest descent*) is by far the most prevalent of this type of search algorithm [140]. According to this algorithm, a decision vector $\boldsymbol{d}$ is iteratively updated according to the rule

$$\boldsymbol{d} \leftarrow \boldsymbol{d} - \kappa \nabla f(\boldsymbol{d}), \tag{3.39}$$

where κ is the step size (or *learning rate*), and $f(\boldsymbol{d})$ is the function to be minimised. The fundamental premise of this method is that the negative gradient of the function $f(\boldsymbol{d})$ at a point $\boldsymbol{r}$ is the direction in which f decreases the *fastest* about the point $\boldsymbol{r}$. Thus, the direction of steepest descent is $-\nabla f(\boldsymbol{r})$. It is, of course, assumed that f is differentiable, which depends on how the objective function is defined (*i.e.* the choice of performance measure and activation functions). In the context of training FNNs within an online learning paradigm, (3.39) becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \kappa \nabla E^q\Big(\boldsymbol{y}^q, f(\boldsymbol{x}^q; \boldsymbol{w})\Big). \tag{3.40}$$

Within a batch learning paradigm, however, (3.39) becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \kappa \nabla \left(\frac{1}{Q}\sum_{q=1}^{Q} E^q\Big(\boldsymbol{y}^q, f(\boldsymbol{x}^q; \boldsymbol{w})\Big)\right). \tag{3.41}$$

In this context, the application of (3.39) yields a procedure for updating the network weights so as to minimise the error function. When every element of the gradient vector is non-negative, the algorithm is said to have converged. The starting point for gradient descent is determined by means of the weight initialisation process discussed in §3.5.6. According to Bishop [6], the starting point will, for some training algorithms, most likely determine the minimum to which they are able to converge. The learning rate, usually a small positive number, is *fairly critical* with respect to the convergence of $\boldsymbol{w}$ to the point where the error function is at a local minimum [6]. If κ is too small, learning will be very slow, whereas if it is too large, divergent oscillations may occur.

When an SFNN is trained within an online learning paradigm, weights in the second layer (*i.e.* between the hidden layer and output layer) are updated according to the rule

$$\begin{aligned} w_{kj}^{(1)} &\leftarrow w_{kj}^{(1)} - \kappa \frac{\partial E^q}{\partial w_{kj}^{(1)}} \\ &= w_{kj}^{(1)} - \kappa \delta_k^{(1)} b_j, \qquad k \in \{1, \ldots, o\},\ j \in \{0, \ldots, m\}. \end{aligned} \tag{3.42}$$

Similarly, weights in the first layer (*i.e.* between the input layer and hidden layer) are updated according to the rule

$$\begin{aligned} w_{ji}^{(1)} &\leftarrow w_{ji}^{(0)} - \kappa \frac{\partial E^q}{\partial w_{ji}^{(0)}} \\ &= w_{ji}^{(0)} - \kappa \delta_j^{(0)} a_i, \qquad j \in \{1, \ldots, m\},\ i \in \{0, \ldots, n\}. \end{aligned} \tag{3.43}$$

When batch learning is employed, however, the derivative of E (*i.e.* the sum of the derivatives across all Q training examples) is used instead of the individual error E^q.

SGD involves the evaluation of either one random training example or, more popularly, a subset (or *mini-batch*) of random training examples of size P. This subset is denoted by $(\hat{\mathcal{X}}, \hat{\mathcal{Y}}) = \{(\hat{\boldsymbol{x}}^1, \hat{\boldsymbol{y}}^1), \ldots, (\hat{\boldsymbol{x}}^p, \hat{\boldsymbol{y}}^p), \ldots, (\hat{\boldsymbol{x}}^P, \hat{\boldsymbol{y}}^P)\}$, where $\hat{\boldsymbol{x}}^p$ and $\hat{\boldsymbol{y}}^p$ represent the independent variables and dependent variables, respectively, of random training example p, therefore $(\hat{\mathcal{X}}, \hat{\mathcal{Y}}) \subset (\mathcal{X}, \mathcal{Y})$.

According to Keskar *et al.* [77], typical mini-batch sizes are $P \in \mathbb{Z} \cap [32, 512]$. The update rule (3.41) is subsequently transformed to represent the premise of SGD. The transformed update rule is

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \kappa \nabla \left(\frac{1}{P} \sum_{p=1}^{P} E^p \left(\hat{\boldsymbol{y}}^p, f(\hat{\boldsymbol{x}}^p; \boldsymbol{w}) \right) \right), \quad \hat{\boldsymbol{x}}^p \in \hat{\mathcal{X}}, \ \hat{\boldsymbol{y}}^p \in \hat{\mathcal{Y}}. \tag{3.44}$$

Efficient information processing and more stable convergence are claimed to be the main advantages of employing SGD, with the latter being attributable to the reduced variance in the parameter update [8]. The former — efficient information processing — can be explained by considering the following hypothetical situation: A training set $\mathcal{S}$ comprises ten identical copies of a set $\mathcal{S}_{sub}$. A model that minimises the error function over the smaller set $\mathcal{S}_{sub}$ is clearly equivalent to a model applied to the larger set $\mathcal{S}$ in this case. In a batch learning paradigm, each evaluation would be *ten times* more computationally expensive than if only one copy of $\mathcal{S}_{sub}$ were to be analysed. SGD, on the other hand, performs the same computations in both scenarios — *i.e.* elements from $\mathcal{S}_{sub}$ are chosen with the same probability as from $\mathcal{S}$. Real-life problem data sets do not (necessarily) contain exact duplicates as considered here, but a notable amount of redundancy is often present. Thus, following a batch learning approach where the entire data set is analysed, is usually relatively inefficient and so SGD is typically preferred in practice [9].

It should be noted that gradient descent (including SGD) does not succumb to the pitfalls presented by saddle points, unlike second-order *line search-based* algorithms. The inability to efficiently deal with saddle points is one of the main contributing factors to their poor performance in respect of large ANNs [53]. Another contributing factor is that these line search algorithms utilise second-order derivatives, which tend to become computationally expensive for large ANNs. It has been shown empirically that gradient descent seems able to escape from saddle points in many cases [53]. Gradient descent, however, also has other notable drawbacks. These drawbacks are, first, the difficulty in choosing a suitable value for κ and, secondly, a general inability to traverse error surfaces that exhibit substantially different curvatures along different directions — leading to slow convergence as the movement exhibits oscillations. In the case of SGD, choosing the size of the *mini-batch* is also regarded as a challenge. Before discussing some heuristic techniques for addressing these drawbacks, a brief discourse on *momentum* is offered. Inclusion of the notion of momentum has the aim of adding inertia to the search motion through the weight space, so as to smooth out unwanted oscillations during the search process [6]. Thus, a technique for addressing the second major drawback mentioned above is as follows: The original/standard version of gradient descent (3.39) is modified so that

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{v}, \tag{3.45}$$

where $\boldsymbol{v}$ represents the so-called *velocity* vector, which accumulates the gradient elements. This accumulation is captured in the update rule

$$\boldsymbol{v} \leftarrow \mu \boldsymbol{v} - \kappa \nabla E^q \left(\boldsymbol{y}^q, f(\boldsymbol{x}^q; \boldsymbol{w}) \right) \tag{3.46}$$

for an online learning paradigm,

$$\boldsymbol{v} \leftarrow \mu \boldsymbol{v} - \kappa \nabla \left(\frac{1}{Q} \sum_{q=1}^{Q} E^q \left(\boldsymbol{y}^q, f(\boldsymbol{x}^q; \boldsymbol{w}) \right) \right) \tag{3.47}$$

for a batch learning paradigm, or

$$\boldsymbol{v} \leftarrow \mu \boldsymbol{v} - \kappa \nabla \left(\frac{1}{P} \sum_{p=1}^{P} E^p \left(\hat{\boldsymbol{y}}^p, f(\hat{\boldsymbol{x}}^p; \boldsymbol{w}) \right) \right) \tag{3.48}$$

for SGD, where $\mu \in [0, 1)$ is the momentum parameter determining the extent of previous gradient decay. The most common value for this parameter is 0.9, but this parameter value is typically chosen based on numerical experimentation [140]. From Newton's laws of motion, momentum is equal to velocity if unit mass is assumed (hence the naming convention). Significant performance improvements may be gained when incorporating momentum, although its inclusion contributes to the innate drawback associated with gradient descent — selecting a suitable value for this hyper-parameter (and the learning rate) is somewhat difficult.

An enhanced version of SGD, which is commonly applied in practice, involves the incorporation of specific improvements in terms of both the learning rate and momentum. Earlier, it was implied that the learning rate κ remains fixed. When using SGD, however, it is common practice to allow this parameter to *decay* gradually over time, as suggested by Goodfellow *et al.* [53]. In this case, the learning rate during iteration h is denoted by κ_h. The intuition behind the incorporation of a decaying learning rate is to account for the perpetual random noise introduced by the stochastic nature of SGD (*i.e.* random sampling of P training examples). Within a traditional batch learning paradigm, the gradient decreases and then becomes zero at the error minima. Consequently, a fixed learning rate is sufficient to ensure convergence. This is not the case when random noise is present. According to Goodfellow *et al.*, κ_h is commonly taken as

$$\kappa_h = \left(1 - \frac{h}{\tau}\right)\kappa_0 + \frac{h}{\tau}\kappa_\tau, \tag{3.49}$$

which represents a linear decay up to iteration τ, after which the learning rate remains constant. It is common to take τ as the number of iterations required to "make a few hundred passes" through the training set [53]. Setting an appropriate value of κ_0 is, however, regarded as a considerable challenge. Empirical studies have shown that the *optimal* learning rate tends to be larger than the learning rate which delivers the best performance during the first 100 iterations. The learning rate should furthermore be small enough to avoid excessive oscillations. The other parameter, κ_τ, should be set to 1 percent of κ_0 [53]. Goodfellow *et al.* suggest that this approach ought to be implemented with care as it is regarded more an art than a science. Regardless, it is one of the standard approaches when employing SGD and seems capable of delivering good performance without being computationally too expensive [53, 176].

In terms of the other key component (*i.e.* momentum), Sutskever *et al.* [159] proposed an enhanced modification to the standard version, expressed in (3.45), which is inspired by *Nesterov's accelerated gradient* (NAG) method [114, 115]. The new SGD update-rule (which includes the aforementioned improvement) becomes

$$\boldsymbol{v} \leftarrow \mu\boldsymbol{v} - \kappa_h \nabla\left(\frac{1}{P}\sum_{p=1}^{P} E^p\left(\hat{\boldsymbol{y}}^p, f(\hat{\boldsymbol{x}}^p; \boldsymbol{w} + \mu\boldsymbol{v})\right)\right) \tag{3.50}$$

in conjunction with

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{v}. \tag{3.51}$$

NAG is very similar to the standard method of incorporating momentum, but with one significant difference — the gradient is evaluated *after* the current velocity has been applied. This addition serves as a *correction factor* to the standard method of incorporating momentum. NAG provides the algorithm with a form of *prescience* — *i.e.* it is given a notion of where it is going, so as to terminate the descent at the minimum (before the slope rises again) [140].

A pseudocode description of the enhanced version of SGD is presented in Algorithm 3.1 and incorporates both of the two aforementioned improvements (in terms of learning rate and momentum). The training algorithm's stopping criterion would typically be the imposition of an

epoch limit or performance threshold. Instead of using a mini-batch, the entire training set (or single example) can be used, which would mean that a batch (or online) learning paradigm is employed.

Algorithm 3.1: Enhanced SGD with Nesterov momentum

Input : Initial weights $\boldsymbol{w}$, a training set $\{\mathcal{X}, \mathcal{Y}\}$, a momentum parameter μ, an initial velocity $\boldsymbol{v}$, and learning rate parameters τ, κ_0 and κ_τ

Output: Updated weights $\boldsymbol{w}$

$h \leftarrow 1$;
while *stopping criterion not met* **do**
  Sample a mini-batch of P examples: $(\hat{\mathcal{X}}, \hat{\mathcal{Y}}) \subset (\mathcal{X}, \mathcal{Y})$;
  Apply the interim update: $\tilde{\boldsymbol{w}} \leftarrow \boldsymbol{w} + \mu\boldsymbol{v}$;
  Compute the gradient (at interim point): $\boldsymbol{g} \leftarrow \nabla(1/P \sum_p E^p(\hat{\boldsymbol{y}}^p, f(\hat{\boldsymbol{x}}^p; \boldsymbol{w})))$;
  Compute the learning rate: $\kappa_h \leftarrow (1 - h/\tau)\kappa_0 + (h/\tau)\kappa_\tau$;
  Compute the velocity update: $\boldsymbol{v} \leftarrow \mu\boldsymbol{v} - \kappa_h\boldsymbol{g}$;
  Apply the update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{v}$;
  $h \leftarrow h + 1$;

Many heuristic techniques have been proposed in the literature for improving the performance of gradient descent, specifically in terms of *adapting* the learning rate, a hyper-parameter which significantly affects model performance [53]. These heuristic improvements are far more sophisticated than the basic decay technique in (3.49). According to Goodfellow *et al.* [53], the most prolific *adaptive learning rate* methods are RMSProp and Adam. They further claimed that no consensus has been reached in respect of the choice of which algorithm to employ and that this is largely dependent on "the user's familiarity with the algorithm." A relatively recent paper by Wilson *et al.* [176], titled *The marginal value of adaptive gradient methods in machine learning* reported that, although adaptive learning rate methods deliver superior performance during training (when compared to SGD), generalisation — the most important performance indicator — is usually poor, and often *significantly* so. Several state-of-the-art deep learning models were used as the basis for the comparative study of Wilson *et al.,* lending further credibility to their findings. They concluded that practitioners should "reconsider the use" of adaptive methods, as reflected in their primary findings, which were:

- In all of the benchmark problems, SGD (with momentum) outperformed the adaptive learning rate methods in terms of generalisation performance.
- Performance of the adaptive methods tend to quickly reach a plateau after an initial strong start.
- There is no significant increase in the amount of tuning any of the methods must undergo in order to obtain good results.

The last finding may be regarded as a challenge to the conventional wisdom that adaptive learning rate methods require less *tuning* than that of the vanilla version of gradient descent or SGD.

On the other hand, Rude [140] found that adaptive learning rate methods deliver superior performance to that of SGD, although it should be noted that the benchmark problems employed were not explicitly stated or described by the author. Nonetheless, the famous NFL theorem

[178] prevails and lends an explanation for the discrepancies, contradictions, disagreement, and overall lack of consensus found in literature on this matter. The NFL essentially states that "an optimisation algorithm that performs particularly well on one set of objective functions, will also perform correspondingly poorly on all [sic] other objective functions" [178]. To account for this phenomenon, all three of these training algorithms are included in the current study and, subsequently, compared with the proposed hyperheuristic. Algorithmic descriptions of both RMSProp and Adam now follow.

RMSProp

Before discussing RMSProp, a brief overview of its muse, AdaGrad, is offered. Proposed by Duchi *et al.* [36], AdaGrad is a *per-dimension learning rate method* for gradient descent (dimensions here refer to the network weights). The main advantages of AdaGrad, according to Duchi *et al.*, are as follows: The learning rate need not be manually set, large gradients/networks are handled well, and there is a minimal increase in computational burden (compared to that of standard gradient descent). Each weight's learning rate is individually adapted and scaled inversely proportionally to the square root of the sum across all previous squared values of the gradient. Weights exhibiting the largest partial derivatives (of the error) are rapidly decreased, whereas weights exhibiting the smallest partial derivatives experience relatively small decreases. As a result, better progress is facilitated in the "more gently sloped directions" of the weight space [53]. Training of large FNNs transpire within a non-convex optimisation setting, however, while AdaGrad performs better in a convex optimisation setting, thus requiring further modifications to ensure good performance.

RMSProp, proposed by Hinton [164], is one such modification and aims to improve non-convex performance by incorporating an exponentially weighted moving average of the accumulated gradients. As a result, RMSProp discards history from the extreme past so as to speed up convergence. Unfortunately, incorporating this exponentially weighted moving average introduces an additional hyper-parameter ϱ which represents the decay. Based on empirical findings, RMSProp has exhibited effectiveness and practicality for training DNNs [53]. The procedure is described in pseudocode form in Algorithm 3.2. Note that a constant $\delta = 10^{-6}$ is introduced to *stabilise* division by small numbers. The accumulation of gradient information is stored in a vector $\boldsymbol{r}$. In line 5, the scaling by a factor $1/(\sqrt{\boldsymbol{r} + \delta})$ is performed element-wise. It should be noted that both RMSProp and Adam may adopt the concept of random sampling (employed by SGD), even though the algorithms are discussed within a batch learning context.

Algorithm 3.2: RMSProp

Input : Initial weights $\boldsymbol{w}$, a training set $\{\mathcal{X}, \mathcal{Y}\}$, a global learning rate κ, a small constant $\delta = 10^{-6}$, and an exponential decay rate ϱ

Output: Updated weights $\boldsymbol{w}$

1 Initialise accumulation variables $\boldsymbol{r} \leftarrow 0$;
2 **while** *stopping criterion not met* **do**
3 Compute the gradient: $\boldsymbol{g} \leftarrow \nabla(1/Q \sum_{q=1} E^q(\boldsymbol{y}^q, f(\boldsymbol{x}^q; \boldsymbol{w})))$;
4 Accumulate the squared gradient: $\boldsymbol{r} \leftarrow \varrho \boldsymbol{r} + (1 - \varrho)\boldsymbol{g} \odot \boldsymbol{g}$;
5 Compute the parameter update: $\Delta \boldsymbol{w} \leftarrow \kappa/(\sqrt{\boldsymbol{r} + \delta}) \odot \boldsymbol{g}$;
6 Apply the update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w}$;

Adam

As mentioned earlier, the other popular adaptive learning rate method (also included in this study) is Adam, which was proposed by Kingma and Ba [79], and holds advantages such as simplicity, computational efficiency, relatively low memory requirements, and the efficient handling of large problems (large in terms of data volume and the number of network weights). Within the context of the earlier algorithmic discussions, Adam (the name of which was derived from the phrase *adaptive moment estimation*) contains elements from both RMSProp and momentum, with some distinctions. Exponentially decaying averages of the gradients and past squared gradients are used as estimates of the first and second-order moments, respectively, and represent elements *borrowed* from momentum and RMSProp. An additional inclusion is that of bias corrections to the estimations of both the first-order and the *uncentered* second-order moments. These corrections to the uncentred second-order moments are not present within the RMSProp algorithm, and so Adam is an improvement in this regard. According to Goodfellow *et al.* [53], Adam may sometimes require further adjustment, in terms of the learning rate, before satisfactory performance is delivered, although it is regarded as "fairly robust." A pseudocode form description of Adam is presented in Algorithm 3.3. Once again, a small constant $\delta = 10^{-8}$ is introduced for numerical stabilisation. A step size $\kappa = 0.001$ as well as two exponential decay rates, denoted by ϱ_1 and ϱ_2, are also introduced, with values of 0.9 and 0.999, respectively, as prescribed by Goodfellow *et al.*

Algorithm 3.3: Adam

Input : Initial weights $\boldsymbol{w}$, a training set $\{\mathcal{X}, \mathcal{Y}\}$, a step size κ, a small constant $\delta = 10^{-8}$, and exponential decay rates ϱ_1 and ϱ_2

Output: Updated weights $\boldsymbol{w}$

$h \leftarrow 0$;
Initialise first and second-order moment variables $\boldsymbol{s} \leftarrow 0$, $\boldsymbol{r} \leftarrow 0$;
while *stopping criterion not met* **do**
  Compute the gradient: $\boldsymbol{g} \leftarrow \nabla(1/Q \sum_{q=1} E^q(\boldsymbol{y}^q, f(\boldsymbol{x}^q; \boldsymbol{w})))$;
  $h \leftarrow h + 1$;
  Update the biased first-order moment estimate: $\boldsymbol{s} \leftarrow \varrho_1 \boldsymbol{s} + (1 - \varrho_1)\boldsymbol{g}$;
  Update the biased second-order moment estimate: $\boldsymbol{r} \leftarrow \varrho_2 \boldsymbol{r} + (1 - \varrho_2)\boldsymbol{g} \odot \boldsymbol{g}$;
  Correct the bias in first-order moment: $\tilde{\boldsymbol{s}} \leftarrow \boldsymbol{s}/(1 - \varrho_1^h)$;
  Correct the bias in second-order moment: $\tilde{\boldsymbol{r}} \leftarrow \boldsymbol{r}/(1 - \varrho_2^h)$;
  Compute the parameter update: $\Delta \boldsymbol{w} \leftarrow -\kappa \tilde{\boldsymbol{s}}/(\sqrt{\tilde{\boldsymbol{r}}} + \delta)$;
  Apply the update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w}$

3.8 Regularisation

In §3.1, the problem of overfitting (*i.e.* good memorisation, but poor generalisation) was briefly discussed. The important task of reducing/preventing overfitting is accomplished by means of regularisation, which is defined by Goodfellow *et al.* [53] as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." Apart from a model's performance on the training set (*i.e.* memorisation), the performance on the testing set (*i.e.* generalisation) may be regarded as the principal priority when designing ML algorithms. This section therefore contains a brief overview of a few of the most prominent regularisation strategies found in the literature, namely L^2 and L^1 regularisation, *data set augmentation*, and *early stopping*.

3.8.1 L^2 parameter regularisation

The first strategy discussed is L^2 parameter regularisation (also referred to as *ridge regression* or *Tikhonov regularisation*), the aim of which is to *drive* the weights closer to the origin[3]. This is achieved by adding the regularisation term

$$\frac{\lambda}{2Q}\sum_{w} w^2 \tag{3.52}$$

to the error function. The additional regularisation term equates to the sum of the squares of all weight values scaled by $\lambda/2Q$, with Q being the size of the data set and $\lambda > 0$ the *regularisation parameter*. Using MSE within a batch learning paradigm as an example, the error function now has the form

$$E = \frac{1}{Q}\sum_{o=1}^{Q}\left(\boldsymbol{y}^q - f(\boldsymbol{x}^q;\boldsymbol{w})\right)^2 + \frac{\lambda}{2Q}\sum_{w} w^2, \tag{3.53}$$

or, more generally,

$$E = E_0 + \frac{\lambda}{2Q}\sum_{w} w^2,$$

where E_0 represents the original, unregularised error function. The purpose of the regularisation term is to predispose the training algorithm towards learning small weights, unless relatively large weights improve the error function notably. From (3.53), a trade-off exists between minimising the error function and finding small weights, with the relative importance of these two conflicting objectives depending on the value of λ — a large value of λ implies greater emphasis on small weight values, whereas a small value of λ implies that greater importance is attached to memorisation. The *best* value for this hyper-parameter is typically ascertained during numerical experimentation [116].

The intuition of rather having many small weights (as opposed to a few large weights) is that, collectively, all the neurons then have a greater capacity to learn the *true* underlying relationships within the data. In addition, a model with regularised parameters tends to ignore noise within the data when learning transpires — the model is, thus, less sensitive to certain inputs. The weights associated with the biases are, for this reason, excluded from the regularisation term as they do not behave similarly to normal weights and sometimes it may be desirable to have large values for these weights as saturation may be required.

According to Nielsen [116], however, a discourse on regularisation touches upon "issues which go to the heart of science." These issues pertain to the brain's generalisation capability — a matter surrounded by conjecture. Although regularisation may provide the necessary means to reduce/avoid overfitting, providing principal and fundamental reasoning *why* these strategies work is not entirely possible as mankind's understanding of the underlying principles is inadequate, especially within this context. A pragmatic approach was adopted by Nielsen, with favourable results from numerical experiments serving as the necessary justification. The same approach is consequently adopted in this dissertation — *i.e.* regularisation is implemented with reckless abandonment.

[3]Zero is chosen as the default value to regularise towards as it is unclear, beforehand, whether the *best* value to regularise towards is positive or negative. According to Goodfellow *et al.* [53], zero is by far the most common regularisation *target*.

3.8.2 L^1 parameter regularisation

The next strategy discussed is L^1 parameter regularisation (also known as *Lasso* regression) which is, intuitively, very similar to L^2 regularisation — *i.e.* large weights are penalised in favour of small weights. The L^1 regularisation term is

$$\frac{\lambda}{Q}\sum_{w}|w|. \tag{3.54}$$

According to Nielsen, when performing theoretical analysis on L^1 regularisation, weights shrink by a constant amount until they reach zero, whereas in the case of L^2 regularisation, weight shrinkage is proportional to w. Thus, when the weight magnitude, denoted by $|w|$, is large, L^2 regularisation shrinks weights faster than L^1 regularisation does (and *vice versa*). L^1 regularisation effectively concentrates the *importance* of the network on a relatively small number of high-priority connections, while the other weights are driven toward zero [116]. As a result of this *sparsity* effect, L^1 regularisation is a popular approach when performing *feature selection* — a technique used to simplify machine learning problems by choosing which subset of the available features should be considered [53].

3.8.3 Data set augmentation

ANNs thrive on a deluge of data. In practice, however, the availability of data may sometimes be a limiting factor. Fortunately, a somewhat trivial regularisation strategy has been proposed which allows for the artificial expansion of the training data, also known as *data set augmentation*, so as to further improve the model's generalisation capabilities [53]. The principal idea is to add *fake data* to the data set, which is a straightforward task for many ML problems, especially classification problems such as speech recognition and image classification. The latter case exhibits a vast number of factors of variation due to an image's high-dimensional nature, which renders the augmentation of current examples in the data set a trivial process. Simple operations, such as translating an image by a small number of pixels in any (allowable) direction, or even rotating/scaling the image, has resulted in significant improvements to generalisation [53]. Care is, however, required in special cases where these transformations would result in the undesired misclassification of examples, *e.g.* rotational transformations of "6" and "9," or "d" and "p." The *injection* of noise is another way of augmenting the original data set — a model's robustness can be improved by applying random noise to the inputs or, if a lower level of abstraction is required, to the hidden neurons. It should be noted that whenever different algorithms are compared, the same data set (with its accompanying augmentations) should be used when comparing performance as the incorporation of fake data or noise may be the main contributing factor towards one algorithm's superiority or inferiority [53].

3.8.4 Early stopping

According to Goodfellow *et al.* [53], early stopping is one of the most common regularisation strategies in the field of ANNs — attributable to its effectiveness and simplicity. The premise of this strategy is as follows: Whilst training the network, the performance on the validation data set is taken as an indicator of when (in terms of number of epochs) overfitting starts to occur. Figure 3.15 illustrates this notion — whilst the training error (loss) continues its decline, the validation error also declines, but eventually starts to increases again after some *critical point* (but it may decrease again thereafter). The critical point is stored and whenever the validation performance starts to improve again (after the critical point), the model parameters may be

stored again. One way of interpreting early stopping is by viewing it as a very efficient *hyper-parameter selection algorithm*, where the number of epochs is treated as the hyper-parameter and the critical point(s) suggest *candidate* values for this hyper-parameter. The standard procedure is to stop the training process after the validation error reaches the critical *minimum* point and no subsequent improvement is attained for some pre-specified number of epochs thereafter.

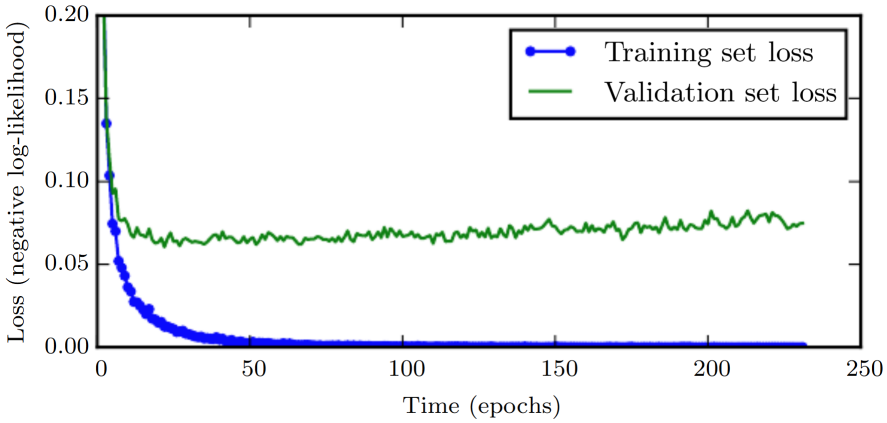


FIGURE 3.15: *A graphical representation of a training algorithm's performance on the training set (blue) and validation set (green), which is used when employing early stopping. Overfitting occurs when the validation performance starts to increase after an initial decrease [53].*

Two additional costs are incurred, the first is the evaluation of the validation error during training (this typically occurs periodically). The second is the storage of the model parameters at the critical point(s). An advantage of this method (over other methods) is that it is unobtrusive, as it does not alter the objective function, and may be used in conjunction with other regularisation strategies without much consideration of its effect on the other strategies implemented [53].

3.9 Meta-learning

An important facet of any ANN implementation pertains to the choice of training algorithm — generally speaking, this predicament refers to the *algorithm selection problem*. The common approach (or heuristic) is to select the algorithm that has achieved SOTA performance with respect to a widely-used benchmark data set (or suite of data sets). Such a selection, however, disregards to a notable extent the similarity between the benchmark data set(s) and the problem at hand[4] — arguably, an uninspired approach given the underlying context of attempting to propose and devise methodologies that "learn from experience." Fortunately, an approach that is more closely aligned with the fundamental underpinnings of ML has manifested itself in the ML domain, as reported by Vilalta and Drissi [169]. This manifestation is represented by *meta-learning* which is self-referential both in respect of its name and in respect of its field — essentially, meta-learning is concerned with the notion of *learning to learn*.

The principal aim of meta-learning is to increase the versatility of the modelling approach by incorporating domain information when deciding upon a suitable (training) algorithm for the problem (or data set) at hand. Meta-learning entails learning which characteristics, *i.e.* meta-features, of a problem (or data set) render certain algorithms more suitable. Wolpert and Macready's [178]

[4] Admittedly, only high-level correspondence is sought. For example, when performing an image recognition task, an SOTA training algorithm with respect to an unrelated image recognition data set is selected. The nuances of the problem at hand and its similarity with respect to the benchmark problem are seldom taken into account.

NFL theorem underlines the importance of an adaptive approach that takes into consideration the characteristics of different problems when selecting an appropriate algorithm — there is no omnipotent algorithm, as per the theorem.

According to Filchenkov and Pendryak [43], meta-learning transforms and reduces the algorithm selection problem[5] into a supervised learning problem in which the independent variables, *i.e.* meta-features[6], are high-level descriptions of the data set at hand and the target variable relates to the selection of an appropriate (training) algorithm, *i.e. base-leaner*. The supervised learning task can be either classification or regression, depending on the nature of the target variable selected. The meta-learning prediction algorithm, *i.e. meta-learner*, is therefore tasked with learning the relationships (*i.e.* correlations) between various high-level problem characteristics and the algorithmic performance of the base-learner(s). Learning is based on knowledge accumulated during previous implementations of the base-learner(s) with respect to different problem instances. Fundamentally speaking, the meta-learner aims to improve the performance of the base-learner by utilising and exploiting algorithmic experience from previous implementations which is learnt in a supervised learning manner, somewhat akin to *transfer learning* [43]. Although the aforementioned utility of meta-learning is expressed in the context of ANN (and ML) training algorithms, the notion of "learning to learn" has been employed successfully in other domains such as evolutionary optimisation [152, 175], constraint satisfaction [68, 83], and combinatorial optimisation [112, 180].

The BOHTA proposed later in this dissertation comprises several sub-algorithms — similar to its muse, *i.e.* AMALGAM. Due to the nature and extent of the accompanying algorithmic performance comparison carried out, a meta-learning investigation can provide insight into the dynamics of the BOHTA's constituent sub-algorithms. Furthermore, the knowledge and experience accumulated can potentially lead to performance improvements when used as part of the BOHTA's algorithmic procedure. The premise of meta-learning within this dissertation is therefore as follows: The sub-algorithms constituting the BOHTA represent the base-learners to which a suitable meta-learner is applied. White-box statistical learning algorithms, such as the tree-based algorithms described in §2.2.2, are good meta-learner candidates, attributable to the rule formulations they generate. Consequently, insight can be gained into the meta-features that contribute the most towards predicting algorithmic performance. An important precursor to such a meta-learning study, however, is the selection of an appropriate set of explanatory meta-features that will enable the meta-learner to perform the supervised meta-learning task at hand.

Castiello *et al.* [18] stated that the predictive capabilities of a meta-learner rely heavily on the quality and diversity of the meta-features selected to form part of the supervised meta-learning task. There are three main categories of standard meta-features, namely: *Generic* (*e.g.* number of features and number of classes), *statistical* (*e.g.* mean and variance of numerical features), and *information theoretic* (*e.g.* average class entropy) [18]. The selection of the meta-features considered in a meta-learning investigation should, in general, satisfy the following two criteria: (1) Usefulness towards discerning relative algorithmic performance of base-learners and (2) computational simplicity. The most popular meta-features in literature that satisfy these criteria are now described [1, 18, 43, 104].

[5] In the context of this dissertation, the algorithm selection problem specifically entails selecting a suitable ANN training algorithm for a data set under consideration.

[6] There is an important distinction between the features of a data set and its meta-features. In the case of the former, the features (*i.e.* base-level tasks) relate to the problem itself, *e.g.* the gender or age in the Titanic data set of §2.2.2. In the case of the latter, on the other hand, the meta-features describe the features themselves, *e.g.* the mean age in the Titanic data set of §2.2.2.

3.9.1 Generic meta-features

General information pertaining to the data set under consideration is assimilated by generic meta-features. Properties relating to the size and extent of the problem at hand (including its constituent base-level tasks), together with a measure of their associated difficulty, are encapsulated by this class of meta-features. Many generic meta-features can be formulated in respect of the data set under consideration; the most widely adopted are:

- *Classification task*: Does the problem comprise two classes (binary) or more classes (multi-class)?
- *Data set size*: How many rows (or instances) constitute the data set?
- *Number of independent variables*: How many features describe the input space?
- *Number of dependent variables*: How many target variables describe the output space?
- *Nature of the input data*: Do the input data comprise numerical or categorical features solely? Or a combination of the two?
- *Number of numerical input features*: How many input features are of a numerical nature?
- *Number of categorical input features*: How many input features are of a categorical nature?
- *Output-input ratio*: What is the quantitative relation between the output and input space?
- *Dimensionality*: What is the quantitative relation between the data set variables (*i.e.* independent and dependent variables) and the size of the data set?

3.9.2 Statistical meta-features

Various properties of a data set's innate distributions and numerical properties can be obtained by means of standard statistical measures. According to Castiello *et al.* [18], the use of statistical meta-features can provide insight into the properties that enable base-learners to identify and utilise the different manifestations of numerical properties (*e.g.* correlations) embedded within the data set. These properties form an integral part of any supervised ML algorithm which searches for a functional mapping from the input space to the output space. The two main statistical meta-features considered in this dissertation are *kurtosis* and *skewness*.

Kurtosis represents a popular measure for describing the "shape characteristic" of a random variable's distribution [74]. Let x_i^q denote some random variable under consideration which corresponds to input feature $i \in \{1, \ldots, n\}$. The random variable's distribution is obtained by observing all $q \in \{1, \ldots, Q\}$ examples (or instances) within the data set at hand. According to DeCarlo [29], a positive kurtosis value corresponds to a distribution that exhibits "heavier" tails and a higher peak when compared with a normal distribution. A negative kurtosis value, on the other hand, corresponds to a distribution that exhibits "lighter" tails and a flatter peak when compared with a normal distribution. The notion of kurtosis is depicted graphically in Figure 3.16.

Mathematically speaking, kurtosis is defined as the fourth central moment with respect to the mean of the random variable, denoted by $\bar{x}_i$, divided by the square of the random variable's variance, after which the kurtosis of a normal distribution, which equates to three, is subtracted. A simple correction can be applied to reduce the bias of the measure — the inclusion of so-called

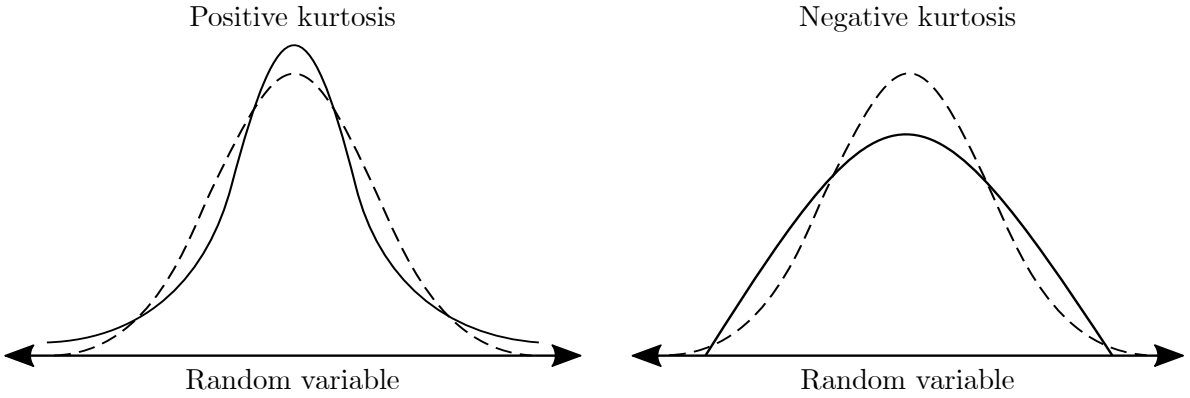


FIGURE 3.16: *Examples of data exhibiting positive kurtosis (left) and negative kurtosis (right). The dotted line represents a normal distribution. (Adapted from DeCarlo [29].)*

unbiased cumulant estimates (expressed as ratios) is typically preferred [74]. Let ξ_i denote the kurtosis of independent variable (or feature) $i \in \{1, \ldots, n\}$ which can be described mathematically as

$$\xi_i = \frac{Q-1}{(Q-2)(Q-3)} \left((Q+1) \frac{\frac{1}{Q}\sum_{q=1}^{Q}(x_i^q - \bar{x}_i)^4}{\left(\frac{1}{Q}\sum_{q=1}^{Q}(x_i^q - \bar{x}_i)^2\right)^2} + 6 \right), \tag{3.55}$$

where $\bar{x}_i$ again denotes the mean value of feature i. The kurtosis with respect to the entire (input) data set, denoted by $\bar{\xi}$, is calculated by averaging (3.55) across all n features. That is,

$$\bar{\xi} = \frac{1}{n}\sum_{i=1}^{n} \xi i. \tag{3.56}$$

Skewness, on the other hand, is a statistical measure for quantifying the degree of symmetry present within the distribution of a random variable — more specifically, it measures the *lack* of symmetry (*i.e.* asymmetry). Accordingly, it represents another measure for describing the shape of the distribution and therefore supplements kurtosis. Data that are *skewed* right have a positive skewness value, *i.e.* the right tail is heavier than the left. Conversely, data that are *skewed* left have a negative skewness value, *i.e.* the left tail is heavier than the right. The notion of skewness is illustrated graphically in Figure 3.17.

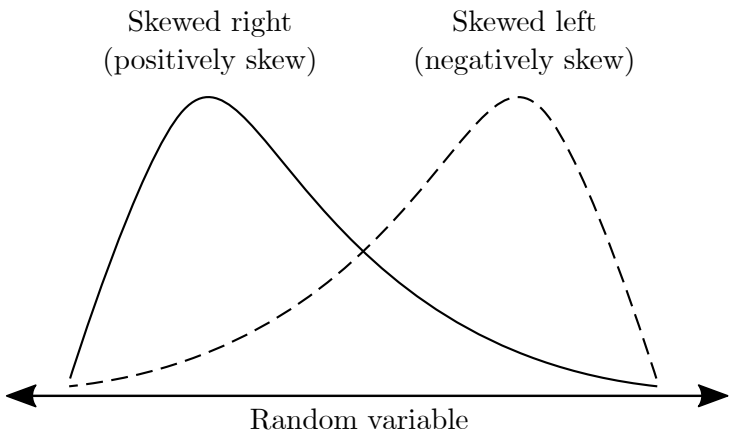


FIGURE 3.17: *Two examples of skewed data. The solid curve represents data that are skewed right (positively skew), whereas the dotted curve represents data that are skewed left (negatively skew).*

According to Joanes and Gill [74], a traditional measure of skewness is simply the third central moment with respect to the mean of the random variable divided by the 1.5^{th} power of its variance. Joanes and Gill further stated that the so-called *adjusted Fisher-Pearson standardised moment coefficient of skewness*[7] is preferred due to its unbiased nature and favourable MSE values with respect to empirical studies. Accordingly, the traditional formulation of skewness is modified by applying a simple correction which involves inserting ratios of unbiased cumulant estimates — similar to the case of kurtosis. Let ζ_i denote the Fisher-Pearson coefficient of skewness for independent variable (or feature) $i \in \{1, \ldots, n\}$ which can be described mathematically as

$$\zeta_i = \frac{\sqrt{Q(Q-1)}}{Q-2} \left(\frac{\frac{1}{Q}\sum_{q=1}^{Q}(x_i^q - \bar{x}_i)^3}{\left(\frac{1}{Q}\sum_{q=1}^{Q}(x_i^q - \bar{x}_i)^2\right)^{\frac{3}{2}}} \right), \tag{3.57}$$

where $\bar{x}_i$ denotes the mean value of feature i. The second fraction in (3.57) comprises the third central moment in the numerator and the 1.5^{th} power of the variance in the denominator. The skewness with respect to the entire (input) data set, denoted by $\bar{\zeta}$, can be calculated by averaging the Fisher-Pearson coefficient of skewness in (3.57) across all n features. That is,

$$\bar{\zeta} = \frac{1}{n}\sum_{i=1}^{n}\zeta_i. \tag{3.58}$$

3.9.3 Information theoretic meta-features

Meta-features that stem from the field of information theory can be of particular importance when attempting to *quantify information* present within some data (or random variable). Castiello *et al.* [18] underlined the utility of information theoretic meta-features especially within the context of discrete (categorical) random variables. Recall from §1.3 that each data set considered in this dissertation pertains solely to classification tasks. As a result, the dependent variables y_k^q are all discrete-valued, where $k \in \{1, \ldots, o\}$ and $q \in \{1, \ldots, Q\}$. Information theoretic meta-features with respect to the target variables of the data set can potentially aid the meta-learner towards learning the (functional) relationship between said features and the algorithmic performance of the base-learners.

Entropy is a powerful information theoretic measure employed towards approximating the level of *uncertainty* present within data (or random variable). Coincidentally, the notion of entropy forms an integral part of the C4.5 decision tree algorithm's working, as stated in §2.2.2. A slight modification is, however, made to the expression in (2.3) so as to contextualise it (for meta-learning). Accordingly, the base-2 logarithm is used instead which consequently translates the units of the information measure into bits — a common convention in meta-learning studies [18]. The resulting mathematical expression for calculating the entropy of a discrete random variable is

$$H = -\sum_{k=1}^{o} p_k \log_2 p_k, \tag{3.59}$$

where p_k denotes the probability that the random variable is equal to k where $k \in \{1, \ldots, o\}$. Furthermore, the mean class entropy, denoted by $\bar{H}$, can be obtained by averaging the expression in (3.59) across all o classes. That is,

$$\bar{H} = -\frac{1}{o}\sum_{k=1}^{o} p_k \log_2 p_k. \tag{3.60}$$

[7] For the sake of brevity, the abbreviated "Fisher-Pearson coefficient of skewness" is henceforth used.

The entropy of a random variable can provide additional insight into the distribution of the data, including the shape thereof, further supplementing the statistical meta-features defined above, *i.e.* kurtosis and skewness. Small entropy values correspond to skewed distributions, ascribed to the lower degree of uncertainty present — one class typically dominates. Large entropy values, on the other hand, correspond to more balanced distributions, ascribed to the higher degree of uncertainty present — each class is represented more equally.

3.10 Chapter summary

This chapter contained a review of the most relevant and pertinent literature pertaining to ANNs (and more specifically, FNNs). The reader was presented with the necessary background information so as to facilitate an understanding of the remainder of the research reported in this dissertation.

In §3.1, it was explained that ANNs are computational models inspired by neurological phenomena and comprise layered interconnected neurons, which, when provided with data, can learn *functional* relationships between these data. The activation functions employed by the neurons were discussed in depth in §3.2, with the more popular activation functions in the literature elaborated upon. The three current main classes of ANNs, namely feedforward, recurrent, and convolutional networks, were discussed in §3.3. The focus of this dissertation — *i.e.* on feedforward networks — was afforded greater attention in §3.4, with an emphasis on their inner working and the relevant notation used to describe these networks.

The most important and, arguably, defining facet of an ANN is its ability to learn during the so-called training process. A discourse on the relevant concepts pertaining to learning/training was conducted in §3.5. Key performance measures for the two main supervised learning problem classes, namely classification and regression, were highlighted in §3.6. In §3.7, the focus shifted to learning procedures, with a specific emphasis on the noteworthy backpropagation method, error surfaces, and the gradient descent method. Regularisation, an important technique for reducing/preventing the problem of overfitting, was discussed in §3.8, with a focus on L^2 and L^1 parameter regularisation along with data set augmentation and early stopping. The domain of meta-learning was finally discussed in §3.9 with an emphasis on the most important generic, statistical, and information theoretic meta-features found in the literature.

CHAPTER 4

Metaheuristics and Hyperheuristics

Contents

The aim in this chapter is to review, specifically within a multi-objective context, the pertinent literature related to *metaheuristic* and *hyperheuristic* optimisation techniques. Fundamental concepts and terminology in the literature are reviewed so as to facilitate an understanding of the work presented in the remainder of this dissertation. This chapter comprises three main parts. The first part focusses on the research field of metaheuristics. The discussion in this first part serves as a precursor to the next main part, which centres on the sub-field of *evolutionary optimisation*. Three key *evolutionary algorithms* (EAs) are discussed in the second part, namely the *non-dominated sorting genetic algorithm II* (NSGA-II), *non-dominated sorting differential evolution* (NSDE) algorithm, and *optimised multi-objective particle swarm optimisation* (OMOPSO) algorithm. The last part focusses on the research field of hyperheuristics, with a specific emphasis on the AMALGAM method. This method serves as the main source of inspiration for the BOHTA proposed in this dissertation. A concise summary concludes the chapter.

4.1 Metaheuristics

An incessant increase in optimisation problem complexity has compelled researchers to adopt approximate optimisation approaches, including that of metaheuristic[1] solution techniques. The term *metaheuristic* appeared for the first time in a paper by Glover [51] published in 1986. Their conception during this time heralded a new era in the domain of mathematical optimisation [118]. Since the arrival of metaheuristics, the research field has matured into one that is both

[1] In 1986, the term *metaheuristics* was coined by Fred Glover and comprises the Greek prefix *meta* (μετα) — meaning *at a higher level*, and the Greek word *heuriskein* (ευρισκειν) — meaning *to find*.

well developed and actively researched. In recent work by Sörensen and Glover [154], the following comprehensive definition was proposed:

> "*A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework.*"

According to Glover and Kochenberger [52], these solution techniques were initially designed to be robust search procedures, capable of escaping local optima by better managing the interaction between "local improvement procedures" and high-level strategies. Osman and Laporte [119], furthermore, claimed that metaheuristics intelligently combine different concepts of exploration (diversification) and exploitation (intensification) so as to better guide "subordinate heuristics" during the search. Lately (and in more general terms), the principal aim of metaheuristics may be regarded as finding — within a reasonable computation time — high-quality solutions to computationally hard optimisation problems in science and engineering [161]. The wide-spread adoption of metaheuristics has provided researchers with the necessary means to circumvent the failings of classical heuristics and exact optimisation methods (due to their ineffectiveness and/or inefficiency).

Metaheuristics, also sometimes referred to as *approximate* solution techniques, typically offer no guarantee with respect to the quality of solutions found (*i.e.* closeness to Pareto optimality), with the single exception of the method of simulated annealing. *Exact* solution techniques, on the other hand, guarantee the establishment of an optimal solution within a finite amount of time. Glover and Kochenberger claimed that exact solution techniques cannot compete with "leading" metaheuristics in many real-world cases — ascribed to the complex nature and intractability of the problems at hand. This promising field of research consequently receives continual interest from researchers.

With respect to the classification of metaheuristics, a fundamental bi-classification is apparent in the literature — the two prevailing classes are: *Single solution-based* metaheuristics and *population-based* metaheuristics [10]. Single solution-based metaheuristics, also called *trajectory methods*, operate iteratively on a single candidate solution, with exploitation being the main focus of a search procedure within this class. Some of the most prominent single solution-based metaheuristics are the method of *simulated annealing* [80], *tabu search* [51], the *variable neighbourhood search* method [108], the GRASP method [42], *guided local search* [170], *iterated local search* [158], and variants of these methods. Population-based metaheuristics, on the other hand, operate iteratively on a set (*i.e.* population) of candidate solutions, with a greater emphasis on exploration during the search. The most prominent methods in this class are GAs [63], *evolution strategies* [161], *evolutionary programming* [47], *genetic programming* [84], DE [157], *ant colony optimisation* [35], and PSO [39]. Due to the BOHTA proposed later in this dissertation being inspired by AMALGAM, and the fact that AMALGAM offers flexibility in the sense that any population-based metaheuristic can be employed as one of its sub-algorithms, this class of metaheuristics forms the basis of discussion in the remainder of this chapter. The focus is, more specifically, on a powerful sub-field of population-based metaheuristics known as *evolutionary optimisation*.

4.2 Evolutionary optimisation

Evolutionary optimisation is characterised and inspired by mechanisms and processes found within the domain of biology. Some of these processes include natural selection, migration of species, flocking of birds, and the foraging behaviour of ant colonies. By drawing inspiration from

these biological phenomena, many researchers have devised powerful optimisation approaches. Evolutionary optimisation may be regarded as the fundamental underpinning of AMALGAM — specifically with respect to the high-level mechanism that governs its lower level sub-algorithms. Furthermore, its sub-algorithms may also be based on evolutionary optimisation — hence their classification as EAs in the literature. The BOHTA proposed later in this dissertation closely emulates AMALGAM and, as a result, may also be regarded as an EA. A short exposition of evolutionary optimisation — and more specifically *multi-objective evolutionary algorithms* (MOEAs) — therefore follows so as to ensure a better understanding of the inner workings of both the sub-algorithms as well as the overarching BOHTA.

Excellent performance and wide-spread usage in the literature serve as adequate justification for researchers' continued interest in MOEAs [38, 185]. The natural result from this attention is the availability of a copious number of metaheuristics, specifically in the class of MOEAs. A good introductory text on this topic is that of Coello Coello *et al.* [21]. The generic algorithmic framework underlying these metaheuristics comprises the following functional elements:

1. *Initialisation*: An initial population of solutions is generated and their fitness levels evaluated.

2. *Selection for reproduction*: A mating pool for reproduction is created. The selection procedure (for choosing entrants to the mating pool) is typically governed by the fitness level of each parent solution, *i.e.* the desirability of being selected.

3. *Reproduction*: A new population of offspring solutions is generated by applying variation operators to the mating pool (*e.g. crossover* and *mutation*). Their fitness is evaluated thereafter.

4. *Selection for replacement*: Solutions from both the previous population and offspring population are selected in order to update the current population. This selection procedure is also governed by the fitness (*i.e.* desirability) of solutions present in the population. An *archive* (secondary population), containing previously found non-dominated solutions, is maintained by some MOEAs.

5. *Stopping criteria*: If the stopping criteria are met, the algorithm is terminated (and the current population or archive of non-dominated solutions returned as candidate solutions); otherwise, the algorithms return to step 2.

Given the aforementioned background information, an in-depth discussion follows on each of the three sub-algorithms employed in the BOHTA proposed later in this dissertation. These three sub-algorithms are the lower level metaheuristics managed by the higher level hyperheuristic. Work related to the application of EAs (including MOEAs) to ANN training is found abundantly in the literature. In this dissertation, a novel approach to training ANNs is, however, proposed. Therefore, the following sub-algorithm discussion transpires in a predominantly general context, *i.e.* each sub-algorithm discussion relates directly to its original conception. Literature related to the application of the sub-algorithms to existing ANN training approaches is therefore not the focal point of this discussion. The relevant sub-algorithm modifications and extensions (in an ANN training context) are addressed later. This sub-algorithm discussion precedes a detailed exposition of the field of hyperheuristics, as well as AMALGAM — the muse of the BOHTA.

4.2.1 The NSGA-II

One of the most prominent population-based metaheuristics, residing in the class of EAs, is the GA. The main source of inspiration for this much-celebrated metaheuristic emanates from

Charles Darwin's biological theory of evolution by natural selection [25]. Petrowski and Taillard [122] summarised the theory as follows: Within a biological species, the survival of the "fittest" individuals by means of competition — *i.e.* "selecting" those that adapt best — represents evolution within the species. By transmitting desirable characteristics from individuals to their offspring (by means of sexual reproduction in a cooperative manner), perpetuation of the species is ensured. According to Petrowski and Taillard, GAs were proposed by Holland [63] in 1975, although it was only towards the end of the 1980s that the seminal work of Goldberg [62] propelled this optimisation approach to the forefront of the field of metaheuristic research.

The *individuals* of a species represent candidate solutions to an optimisation problem, whereas the *population* represents a subset of individuals considered concurrently. Selecting an individual for reproduction or replacement depends on its desirability, which is specified by an associated *fitness* level. In order to ascertain an individual's fitness level, a *fitness function* is used. This fitness function is naturally analogous to the objective function associated with the optimisation problem at hand. Akin to the evolution of species, a population of solutions evolves over time. This evolution transpires iteratively, with each iteration being referred to as a *generation*. During each generation, a new population is created by applying evolutionary operators to solutions in the current (or previous) population. These operators represent the reproduction, disappearance, or survival of solutions in the current population. The term *parents* is used to represent the existing solutions employed for reproduction, while *offspring* represents the new solutions created.

EAs, including GAs, are fundamentally governed by their evolutionary operators. Figure 4.1 contains a graphical illustration of the two main categories of evolutionary operators, namely: *Selection operators* and *variation operators*. Each of the respective categories, furthermore, comprises two operators. With reference to selection operators, there is 1) *reproduction*, which determines a solution's likelihood for reproduction; and 2) *replacement*, which ensures that a fixed population size is maintained by selecting specific solutions to keep (or replace). The second category, on the other hand, contains 1) *mutation*, which alters a solution with a view to create a new one; and 2) *crossover*, which creates one or more offspring by combining two or more parents, thus performing the role of sexual reproduction. In the evolutionary computation literature, many different operators can be found; a good summary is provided by Talbi [161].

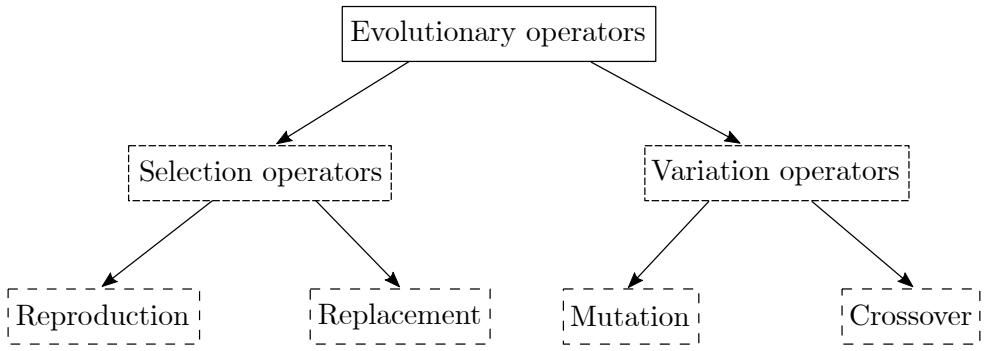


FIGURE 4.1: *The evolutionary operators of a generic GA.*

The flow diagram in Figure 4.2 encapsulates the fundamental working of a generic GA. The oval shapes in the flow diagram represent the selection operators (either reproduction or replacement), whereas the hexagonal shapes represent the variation operators (either mutation or crossover).

A GA may be applied in the contexts of both single-objective and multi-objective optimisation problems, with many different variants prevailing in the literature. AMALGAM — the muse of the BOHTA — employs a powerful variant of the GA designed for use in an MOO context,

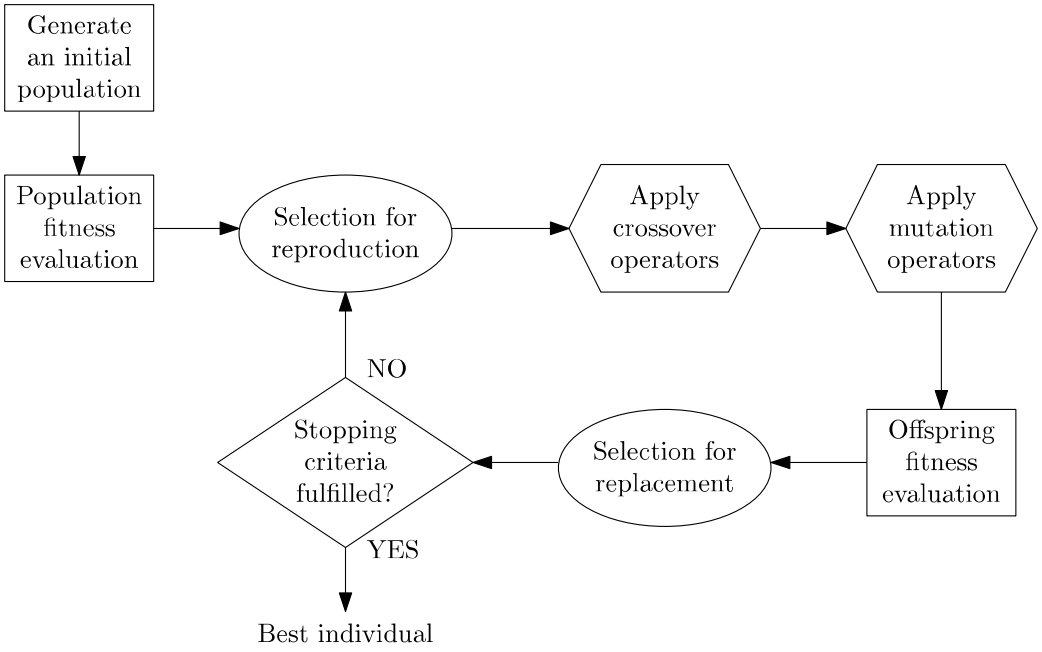


FIGURE 4.2: *The working of a generic GA [145].*

namely the NSGA-II. A number of criticisms aimed at the original version of this EA (the NSGA) — such as its lack of incorporating elitism, the computational complexity of its non-dominated sorting algorithm, and its dependence on a fitness sharing parameter — have been addressed explicitly in its successor, the NSGA-II [28]. Developed in 2002 by Deb *et al.* [28], the NSGA-II quickly became an influential EA in the field of metaheuristic solution techniques (within an MOO context). Its consideration in this dissertation may be attributed to both its proven track record in many optimisation studies as well as its previous use in the hyperheuristic at the centre of this dissertation — AMALGAM, proposed by Vrugt and Robinson [171].

The NSGA-II follows the generic algorithmic framework of an EA (discussed earlier), but with notable differences in respect of the fitness assignment and selection procedure. Within the NSGA-II, two attributes comprise a solution's fitness — its Pareto rank and its crowding distance. The first of these attributes, *i.e.* Pareto rank, is determined by the FNSA, discussed in §2.1.1. The FNSA — presented in pseudo-code form as Algorithm 2.1 — was developed by Deb *et al.* [28] with the aim of addressing the poor computational complexity associated with the sorting algorithm of the original NSGA. As a result of the incorporation of the FNSA, the computational complexity of the metaheuristic improved from $\mathcal{O}(eM^3)$ to $\mathcal{O}(eM^2)$, where e denotes the number of objectives and M the population size. The second attribute (crowding distance) is determined by means of a crowding distance assignment procedure (also discussed in §2.1.2, and presented in pseudo-code form as Algorithm 2.2). A method for distinguishing between two solutions, based on their Pareto ranks and crowding distances, is facilitated by the adoption of the crowded comparison operator, expressed in (2.2). According to Deb *et al.* [28], the incorporation of the crowded comparison operator allows for the exploration of more diverse solutions by the NSGA-II. A complete explanation of the NSGA-II, accompanied by a pseudocode description in Algorithm 4.1, now follows.

The algorithm's initialisation procedure (spanning steps 1–6) starts with the random generation of an initial parent population $\mathcal{P}_0$ of size M, which is subsequently ranked and sorted using the FNSA. Initially, a solution's fitness comprises only its Pareto rank. In order to generate the

Algorithm 4.1: NSGA-II [28, 145]

Input : An MOP, a population size M, and a maximum number of generations t_{max}.
Output: An approximate Pareto set $\tilde{\mathcal{P}}_S$ for the input MOP.

1 Generate an initial parent population $\mathcal{P}_0$ of M random candidate solutions;
2 Rank and sort the population $\mathcal{P}_0$ using the FNSA (Algorithm 2.1);
3 Assign a fitness value to each solution in $\mathcal{P}_0$ equal to its Pareto rank;
4 Create a mating pool of solutions from $\mathcal{P}_0$ by applying the relevant selection for reproduction operators;
5 Generate an offspring population $\mathcal{Q}_0$ of size M by applying a crossover operator (with probability p_c) and a mutation operator (with probability p_m) to solutions in the mating pool;
6 $t \leftarrow 0$;
7 **while** $t < t_{max}$ **do**
8 $\quad \mathcal{R}_t \leftarrow \mathcal{P}_t \cup \mathcal{Q}_t$;
9 $\quad$ Partition $\mathcal{R}_t$ into non-dominated fronts $\mathcal{F}_1, \mathcal{F}_2, \ldots$ using the FNSA (Algorithm 2.1);
10 $\quad \mathcal{P}_{t+1} \leftarrow \emptyset$;
11 $\quad j \leftarrow 1$;
12 $\quad$ **while** $|\mathcal{P}_{t+1}| < M$ **do**
13 $\quad\quad$ **if** $|\mathcal{P}_{t+1}| + |\mathcal{F}_j| \leq M$ **then**
14 $\quad\quad\quad \mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \mathcal{F}_j$;
15 $\quad\quad\quad j \leftarrow j + 1$;
16 $\quad\quad$ **else**
17 $\quad\quad\quad$ Calculate the crowding distance for each solution in $\mathcal{F}_j$ by using the crowding distance assignment procedure (Algorithm 2.2);
18 $\quad\quad\quad$ Sort $\mathcal{F}_j$ in decreasing order of crowding distance;
19 $\quad\quad\quad \mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \{\text{the first } M - |\mathcal{P}_{t+1}| \text{ solutions in } \mathcal{F}_j\}$;
20 $\quad$ Calculate the crowding distance for each solution in $\mathcal{P}_{t+1}$ using the crowding distance assignment procedure (Algorithm 2.2);
21 $\quad$ Create a mating pool of solutions from $\mathcal{P}_{t+1}$ by applying the relevant selection for reproduction operators;
22 $\quad$ Generate an offspring population $\mathcal{Q}_{t+1}$ of size M by applying a crossover operator (with probability p_c) and a mutation operator (with probability p_m) to solutions in the mating pool;
23 $\quad t \leftarrow t + 1$;
24 $\tilde{\mathcal{P}}_S \leftarrow \mathcal{P}_{t_{max}}$

offspring population $\mathcal{Q}_0$ (also of size M), a selection for reproduction operator is applied so as to select parents for reproduction, and this is followed by the application of the relevant crossover and mutation operators with the relevant crossover and mutation probabilities, denoted by p_c and p_m, respectively. This procedure is performed as many times as required, *i.e.* until the M offspring solutions have been generated. The optimisation context in which the aforementioned procedure is carried out is that of fitness minimisation (lower Pareto ranks are preferred). The generation counter t is now set to zero. The main procedure (spanning the loop in steps 7–23) is iterated until some stopping criterion is met (*e.g.* a maximum number of generations t_{max} is reached). First, the parent and offspring population are combined to create the combined population $\mathcal{R}_t \leftarrow \mathcal{P}_t \cup \mathcal{Q}_t$. This is followed by the ranking and sorting of the combined population $\mathcal{R}_t$ into non-dominated fronts $\mathcal{F}_1, \ldots, \mathcal{F}_k$ using FNSA. Crowding distances are then assigned to each solution in the

respective non-dominated fronts by means of the crowding distance assignment algorithm. The next population $\mathcal{P}_{t+1}$ is created by inserting solutions from successive fronts $\mathcal{F}_1, \mathcal{F}_2, \ldots$ until the insertion of solutions from the next front would lead to a population size exceeding M. Solutions in the last non-dominated front (*i.e.* the front that would exceed the population size) are then sorted in descending order of crowding distance and inserted one-by-one until $|\mathcal{P}_{t+1}| = M$. A mating pool of solutions is next created by applying a selection for reproduction operator (which incorporates the crowded comparison operator $\succ_{cc}$ expressed in (2.2)). Penultimately, the next offspring population $\mathcal{Q}_{t+1}$ is generated by applying the relevant crossover and mutation operators to solutions in the mating pool. Finally, the value of the generation counter is incremented, *i.e.* $t \leftarrow t + 1$.

4.2.2 NSDE

During the mid 1990s, an EA known as DE was proposed by Price and Storn [157] for solving optimisation problems over continuous domains. In an attempt to solve the *Tchebycheff polynomial fitting problem*, which was propounded by Storn, Price devised the notion of using vector differences to perturb a population of decision vectors. According to Petrowski and Taillard [122], the DE algorithm is founded on "the principles of mutation, crossover, and selection." After the conception of this algorithm, improvements and refinements followed as a direct result of numerous discussions between Price and Storn. Today, DE is regarded as a simple, yet powerful, and robust search procedure, especially within the context of continuous optimisation — *i.e.* optimisation problems in which decision variables are real-valued. DE closely follows the generic algorithmic framework of an EA (discussed earlier). The fundamental premise of DE may best be described by elucidating the mechanisms that govern its evolutionary operators. The notational convention adopted in this discussion closely emulates that of Talbi [161].

Before discussing the working of the DE algorithm, a naming convention widely adopted in the literature is first addressed. A parent vector[2] from a current population is called a *target* vector. After a target vector is mutated, it is called a *donor* vector. A so-called *trial* vector is the term used to refer to a donor vector that has undergone crossover. The DE algorithm commences with the generation of an initial population of random target vectors. In order to generate an offspring population, a donor vector is first created for each corresponding target vector by means of mutation. The mechanisms that govern the mutation operators are based on the distribution of target vectors in the population. The search directions and possible step sizes are, thus, dictated by the locations of the different target vectors selected by the mutation operators, which help direct the algorithm towards finding "good" solutions [161]. A relatively simple mutation operator is given by

$$\boldsymbol{v}_j^{t+1} = \boldsymbol{d}_{r_1}^t + F \cdot (\boldsymbol{d}_{r_2}^t - \boldsymbol{d}_{r_3}^t), \tag{4.1}$$

where $\boldsymbol{v}_j^t$ denotes donor vector $j \in \{1, \ldots, M\}$ during generation t. Furthermore, $\boldsymbol{d}_{r_1}^t$,$\boldsymbol{d}_{r_2}^t$, and $\boldsymbol{d}_{r_3}^t$ denote three random, yet distinct, target vectors (*i.e.* $j \neq r_1 \neq r_2 \neq r_3$), while F is the so-called *amplification factor*. According to (4.1), the perturbations diminish as the distance between the target vectors that constitute the difference vector $\boldsymbol{d}_{r_2}^t - \boldsymbol{d}_{r_3}^t$ becomes smaller. The real-valued amplification factor aids the algorithm in avoiding stagnation during the search process [161].

Mutation is followed by the application of a crossover procedure, which involves "mixing" a donor vector $\boldsymbol{v}_j^t$ with a corresponding target vector $\boldsymbol{d}_j^t$. Crossover essentially entails the recombination of the decision variables of the donor vector

$$\boldsymbol{v}_j^t = [v_{j,1}^t, \ldots, v_{j,i}^t, \ldots, v_{j,n}^t], \qquad j \in \{1, \ldots, M\},$$

[2] The terms vector and decision vector are used interchangeably for the sake of brevity.

and the decision variables of the target vector

$$\boldsymbol{d}_j^t = [d_{j,1}^t, \ldots, d_{j,i}^t, \ldots, d_{j,n}^t], \qquad j \in \{1, \ldots, M\}.$$

This recombination procedure produces a trial vector and is governed by a crossover operator. An example of a commonly used crossover operator (employed in the original version of DE) is

$$u_{j,i}^{t+1} = \begin{cases} v_{j,i}^{t+1}, & \text{if } \mathsf{rand}_i[0,1] \leq C_R \text{ or } j = j_{rand}, \qquad i \in \{1, \ldots, n\}, \\ d_{j,i}^t, & \text{if } \mathsf{rand}_i[0,1] > C_R \text{ and } j \neq j_{rand}, \qquad i \in \{1, \ldots, n\}, \end{cases} \tag{4.2}$$

where $u_{j,i}^{t+1}$ denotes decision variable $i \in \{1, \ldots, n\}$ in trial vector $j \in \{1, \ldots, M\}$. The parameter C_R denotes the *crossover rate*. The condition $j = j_{rand}$, where $j_{rand} = \mathsf{int}(\mathsf{rand}_i[0,1]M) + 1$, ensures that the trial vector inherits at least one decision variable from the donor vector. In order to update the current population (*i.e.* generate the offspring population) a replacement operator is applied. One simple replacement operator (also employed in the original version of DE) is

$$\boldsymbol{d}_j^{t+1} = \begin{cases} \boldsymbol{u}_j^{t+1}, & \text{if } h(\boldsymbol{u}_j^{t+1}) > h(\boldsymbol{d}_j^t), \\ \boldsymbol{d}_j^t, & \text{otherwise}, \end{cases} \tag{4.3}$$

which is applicable to a single-objective, maximisation problem. According to this operator, a trial vector only replaces its corresponding target (parent) vector if it exhibits an improvement with respect to fitness level. This replacement procedure concludes the operations performed in respect of the current generation. In summary: During each generation, a corresponding donor vector is created for each target vector *via* mutation. Each donor vector is subsequently recombined with its target vector during crossover, so as to generate a trial vector. A trial vector only replaces its corresponding target vector if its fitness level is sufficiently superior. This procedure is repeated until some stopping criterion is met. Figure 4.3 illustrates graphically the fundamental working of the DE algorithm.

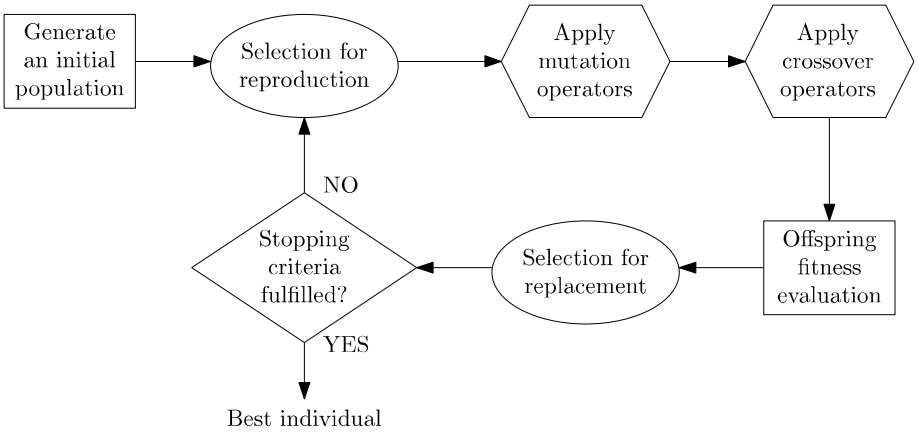


FIGURE 4.3: *The working of a generic DE algorithm.*

It is common practice (in the literature) to employ a so-called DE *scheme* which follows the notational convention: *DE*/a/b/c. According to this notation, the placeholder a specifies the nature of the mutation operator (rand and best are popular options, where the former denotes a randomly chosen target vector and the latter denotes the best target vector in the population). The placeholder b specifies the number of difference-vectors employed during mutation. Lastly, c specifies the nature of the crossover operator employed.

A DE scheme that employs the mutation operator in (4.1) and the crossover operator in (4.2) is denoted by *DE*/rand/1/bin. This specific denotation is attributed to the fact that a random

trial vector (*i.e.* $\boldsymbol{d}_{r_1}^t$) is mutated using one difference vector (*i.e.* $\boldsymbol{d}_{r_2}^t - \boldsymbol{d}_{r_3}^t$) and due to the application of a binomial crossover operator. Price and Storn [157] proposed numerous variants to the mutation scheme in *DE*/rand/1/bin. The last placeholder, which specifies the crossover operator, is omitted in the subsequent discussion as the focus is on the mutation operator. The first variant is *DE*/rand/2, and the corresponding mutation operator is given by

$$\boldsymbol{v}_j^{t+1} = \boldsymbol{d}_{r_1}^t + F \cdot (\boldsymbol{d}_{r_2}^t - \boldsymbol{d}_{r_3}^t) + F \cdot (\boldsymbol{d}_{r_4}^t - \boldsymbol{d}_{r_5}^t). \tag{4.4}$$

In (4.4), two difference vectors are employed and the condition $j \neq r_1 \neq r_2 \neq r_3 \neq r_4 \neq r_5$ must hold. The second variant is *DE*/best/1, and its mutation operator is given by

$$\boldsymbol{v}_j^{t+1} = \boldsymbol{d}_{best}^t + F \cdot (\boldsymbol{d}_{r_1}^t - \boldsymbol{d}_{r_2}^t), \tag{4.5}$$

where $\boldsymbol{d}_{best}^t$ represents the target vector with the best fitness level in the current population. Only one difference vector is used to perturb the best target vector. The condition $r_1 \neq r_2$ must also hold. The third variant is *DE*/best/2, and its mutation operator is given by

$$\boldsymbol{v}_j^{t+1} = \boldsymbol{d}_{best}^t + F \cdot (\boldsymbol{d}_{r_1}^t - \boldsymbol{d}_{r_2}^t) + F \cdot (\boldsymbol{d}_{r_3}^t - \boldsymbol{d}_{r_4}^t). \tag{4.6}$$

In (4.6), two difference vectors are employed and the condition $r_1 \neq r_2 \neq r_3 \neq r_4$ must hold. The last mutation operator variant proposed by Price and Storn is *DE*/target-to-best/1, and its mutation operator is given by

$$\boldsymbol{v}_j^{t+1} = \boldsymbol{d}_j^t + F \cdot (\boldsymbol{d}_{r_1}^t - \boldsymbol{d}_{r_2}^t). \tag{4.7}$$

In (4.7), the vector mutated is the corresponding target vector $\boldsymbol{d}_j^t$ and the condition $r_1 \neq r_2$ must hold. From a relatively recent SOTA survey by Das and Suganthan [26], the scheme *DE*/rand/2/bin was found to outperform all other schemes in respect of thirteen benchmark problems. This scheme employs the mutation operator

$$\boldsymbol{v}_j^{t+1} = \boldsymbol{d}_{r_1}^t + \frac{F}{2} \cdot (\boldsymbol{d}_{r_1}^t - \boldsymbol{d}_{r_2}^t - \boldsymbol{d}_{r_3}^t), \tag{4.8}$$

where $\boldsymbol{d}_{r_1}^t$, $\boldsymbol{d}_{r_2}^t$, and $\boldsymbol{d}_{r_3}^t$ are distinct random target vectors subject to the conditions: $h(\boldsymbol{d}_{r_1}^t) \geq h(\boldsymbol{d}_{r_2}^t)$ and $h(\boldsymbol{d}_{r_1}^t) \geq h(\boldsymbol{d}_{r_3}^t)$ within a single-objective, maximisation context. Based on the experimental results of the survey, this scheme performed the best in terms of "final accuracy and robustness," regardless of the characteristics of the benchmark problem at hand [26].

The two most prominent crossover operators are *binomial crossover* (denoted by bin) and *exponential crossover* (denoted by exp). Binomial crossover (introduced in (4.3)) is a relatively simple operator, whereas exponential crossover is more complex. According to the SOTA survey by Das and Suganthan [26], however, the best performing DE scheme, given in (4.8), employs a binomial crossover operator, and so an in-depth discussion on exponential crossover is excluded.

According to the SOTA surveys by Das and Suganthan [26] and by Mezura-Montes *et al.* [103], the NSDE algorithm designed by Iorio and Li [70] is one of the best-performing versions of DE in an MOO context, outperforming the celebrated NSGA-II in respect of numerous benchmarks. NSDE may be regarded as a simple modification of the NSGA-II. The main distinction pertains to how the offspring solutions are created — more specifically, the nature of the different evolutionary operators applied (*i.e.* reproduction, replacement, crossover, and mutation). The full NSDE is now described, and a pseudocode description of the method is presented as Algorithm 4.2.

The algorithm's initialisation procedure (spanning steps 1–7) starts with the generation of an initial parent population $\mathcal{P}_0$ comprising M random target (parent) vectors. This population is

Algorithm 4.2: NSDE [70]

Input : An MOP, a population size M, and a maximum number of generations t_{max}.
Output: An approximate Pareto set $\tilde{\mathcal{P}}_S$ for the input MOP.

1 Generate an initial parent population $\mathcal{P}_0$ of M random target vectors;
2 Rank and sort the population $\mathcal{P}_0$ using the FNSA (Algorithm 2.1);
3 Assign a fitness value to each solution in $\mathcal{P}_0$ equal to its Pareto rank;
4 Generate a donor population $\mathcal{V}_0$ of size M by applying mutation operators to $\mathcal{P}_0$;
5 Generate a trial population $\mathcal{U}_0$ of size M by applying crossover operators to $\mathcal{V}_0$;
6 Generate an offspring population $\mathcal{Q}_0$ of size M by applying replacement operators to $\mathcal{U}_0$;
7 $t \leftarrow 0$;
8 **while** $t < t_{max}$ **do**
9 $\quad \mathcal{R}_t \leftarrow \mathcal{P}_t \cup \mathcal{Q}_t$;
10 $\quad$ Partition $\mathcal{R}_t$ into non-dominated fronts $\mathcal{F}_1, \mathcal{F}_2, \ldots$ using the FNSA (Algorithm 2.1);
11 $\quad \mathcal{P}_{t+1} \leftarrow \emptyset$;
12 $\quad j \leftarrow 1$;
13 $\quad$ **while** $|\mathcal{P}_{t+1}| < M$ **do**
14 $\quad\quad$ **if** $|\mathcal{P}_{t+1}| + |\mathcal{F}_j| \leq M$ **then**
15 $\quad\quad\quad \mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \mathcal{F}_j$;
16 $\quad\quad\quad j \leftarrow j + 1$;
17 $\quad\quad$ **else**
18 $\quad\quad\quad$ Calculate the crowding distance for each solution in $\mathcal{F}_j$ using the crowding distance assignment procedure (Algorithm 2.2);
19 $\quad\quad\quad$ Sort $\mathcal{F}_j$ in decreasing order of crowding distance;
20 $\quad\quad\quad \mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \{\text{the first } M - |\mathcal{P}_{t+1}| \text{ solutions in } \mathcal{F}_j\}$;
21 $\quad$ Calculate the crowding distance for each solution in $\mathcal{P}_{t+1}$ using the crowding distance assignment procedure (Algorithm 2.2);
22 $\quad$ Generate a donor population $\mathcal{V}_{t+1}$ of size M by applying mutation operators to $\mathcal{P}_{t+1}$;
23 $\quad$ Generate a trial population $\mathcal{U}_{t+1}$ of size M by applying crossover operators to $\mathcal{V}_{t+1}$;
24 $\quad$ Generate an offspring population $\mathcal{Q}_{t+1}$ of size M by applying replacement operators to $\mathcal{U}_{t+1}$;
25 $\quad t \leftarrow t + 1$;
26 $\tilde{\mathcal{P}}_S \leftarrow \mathcal{P}_{t_{max}}$

subsequently ranked and sorted according to the FNSA in Algorithm 2.1. Initially, a vector's fitness comprises only its Pareto rank, which is used by the mutation operators to identify the required target vectors for its mutation procedure. A population of donor vectors $\mathcal{V}_0$ is then created by applying mutation operators to each target vector in $\mathcal{P}_0$. This procedure is followed by the application of crossover operators to each donor vector in $\mathcal{V}_0$ so as to create a population of trial vectors $\mathcal{U}_0$. The offspring population $\mathcal{Q}_0$ can next be generated by applying the replacement operators to solutions in $\mathcal{U}_0$. The generation counter t is then set to zero.

The main procedure (spanning the loop in steps 8–25) is iterated until some stopping criterion is met (*e.g.* a maximum number of generations t_{max} is reached). First, the parent and offspring populations are combined to create the population $\mathcal{R}_t \leftarrow \mathcal{P}_t \cup \mathcal{Q}_t$. This is followed by the ranking and sorting of the population $\mathcal{R}_t$ into non-dominated fronts $\mathcal{F}_1, \ldots, \mathcal{F}_k$ using the FNSA in Algorithm 2.1. Crowding distances are then assigned to each vector in the respective non-dominated fronts by invoking the crowding distance assignment procedure in Algorithm 2.2. The next population $\mathcal{P}_{t+1}$ is created by inserting vectors from successive fronts $\mathcal{F}_1, \mathcal{F}_2, \ldots$ until the

insertion of vectors from the next front would lead to a population size exceeding M. Vectors in the last non-dominated front (*i.e.* the front that would exceed the population size) are then sorted in descending order of crowding distance and inserted one-by-one until $|\mathcal{P}_{t+1}| = M$. The next offspring population $\mathcal{Q}_{t+1}$ is generated by first applying the relevant mutation operators to $\mathcal{P}_{t+1}$ so as to generate $\mathcal{V}_{t+1}$. The mutation operators employ the crowded comparison operator when selecting target vectors to form part of the mutation procedure. This is followed by the application of crossover operators to $\mathcal{V}_{t+1}$ so as to generate $\mathcal{U}_{t+1}$. Replacement operators (which also incorporate the crowded comparison operator) are subsequently applied to $\mathcal{U}_{t+1}$ in order to generate $\mathcal{Q}_{t+1}$. Finally, the value of the generation counter is incremented, *i.e.* $t \leftarrow t + 1$.

4.2.3 OMOPSO

The phenomenon of birds flocking served as inspiration for Kennedy and Eberhart [39] when they devised a novel, influential population-based metaheuristic in 1995, called PSO. The muse was, more specifically, a computer simulation study in which a coordinated flock of birds searched for food by means of simple information-sharing rules. Before discussing the inner workings of PSO, a peculiar discrepancy in the literature is first addressed. PSO is classified as a population-based metaheuristic that resides within the broader class of optimisation approaches known as *swarm intelligence* (SI) methods. Techniques within SI are inspired by the collective behavioural dynamics of insect swarms (*i.e.* colonies), and are typically characterised by underlying principles from the domains of *self-organisation* and local/indirect *information exchange* [122, 137]. Many researchers view EAs and SI as disparate fields, a finding corroborated by Simon [151]. Simon further stated, however, that there are also many researchers who view SI as a sub-field of EAs — one prominent instance is one of the inventors of PSO, Russel Eberhart [149]. When one considers the general procedure of PSO it ought to be apparent that, on a fundamental level, SI operates similar to EAs — a population of candidate solutions are "evolved" and improve iteratively over a number of generations. SI (including PSO) is therefore also viewed as a sub-field of EAs in this dissertation.

In PSO, solutions — *i.e.* points or locations in decision space — are represented by *particles*. A *swarm* of these particles constitutes a population, which is maintained throughout the algorithm's execution. Like a flock of birds or insects flying over some region (foraging for food), the swarm of particles analogously moves through the multi-dimensional decision space, searching for optimal locations. There are two main factors that affect the flight path of a bird, namely: *Cognitive* factors and *social* factors [137]. Cognitive factors are modelled by the effect of the bird's own location history, whereas social factors are modelled by the effect of the other birds' location histories. A bird may, consequently, have its flight path directed towards a favourable location (containing food), or directed towards a favourable location that the rest of the swarm knows about. In PSO, these notions may be represented algorithmically as follows: A particle's movement is dictated by its current *velocity* and the *positions* of favourable locations already identified by the particle itself (cognitive factors) or by those of other particles (social factors). The context in which social factors are considered is usually characterised by a neighbourhood topology — *i.e.* the swarm's social network. The topology provides a formal specification for each particle, defining with whom it can exchange information (*i.e.* communicate). Typically, all particles can communicate with each other. This network topology is known as fully connected.

Execution of a generic PSO algorithm involves the following: A swarm of particles is initialised — *i.e.* each particle is assigned a random position and velocity. The *personal best* position of each particle is recorded along with the *global best* position of the swarm. If the neighbourhood topology is not fully connected (*i.e.* limited), then the global best with respect to the neighbourhood

is recorded. Updating a particle's velocity involves combining the particle's personal best with the global best solution (with respect to the swarm or neighbourhood) and is used to adjust its velocity iteratively, with a random component incorporated into the update rule. This generic PSO procedure is illustrated graphically in Figure 4.4.

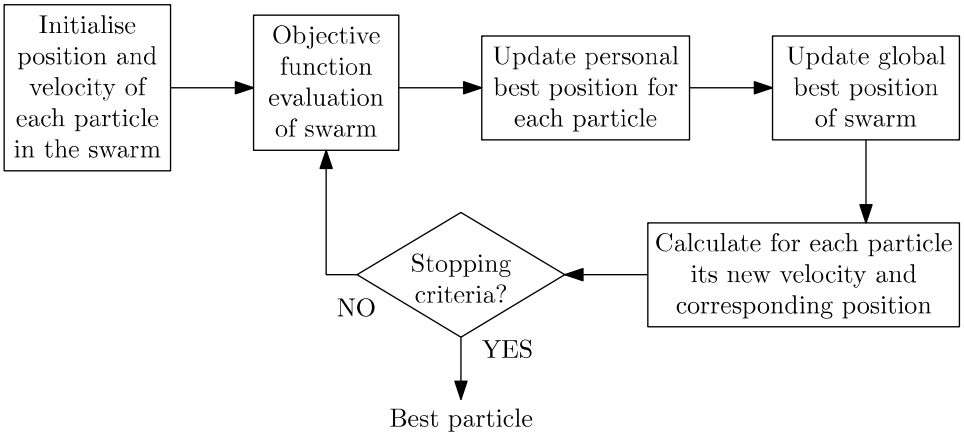


FIGURE 4.4: *The working of a generic PSO algorithm [145].*

Many different versions of PSO are present in the literature. Vrugt and Robinson [171], however, employed a relatively simple implementation of PSO in their AMALGAM method. The simplicity of their approach can be attributed to the manner in which the personal best and global best positions are determined. The following intuition underlies their approach. According to the authors, it is natural that the quality of a solution should be assessed in terms of how far its objective function vector is from that of a Pareto optimal decision vector in objective function space. The location of a Pareto optimal decision vector is not always known and, as a result, the identification of the personal and global best positions is instead based on the Euclidean distance between the respective particles and the best objective function values found thus far during the search. In this dissertation, however, a more sophisticated variant of PSO (in an MOP context) is employed to form part of the BOHTA. This choice is attributed to findings in a relatively recent survey of SOTA PSO algorithms, conducted by Durillo *et al.* [37]. It was found empirically that the OMOPSO algorithm developed by Sierra and Coello Coello [150] outperformed five other SOTA PSO algorithms in respect of several benchmark problems. The adoption of OMOPSO in this dissertation is therefore warranted.

According to Sierra and Coello Coello [150], the OMOPSO algorithm specifically addresses issues related to the application of PSO in an MOO context. In order to apply the standard PSO algorithm to MOPs, modifications are typically made with respect to the selection of the personal best and global best positions. According to its creators, the OMOPSO algorithm is afforded greater robustness and effectiveness as a result of the strategy it employs when selecting these positions. One of the key differences between the OMOPSO algorithm and the standard PSO algorithm is the use of an additional swarm, called the *leader swarm*. This leader swarm is an archive that contains strictly non-dominated solutions and employs the concept of ϵ-dominance (described in §2.1). During every iteration, different global best positions are selected from the leader swarm and assigned to each particle by applying selection for reproduction operators. The concept of crowding distance (as defined in §2.1.2) is employed to facilitate the selection process. The OMOPSO algorithm employs different selection for reproduction operators to select, within the context of minimising crowding distance, a globally best position for each particle. After each particle in the swarm has had its position updated by means of a so-called

flight operator, the swarm is partitioned into three subsets. The first subset remains unaltered, although different mutation operators, borrowed from the evolutionary computation literature (mentioned in §4.2.1), are applied to the second and third subsets.

The notation adopted in this section closely emulates that of Sierra and Coello Coello [150]. As may be seen in Figure 4.4, the algorithm commences with the initialisation of a regular swarm of particles. This procedure involves the assignment of a position vector, denoted by $\boldsymbol{d}_j^t$, and a velocity vector, denoted by $\boldsymbol{v}_j^t$, to each particle j during generation t. The position of a particle is updated by means of the flight operator as

$$\boldsymbol{d}_j^{t+1} = \boldsymbol{d}_j^t + \boldsymbol{v}_j^{t+1}. \tag{4.9}$$

Before discussing the flight operator responsible for updating the velocity of a particle, the notations $\boldsymbol{d}_{pb,j}$ and $\boldsymbol{d}_{gb}$ are introduced, denoting the personal best solution uncovered by particle j and the global best position of the swarm, respectively. The velocity of a particle is therefore updated by applying the flight operator

$$\boldsymbol{v}_j^{t+1} = W\boldsymbol{v}_j^t + C_1 r_1(\boldsymbol{d}_{pb,j} - \boldsymbol{d}_j^t) + C_2 r_2(\boldsymbol{d}_{gb} - \boldsymbol{d}_j^t), \tag{4.10}$$

where W is the inertia weight, C_1 and C_2 are so-called learning factors, and $r_1, r_2 \in [0, 1]$ are random numbers. In the original version of OMOPSO, the same value of ϵ is associated in the context of ϵ-dominance with each objective function for the sake of simplicity. Furthermore, this parameter is typically tuned based on the desired number of points in the final Pareto front [150].

The full OMOPSO algorithm may now be delineated, and a pseudocode description thereof is presented in Algorithm 4.3. The algorithm commences with the generation of a regular swarm $\mathcal{R}$ comprising M random solutions. The personal best positions of each particle is then set to its randomly initialised position, *i.e.* $\boldsymbol{d}_{pb,j} \leftarrow \boldsymbol{d}_j^0$. This is followed by the creation of the leader swarm $\mathcal{L}_0$, which has a maximum size of M and contains a copy of each non-dominated solution in $\mathcal{R}$. The crowding distance of each solution within the leader swarm is calculated by applying Algorithm 2.2. These solutions are next used to create the ϵ-archive, denoted by $\mathcal{A}_0^\epsilon$. The generation counter is set to zero before the algorithm's main procedure (spanning steps 7–23) is iterated until a stopping criterion is met (*e.g.* a maximum number of generations t_{max} is reached). From the leader swarm $\mathcal{L}_t$, selection for reproduction operators (based on crowding distance) are applied so as to identify a globally best solution $\boldsymbol{d}_{gb}$ for each particle $j \in \mathcal{R}$. The flight operators (4.10) and (4.9) are next applied to update the velocity and position of particle j, respectively. Thereafter, the regular swarm $\mathcal{R}$ is partitioned into three subsets of approximately equal size, denoted by $\mathcal{Q}_1$, $\mathcal{Q}_2$, and $\mathcal{Q}_3$. Different mutation operators are subsequently applied to particles in the second and third subsets. Each particle $j \in \mathcal{R}$ has its personal best position updated according to the following procedure: if $\boldsymbol{d}_j^{t+1} \succ \boldsymbol{d}_{pb,j}$ or if $\boldsymbol{d}_j^{t+1}$ and $\boldsymbol{d}_{pb,j}$ are not dominated by one another; then $\boldsymbol{d}_{pb,j} \leftarrow \boldsymbol{d}_j^{t+1}$. The non-dominated solutions in the combined swarm $\mathcal{L}_t \cup \mathcal{R}$ are identified and then used to create the new leader swarm $\mathcal{L}_{t+1}$. Algorithm 2.2 is subsequently executed to calculate the crowding distance of each solution in the leader swarm. If $|\mathcal{L}_{t+1}| > M$, then solutions with the smallest crowding distance are removed from the leader swarm until $|\mathcal{L}_{t+1}| = M$. The penultimate step is to update the ϵ-archive $\mathcal{A}_{t+1}^\epsilon$. Finally, the generation counter is incremented accordingly, *i.e.* $t \leftarrow t + 1$.

In this dissertation, a modified version of the original OMOPSO algorithm is implemented. Two changes are made with respect to the original algorithm. These changes are motivated by the experimental comparisons in the previously mentioned survey of SOTA PSO algorithms [37]. The two changes are as follows: First, the regular swarm is partitioned into only two subsets, with the mutation operators only being applied to particles in the second subset (the first subset remains unchanged). Secondly, the leader swarm is not subject to the notion of ϵ-dominance, and so the final leader swarm represents the approximate Pareto set.

Algorithm 4.3: OMOPSO [145, 150]

Input : An MOP, a swarm size M, a maximum number of generations t_{max}, and an ϵ-value.

Output: An approximate Pareto set, $\tilde{\mathcal{P}}_S$ for the input MOP.

```
Generate an initial regular swarm R comprising M random solutions;
For each solution, set personal best as d_{pb,j} ← d_j^0;
Create an initial leader swarm L_0 by copying non-dominated solutions from R;
Calculate the crowding distance for each solution in L_0 using Algorithm 2.2;
Determine the initial ε-archive A_0^ε using L_0;
t ← 0;
while t < t_max do
    foreach particle j ∈ R do
        Use selection for reproduction operators (based on crowding distance) to identify a
        global best position d_{gb};
        Obtain v_j^{t+1} by applying velocity flight operator (4.10);
        Obtain d_j^{t+1} by applying position flight operator (4.9);
    Partition R into three sub-sets Q_1, Q_2, and Q_3;
    Apply different mutation operators to sub-sets Q_2 and Q_3, respectively;
    foreach particle j ∈ R do
        if d_j^{t+1} ≻ d_{pb,j} or d_j^{t+1}, d_{pb,j} non-dominated by each other then
            d_{pb,j} ← d_j^{t+1};
    Create L_{t+1} by copying all non-dominated solutions from L_t ∪ R;
    Calculate the crowding distance for each solution in L_{t+1} using Algorithm 2.2;
    if |L_{t+1}| > M then
        Sort L_{t+1} in decreasing order of crowding distance;
        Remove |L_{t+1}| − M solutions from L_{t+1};
    Update ε-archive as A_{t+1}^ε ← L_{t+1} ∪ A_t^ε;
    t ← t + 1;
P̃_S ← A_{t_max}^ε
```

4.3 Hyperheuristics

According to the seminal paper by Burke *et al.* [16] titled *Hyperheuristics: A survey of the state of the art*, when hyperheuristics were officially[3] conceived (in 2000), they were regarded — within the field of combinatorial optimisation — as *heuristics to choose heuristics*. Traditionally, hyperheuristics search for good optimisation methods, rather than for good solutions to optimisation problems, and only use limited problem-specific information to inform the search process. As illustrated graphically in Figure 4.5, the main distinguishing feature between a hyperheuristic optimisation approach and a metaheuristic optimisation approach is that the former operates first on a heuristic search space and then indirectly on the solution search space, whereas in the case of the latter, the solution search space is directly operated upon. Hyperheuristics, thus, manage a set of *low-level* heuristics. An advantage of operating on a heuristic space is the potential achievement of greater search effectiveness as heuristics may provide an "advantageous search space structure" [16].

[3]The origins of hyperheuristics can be traced back to the 1960s, although the notion of automating heuristic-design was only officially referred to as a hyperheuristic approach in 2000.

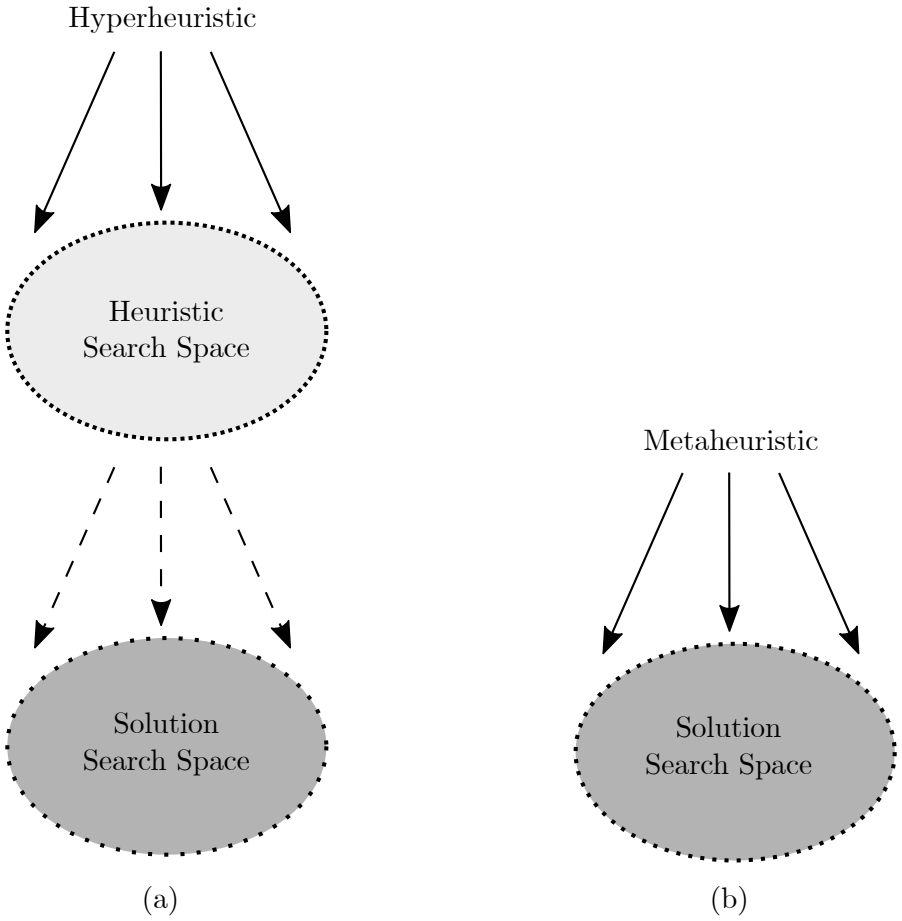


FIGURE 4.5: *(a) A hyperheuristic search approach* versus *(b) a metaheuristic search approach [16].*

One key motivation behind the development of hyperheuristics was the idea of solving computationally hard search problems by means of *automated heuristic-design* — in the process, raising the level of general applicability of the newly constructed *optimisation strategy*. Other noteworthy contributing factors pertaining to the emergence of this relatively new field of research was the difficulty of *easily* applying heuristic and other search methods to newly encountered problems (or, even, new instances of familiar problems). This difficulty may be attributed to the considerable range of both parameter and algorithm choices, as well as the general lack of guidance in this respect within the literature [16]. The scientific community's level of understanding of why different heuristic approaches work effectively (or ineffectively) in different scenarios failed to facilitate straightforward advice as to which approach to follow in a given situation.

"A search method or learning mechanism for selecting or generating heuristics to solve computational search problems" is a more formal definition of a hyperheuristic proposed by Burke *et al.* [17] which further elucidates their nature. From this definition, two main hyperheuristic classes can be identified, namely *generative* and *selective* hyperheuristics. The former focusses on methodologies for constructing new heuristics using components from existing heuristics, whereas the latter focusses on methodologies for selecting existing heuristics. The graphical illustration in Figure 4.6 indicates two further sub-classifications: *Constructive* and *perturbative* methods. Perturbative methods operate in respect of *complete* candidate solutions and modify one or more of their components, whereas constructive methods consider *partial* candidate solutions (*i.e.* solutions missing one or more components), iteratively constructing them until a complete solution is obtained.

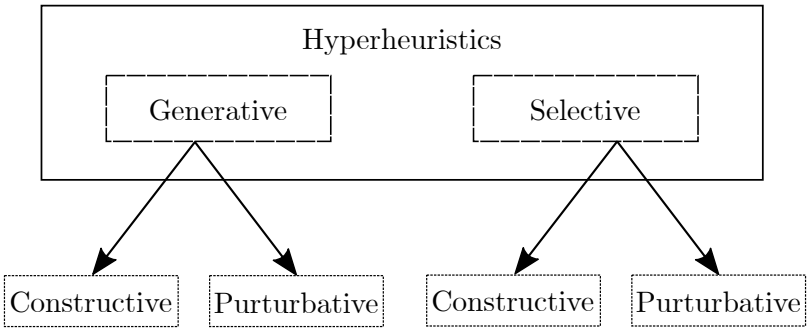


FIGURE 4.6: *A classification of hyperheuristic search approaches [16].*

From the definition, a hyperheuristic may also be regarded as a *learning mechanism* (instead of a search method), but only if some feedback from the search process is utilised. Two paradigms constitute learning: It either occurs *online* or *offline* (similar to the paradigms found within the field of machine learning). In the former case, learning transpires while the algorithm is solving a problem instance — *i.e.* modifying or constructing a candidate solution. In an offline learning paradigm, on the other hand, learning transpires when knowledge (in the form of a set of rules) is attained from a set of training instances — *i.e.* their modifications to constructions of candidate solutions.

As stated in the scope delimitation of §1.3, selective hyperheuristics are the focus of this dissertation. Consequently, an emphasis is specifically placed on this class of hyperheuristics in the current discussion. Selective hyperheuristics of a constructive nature are typically presented with an empty or partial solution, and the algorithm progressively and systematically builds upon such a partial solution until the final state (*i.e. a complete solution*) is obtained. The hyperheuristic has access to a set of problem-specific, pre-existing heuristics and the task is to select a single heuristic (or sequence of heuristics) for constructing the solution until the final state is reached. Due to the final state being a complete solution, the heuristic sequence has a finite length, which usually depends on the complexity of the underlying combinatorial optimisation problem. This class of hyperheuristics has been applied (within both the online and offline learning paradigms) to many combinatorial optimisation problems, such as production scheduling, educational timetabling, strip packing, workforce scheduling, constraint satisfaction, and vehicle routing [16].

In terms of selective hyperheuristics of a perturbative nature, the objective is to improve "full" candidate solutions by means of the automated selection and application of heuristics. This class of hyperheuristics has been applied to personnel scheduling, educational timetabling, space allocation, cutting and packing, vehicle routing, and sports scheduling [16]. When employing this methodology, two options are available with respect to the number of solutions that are processed. The first is *single-point* or *trajectory-based* search, which involves a single candidate solution undergoing consecutive perturbations until a final step (*i.e.* termination) is reached. The alternative approach involves processing multiple solutions simultaneously, referred to as *multi-point* or *population-based* search. According to Burke *et al.* [16], single-point search is the more popular of the two. Regardless of the approach followed, the underlying procedure remains the same — a single heuristic (or subset of heuristics) is selected from a set of low-level perturbative heuristics and applied to the candidate solution(s). After the solutions have been perturbed, a decision is made as to whether or not to accept them. Evidently, two components comprise a selective-perturbative hyperheuristic, namely a *heuristic selection method* and a *move acceptance method*. Different strategies are available for each of these components.

Whenever a learning mechanism is not incorporated in a heuristic selection method, some *random* or *exhaustive* process dictates the selection strategy. If learning is incorporated, however, the majority of implementations are online learning-based. Within this paradigm, an online score accompanies each heuristic, and is methodically processed as part of the selection strategy. Such a score-based hyperheuristic framework comprises a number of components, namely:

(i) The initial score,

(ii) a memory length adjustment,

(iii) a heuristic selection strategy (scored-based),

(iv) a score updating-rule (for the purpose of solution improvement), and

(v) another score updating-rule (for the purpose of solution degradation).

Initially, each of the available perturbative heuristics is assigned a score, typically the same value. The second component — the memory length adjustment procedure — dictates the extent of the influence that previous heuristic performances have on the heuristic selection strategy (at some decision point). According to Burke *et al.* [16], typical strategies include the *max* strategy, where selection is based on the best performing heuristic, and the *Roulette-wheel* strategy, where probabilities (corresponding to the relative performances of heuristics) are assigned to each heuristic and selection is performed stochastically (based on these probabilities).

Move acceptance — the other important component of selective-perturbative hyperheuristics — may also be either deterministic or non-deterministic. In a deterministic setting, acceptance is always the same, regardless of the decision point during the search (but pertaining to specific current and new candidate solutions). In a non-deterministic setting, however, different decisions may be generated for the same input at a specific decision point. Additional parameters may be required when implementing a non-deterministic move acceptance strategy.

A selective-perturbative hyperheuristic comprising evolutionary sub-algorithms and employing a multi-point online search strategy as well as a deterministic move acceptance method is the focus of this dissertation. Although the traditional aim of hyperheuristics is raising the level of general applicability of the optimisation algorithm, Vrugt and Robinson [171] additionally reported significant improvements (approaching a factor of ten) when applying this very type of hyperheuristic — these promising claims warrant its consideration and subsequent further investigation. This type of hyperheuristic, called AMALGAM, is discussed in greater detail in the following section.

4.4 The AMALGAM method

Vrugt and Robinson [171] proposed AMALGAM with *evolutionary optimisation* forming the foundation of its operation. As stated by Vrugt and Robinson, the use of evolutionary algorithms is substantiated by their superior performance in *search and optimisation problems* when multiple conflicting objectives are considered. It is further claimed that these performance benefits are attributable to the following characteristics of evolutionary algorithms: Their ability to *traverse* intractably large search spaces, their perpetuation of a set of diverse solutions, and their utilisation of similarities among solutions by means of recombination.

When solving a multi-objective optimisation problem (approximately) by means of a population-based method, a *single* algorithm is typically implemented so as to evolve the population during

an iterative search process. Based on the result of the NFL theorem (see §1.1), however, this approach may not be the most efficient — it is impossible to develop a single algorithm that outperforms all other algorithms for a diverse set of optimisation problems [178]. Consequently, the design of new algorithms is aimed at mitigating this predicament explicitly. AMALGAM is one such attempt at developing an algorithm which not only raises the level of general applicability, but also delivers adequate performance (at least as good as the best lower level heuristic/sub-algorithm incorporated within it). For problems of a more complex nature (*i.e.* of high dimensions), this hyperheuristic reportedly achieves orders of magnitude of improvement in solution quality over the implementations of its individual sub-algorithms [171].

Vrugt and Robinson's research hypothesis (for their proposed hyperheuristic) was that a dynamic and adaptive search procedure, incorporating an offspring-generation procedure that is based on the shape and "local peculiarities" of the search space (or *fitness landscape*), will result in a notably more efficient evolutionary search procedure. The main reasoning behind this conjecture is that for different optimisation problems, the corresponding search spaces may exhibit notably different characteristics (with respect to shape and problem peculiarities). The aforementioned procedure may therefore indeed be a fruitful one. AMALGAM consists of two main components, namely *simultaneous multi-method search* and *self-adaptive offspring creation*, which are said to help solve multi-objective optimisation problems in a reliable and computationally efficient manner. Ultimately, the shared advantages of the individual sub-algorithms are exploited, and their respective weaknesses are compensated for.

Based on AMALGAM's self-adaptive offspring creation, significant reproductive success of the sub-algorithms is rewarded when *allotting* computation resources to each of the k sub-algorithms — *i.e.* deciding on the number of solutions each sub-algorithm may generate during subsequent generations. Suppose sub-algorithm $i \in \{1, \ldots, k\}$ generates S_{t+1}^i solutions as its contribution to the next parent population $\mathcal{P}_{t+1}$ during generation t. Furthermore, let the number of solutions generated by sub-algorithm i during generation $t+1$ be given by

$$N_{t+1}^i = \frac{\left(\frac{S_{t+1}^i}{N_t^i}\right) M}{\sum_{i=1}^{k} \left(\frac{S_{t+1}^i}{N_t^i}\right)}, \tag{4.11}$$

where M is the total number of solutions in the population. It follows from (4.11) that S_{t+1}^i/N_t^i represents the ratio of the number of *successful* offspring solutions to the total number of offspring solutions initially created. This ratio is, therefore, a measure of the reproductive success of sub-algorithm i, and ensures that the "best" sub-algorithms are rewarded accordingly. Furthermore, this measure is scaled by the combined success of the entire set of sub-algorithms. According to Vrugt and Robinson, an important step is to impose a minimum constraint on the value of N_t^i so as to ensure that sub-algorithm i is not entirely *deactivated* during the hyperheuristic's execution. It should be noted that Vrugt and Robinson enforced a minimum constraint of $N_t^i \geq 5$, although no motivation was provided for this specific choice.

AMALGAM uses both the notions of Pareto rank and *crowding distance* to rank and sort solutions in the population. The Pareto rank of each solution in the population is determined by the FNSA, presented in pseudocode form as Algorithm 2.1. Crowding distance, another important part of the NSGA-II and NSDE, facilitates the ranking and sorting process, and was presented in pseudocode form as Algorithm 2.2. A pseudocode description of AMALGAM is presented in Algorithm 4.4 so as to facilitate the following discussion.

AMALGAM initialises with the generation of an initial parent population $\mathcal{P}_0$ consisting of M random candidate solutions. $\mathcal{P}_0$ is then ranked and sorted according to the FNSA. The generation of an offspring population $\mathcal{Q}_0$ takes place thereafter with $N_0^i = M/k$, where each sub-algorithm

Algorithm 4.4: AMALGAM [145, 171]

Input : An MOP, a population size M, a maximum number of generations t_{max}, a set of k sub-algorithms

Output: An approximate Pareto set, $\tilde{\mathcal{P}}_S$

1 Generate an initial parent population $\mathcal{P}_0$ of M random solutions;
2 Rank and sort the population $\mathcal{P}_0$ according to the FNSA in Algorithm 2.1;
3 Set $N_0^i = M/k$;
4 Create an offspring population $\mathcal{Q}_0$ of size M using each sub-algorithm $i \in \{1, \ldots, k\}$;
5 $t \leftarrow 0$;
6 **while** $t < t_{max}$ **do**
7 $\quad \mathcal{R}_t \leftarrow \mathcal{P}_t \cup \mathcal{Q}_t$;
8 $\quad$ Partition $\mathcal{R}_t$ into non-dominated fronts $\mathcal{F}_1, \mathcal{F}_2, \ldots$ using the FNSA in Algorithm 2.1;
9 $\quad \mathcal{P}_{t+1} \leftarrow \emptyset$;
10 $\quad j = 0$;
11 $\quad$ **while** $|\mathcal{P}_{t+1}| < M$ **do**
12 $\quad\quad$ **if** $|\mathcal{P}_{t+1}| + |\mathcal{F}_j| \leq M$ **then**
13 $\quad\quad\quad \mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \mathcal{F}_j$;
14 $\quad\quad\quad j \leftarrow j + 1$;
15 $\quad\quad$ **else**
16 $\quad\quad\quad$ Calculate the crowding distance for each solution in $\mathcal{F}_j$ using Algorithm 2.2;
17 $\quad\quad\quad$ Sort $\mathcal{F}_j$ in decreasing order of crowding distance;
18 $\quad\quad\quad \mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \{$the first $M - |\mathcal{P}_{t+1}|$ solutions in $\mathcal{F}_j\}$;
19 $\quad$ Calculate the crowding distance for each solution in $\mathcal{P}_{t+1}$ using Algorithm 2.2;
20 $\quad$ Calculate the number of successful solutions S_{t+1}^i for each $i \in \{1, \ldots, k\}$;
21 $\quad$ Calculate N_{t+1}^i according to (4.11), for each $i \in \{1, \ldots, k\}$;
22 $\quad$ Generate N_{t+1}^i new offspring solutions for each $i \in \{1, \ldots, k\}$ and create the population $\mathcal{Q}_{t+1}$ of size M;
23 $\quad t \leftarrow t + 1$;
24 $\tilde{\mathcal{P}}_S \leftarrow \mathcal{P}_{t_{max}}$

has access to the entire parent population. The generation counter t is then set to zero. Following AMALGAM's initialisation procedure, the main loop, spanning steps 6–23, is iterated until a stopping criterion is met (the execution of a maximum number of iterations t_{max}). This main procedure commences with the creation of a combined population $\mathcal{R}_t \leftarrow \mathcal{P}_t \cup \mathcal{Q}_t$. The FNSA is then used to rank and sort population $\mathcal{R}_t$ into non-dominated fronts $\mathcal{F}_1, \ldots, \mathcal{F}_n$. Following this, the crowding distance for each solution is calculated. The next population $\mathcal{P}_{t+1}$ is created by inserting all the solutions from successive fronts, and this continues until the insertion of an entire front would result in the population size, M, being exceeded. Whenever this happens, the relevant front is sorted in decreasing order of crowding distance, and solutions are subsequently inserted one-by-one in this order, until the population comprises exactly M solutions. The number of successful solutions generated by each sub-algorithm S_{t+1}^i is then calculated. Following this, the number of solutions generated by each sub-algorithm during the next generation, N_{t+1}^i, is calculated according to (4.11). The next offspring solution population $\mathcal{Q}_{t+1}$ (again of size M) is created by each sub-algorithm. Finally, the generation counter is incremented.

As mentioned, the original version of AMALGAM contained the following evolutionary sub-algorithms: The NSGA-II, PSO, an AMS algorithm, and DE. This choice of algorithms was based on the results of numerical experiments spanning ten multi-objective optimisation problems

differing with respect to convexity and spatial characteristics [171]. To the best of the author's knowledge, an in-depth study of AMALGAM's application in respect of training ANNs has not yet been attempted in the literature, although the individual applications of a GA [106, 110, 174], PSO [143, 54, 183], and DE [69, 98, 182] towards training ANNs can be found in the literature. There is a complete lack of literature related to the successful application of AMS in respect of ANN training — consequently it is excluded from further discussion.

4.5 Chapter summary

This chapter contained a review of the relevant literature related to the respective fields of metaheuristics and hyperheuristics. The chapter opened in §4.1 with a brief discussion on the general notion of a metaheuristic approach towards solving optimisation problems. This was followed in §4.2 by a discussion on a powerful sub-field of metaheuristics known as evolutionary optimisation. This discussion included detailed descriptions of three key MOEAs, namely: NSGA-II, NSDE, and OMOPSO. These algorithms form a fundamental part of the BOHTA solution methodology proposed later in this dissertation. An overview of the relatively new and promising field of hyperheuristics followed in §4.3, with an emphasis on hyperheuristics of a selective nature. Finally, the algorithm that serves as the dissertation's muse — *i.e.* AMALGAM — was discussed in detail in §4.4, which included a description of its fundamental mechanisms (*i.e.* simultaneous multi-method search and self-adaptive offspring creation).

Part II

Model Design

CHAPTER 5

Model Formulation

Contents

The purpose of this chapter is to formulate an appropriate mathematical model of the problem at hand — *i.e.* training an FNN in respect of its network weights, network structure, and activation functions in order to facilitate its effective solution of various classification problems (represented by data sets). The chapter opens with an overview of the proposed model, and this is followed by a discussion on the decision variables, constraints, and objective functions. There are three main sets of decision variables: Network weights, network structure, and activation functions. The mathematical model contains two objective functions — a main objective function and a so-called helper objective function. This model is solved by the BOHTA (as well as its individual constituent sub-algorithms) later in this dissertation. The chapter closes with a concise summary of its contents.

5.1 Model overview

The aim of the discussion in this section is to elucidate the salient parts of the mathematical model to be solved by the BOHTA proposed in this dissertation. This discussion is facilitated by the graphical illustration in Figure 5.1, which represents by and large a general FNN, underpinned by a notably low level of abstraction. This level of abstraction — a level that simply renders a typical gradient-based approach non-viable — is appropriate due to the nature of the proposed BOHTA. The aim is to exploit the immanent advantages associated with modelling at this very level. Depicted in Figure 5.1 is an FNN with n input neurons, o output neurons, h hidden layers, and m hidden neurons per hidden layer. The newly introduced bracketed superscript $f \in \{1, \ldots, h\}$ (depicted in the figure) denotes an arbitrary hidden layer of the network. The

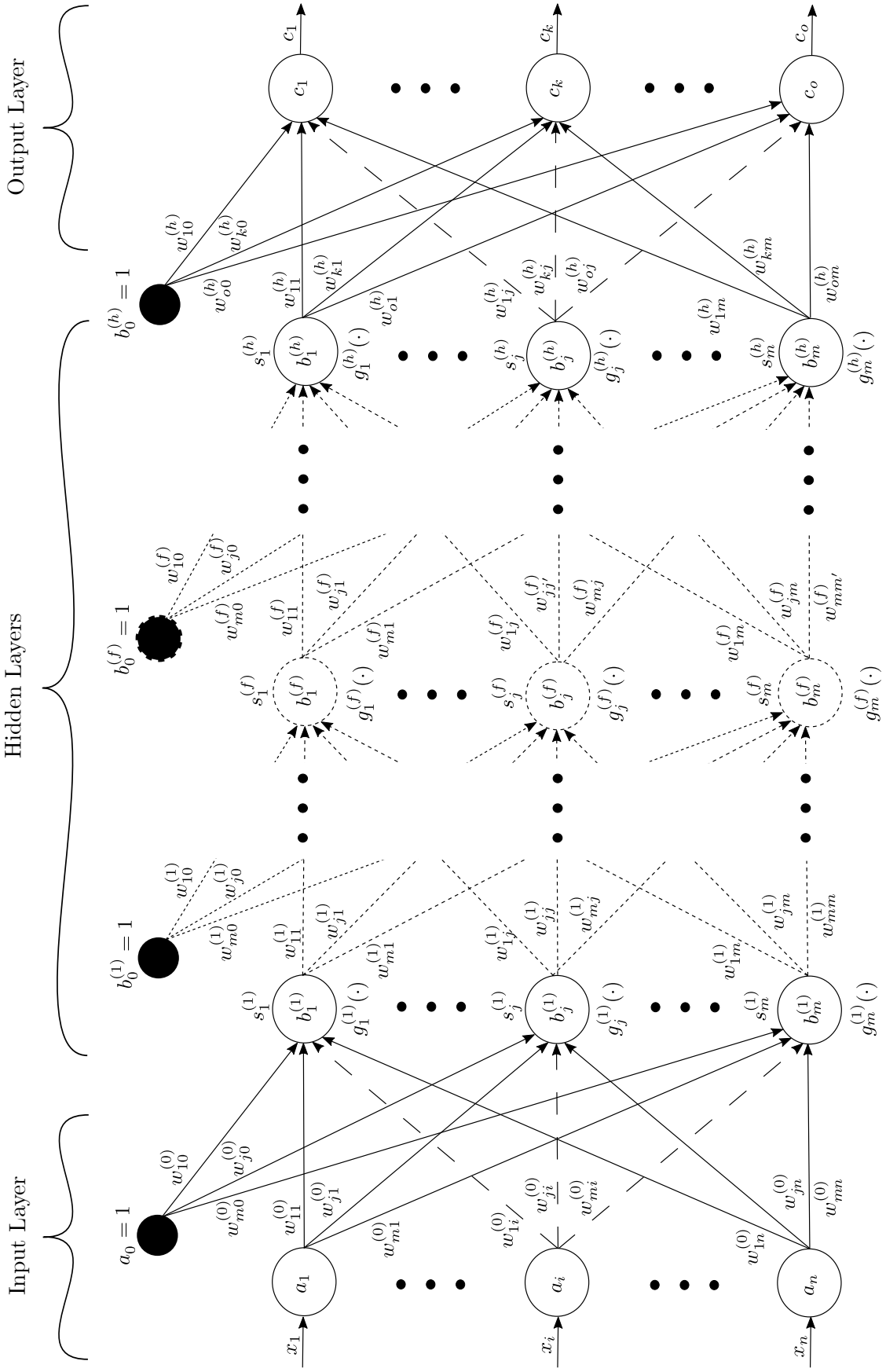


FIGURE 5.1: *Complete FNN used as basis for the mathematical model formulation.*

parameters n and o are problem-specific and represent the number of independent and dependent variables, respectively. The number of hidden layers and the number of hidden neurons per layer (denoted by h and m, respectively) are pre-defined and, essentially, determine the maximum size of the network (a matter to be addressed shortly).

5.2 Decision variables and constraints

Figure 5.1 contains most of the relevant decision variables, such as the network weights and network size (*i.e.* variables associated with the hidden layers and the hidden neurons), explicitly. The remainder of the variables, such as those related to the activation function have, however, been omitted so as to minimise clutter. Regardless, the discourse in this section is of an exhaustive and comprehensive nature, addressing all of the relevant decision variables and constraints. As alluded to above, superscripts and subscripts have newly been introduced for hidden layer activations and hidden neuron activation functions, respectively. These introductions, necessitated by the lower level of abstraction in this chapter than in the standard approach discussed in Chapter 2, are motivated and described in this section.

5.2.1 Network weights

Let $\boldsymbol{w} = \{\boldsymbol{W}^{(0)}, \ldots, \boldsymbol{W}^{(f)}, \ldots, \boldsymbol{W}^{(h)}\}$ represent an ordered list of *weight-matrices*, *i.e.* a list containing all $h+1$ weighted-connection layers[1]. Consider first the weight-matrix $\boldsymbol{W}^{(0)}$ representing the weighted-connection layer between the input layer and first hidden layer of the FNN, given by

$$\begin{bmatrix} w_{10}^{(0)} & w_{11}^{(0)} & \cdots & w_{1i}^{(0)} & \cdots & w_{1n}^{(0)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j0}^{(0)} & w_{j1}^{(0)} & \cdots & w_{ji}^{(0)} & \cdots & w_{jn}^{(0)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{m0}^{(0)} & w_{m1}^{(0)} & \cdots & w_{mi}^{(0)} & \cdots & w_{mn}^{(0)} \end{bmatrix}. \tag{5.1}$$

In this matrix, column $\phi \in \{0, \ldots, n\}$ represents the outgoing weights from neuron $i \in \{0, \ldots, n\}$ in the input layer, whereas row $\varphi \in \{1, \ldots, m\}$ represents the incoming weights to neuron $j \in \{1, \ldots, m\}$ in the first hidden layer. Figure 5.2 provides a graphical illustration so as to further elucidate the convention used in the weight-matrices.

Consider next the weight-matrix $\boldsymbol{W}^{(f)}$ representing a hidden weighted-connection layer contained solely within the hidden layers of the FNN, more specifically between layer $f \in \{1, \ldots, h-1\}$ and layer $f+1$, given by

$$\begin{bmatrix} w_{10}^{(f)} & w_{11}^{(f)} & \cdots & w_{1j}^{(f)} & \cdots & w_{1m}^{(f)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j0}^{(f)} & w_{j1}^{(f)} & \cdots & w_{jj'}^{(f)} & \cdots & w_{jm}^{(f)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{m0}^{(f)} & w_{m1}^{(f)} & \cdots & w_{mj}^{(f)} & \cdots & w_{mm'}^{(f)} \end{bmatrix}. \tag{5.2}$$

[1] A "weighted-connection layer" refers to the weights between two successive layers, *i.e.* between the input and first hidden layer, between two arbitrary (but successive) hidden layers, or between the last hidden layer and output layer.

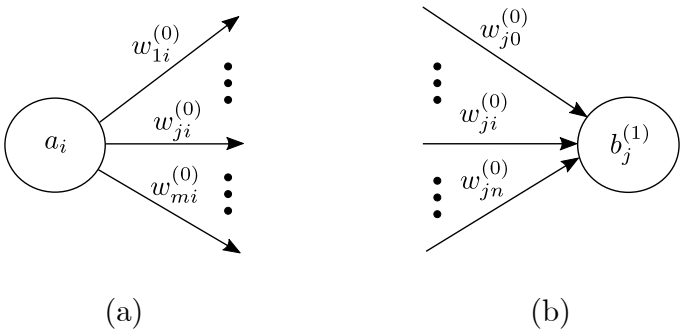


FIGURE 5.2: *A graphical illustration facilitating the interpretation of the weight-matrix* $\boldsymbol{W}^{(0)}$ *in* (5.1)*, where (a) represents the outgoing weights contained within column* $\phi \in \{0, \ldots, n\}$ *and (b) represents the incoming weights contained within row* $\varphi \in \{1, \ldots, m\}$*. The same interpretation may be applied to the other weight-matrices.*

The indices j' and m' are used to nuance a distinction between two successive layers whenever the subscripts contain two of the same indices. For example, $w_{jj'}^{(f)}$ refers to the weighted connection from neuron j' in layer f to neuron j in layer $f+1$. Column $\phi \in \{0, \ldots, m\}$ thus represents the outgoing weights from neuron $j' \in \{0, \ldots, m'\}$ in hidden layer f, whereas row $\varphi \in \{1, \ldots, m\}$ represents the incoming weights to neuron $j \in \{1, \ldots, m\}$ in layer $f+1$.

Finally, consider weight-matrix $\boldsymbol{W}^{(h)}$, which represents the weighted-connection layer between the last hidden layer and the output layer of the FNN, given by

$$\begin{bmatrix} w_{10}^{(h)} & w_{11}^{(h)} & \cdots & w_{1j}^{(h)} & \cdots & w_{1m}^{(h)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{k0}^{(h)} & w_{k1}^{(h)} & \cdots & w_{kj}^{(h)} & \cdots & w_{km}^{(h)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{o0}^{(h)} & w_{o1}^{(h)} & \cdots & w_{oj}^{(h)} & \cdots & w_{om}^{(h)} \end{bmatrix}. \tag{5.3}$$

Correspondingly, column $\phi \in \{0, \ldots, m\}$ represents the outgoing weights from neuron $j \in \{0, \ldots, o\}$ in the last hidden layer, whereas row $\varphi \in \{1, \ldots, n\}$ represents the incoming weights to neuron $k \in \{1, \ldots, o\}$ in the output layer.

The weights contained in the respective weight-matrices of $\boldsymbol{w}$ are subject to a single constraint, requiring that all weights are real-valued. Starting with weight matrix $\boldsymbol{W}^{(0)}$ in (5.1), the constituent weights are therefore subject to the constraint set

$$w_{ji}^{(0)} \in \mathbb{R}, \qquad j \in \{1, \ldots, m\},\ i \in \{0, \ldots, n\}.$$

The weights contained in the weight matrix $\boldsymbol{W}^{(f)}$ in (5.2) are similarly subject to the constraint set

$$w_{jj'}^{(f)} \in \mathbb{R}, \qquad j \in \{1, \ldots, m\},\ j' \in \{0, \ldots, m'\},\ f \in \{1, \ldots, h-1\}.$$

Lastly, the weights in the weight matrix $\boldsymbol{W}^{(h)}$ in (5.3) are subject to the constraint set

$$w_{kj}^{(h)} \in \mathbb{R}, \qquad k \in \{1, \ldots, o\},\ j \in \{0, \ldots, m\}.$$

When following a conventional gradient-based network training approach, these weight-matrices represent the only decision variables considered by the training algorithm during an optimisation run (in the case of hyper-parameter optimisation, the pre-optimised weights remain fixed).

Treating only the network weights as decision variables is a restriction attributable to the nature of this gradient-based approach and the accompanying limitation it presents — *i.e.* that of differentiability. In the case of a metaheuristic or hyperheuristic network training approach, however, these weight-matrices are but one facet of the optimisation paradigm.

5.2.2 Network structure

The next set of decision variables pertains to the structure of the network, *i.e.* the number of hidden layers and the number of hidden neurons in each hidden layer. As illustrated graphically in Figure 5.1, *structural switching-variables*, denoted by

$$s_j^{(f)} = \begin{cases} 1, & \text{if neuron } j \in \{1,\dots,m\} \text{ in layer } f \in \{1,\dots,h\} \text{ is } \textit{on} \\ 0, & \text{otherwise,} \end{cases} \tag{5.4}$$

are introduced. The term "on" here indicates that a neuron is active and that its net input can be evaluated by its activation function, with the resulting output being transmitted onwards for further processing. Whenever, say, neuron j in hidden layer f is inactive, *i.e.* if $s_j^{(f)} = 0$, then its resulting activation is consequently nullified (*i.e.* $b_j^{(f)} = 0$) and, as a result, the corresponding weights $w_{1j}^{(f)}$, $w_{jj'}^{(f)}$, and $w_{mj}^{(f)}$ are also nullified. In essence, the binary variable $s_j^{(f)}$ controls the size of hidden layer f (*i.e.* the number of active neurons in the layer) implicitly. The network's switching-variables are stored in a matrix $\boldsymbol{S} = [\boldsymbol{s}^{(1)} \cdots \boldsymbol{s}^{(f)} \cdots \boldsymbol{s}^{(h)}]$, where the column vector $\boldsymbol{s}^{(f)}$ represents the switching-variables in hidden layer f. That is,

$$\left[\boldsymbol{s}^{(1)} \cdots \boldsymbol{s}^{(f)} \cdots \boldsymbol{s}^{(h)}\right] = \begin{bmatrix} s_1^{(1)} & \cdots & s_1^{(f)} & \cdots & s_1^{(h)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ s_j^{(1)} & \cdots & s_j^{(f)} & \cdots & s_j^{(h)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ s_m^{(1)} & \cdots & s_m^{(f)} & \cdots & s_m^{(h)} \end{bmatrix}.$$

Whenever a *redundancy criterion*, given by

$$\sum_{j=1}^{m} s_j^{(f)} = 0, \qquad f \in \{1,\dots,h\}, \tag{5.5}$$

is met, no active neurons are present within hidden layer f, rendering the entire layer redundant. In this case, the hidden layer's neurons together with the corresponding weights $\boldsymbol{W}^{(f)}$, are consequently suppressed (*i.e.* not used to calculate subsequent activations). The net input to the neurons within this layer is thus only processed by the next non-redundant layer. The entries in $\boldsymbol{S}$ therefore control both the number of hidden neurons in each layer and the number of hidden layers within the network.

Criterion (5.5) evidently includes all hidden layers, *i.e.* $f \in \{1,\dots,h\}$. This presents the potential situation where *all* of the hidden layers may be rendered inactive if (5.5) holds true for each $f \in \{1,\dots,h\}$. An obvious workaround would be to introduce the constraint

$$\sum_{f=1}^{h} \sum_{j=1}^{m} s_j^{(f)} \geq 1.$$

This constraint would ensure that the network contains at least one active neuron and, consequently, at least one active layer. Such a constraint incorporation would, however, place an additional computational burden on the optimisation algorithm — an issue to be avoided if possible. An alternative approach is therefore proposed — one that avoids the inclusion of any additional constraints. According to the proposed approach, the network is allowed to have no active neurons *per se*, although whenever this is the case, the network output is simply given by

$$\boldsymbol{c} = \boldsymbol{W}^{(h)} \left(\boldsymbol{W}^{(0)} \tilde{\boldsymbol{a}} \right). \tag{5.6}$$

The activation functions are thus excluded from the calculation of the network output. This approach is preferred due to the computational simplicity it affords, whilst still ensuring that the network does not lose its computational capability — a consequence one ought to expect from a network containing no active neurons. Training a network with no hidden layers is typically treated as a linear regression problem, in which only one weighted-connection layer is utilised. According to the proposed approach, however, two weighted-connection layers (as expressed in (5.6)) are utilised, so as to afford the network a greater degree of computational capability. ANNs are typically applied to supervised learning problems that are *linearly inseparable*[2]. Appropriately, the non-linearities "enforced" by the proposed approach in (5.6) ensure that the BOHTA does not oversimplify the network, impeding its capabilities unnecessarily.

5.2.3 Activation functions

The last set of decision variables is related to the activation functions employed by each neuron. As mentioned earlier, details pertaining to the activation functions have been omitted from Figure 5.1 so as to minimise clutter. *Activation functional variables*, given by

$$g_j^{(f)}(\eta_j^{(f)}) = \begin{cases} \alpha_j^{(f)} \eta_j^{(f)}, & \eta_j^{(f)} < 0, \\ \beta_j^{(f)} \eta_j^{(f)}, & \eta_j^{(f)} \geq 0, \end{cases} \tag{5.7}$$

are nevertheless introduced and represent piecewise linear functions employed by hidden neuron $j \in \{1, \ldots, m\}$ in hidden layer $f \in \{1, \ldots, h\}$, for a given net input $\eta_j^{(f)}$. The adjustable constituent slope parameters are represented by $\alpha_j^{(f)}$ and $\beta_j^{(f)}$, and are subject to the following constraints: For a negative input, *i.e.* if $\eta_j^{(f)} < 0$, the relevant slope variables are subject to the constraint

$$\alpha_j^{(f)} \in \mathbb{R}, \qquad j \in \{1, \ldots, m\},\ f \in \{1, \ldots, h\},$$

whereas for a non-negative input, *i.e.* if $\eta_j^{(f)} \geq 0$, the relevant slope variables are subject to

$$\beta_j^{(f)} \in \mathbb{R}, \qquad j \in \{1, \ldots, m\},\ f \in \{1, \ldots, h\}.$$

The PReLU in (3.6) serves as the muse for this activation function due to its computational simplicity and the favourable performance reported in the literature when employing a gradient-based network training approach [59, 75]. A modification to the PReLU is, however, made, *i.e.* a slope parameter for non-negative input is incorporated and, furthermore, the parameters for both non-negative and negative inputs are now treated as decision variables. The BOHTA is thus tasked with finding neuron-specific piecewise linear functions that best emulate and model the neuron firing process. Figure 5.3 contains a graphical illustration of the function $g_j^{(f)}(\eta_j^{(f)})$, with the slope parameter settings $\alpha_j^{(f)} = 0.5$ and $\beta_j^{(f)} = 1.25$.

[2] In the case of linearly inseparable classification problems, a linear decision boundary cannot separate the different classes (or categories) at hand.

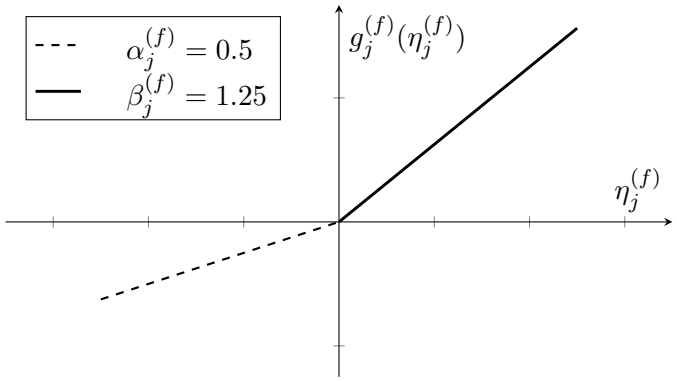


FIGURE 5.3: *The proposed neuron-specific piecewise linear activation function employed by hidden neuron $j \in \{1, \ldots, m\}$ in hidden layer $f \in \{1, \ldots, h\}$ with the slope variable values in (5.7) indicated.*

The hidden neuron activation functions are stored in a matrix $\boldsymbol{G} = [\boldsymbol{g}^{(1)} \cdots \boldsymbol{g}^{(f)} \cdots \boldsymbol{g}^{(h)}]$, where $\boldsymbol{g}^{(f)}$ represents the column vector of activation functions employed in hidden layer $f \in \{1, \ldots, h\}$. That is,

$$\left[\boldsymbol{g}^{(1)} \cdots \boldsymbol{g}^{(f)} \cdots \boldsymbol{g}^{(h)}\right] = \begin{bmatrix} g_1^{(1)}(\cdot) & \cdots & g_1^{(f)}(\cdot) & \cdots & g_1^{(h)}(\cdot) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_j^{(1)}(\cdot) & \cdots & g_j^{(f)}(\cdot) & \cdots & g_j^{(h)}(\cdot) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ g_m^{(1)}(\cdot) & \cdots & g_m^{(f)}(\cdot) & \cdots & g_m^{(h)}(\cdot) \end{bmatrix}.$$

The activation functions employed in Figure 5.1 by the input neurons are identity functions and, as a result, are not explicitly contained in $\boldsymbol{G}$. As stated in §3.2, the use of identity functions in the input layer is the standard approach and is deemed sufficient — the inputs should only be transmitted onwards as-is.

The applicable class of supervised learning problems (classification or regression) determines the choice of which activation function type to employ by the output neurons. For classification problems, the number of output neurons ought to reflect the number of classes (dependent variables) in question. Due to the prolific nature of the softmax function (as stated in §3.2), its subsequent inclusion in the model is well-warranted, although a slight graphical modification is necessitated in order to accommodate its inclusion. Figure 5.4 contains an illustration of the modification required at the output layer.

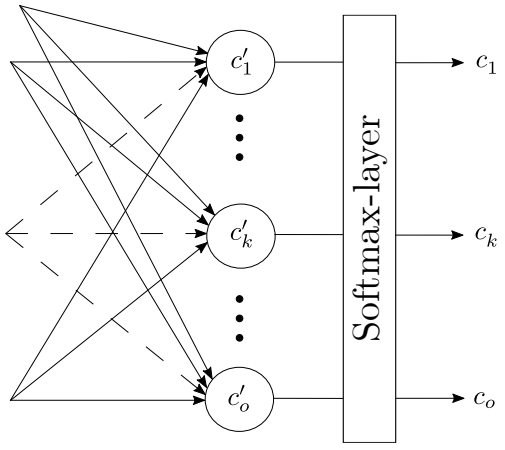


FIGURE 5.4: *The softmax-layer employed in the output layer.*

An intermediate output, denoted by $\boldsymbol{c}' = [c'_1 \cdots c'_k \cdots c'_o]^T$, is calculated by applying an identity function to the net input associated with each output neuron. The intermediate output is subsequently passed through a so-called *softmax-layer*. Within this layer, the softmax activation function (3.5) evaluates $\boldsymbol{c}'$, outputting a vector of probabilities (hence the term "layer"). This vector represents the categorical probabilities of each class (represented by the output neurons) corresponding to the presented input. The output is denoted by $\boldsymbol{c} = [c_1 \cdots c_k \cdots c_o]^T$ with the requirement that $\sum_k c_k = 1$.

5.3 Objective functions

Two objective functions are adopted in the bi-objective optimisation network training approach proposed in this dissertation. Accordingly, there are two main parts in this section. The first part focusses on the *main* objective function, which represents the network performance measure and provides an indication as to how well the network is learning the underlying functional representation of the data. The second part focusses on the *helper* objective function employed to guide the search so as to address the problem of overfitting, *i.e.* a regularisation technique. It may therefore be regarded as a *regularising* objective function.

The reasoning behind this bi-objective optimisation approach, as opposed to a singe-objective optimisation approach, stems from the muse at the centre of this dissertation — *i.e.* AMALGAM. Vrugt and Robinson's [171] hyperheuristic comprises various MOEAs; therefore, its emulation and subsequent application necessitate the formulation of optimisation problems that contain $e \geq 2$ objective functions. In this dissertation, e assumes a value of two — hence the bi-objective classification. An argument could be made for the consideration of multiple objective functions, *i.e.* $(e \geq 3)$, but the optimisation abstraction level adopted in this dissertation far exceeds most network training approaches in literature, therefore warranting an arguably conservative (or moderate) approach with respect to other facets. Given the scope underpinning the research conducted in this dissertation, the inclusion of additional objectives is therefore relegated to future work.

5.3.1 Main objective function

Suppose the network is presented with a training data set $(\mathcal{X}, \mathcal{Y})$, comprising Q input-output vectors of the form $\{(\boldsymbol{x}^1, \boldsymbol{y}^1), \ldots, (\boldsymbol{x}^q, \boldsymbol{y}^q), \ldots, (\boldsymbol{x}^Q, \boldsymbol{y}^Q)\}$. For an arbitrary training example q, the input activations, denoted[3] by $\boldsymbol{a} = [a_1 \cdots a_i \cdots a_n]^T$, are equal to the training example's input features, denoted by $\boldsymbol{x} = [x_1 \cdots x_i \cdots x_n]$. This equality is attributed to the fact that identity functions are employed in the input layer, resulting in the output $a_i = x_i$ for $i \in \{1, \ldots, n\}$, or alternatively $\boldsymbol{a} = [x_1 \cdots x_i \cdots x_n]^T$.

The matrix $\boldsymbol{B} = [\boldsymbol{b}^{(1)} \cdots \boldsymbol{b}^{(f)} \cdots \boldsymbol{b}^{(h)}]$ contains all h hidden layer activations, where

$$\left[\boldsymbol{b}^{(1)} \cdots \boldsymbol{b}^{(f)} \cdots \boldsymbol{b}^{(h)}\right] = \begin{bmatrix} b_1^{(1)} & \cdots & b_1^{(f)} & \cdots & b_1^{(h)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ b_j^{(1)} & \cdots & b_j^{(f)} & \cdots & b_j^{(h)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ b_m^{(1)} & \cdots & b_m^{(f)} & \cdots & b_m^{(h)} \end{bmatrix}.$$

[3] The superscript denoting the training example used is omitted here and in the subsequent discussion so as to minimise clutter.

Note that the biases have been excluded from the above expression (as well as from the vector $\boldsymbol{a}$ mentioned earlier). The notation $\tilde{\boldsymbol{b}}^{(f)}$ is introduced so as to indicate whenever the vector includes the relevant bias, *e.g.* $\tilde{\boldsymbol{b}}^{(f)} = [b_0^{(f)} \cdots b_j^{(f)} \cdots b_m^{(f)}]^T$, where $b_0^{(f)} = 1$ for $f \in \{1, \ldots, h\}$. The same convention also applies to $\tilde{\boldsymbol{a}}$.

The calculation of each of the constituent activation vectors within $\boldsymbol{B}$ are presented next. Starting with the first hidden layer (where $f = 1$), the relevant calculations are

$$\boldsymbol{b}^{(1)} = \left(\boldsymbol{s}^{(1)} \odot \boldsymbol{g}^{(1)}\left(\boldsymbol{W}^{(0)}\tilde{\boldsymbol{a}}\right)\right)\left\lceil \frac{\sum_{j=1}^{m} s_j^{(1)}}{m} \right\rceil + \left(\boldsymbol{W}^{(0)}\tilde{\boldsymbol{a}}\right)\left\lfloor 1 - \frac{\sum_{j=1}^{m} s_j^{(1)}}{m} \right\rfloor, \tag{5.8}$$

where $\odot$ denotes the *Hadamard product* (or element-wise multiplication) of the switching variables $\boldsymbol{s}^{(1)}$ by the net input activations $\boldsymbol{g}^{(1)}\left(\boldsymbol{W}^{(0)}\tilde{\boldsymbol{a}}\right)$. The net input $\boldsymbol{\eta}^{(1)}$ is thus given by $\boldsymbol{W}^{(0)}\tilde{\boldsymbol{a}}$.

Whenever the first hidden layer is deemed non-redundant according to criterion (5.5), at least one neuron within the layer is active, *i.e.* $\sum_{j=1}^{m} s_j^{(1)} \geq 1$. This results in the values

$$\left\lceil \frac{\sum_{j=1}^{m} s_j^{(1)}}{m} \right\rceil = 1 \text{ and } \left\lfloor 1 - \frac{\sum_{j=1}^{m} s_j^{(1)}}{m} \right\rfloor = 0$$

in (5.8). Therefore, (5.8) evaluates to

$$\boldsymbol{b}^{(1)} = \left(\boldsymbol{s}^{(1)} \odot \boldsymbol{g}^{(1)}\left(\boldsymbol{W}^{(0)}\tilde{\boldsymbol{a}}\right)\right)$$

in this case. If this hidden layer's redundancy criterion is, however, met and the layer is deemed redundant, no active neurons are present within the layer and so $\sum_{j=1}^{m} s_j^{(1)} = 0$ and $\boldsymbol{s}^{(1)} = \boldsymbol{0}$. This results in the values

$$\left\lceil \frac{\sum_{j=1}^{m} s_j^{(1)}}{m} \right\rceil = 0 \quad \text{and} \quad \left\lfloor 1 - \frac{\sum_{j=1}^{m} s_j^{(1)}}{m} \right\rfloor = 1$$

in (5.8). Therefore, (5.8) evaluates to

$$\boldsymbol{b}^{(1)} = \boldsymbol{W}^{(0)}\tilde{\boldsymbol{a}}$$

in this case. As a result of this redundancy, both the activation functions $\boldsymbol{g}^{(1)}$ (including the constituent slope parameters) as well as the subsequent weights $\boldsymbol{W}^{(1)}$ are disregarded. The hidden layer is effectively suppressed, and only its net input is processed by the next non-redundant layer.

Before moving on to the subsequent layers' calculations, it should be noted that the term

$$\left\lceil \frac{\sum_{j=1}^{m} s_j^{(1)}}{m} \right\rceil$$

could have been omitted from (5.8), as the switching variables $\boldsymbol{s}^{(1)}$ already provide the same functionality — *i.e.* if the layer is redundant, $\boldsymbol{s}^{(1)}$ equates to zero, nullifying the first term. This term is nevertheless included for the purpose of uniformity in the subsequent analysis.

With respect to the calculation of the activations within the subsequent hidden layers (where $f \geq 2$), it is important to note that it is now possible for one, or perhaps more, preceding hidden layer(s) to be redundant, a situation not possible in the first hidden layer (the input layer cannot be redundant). When evaluating each hidden layer's activations, only the current layer (denoted by f) and the preceding layer (denoted by $f - 1$) are considered, because layer $f - 1$ already contains the necessary calculations pertaining to the preceding layers.

Hence there are four different combinations to consider:

(1) Both layers f and $f-1$ are non-redundant,

(2) layer f is non-redundant and layer $f-1$ is redundant,

(3) layer f is redundant and layer $f-1$ is non-redundant, or

(4) both layers f and $f-1$ are redundant.

Consider first the activations within the second hidden layer (where $f=2$). In this case,

$$\begin{aligned}\boldsymbol{b}^{(2)} = &\left(\boldsymbol{s}^{(2)} \odot \boldsymbol{g}^{(2)}\left(\boldsymbol{W}^{(1)}\tilde{\boldsymbol{b}}^{(1)}\right)\right)\left\lceil\frac{\sum_{j=1}^{m} s_j^{(2)}}{m}\right\rceil\left\lceil\frac{\sum_{j=1}^{m} s_j^{(1)}}{m}\right\rceil \\ &+\left(\boldsymbol{s}^{(2)} \odot \boldsymbol{g}^{(2)}\left(\boldsymbol{b}^{(1)}\right)\right)\left\lceil\frac{\sum_{j=1}^{m} s_j^{(2)}}{m}\right\rceil\left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(1)}}{m}\right\rfloor \\ &+\left(\boldsymbol{W}^{(1)}\tilde{\boldsymbol{b}}^{(1)}\right)\left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(2)}}{m}\right\rfloor\left\lceil\frac{\sum_{j=1}^{m} s_j^{(1)}}{m}\right\rceil \\ &+\left(\boldsymbol{b}^{(1)}\right)\left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(2)}}{m}\right\rfloor\left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(1)}}{m}\right\rfloor. \end{aligned} \tag{5.9}$$

For any arbitrary hidden layer other than the first hidden layer, however,

$$\begin{aligned}\boldsymbol{b}^{(f)} = &\left(\boldsymbol{s}^{(f)} \odot \boldsymbol{g}^{(f)}\left(\boldsymbol{W}^{(f-1)}\tilde{\boldsymbol{b}}^{(f-1)}\right)\right)\left\lceil\frac{\sum_{j=1}^{m} s_j^{(f)}}{m}\right\rceil\left\lceil\frac{\sum_{j=1}^{m} s_j^{(f-1)}}{m}\right\rceil \\ &+\left(\boldsymbol{s}^{(f)} \odot \boldsymbol{g}^{(f)}\left(\boldsymbol{b}^{(f-1)}\right)\right)\left\lceil\frac{\sum_{j=1}^{m} s_j^{(f)}}{m}\right\rceil\left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(f-1)}}{m}\right\rfloor \\ &+\left(\boldsymbol{W}^{(f-1)}\tilde{\boldsymbol{b}}^{(f-1)}\right)\left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(f)}}{m}\right\rfloor\left\lceil\frac{\sum_{j=1}^{m} s_j^{(f-1)}}{m}\right\rceil \\ &+\left(\boldsymbol{b}^{(f-1)}\right)\left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(f)}}{m}\right\rfloor\left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(f-1)}}{m}\right\rfloor, \end{aligned} \tag{5.10}$$

where $f \in \{2,\ldots,h\}$. If the current layer, denoted by $f \in \{2,\ldots,h\}$, is non-redundant, then

$$\left\lceil\frac{\sum_{j=1}^{m} s_j^{(f)}}{m}\right\rceil = 1 \quad \text{and} \quad \left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(f)}}{m}\right\rfloor = 0.$$

If, however, layer f is redundant, then

$$\left\lceil\frac{\sum_{j=1}^{m} s_j^{(f)}}{m}\right\rceil = 0 \quad \text{and} \quad \left\lfloor 1-\frac{\sum_{j=1}^{m} s_j^{(f)}}{m}\right\rfloor = 1.$$

The value of $\boldsymbol{b}^{(f)}$ in each of the four aforementioned cases is as follows: If both layers f and $f-1$ are non-redundant, then (5.10) reduces to

$$\boldsymbol{b}^{(f)} = \boldsymbol{s}^{(f)} \odot \boldsymbol{g}^{(f)}\left(\boldsymbol{W}^{(f-1)}\tilde{\boldsymbol{b}}^{(f-1)}\right),$$

whereas if layer f is non-redundant and layer $f-1$ is redundant, then (5.10) reduces to

$$\boldsymbol{b}^{(f)} = \boldsymbol{s}^{(f)} \odot \boldsymbol{g}^{(f)} \left(\boldsymbol{b}^{(f-1)} \right).$$

Similarly, if layer f is redundant and layer $f-1$ is non-redundant, then (5.10) reduces to

$$\boldsymbol{b}^{(f)} = \boldsymbol{W}^{(f-1)} \tilde{\boldsymbol{b}}^{(f-1)},$$

whereas if both layers f and $f-1$ are redundant, then (5.10) reduces to

$$\boldsymbol{b}^{(f)} = \boldsymbol{b}^{(f-1)}.$$

If the last hidden layer is redundant, only the layer's activation functions $\boldsymbol{g}^{(h)}$ are disregarded, and not the subsequent weights $\boldsymbol{W}^{(h)}$ as they are fundamental to the calculation of the network output — *i.e.* in order to perform the necessary matrix operations, the dimensions of the relevant matrices need to correspond. This stands in contrast to the preceding convention according to which the subsequent weights are disregarded whenever the layer is deemed redundant. The relevant calculations pertaining to the intermediate network output are therefore simply

$$\boldsymbol{c}' = \boldsymbol{W}^{(h)} \tilde{\boldsymbol{b}}^{(h)}. \tag{5.11}$$

The softmax activation function in (3.5) is subsequently applied in order to calculate the network's actual output. This calculation is given by

$$c_k = \frac{e^{c'_k}}{\sum_{r=1}^{o} e^{c'_r}}, \qquad k \in \{1, \ldots, o\}. \tag{5.12}$$

In §3.7.3, it was mentioned that the use of a random subset or mini-batch of training examples (for evaluating the performance of the network) results in favourable convergence rates. Furthermore, the use of mini-batches, rather than the entire training set, lessens the computational burden — a matter that warrants consideration given the level of abstraction at which optimisation transpires. As a result of these key advantages, the premise of mini-batch learning is adopted in the BOHTA network training approach.

Before elucidating the manner according to which the main objective function is evaluated, the notational convention adopted with respect to the different data sets under consideration (*i.e.* the training, validation, and testing sets) are addressed. The training data set, from which the random mini-batches $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$ are sampled, is denoted by $(\mathcal{X}, \mathcal{Y})$. The validation data set, which is central to the BOHTA's stopping criterion, is denoted by $(\mathcal{X}^{\dagger}, \mathcal{Y}^{\dagger})$. Finally, the testing data set, from which the final unbiased evaluation of the trained networks are obtained, is denoted by $(\mathcal{X}^{\ddagger}, \mathcal{Y}^{\ddagger})$. The original data set under consideration is therefore partitioned into the three aforementioned data subsets, according to the most prevalent convention — *i.e.* a 60%:20%:20% split (as discussed in §3.5.5). The partitioning of a data set according to this split convention is illustrated graphically in Figure 5.5.

Suppose a mini-batch of random training examples is evaluated so as to estimate the network's performance, *i.e.* predictive capabilities. This subset, denoted by $(\hat{\mathcal{X}}, \hat{\mathcal{Y}}) \subset (\mathcal{X}, \mathcal{Y})$, is given by $\{(\hat{\boldsymbol{x}}^1, \hat{\boldsymbol{y}}^1), \ldots, (\hat{\boldsymbol{x}}^p, \hat{\boldsymbol{y}}^p), \ldots, (\hat{\boldsymbol{x}}^P, \hat{\boldsymbol{y}}^P)\}$, where $\hat{\boldsymbol{x}}^p = [\hat{x}_1^p \cdots \hat{x}_i^p \cdots \hat{y}_n^p]$ and $\hat{\boldsymbol{y}}^p = [\hat{y}_1^p \cdots \hat{y}_k^p \cdots \hat{y}_o^p]$. Whenever a candidate solution is evaluated, a mini-batch of training examples is randomly sampled from a discrete uniform distribution without replacement and presented to the network so as to determine its predictive capabilities. Sampling without replacement is performed so as to ensure that the network is presented with the entire training set. The intuition underlying this approach is as follows: For a network to generalise (to unseen instances) effectively it must

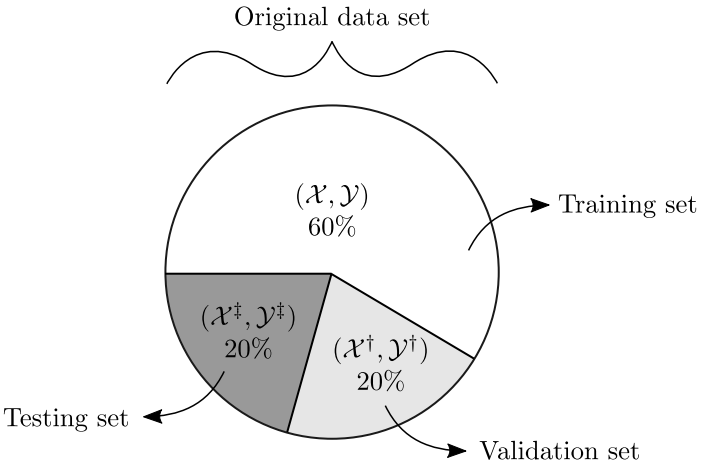


FIGURE 5.5: *The partitioning of a data set according to a 60%:20%:20% split (together with the notational convention adopted).*

first be presented with a representative sample of the true underlying distribution. A network's capacity to generalise is therefore inhibited if it is not subjected to the entire (available) training set. The procedure is depicted graphically in Figure 5.6 and it is shown that this approach ensures that each training example is evaluated once during an epoch. As was stated in §3.5.4, an epoch terminates whenever the entire training set has been evaluated by the training algorithm — accordingly, $\lceil Q/P \rceil$ iterations constitute a single epoch, where Q and P denote the training set size and the mini-batch size, respectively. The notion of an epoch plays an important role in the proposed network training context — a matter elucidated later in this dissertation.

Given the aforementioned derivations, the main part of the BOHTA training problem can now be expressed as

$$\left.\begin{array}{lll} \text{minimise} & h_1\left(\hat{\mathcal{X}}, \hat{\mathcal{Y}}; \boldsymbol{w}, \boldsymbol{S}, \boldsymbol{G}\right) = \dfrac{\frac{1}{P}\sum_{p=1}^{P}\sum_{k=1}^{o}|\hat{y}_k^p - c_k^p|}{F_1} & \\ \text{subject to} & w_{ji}^{(0)} \in \mathbb{R}, & j \in \{1,\ldots,m\},\ i \in \{0,\ldots,n\}, \\ & w_{jj'}^{(f)} \in \mathbb{R}, & j \in \{1,\ldots,m\},\ j' \in \{0,\ldots,m'\}, \\ & & f \in \{1,\ldots,h-1\}, \\ & w_{kj}^{(h)} \in \mathbb{R}, & k \in \{1,\ldots,o\},\ j \in \{0,\ldots,m\}, \\ & \alpha_j^{(f)} \in \mathbb{R}, & j \in \{1,\ldots,m\},\ f \in \{1,\ldots,h\}, \\ & \beta_j^{(f)} \in \mathbb{R}, & j \in \{1,\ldots,m\},\ f \in \{1,\ldots,h\}, \\ & s_j^{(f)} \in \{0,1\}, & j \in \{1,\ldots,m\},\ f \in \{1,\ldots,h\}, \end{array}\right\} \tag{5.13}$$

where c_k^p denotes the output of neuron k, given an input vector $\hat{\boldsymbol{x}}^p$, that ought to correspond to target variable $\hat{y}_k^p$. The objective function comprises two main components: The MAE (3.16) (expressed in the numerator), and the F_1-score (expressed in the denominator). Both of these performance measures (discussed in §3.6) are assessed in respect of the same random mini-batch. The MAE provides a measure of how close or accurate the prediction c_k^p is to the corresponding target value y_k^p for $k \in \{1,\ldots,o\}$. The computationally inexpensive nature of this loss function lends further justification for its inclusion in the model. As stated in §3.6, the cross-entropy performance measure (3.20) and the negative log-likelihood performance measure (3.21) are preferred when adopting gradient-based optimisation techniques due to the favourable shape of

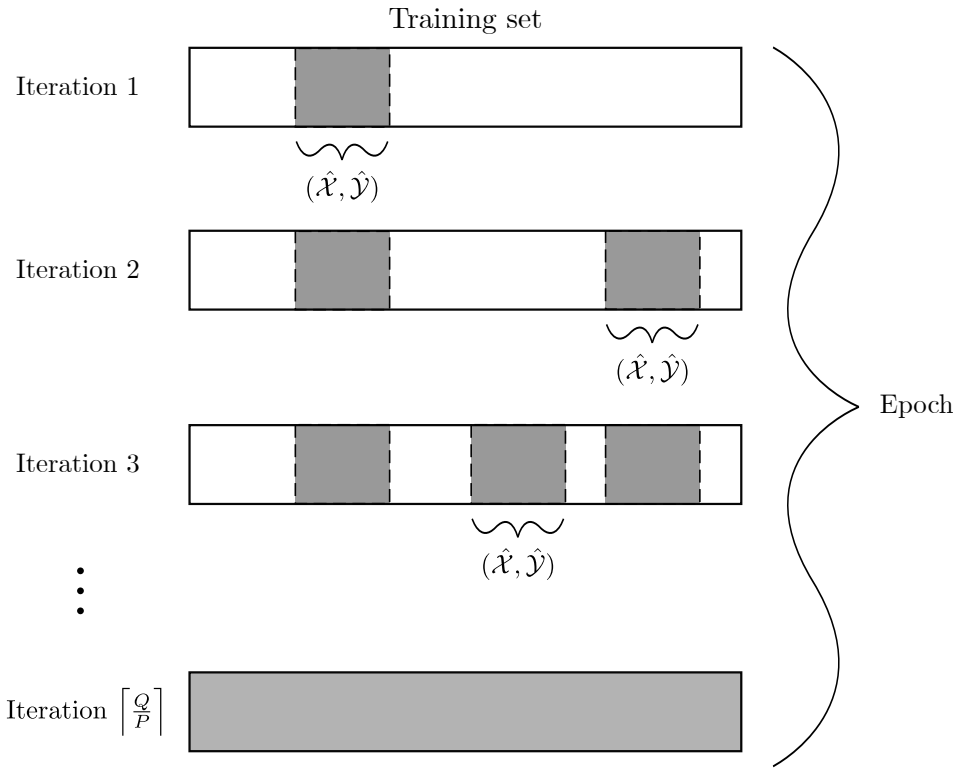


FIGURE 5.6: *The random sampling procedure employed to evaluate a network's performance. During each iteration of an epoch, a random mini-batch* $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$ *is sampled from a discrete uniform distribution without replacement. During subsequent iterations of the sampling procedure, previously sampled training examples are excluded which ensures that by the end of an epoch all training examples have been considered. Accordingly, an epoch comprises* $\lceil Q/P \rceil$ *iterations, where* Q *and* P *denote the training set size and the mini-batch size, respectively.*

the resulting fitness landscape (*i.e.* less severe plateaus). The proposed BOHTA is, however, a gradient-free method, rendering the severity of plateaus inconsequential. Due to the computationally expensive nature of a metaheuristic approach, it is paramount to prioritise computation speed where possible. This performance measure therefore warrants consideration in this regard as it will be evaluated many times throughout the optimisation process. A quantitative pilot study indicated that the MAE function provides the most computationally efficient loss function with respect to reducing computation time when compared with the other prominent functions mentioned in §3.6. In Table 5.1, a summary of the results obtained from the pilot study is provided. Box plots, also known as box-whisker plots, supplement[4] the analysis. The box plots in Figure 5.7 provide further insight into the computational speed afforded by the MAE function. Based on the computational performance achieved by the MAE, its inclusion in the main objective function in (5.13) is well justified.

The F_1-score, on the other hand, is employed to address the challenge posed by data sets that exhibit class imbalance. As discussed in §3.6, a large F_1-score indicates that the prediction is characterised by few false positives and few false negatives in respect of each class (or category). Most real-world problems (*i.e.* data sets) are characterised by a high degree of class imbalance, hence the use of this performance measure is well warranted.

[4]Box plots convey more information than the mere use of sample mean and standard deviation information [86]. The mean (indicated by a diamond), the median, the inter-quartile range, as well as the sample minimum and the sample maximum, are illustrated visually in box plots, and thus provide a more comprehensive representation of the central tendency and spread of the data samples.

TABLE 5.1: *The mean computation time (in seconds) per 100 000 function evaluations for the most prominent absolute value and square loss functions, as discussed in §3.6. In addition, the magnitude of the computation time improvement by each loss function relative to MAE is also summarised. Each experiment of 100 000 function evaluations was repeated fifty times so as to account for possible variation; the respective standard deviations are also indicated.*

Loss function	Mean computation time	Relative performance
MAE (3.16)	$0.113_{\pm 0.001}$	—
SSE (3.17)	$0.146_{\pm 0.002}$	-29.31%
MSE (3.18)	$0.145_{\pm 0.001}$	-27.87%
RMSE (3.19)	$0.150_{\pm 0.001}$	-32.48%

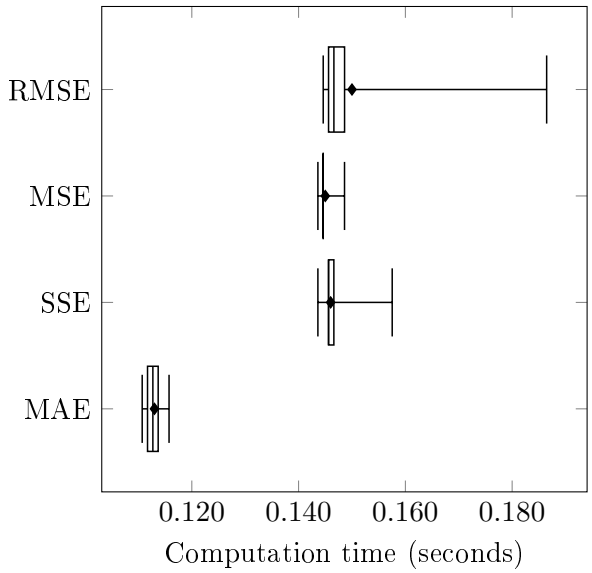


FIGURE 5.7: *Box plots illustrating the computation time for the most prominent absolute value and square loss functions.*

The F_1-score in (5.13) corresponds to the simple arithmetic mean in respect of the o classes and is given by

$$F_1 = \frac{1}{o}\sum_{k=1}^{o} F_1^{(k)}, \tag{5.14}$$

where $F_1^{(k)}$ denotes the F_1-score calculated with respect to class $k \in \{1, \ldots, o\}$. More specifically, after the candidate network's predictions have been determined in respect of the random mini-batch $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$, a binary classification context is assumed — *i.e.* a 2×2 confusion matrix is created for each of the o classes which contains the corresponding TP, TN, FP, and FN values. Thereafter, the precision score P and the recall score R are calculated according to (3.22)–(3.23) and this is followed by the F_1-score calculation according to (3.24) — these calculations are all performed in respect of the individual classes. Finally, the class-specific F_1-scores, *i.e.* $F_1^{(k)}$ for $k \in \{1, \ldots, o\}$, are averaged using the simple arithmetic mean calculation in (5.14). Each class is therefore given equal weighting — it is assumed that each class is assigned equal importance in respect of the classification task at hand.

The objective function proffered in (5.13) — and its subsequent use in the proposed bi-objective optimisation context — is both novel (to the best of the author's knowledge) and intuitive in its formulation. Salient aspects pertaining to both computational efficiency and predictive

performance are considered and addressed by the numerator and denominator, respectively. Accordingly, the main objective of the training algorithm is to find networks that have good weights, structure, and neuron-specific activation functions which deliver accurate predictions unaffected by class imbalance whilst expending computational resources conservatively. Admittedly, a mere qualitative analysis of the proposed approach does not suffice entirely. Therefore, quantitative and empirically substantiated evidence is provided later in this dissertation.

5.3.2 Regularising objective function

A helper objective is employed so as to better guide the search whilst the network is being trained. The helper objective aims to mitigate the problem of overfitting; it is appropriately called the regularising objective function, attributed to its regularisation capabilities. Minimisation of the objective function

$$h_2(\boldsymbol{S}) = \sum_{f=1}^{h} \sum_{j=1}^{m} s_j^{(f)} \tag{5.15}$$

is aimed at reducing the size of the network with respect to the number of hidden neurons, denoted by m, and the number of hidden layers, denoted by h. The intuition behind the incorporation of this objective function is to prefer candidate solutions exhibiting smaller network structures. This regularisation strategy attempts to minimise the number of adjustable parameters — the goal being to mitigate the problem of overfitting. Memorisation (performing well solely on the training set) is rendered a more difficult task when the network structure comprises fewer parameters. Consequently, the network is less inclined to overfit and more predisposed towards learning the "true" underlying functional representation hidden within the data set. It is expected that the generalisation performance (*i.e.* unbiased performance evaluation based on an unseen data set) will improve as a result of favouring smaller network structures.

Another benefit associated with the inclusion of this objective function pertains to computational efficiency. Due to the embedded preference for smaller networks, function evaluations (*i.e.* evaluating the performance of a candidate solution or network) will naturally incur a smaller burden from a computational perspective. As the network structure decreases in size, more weights are nullified by the "off" structural variables, resulting in sparse matrices and rendering matrix operations computationally efficient (relatively speaking). This feature helps mitigate the slow nature of a metaheuristic-based approach and the optimisation abstraction level. A quantitative pilot study indicated substantial improvements when matrix multiplications are performed in respect of sparse matrices.

5.4 Chapter summary

The aim in this chapter was to formulate an appropriate mathematical model of the problem at hand — *i.e.* training an FNN in respect of its network weights, network structure, and activation functions simultaneously. In §5.1, an overview of the mathematical representation was provided and discussed by means of a detailed graphical illustration. This model represents a notably lower level of abstraction than typical approaches in the literature. The incorporation of structural variables and activation functional variables highlight the main distinction — conventionally, the network weights are optimised separately. This was followed in §5.2 by detailed descriptions of the decision variables and constraints related to the network weights, network structure, and activation functions. Key notations and concepts were introduced and discussed, the first of which was the redundancy condition (discussed in §5.2.2). This was followed by the introduction and detailed description of the piece-wise linear activation function employed (in §5.2.3).

A bi-objective optimisation approach is followed in this dissertation. Accordingly, the two objective functions were discussed in §5.3. The mathematical derivation in §5.3.1 elucidated the procedure by which signals propagate throughout the network under different conditions — a necessary precursor to the discussion on the main objective function thereafter. A quantitative pilot study provided evidence of the computational efficiency afforded by the choice of performance measure in (5.13). In §5.3.2, the regularising objective function was introduced and discussed. This objective function aims to guide the search process by favouring smaller networks — a regularisation technique that helps mitigate overfitting. A key advantage associated with the working of the regularising objective function — *i.e.* reduced computation time — was also discussed.

CHAPTER 6

Solution Methodology

Contents

This chapter is devoted to an in-depth discussion on the solution methodology proposed in this dissertation for solving the model of the previous chapter — a methodology called the BOHTA approach. The aim of this hyperheuristic solution approach is to provide high-quality solutions to instances of the mathematical model formulated in Chapter 5, whilst demonstrating an enhanced level of general applicability compared to the application of the individual sub-algorithms. The chapter comprises two main sections. The principal components of the BOHTA approach are discussed in the first main section. This includes a high-level overview of the approach and descriptions of the procedures by which solutions are initialised. This is followed by a discussion on solution representation, which is a necessary precursor to the delineation of the evolutionary operators (employed by the various sub-algorithms) in the following subsection. In the second main section, the focus then turns to a description of the data sets employed as test suite in this dissertation. The test suite, comprising multiple data sets, is selected so as to facilitate the process of demonstrating the BOHTA approach and gaining insight into its working, strengths, and weaknesses. The relevant data pre-processing steps are described next. Penultimately, an important matter related to random sampling is addressed. The chapter closes with a brief summary.

6.1 The BOHTA approach

The model formulated in Chapter 5 is deemed an appropriate mathematical representation of the task of training FNNs so as to perform the supervised learning task of classification. The BOHTA approach is proposed as a powerful and robust solution methodology for providing high-quality solutions to the *bi-objective optimisation problem* (BOP) expressed in §5.3. In order to demonstrate the capabilities of the proposed approach and infer pertinent insight into its working,

a test suite of supervised learning problem instances (represented by data sets) is selected. Each data set gives rise to an instance of the BOP in §5.3. The proposed BOHTA is essentially tasked with finding networks — characterised by their weights, structure, and activation functions — that best approximate the underlying functional representations in respect of these data sets. A meticulous discourse on the most salient features of the proposed BOHTA approach follows a high-level overview of the methodology in this section.

6.1.1 High-level overview

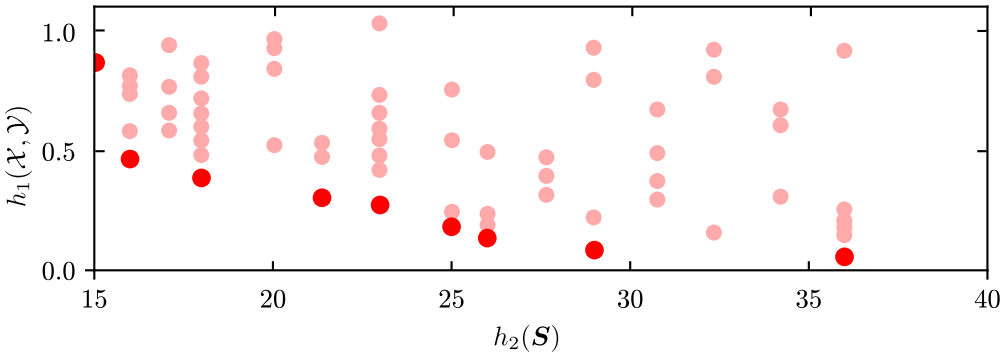
As mentioned, the hyperheuristic known as AMALGAM [171] serves as the foundational source of inspiration for the proposed solution methodology. AMALGAM is a selective-perturbative hyperheuristic, comprising evolutionary sub-algorithms, that has, to the best of the author's knowledge, not been applied to the task of training FNNs in this specific optimisation context. According to Vrugt and Robinson [171], AMALGAM is a powerful and robust optimisation approach that delivers high-quality solutions — whilst raising the level of general applicability — in respect of various benchmark MOPs. An investigation of an AMALGAM-inspired optimisation approach towards providing good solutions to instances of the mathematical model formulated in Chapter 5 is therefore well warranted. AMALGAM's pseudocode description in Algorithm 4.4 facilitates the exposition of the BOHTA in this section because of the notable similarities between these two approaches.

There are, however, a few key differences between these two approaches that require discussion. The most pronounced difference relates to the sub-algorithms employed in the proposed approach. In AMALGAM's original implementation, the NSGA-II, DE, PSO, and AMS were the four sub-algorithms employed. The proposed BOHTA approach omits AMS as it lacks convincing evidence in respect of its successful application towards ANN training in the literature. Vrugt and Robinson [171], furthermore, did not elucidate the MOO-related modifications in respect of DE and PSO, which leads (and contributes) to the conjecture that the MOO versions of DE and PSO employed are rudimentary. This is especially apparent when comparing the MOO GA version employed — *i.e.* the much celebrated NSGA-II. Consequently, enhanced and well-founded versions of DE and PSO — specifically within an MOO context — are rather preferred. The choice of NSDE and OMOPSO as the preferred BOHTA sub-algorithms is based on findings of SOTA surveys, as discussed in §4.2.2 and §4.2.3.

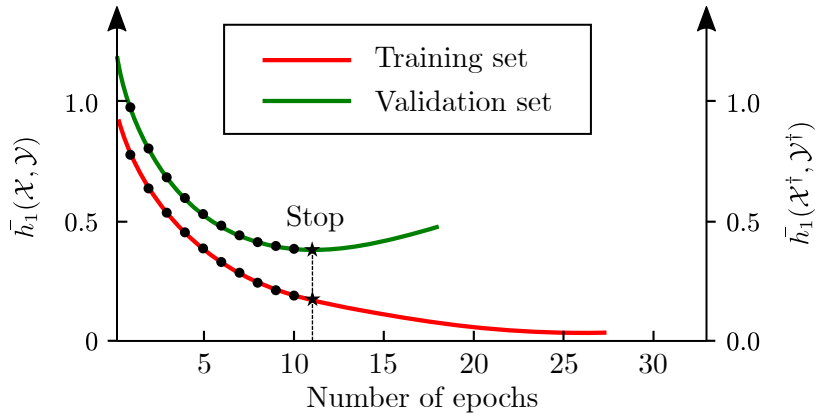
According to the pseudocode description in Algorithm 4.4, AMALGAM requires a set of k sub-algorithms. The BOHTA similarly employs $k = 3$ sub-algorithms, *i.e.* the NSGA-II, NSDE, and OMOPSO, and their pseudocode descriptions are presented in Algorithms 4.1, 4.2, and 4.3, respectively. AMALGAM's input also includes an MOP instance, a population size M, and a stopping criterion, *i.e.* a maximum number of generations t_{max}. The mathematical model formulated in §5.3 comprises two objective functions, *i.e.* $h_1(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$ and $h_2(\boldsymbol{S})$. A BOP instance, instead of an MOP instance, therefore serves as input to the BOHTA (and its constituent sub-algorithms). The data sets contained within the test suite represent the different BOP instances in terms of which the BOHTA is evaluated. Appropriate values for the population size M are determined by means of a sub-algorithm parameter evaluation (discussed later). Lastly, the stopping criterion employed by the BOHTA differs from AMALGAM's — accordingly, the maximum number of generations (or iterations) is determined by means of early stopping.

Recall from §5.3.1 that an epoch comprises $\lceil Q/P \rceil$ training iterations, where Q and P denote the training set size and the mini-batch size, respectively. Each generation of the BOHTA represents one training iteration. The training performance of a network with respect to a mini-batch is denoted by $h_1(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$, *i.e.* the main objective function in (5.13). The training

performance of a network with respect to the *entire* training set, on the other hand, is denoted by $h_1(\mathcal{X}, \mathcal{Y})$. Furthermore, the validation performance with respect to the *entire* validation set is denoted by $h_1(\mathcal{X}^\dagger, \mathcal{Y}^\dagger)$. The BOHTA terminates when, after a certain number of epochs, the average validation performance, denoted by $\bar{h_1}(\mathcal{X}^\dagger, \mathcal{Y}^\dagger)$, of the networks (or solutions) in the approximation front fails to improve during subsequent epochs. The process of early stopping is illustrated graphically in Figure 6.1.



(a) Training performance of the population after an arbitrary number of epochs



(b) Average training and validation performance of each approximation front

FIGURE 6.1: *The notion of early stopping illustrated graphically in the context of the BOHTA, where (a) represents the training performance of all solutions within the population after an arbitrary number of epochs have elapsed. Bright red circles represent solutions in the approximate Pareto front, whereas the semi-transparent red circles represent dominated solutions. The vertical axis represents the main objective function in* (5.13) *evaluated in respect of the entire training set, whereas the horizontal axis represents the regularising objective function in* (5.15)*. The average approximation front performance achieved during each epoch is plotted in (b). The vertical axis on the left, together with the red curve, represents the average approximation performance with respect to the entire training set, whereas the vertical axis on the right, together with the green curve, represents the average performance of the approximation front with respect to the entire validation set. Black circles represent individual epochs, whereas the black star represents the epoch at which performance stops improving (i.e. the epoch after which the BOHTA is terminated). In the illustrated example, the training process is stopped after eleven epochs.*

A so-called *patience* parameter accompanies the process of early stopping which determines the number of successive non-improving epochs considered before stopping the training process. The inclusion of this parameter ensures that the optimisation procedure is not halted prematurely — it is reasonable to assume that the optimisation approach will most likely have to undergo some degradation before finding favourable regions in the search space. In Figure 6.1, a patience value

of one is adopted, *i.e.* the training process is stopped after the first non-improving epoch occurs. There is, however, an intricate trade-off between computational expenditure and the quality of solutions generated when deciding upon an appropriate patience value. Unfortunately, there is no widely-used heuristic for deciding upon a suitable patience value. Consequently, a separate qualitative pilot study was conducted which indicated that a patience value of five resulted in good convergence without excessive training times. This finding is corroborated by various other findings in literature according to which a patience value of five results in a good trade-off between performance and computation burden [4, 53, 125].

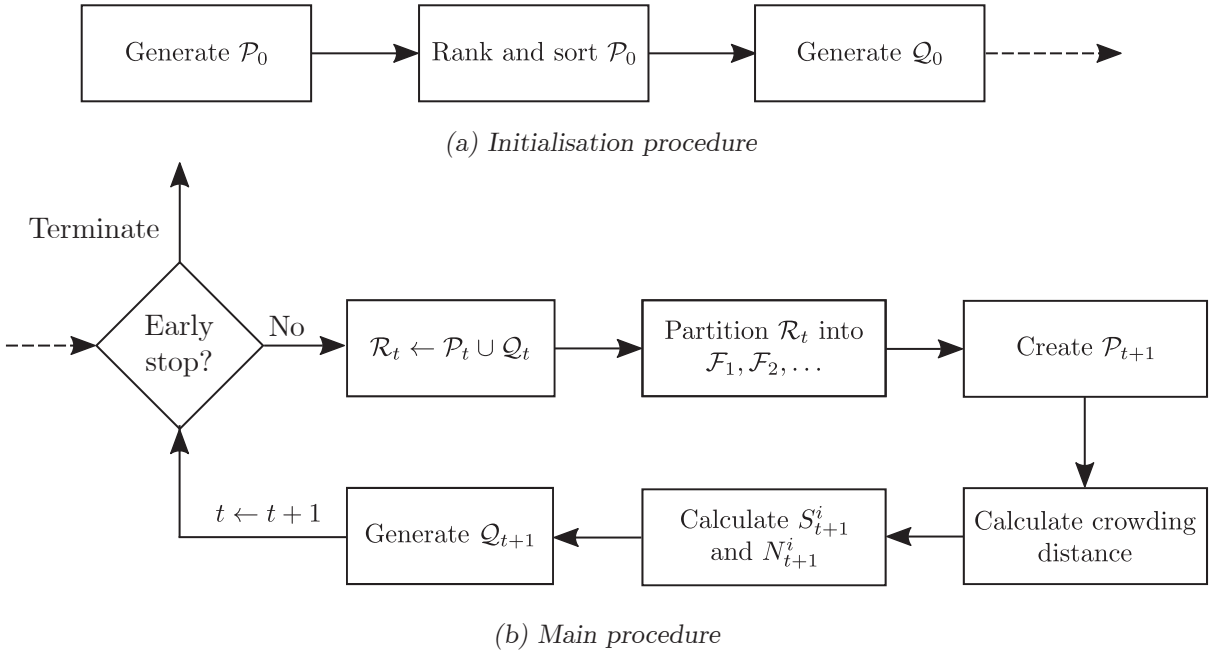
The proposed stopping criterion is therefore *dynamic* and changes from one BOP instance to another — its use can be ascribed to the context in which the BOHTA is applied, *i.e.* training FNNs, together with the notable popularity and success associated with early stopping. The advantages of early stopping is threefold: First, it is a simple stopping criterion, secondly, it negates (to an extent) overfitting, therefore improving generalisation performance, as discussed in §3.8.4, and, thirdly, it is problem-specific, therefore facilitating a more robust performance comparison.

Given a BOP instance, a population size M, and $k = 3$ sub-algorithms, the BOHTA commences with an initialisation procedure, as illustrated graphically in Figure 6.2(a). The generation counter t is set to zero. This initialisation procedure starts with the creation of a parent population $\mathcal{P}_0$, comprising M randomly generated candidate solutions (or networks). The process of random initialisation is discussed later in greater depth. The solutions in $\mathcal{P}_0$ are then assigned Pareto ranks by the FNSA in Algorithm 2.1, after which an offspring population $\mathcal{Q}_0$ of size M is generated. Accordingly, each sub-algorithm generates approximately $N_0^i = M/3$ solutions, for $i \in \{G, D, P\}$, where G, D, and P correspond to the NSGA-II, NSDE, and OMOPSO, respectively. The creation of the offspring population $\mathcal{Q}_0$ concludes the initialisation procedure. The BOHTA's main (iterative) procedure then follows and is illustrated graphically in Figure 6.2(b). This procedure, which is, by and large, identical to that of AMALGAM (described in §4.4), is iterated until the stopping criterion is met, *i.e.* the average validation performance of solutions in the approximation front fails to improve during five successive epochs. An approximate Pareto set $\tilde{\mathcal{P}}_S$, comprising computationally promising networks in respect of the BOP at hand, is returned as high-quality trade-off solutions thereafter.

As mentioned, each sub-algorithm is required to generate an equal number of solutions during the BOHTA's initialisation procedure, which is only possible if M is a multiple of 3. The manner according to which AMALGAM handles the situation when this is not the case was not addressed by Vrugt and Robinson [171]. The mechanism governing AMALGAM's self-adaptive offspring generation, given in (4.11), can result in values for

$$N_{t+1}^i = \frac{\left(\frac{S_{t+1}^i}{N_t^i}\right)}{\sum_{i=1}^k \left(\frac{S_{t+1}^i}{N_t^i}\right)},$$

not being integers. The BOHTA employs a simple procedure for dealing with this situation (including during initialisation). Accordingly, the number of solutions that each sub-algorithm is required to generate is first set to $N_t^i = N_t^j = N_t^k = \lfloor M/3 \rfloor$, which is followed by the random allocation of the remaining $M - 3\lfloor M/3 \rfloor$ solutions, according to a uniform distribution. If only one solution has to be allocated randomly (*i.e.* $M - 3\lfloor M/3 \rfloor = 1$), then a random sub-algorithm $r \in \{G, D, P\}$ is selected and its population size is incremented (*i.e.* $N_t^r \leftarrow N_t^r + 1$). If, however, two solutions have to be allocated randomly (*i.e.* $M - 3\lfloor M/3 \rfloor = 2$), then random sub-algorithms $r \in \{G, D, P\}$ and $r' \in \{\{G, D, P\} \backslash \{r\}\}$ have their respective population sizes incremented (*i.e.* $N_t^r \leftarrow N_t^r + 1$ and $N_t^{r'} \leftarrow N_t^{r'} + 1$).



(a) Initialisation procedure

(b) Main procedure

FIGURE 6.2: *(a) The initialisation procedure and (b) the main (iterative) procedure of the BOHTA. These procedures closely emulate those of AMALGAM, as described in Algorithm 4.4.*

The operation of the BOHTA is characterised by a competition amongst its three sub-algorithms — essentially, a competition for computation opportunity. The higher the relative reproductive success of a sub-algorithm, *i.e.* the value of S^i_{t+1} in (4.11), the more opportunity it is afforded to generate solutions (or networks), *i.e.* the value of N^i_{t+1}. Because the population size remains constant, this situation is akin to a *zero-sum game*, *i.e.* if one sub-algorithm's relative performance increases, the other sub-algorithms' combined allocated computation budgets decrease accordingly. The underlying notion of the BOHTA is simple: Instead of allocating the entire computation budget to a single training algorithm, the budget is rather allotted to multiple training algorithms and the allotment is based on the training algorithms' capability of producing high-quality solutions to the problem at hand. During the optimisation process, it is conjectured that certain sub-algorithms will perform better than others, depending on where in the solution space the BOHTA operates. These locations in the solution space are represented by the networks themselves. Therefore, given a network that has *some* structure, comprising *some* set of network weights, and employing *some* set of activation functions, certain sub-algorithms may be better equipped to evolve this network into a higher-quality solution. This conjecture can, to an extent, be corroborated by the NFL theorem — no single optimisation algorithm will always outperform another. It is therefore sensible not just to rely on one training algorithm, as it cannot consistently outperform all other training algorithms during the entire optimisation process. The age-old adage of "Two heads are better than one," applies, even in the context of solving optimisation problems. The BOHTA embodies this philosophy algorithmically and a degree of synergy is expected from the implicit cooperation facilitated by the BOHTA — *i.e.* a combined effect, which is greater than the sum of the respective effects, is expected.

6.1.2 Solution initialisation

The initialisation procedure of the BOHTA includes the process of generating a population of random feasible solutions (or networks), denoted by $\mathcal{P}_0$ in Figure 6.2(a). As elucidated in

§5.2, a network comprises three main classes of decision variables, *i.e.* network weights, network structure, and activation functions. The manner by which random values are assigned to these decision variables is discussed in greater depth in this section.

Network structure initialisation

The structure of a network relates to its width and depth, *i.e.* the number of hidden layers and the number of hidden neurons. A random network therefore represents a network comprising a random number of non-redundant hidden layers as well as a random number of active hidden neurons within each non-redundant hidden layer. The switching variables defined in §5.2.2 determine the size of the network, that is

$$\sum_{j=1}^{m} s_j^{(f)}, \qquad s_j^{(f)} \in \{0,1\},\ f \in \{1,\ldots,h\}$$

expresses the number of active hidden neurons and, as a result, also expresses the number of non-redundant hidden layers, subject to the redundancy condition in (5.5). Consequently, the parameters h and m determine an upper bound on the network structure. The largest network can therefore comprise at most h non-redundant hidden layers and at most m active hidden neurons per hidden layer, *i.e.* the largest number of hidden neurons (across all hidden layers) that can be active is mh. The pseudocode description in Algorithm 6.1 outlines the procedure for randomly generating the entries of the matrix $\boldsymbol{S}$ (the $m \times h$ matrix in which all structural variables are stored). The input to this algorithm is the matrix $\boldsymbol{S}$ in which all hidden neurons are initially inactive (or switched "off"), *i.e.* $s_j^{(f)} = 0$ for all $j \in \{1,\ldots,m\}$ and all $f \in \{1,\ldots,h\}$. The random integer r (generated in line 1) determines the number of hidden neurons in $\boldsymbol{S}$ that should be activated and is randomly sampled from a discrete uniform distribution on the interval $[\lceil mh/3 \rceil, mh]$, denoted by $\mathcal{U}[\lceil mh/3 \rceil, mh]$. The lower bound of $[\lceil mh/3 \rceil, mh]$ was determined during a separate qualitative pilot study. According to the findings of this pilot study, a lower bound of 0 consistently resulted in the training algorithm converging towards networks comprising no active neurons, *i.e.* $h_2(\boldsymbol{S}) = 0$. These networks tend to perform poorly and are classified as undesired local minima. Consequently, alternative lower bounds were explored so as to afford the training algorithm an improved opportunity to prevent premature convergence towards these local minima. A lower bound of $[\lceil mh/3 \rceil, mh]$ was found to deliver the most promising results, hence its employment.

Algorithm 6.1: Network structure initialisation

Input : The number of hidden neurons m, the number of hidden layers h, and an $m \times h$ matrix $\boldsymbol{S}$ comprising distinct, strictly inactive switching variables.

Output: An $m \times h$ matrix $\boldsymbol{S}$ comprising randomly activated or inactive switching variables.

1 Generate a random integer $r \sim \mathcal{U}[\lceil mh/3 \rceil, mh]$;
2 Generate a set $\mathcal{L}$ of length r comprising unique random integers from $\mathcal{U}[1, hm]$;
3 Sort $\mathcal{L}$ in ascending order;
4 **for** $f \leftarrow 1$ **to** h **do**
5 **for** $j \leftarrow 1$ **to** m **do**
6 **if** $\mathcal{L}_{h(f-1)+j} = h(f-1)+j$ **then**
7 $s_j^{(f)} \leftarrow 1$;

The set $\mathcal{L}$ of length r (generated in line 2) determines the specific random neurons within the network that should be activated and comprises distinct random integers sampled from the range $[1, mh]$, denoted by $\mathcal{L} \sim \mathcal{U}[1, mh]$ without replacement. Sampled values are not replaced so as to prevent the same neurons from being activated multiple times. The procedure spanning lines 4–7 iterates through each hidden neuron within each hidden layer, activating the relevant neurons according to the entries of $\mathcal{L}$. The lower and upper bounds of the distribution $\mathcal{U}[1, mh]$ (from which the entries of $\mathcal{L}$ are sampled) indicate that any hidden neuron within any hidden layer can be activated. The matrix $\boldsymbol{S}$, comprising the r activated neurons, is returned at the end of the algorithm's execution.

A simple example of a possible random network structure is as follows: For parameter values $h = 3$ and $m = 5$ as well as a randomly generated value of $r = 7$ and sorted set $\mathcal{L} = \{2, 4, 7, 9, 10, 11, 13\}$, the corresponding matrix $\boldsymbol{S}$ may be

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Accordingly, hidden layers $f = 1$ and $f = 3$ each contain two active hidden neurons and three inactive hidden neurons, whereas hidden layer $f = 2$ contains three active hidden neurons and two inactive hidden neurons. Another simple example is as follows: For parameter values $h = 5$ and $m = 3$ as well as a randomly generated value of $r = 10$ and sorted set $\mathcal{L} = \{1, 2, 3, 7, 9, 11, 12, 13, 14, 15\}$, the corresponding matrix $\boldsymbol{S}$ may be

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

A key remark is that hidden layer $f = 2$ is redundant, attributed to the fact that $\sum_{j=1}^{3} s_j^{(2)} = 0$. Therefore, this hidden layer's activation functions and weights, denoted by $\boldsymbol{g}^{(2)}$ and $\boldsymbol{W}^{(2)}$, respectively, are disregarded and the net input to this layer is only processed by the next hidden layer.

Network weights initialisation

A network comprises $h + 1$ weighted-connection layers and each corresponding weight-matrix is contained within the ordered list $\boldsymbol{w} = \{\boldsymbol{W}^{(0)}, \ldots, \boldsymbol{W}^{(f)}, \ldots, \boldsymbol{W}^{(h)}\}$. The initialisation of the BOHTA therefore includes the assignment of random values to each network weight contained within the weight-matrices of $\boldsymbol{w}$. These weight-matrices are initially only populated with zeros, before being subjected to the weight initialisation procedure. The vast majority of studies related to weight initialisation have been conducted within the context of a gradient-based training approach. There is, furthermore, a noteworthy lack of consensus on favourable weight initialisation procedures within a non-gradient-based training approach (such as that adopted in this dissertation). Consequently, the most popular initialisation procedures within a gradient-based context are rather considered — attributable to their superior performance. A future investigation aimed at uncovering good weight initialisation procedures in a non-gradient-based context may, however, prove worthwhile.

The two most popular weight initialisation procedures (within a gradient-based context) were discussed in §3.5.6. The applicability of these procedures depends on the type of activation function

employed — *e.g.* a sigmoid activation function or a PReLU activation function. The activation function proposed in the mathematical model of Chapter 5 is a piecewise linear function, as discussed in §5.2.3. Greater similarity is exhibited between the proposed activation function and the PReLU activation function (as opposed to the sigmoid activation function). Consequently, the weight initialisation procedure corresponding to the PReLU function is employed in the BOHTA approach. As discussed in §3.5.6, He *et al.* [59] described this procedure as follows: The (continuous) normal distribution, from which the real-valued random values are drawn, must have a mean of zero and a standard deviation of $\sqrt{2/\tilde{m}}$, where $\tilde{m}$ denotes the fan-in number, *i.e.* the number of connections entering the neuron. The corresponding notation for this distribution is $\mathcal{N}(0, 2/\tilde{m})$. The weights associated with the biases are drawn from a normal distribution with a standard deviation of 1. The corresponding notation is $\mathcal{N}(0, 1)$.

A modified notation, given by $\tilde{m}^{(f)}$, is newly introduced and denotes the fan-in number corresponding to weight-matrix $\boldsymbol{W}^{(f)}$, for $f \in \{0, \ldots, h\}$. Accordingly, the new notation for the distribution is given by $\mathcal{N}(0, 2/\tilde{m}^{(f)})$. The pseudocode description in Algorithm 6.2 outlines the weight initialisation procedures for the different weight-matrices. Three distinct cases — each corresponding to a different weight-matrix — are considered:

1. Weight-matrix $\boldsymbol{W}^{(0)}$ — containing the weights between the input layer and the first hidden layer — is initialised according to lines 2–7. The fan-in number depends on the number of input neurons n, which remains fixed. The corresponding calculation is $\tilde{m}^{(0)} \leftarrow n$. Only the weights that are connected to active hidden neurons in the first hidden layer are initially assigned random values. The relevant weight initialisation procedure is therefore $w_{ji}^{(0)} \leftarrow \mathcal{N}(0, 2/\tilde{m}^{(0)})$, for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\} : s_j^{(1)} = 1$. The initialisation procedure for weights associated with the input layer bias is $w_{j0}^{(0)} \leftarrow \mathcal{N}(0, 1)$ for $j \in \{1, \ldots, m\} : s_j^{(1)} = 1$, as indicated in line 7.

2. Weight-matrix $\boldsymbol{W}^{(f)}$ for $f \in \{1, \ldots, h-1\}$ — containing the weights between any two successive hidden layers — is initialised according to lines 8–14. The fan-in number depends on the number of active hidden neurons in the hidden layer at the start of the weighted-connection layer. The corresponding calculation is $\tilde{m}^{(f)} \leftarrow \sum_{j=1}^{m} s_j^{(f)}$, for $f \in \{1, \ldots, h-1\}$. Only the weights that connect active hidden neurons in the preceding hidden layer f to active hidden neurons in the succeeding hidden layer $f+1$ are initially assigned random values. The relevant weight initialisation procedure is therefore $w_{jj'}^{(f)} \leftarrow \mathcal{N}(0, 2/\tilde{m}^{(f)})$, for $j' \in \{1, \ldots, m\} : s_{j'}^{(f)} = 1$ and $j \in \{1, \ldots, m\} : s_j^{(f+1)} = 1$. The initialisation procedure for weights associated with the bias of hidden layers $f \in \{1, \ldots, h-1\}$ is $w_{j0}^{(f)} \leftarrow \mathcal{N}(0, 1)$ for $j \in \{1, \ldots, m\} : s_j^{(f+1)} = 1$, as indicated in line 14.

3. Weight-matrix $\boldsymbol{W}^{(h)}$ — containing the weights between the last hidden layer and the output layer — is initialised according to lines 15–20. The fan-in number depends on the number of active hidden neurons in the last hidden layer. The corresponding calculation is $\tilde{m}^{(h)} \leftarrow \sum_{j=1}^{m} s_j^{(h)}$. Only the weights that connect active hidden neurons in the last hidden layer to the output neurons are initially assigned random values. The relevant weight initialisation procedure is therefore $w_{kj}^{(h)} \leftarrow \mathcal{N}(0, 2/\tilde{m}^{(h)})$, for $j \in \{1, \ldots, m\} : s_j^{(h)} = 1$ and $k \in \{1, \ldots, o\}$. The initialisation procedure for weights associated with the last hidden layer bias is $w_{k0}^{(h)} \leftarrow \mathcal{N}(0, 1)$ for $k \in \{1, \ldots, o\}$, as indicated in line 20.

The different weight initialisation procedures discussed above only assign random values initially, *i.e.* when $\mathcal{P}_0$ is generated. Presumably, the BOHTA (and its sub-algorithms) will activate and

Algorithm 6.2: Network weight initialisation

Input : An ordered list $\boldsymbol{w} = \{\boldsymbol{W}^{(0)}, \ldots, \boldsymbol{W}^{(f)}, \ldots, \boldsymbol{W}^{(h)}\}$ comprising zero-valued weights, the number of input neurons n, the number of output neurons o, the network structure matrix $\boldsymbol{S}$.

Output: An ordered list $\boldsymbol{w} = \{\boldsymbol{W}^{(0)}, \ldots, \boldsymbol{W}^{(f)}, \ldots, \boldsymbol{W}^{(h)}\}$ comprising randomly initialised weight matrices.

foreach $f \in \{0, \ldots, h\}$ **do**
  if $f = 0$ **then**
    $\tilde{m}^{(0)} \leftarrow n$;
    foreach $i \in \{1, \ldots, n\}$ **do**
      foreach $j \in \{1, \ldots, m\} : s_j^{(1)} = 1$ **do**
        $w_{ji}^{(0)} \leftarrow \mathcal{N}(0, 2/\tilde{m}^{(0)})$;
        $w_{j0}^{(0)} \leftarrow \mathcal{N}(0, 1)$;
  if $f > 0$ **and** $f \neq h$ **then**
    foreach $f \in \{1, \ldots, h-1\}$ **do**
      $\tilde{m}^{(f)} \leftarrow \sum_{j=1}^{m} s_j^{(f)}$;
      foreach $j' \in \{1, \ldots, m\} : s_{j'}^{(f)} = 1$ **do**
        foreach $j \in \{1, \ldots, m\} : s_j^{(f+1)} = 1$ **do**
          $w_{jj'}^{(f)} \leftarrow \mathcal{N}(0, 2/\tilde{m}^{(f)})$;
          $w_{j0}^{(f)} \leftarrow \mathcal{N}(0, 1)$;
  if $f = h$ **then**
    $\tilde{m}^{(f)} \leftarrow \sum_{j=1}^{m} s_j^{(f)}$;
    foreach $j \in \{1, \ldots, m\} : s_j^{(f)} = 1$ **do**
      foreach $k \in \{1, \ldots, o\}$ **do**
        $w_{kj}^{(f)} \leftarrow \mathcal{N}(0, 2/\tilde{m}^{(f)})$;
        $w_{k0}^{(f)} \leftarrow \mathcal{N}(0, 1)$;

inactivate hidden neurons when generating the offspring population $\mathcal{Q}_t$, as indicated in Figure 6.2, aiming to find network structures that deliver superior performance in respect of the main objective function. Whenever an inactivated neuron is activated (switched "on"), the same weight initialisation procedures, as described above, are applied when assigning random values to the weights that are connected to the newly activated neurons. The changes in the network structure matrix $\boldsymbol{S}$ indicate which weights should be initialised. These procedures are, however, only applicable to hidden neurons that are activated for the first time, *i.e.* hidden neurons that were previously inactive.

The aim of the modelling approach described above is to provide initialised hidden neurons (and their accompanying weights) the best computational capability to contribute positively towards minimising the main objective function. It is conjectured that the BOHTA is less inclined to inactivate these suitably initialised hidden neurons prematurely. This modelling approach with respect to weight initialisation, given a changing network structure, is, to the best of the author's knowledge, novel.

Activation function initialisation

The last class of decision variables that are randomly initialised (when generating $\mathcal{P}_0$) relates to the activation functions employed by the hidden neurons in the network. Recall from §5.2.3 that so-called activation functional variables represent the piecewise linear functions employed by the hidden neurons. Accordingly, each hidden neuron has a corresponding activation functional variable $g_j^{(f)}(\eta_j^{(f)})$ for $j \in \{1, \ldots, m\}$ and $f \in \{1, \ldots, h\}$, comprising a slope variable for negative input, *i.e.* $\alpha_j^{(f)}$ if $\eta_j^{(f)} < 0$, and a slope variable for non-negative input, *i.e.* $\beta_j^{(f)}$ if $\eta_j^{(f)} \geq 0$. This modelling approach of finding neuron-specific piecewise linear activation functions is, to the best of the author's knowledge, also novel, although it is not drastically different from current approaches. Similarities shared with the PReLU activation function specifically, are somewhat apparent, while greater flexibility is nevertheless facilitated by the proposed approach — attributable to both $\alpha_j^{(f)} \in \mathbb{R}$ and $\beta_j^{(f)} \in \mathbb{R}$. Accordingly, it is conjectured that more computational freedom is afforded to the BOHTA when attempting to best emulate the firing process of biological neurons. The net input to hidden neurons can be scaled in a continuous fashion, allowing for a notably lower level of abstraction when attempting to emulate the inhibitory or excitatory relationships that exist between the presented input vector $\hat{\boldsymbol{x}}^p$ and the corresponding output vector $\hat{\boldsymbol{y}}^p$.

The initialisation of the BOHTA therefore includes the assignment of random values to the respective slope variables of the activation functions employed by hidden neurons within the network. Random values are assigned to the slope variables of *all* hidden neurons, regardless of their state, *i.e.* whether they are active or inactive. Presumably, the BOHTA will activate most, if not all, hidden neurons during the optimisation process. Therefore, all neurons have already been assigned random values at the time of their initialisation. The weight initialisation procedures (discussed in this section) inherently depend on the network structure, and so all network weights are not initialised at the start, which stands in contrast to the approach for activation functions. In order to facilitate the greater flexibility claimed, the slope variables $\alpha_j^{(f)}$ and $\beta_j^{(f)}$ are randomly sampled from the uniform continuous distribution $\mathcal{U}(-1, 1)$. The reasoning behind the use of unit-based bounds is twofold: First, the input data $\mathcal{X}$ are standardised and, secondly, the target variables in $\mathcal{Y}$ are discrete (attributable to the nature of the problem at hand, *i.e.* classification). Unit-based bounds therefore naturally coincide with the input and output data ranges. Larger bounds are possible, although their adoption may lead to unnecessarily large slope variables and, in the process, biassing the net input to the neuron. The bounds $(-1, 1)$ help mitigate this undesirable situation.

6.1.3 Solution encoding

An essential part of the proposed solution methodology pertains to the evolutionary operators employed by the constituent sub-algorithms. A necessary precursor to a discussion on the working of these evolutionary operators is a delineation of the solution representation adopted, *i.e.* the *encoding scheme* employed. In order for a sub-algorithm to *evolve* solutions (or networks), an appropriate encoding format is required. Candidate solutions should be represented in a format that enables and facilitates the application of the respective variation operators of the NSGA-II, NSDE, and OMOPSO. Accordingly, a vector (or string) encoding scheme provides the necessary means to this end and is therefore proposed for use in this dissertation.

A hypothetical network underpins the subsequent explication so as to effectively demonstrate the proposed encoding scheme. The parameters of this hypothetical network are as follows:

- An input and output layer size of $n = 3$ and $o = 2$, respectively, and

- a network depth and width of $h = 3$ and $m = 3$, respectively.

Given this arbitrary network set-up, the network structure is randomly initialised by means of Algorithm 6.1, resulting in the matrix

$$\boldsymbol{S} = \begin{bmatrix} s_1^{(1)} & s_1^{(2)} & s_1^{(3)} \\ s_2^{(1)} & s_2^{(2)} & s_2^{(3)} \\ s_3^{(1)} & s_3^{(2)} & s_3^{(3)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}. \tag{6.1}$$

A corresponding graphical illustration of the network structure is provided in Figure 6.3. It may be observed that certain neurons and weights are greyed out, attributable to the fact that their corresponding switching variables are zero, *i.e.* these neurons are inactive, as expressed in (6.1).

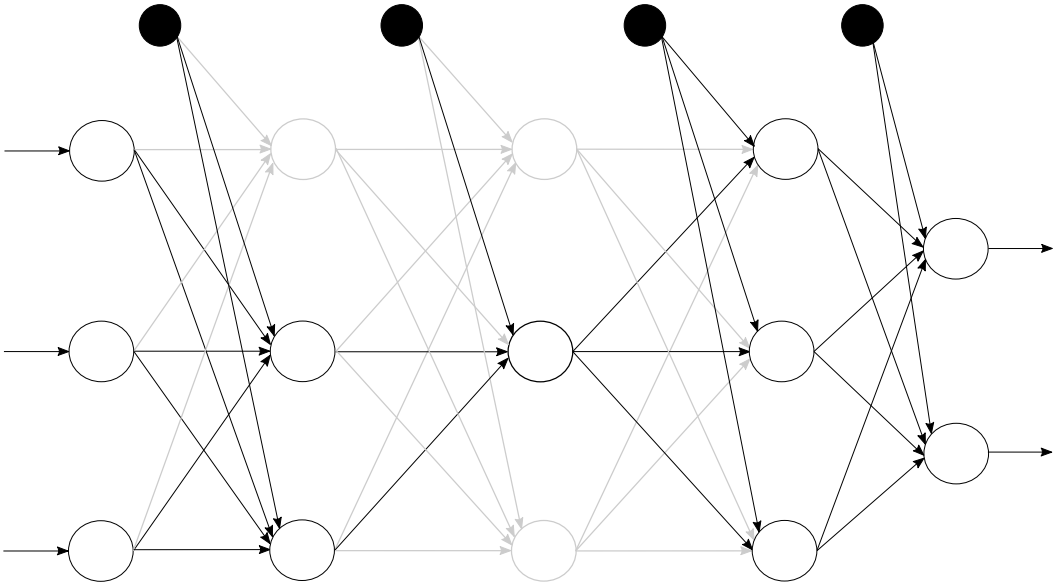


FIGURE 6.3: *An illustration of a hypothetical network in which $n = 3$, $o = 2$, $h = 3$, and $m = 3$. The inactive neurons are greyed out.*

After randomly initialising the network structure, the next step is to initialise the weights randomly between the active neurons within the network. Suppose Algorithm 6.2 is employed for this purpose, resulting in the weight matrices

$$\boldsymbol{W}^{(0)} = \begin{bmatrix} w_{10}^{(0)} & w_{11}^{(0)} & w_{12}^{(0)} & w_{13}^{(0)} \\ w_{20}^{(0)} & w_{21}^{(0)} & w_{22}^{(0)} & w_{23}^{(0)} \\ w_{30}^{(0)} & w_{31}^{(0)} & w_{32}^{(0)} & w_{33}^{(0)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -0.20 & 0.76 & -0.17 & -0.45 \\ -0.43 & 0.67 & -0.08 & -0.01 \end{bmatrix}, \tag{6.2}$$

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{30}^{(1)} & w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1.72 & 0 & -1.01 & 0.37 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{6.3}$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{10}^{(2)} & w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ w_{20}^{(2)} & w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \\ w_{30}^{(2)} & w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)} \end{bmatrix} = \begin{bmatrix} 0.43 & 0 & -1.27 & 0 \\ 0.21 & 0 & -0.81 & 0 \\ 2.50 & 0 & -0.80 & 0 \end{bmatrix}, \tag{6.4}$$

and

$$\boldsymbol{W}^{(3)} = \begin{bmatrix} w_{10}^{(3)} & w_{11}^{(3)} & w_{12}^{(3)} & w_{13}^{(3)} \\ w_{20}^{(3)} & w_{21}^{(3)} & w_{22}^{(3)} & w_{23}^{(3)} \end{bmatrix} = \begin{bmatrix} -0.26 & -0.40 & 0.58 & -0.73 \\ 0.14 & 0.72 & -0.23 & -1.05 \end{bmatrix}. \tag{6.5}$$

The zero-valued weights correspond to the greyed out weighted connections in Figure 6.3, and are only initialised when the corresponding switching variables are changed by the training algorithm, therefore activating the relevant hidden neurons, as discussed in §6.1.2. The last step is to initialise the activation functional variables randomly. Unlike the initialisation procedure for the network weights, this initialisation procedure does not depend on the network structure, and hence all activation functions are initialised with random values. Accordingly, the network's activation functional variables are given by

$$\boldsymbol{G} = \begin{bmatrix} g_1^{(1)}(\cdot) & g_1^{(2)}(\cdot) & g_1^{(3)}(\cdot) \\ g_2^{(1)}(\cdot) & g_2^{(2)}(\cdot) & g_2^{(3)}(\cdot) \\ g_3^{(1)}(\cdot) & g_3^{(2)}(\cdot) & g_3^{(3)}(\cdot) \end{bmatrix}.$$

The procedure described in §6.1.2 is employed to assign random values to the slope variables, yielding, for example,

$$\begin{bmatrix} \alpha_1^{(1)} & \alpha_1^{(2)} & \alpha_1^{(3)} \\ \alpha_2^{(1)} & \alpha_2^{(2)} & \alpha_2^{(3)} \\ \alpha_3^{(1)} & \alpha_3^{(2)} & \alpha_3^{(3)} \end{bmatrix} = \begin{bmatrix} 0.48 & -0.75 & -0.38 \\ -0.57 & -0.51 & 0.46 \\ -0.11 & -0.20 & -0.39 \end{bmatrix}, \tag{6.6}$$

for negative input (*i.e.* $\eta_j^{(f)} < 0$ for all $j \in \{1, \ldots, 3\}$ and all $f \in \{1, \ldots, 3\}$), and

$$\begin{bmatrix} \beta_1^{(1)} & \beta_1^{(2)} & \beta_1^{(3)} \\ \beta_2^{(1)} & \beta_2^{(2)} & \beta_2^{(3)} \\ \beta_3^{(1)} & \beta_3^{(2)} & \beta_3^{(3)} \end{bmatrix} = \begin{bmatrix} -0.99 & 0.68 & -0.66 \\ 0.06 & -0.64 & -0.36 \\ 0.10 & 0.26 & 0.62 \end{bmatrix} \tag{6.7}$$

for non-negative input (*i.e.* $\eta_j^{(f)} \geq 0$ for all $j \in \{1, \ldots, 3\}$ and all $f \in \{1, \ldots, 3\}$).

In order to facilitate the application of the various variation operators of the sub-algorithms, *i.e.* to evolve this network, a vector or string format is proposed, in which the decision variables are collectively encoded as decision vectors. Three different decision vectors are defined — one for each class of decision variables. Accordingly, the vectors $\boldsymbol{d}_{\boldsymbol{S},j}^t$, $\boldsymbol{d}_{\boldsymbol{w},j}^t$, and $\boldsymbol{d}_{\boldsymbol{G},j}^t$ denote the decision vectors $j \in \{1, \ldots, M\}$ during generation t, which corresponds to the network structure, network weights, and activation functions, respectively. The subscript j and the superscript t are subsequently omitted so as to reduce clutter. In the case of the network structure expressed in (6.1), the corresponding decision vector is given by

$$\boldsymbol{d}_{\boldsymbol{S}} = \begin{bmatrix} s_1^{(1)} & s_2^{(1)} & s_3^{(1)} & s_1^{(2)} & s_2^{(2)} & s_3^{(2)} & s_1^{(3)} & s_2^{(3)} & s_3^{(3)} \end{bmatrix}, \tag{6.8}$$

while in the case of the network weights expressed in (6.2), (6.3), (6.4), and (6.5), the corresponding decision vector is given by

$$\begin{aligned} \boldsymbol{d}_{\boldsymbol{w}} = \big[& w_{10}^{(0)} \; w_{11}^{(0)} \; w_{12}^{(0)} \; w_{13}^{(0)} \; w_{20}^{(0)} \; w_{21}^{(0)} \; w_{22}^{(0)} \; w_{23}^{(0)} \; w_{30}^{(0)} \; w_{31}^{(0)} \; w_{32}^{(0)} \; w_{33}^{(0)} \\ & w_{10}^{(1)} \; w_{11}^{(1)} \; w_{12}^{(1)} \; w_{13}^{(1)} \; w_{20}^{(1)} \; w_{21}^{(1)} \; w_{22}^{(1)} \; w_{23}^{(1)} \; w_{30}^{(1)} \; w_{31}^{(1)} \; w_{32}^{(1)} \; w_{33}^{(1)} \\ & w_{10}^{(2)} \; w_{11}^{(2)} \; w_{12}^{(2)} \; w_{13}^{(2)} \; w_{20}^{(2)} \; w_{21}^{(2)} \; w_{22}^{(2)} \; w_{23}^{(2)} \; w_{30}^{(2)} \; w_{31}^{(2)} \; w_{32}^{(2)} \; w_{33}^{(2)} \\ & w_{10}^{(3)} \; w_{11}^{(3)} \; w_{12}^{(3)} \; w_{13}^{(3)} \; w_{20}^{(3)} \; w_{21}^{(3)} \; w_{22}^{(3)} \; w_{23}^{(3)} \big]. \end{aligned} \tag{6.9}$$

Finally, in the case of the slope variables expressed in (6.6) and (6.7), the corresponding decision vector is given by

$$\begin{aligned} \boldsymbol{d}_{\boldsymbol{G}} = \big[& \alpha_1^{(1)} \; \beta_1^{(1)} \; \alpha_2^{(1)} \; \beta_2^{(1)} \; \alpha_3^{(1)} \; \beta_3^{(1)} \; \alpha_1^{(2)} \; \beta_1^{(2)} \; \alpha_2^{(2)} \\ & \beta_2^{(2)} \; \alpha_3^{(2)} \; \beta_3^{(2)} \; \alpha_1^{(3)} \; \beta_1^{(3)} \; \alpha_2^{(3)} \; \beta_2^{(3)} \; \alpha_3^{(3)} \; \beta_3^{(3)} \big]. \end{aligned} \tag{6.10}$$

The grouping of decision variables within each of these decision vectors is not arbitrary — an intuitive correspondence is present. The grouping of the weights in $\boldsymbol{d_w}$, as well as that of the slope variables in $\boldsymbol{d_G}$, corresponds to the grouping of the structural variables in $\boldsymbol{d_S}$. The structure of the network itself therefore governs the "structure" of the decision vectors. The following conventions are adopted:

- In the case of $\boldsymbol{d_S}$, as expressed in (6.8), structural variables belonging to the same hidden layer are grouped together (*e.g.* structural variables $s_1^{(1)}$, $s_2^{(1)}$, and $s_3^{(1)}$ all belong to the first hidden layer, and therefore they are grouped together). The vector is populated by iterating through each hidden layer.
- In the case of $\boldsymbol{d_w}$, as expressed in (6.9), weights connected to the same neuron in the succeeding layer (hidden or output layer) are grouped together (*e.g.* weights $w_{10}^{(0)}$, $w_{11}^{(0)}$, $w_{12}^{(0)}$, and $w_{13}^{(0)}$ are all connected to the first neuron in the first hidden layer, corresponding to structural variable $s_1^{(1)}$, and so they are grouped together). The vector is populated by iterating through each neuron and each layer (first the hidden neurons and then the output neurons).
- In the case of $\boldsymbol{d_G}$, as expressed in (6.10), slope variables belonging to the same hidden neuron are grouped together (*e.g.* slope variables $\alpha_1^{(1)}$ and $\beta_1^{(1)}$ belong to the first hidden neuron in the first hidden layer, *i.e.* corresponding to structural variable $s_1^{(1)}$, and so they are grouped together). The vector is populated by iterating through each hidden neuron and each hidden layer.

The proposed vector (or string) encoding scheme facilitates the effective application of the sub-algorithms' evolutionary operators so as to evolve networks, *i.e.* generate new solutions. The different grouping conventions adopted within these vectors allow for a more intuitive exposition of the variation operators related to the NSGA-II. In the case of the NSDE and OMOPSO, on the other hand, the variation operators are well suited to (and equipped for) operating on solutions that are in vector format — a matter that is further substantiated and elucidated in the following discourse.

6.1.4 Evolutionary operators

The fundamental working of an EA is governed by its evolutionary operators — selection and variation. Consequently, an essential part of the proposed solution methodology relates directly to the evolutionary operators employed by the NSGA-II, NSDE, and OMOPSO. The aim in this section is to elucidate these operators. The working of each sub-algorithm was discussed in general in §4.2, but a comprehensive description with respect to the evolutionary operators of the NSGA-II was omitted due to their context-dependent nature — a meaningful discussion on these operators presupposes a formulated mathematical model, along with an appropriate solution encoding scheme. Greater focus is therefore placed on the implementation of the NSGA-II. The evolutionary operators of NSDE and OMOPSO, on the other hand, were discussed in sufficient detail, although a brief discussion is included here for clarification purposes.

NSGA-II

Before the variation operators of the NSGA-II are applied, a mating pool is first created, comprising solutions from $\mathcal{P}_t$. A suitable selection for reproduction operator is required so as to

select solutions that should form part of this mating pool. One of the most popular selection operators is *deterministic binary tournament selection* [122, 138]. According to this technique, two random[1] solutions are sampled from $\mathcal{P}_t$, and this is followed by the selection of the best performing solution with respect to their fitness values. During the initialisation procedure of the BOHTA, as presented in Figure 6.2(a), a solution's fitness value is its Pareto rank (determined using the FNSA). As part of the main procedure of the BOHTA, presented in Figure 6.2(b), a solution's fitness is, however, evaluated by means of the crowded comparison operator (as expressed in (2.2)), and so the notions of both Pareto rank and crowding distance are employed. This tournament selection procedure is repeated until enough parent solutions have been selected so that the variation operators can be applied. Note that all three sub-algorithms employ this selection for reproduction operator.

In order for the NSGA-II to produce new (offspring) solutions, the variation operators crossover and mutation are employed. Crossover is a probabilistic mechanism for exchanging genetic information between two (or more) solutions so as to effectively explore recombinations of promising and advantageous structures that have manifested themselves in the current population. A crossover probability, denoted by p_c, accompanies this operator and is typically large — attributable to the fact that crossover is the primary mechanism for variation [161]. According to the evolutionary computation literature [161], the most influential operators are *n-point crossover* (a generalisation of the 1-*point crossover*), *uniform crossover*, *mean-centric recombination* (which includes *intermediate crossover*, *geometric crossover*, *uni-modal normal distribution crossover*, and *simplex crossover*) and *parent-centric recombination* (which includes *simulated binary crossover* and *parent-centric crossover*). Permutation-based crossover operators are disregarded as they do not apply to the BOP of Chapter 5.

The n-point crossover operator, where $n = 2$, is employed abundantly within the context of ANN training and delivers promising performance, as reported by Kitano [81] and Tsai *et al.* [166]. Its adoption in the proposed solution methodology is therefore well warranted. Three distinct crossover operations are performed — one for each decision vector (*i.e.* $\boldsymbol{d_S}$, $\boldsymbol{d_w}$ and $\boldsymbol{d_G}$). The operation itself remains identical over the three classes. According to the 2-point crossover operator, two *cut-points* are randomly selected along the vector encodings of the parent solutions, after which the components between these two points are interchanged between the two parents. A simple example follows, illustrating the working of this crossover operator in respect of the network structure decision vector $\boldsymbol{d_S}$. For the given hypothetical network set-up in §6.1.3, consider the network structures of the two parents (selected using tournament selection) encoded by the decision vectors

$$\boldsymbol{d_S}_{,\text{parent}1} : \big[0\ \ 1\ \ 1\ \mid\ 1\ \ 1\ \mid\ 1\ \ 1\ \ 0\ \ 1\big],$$
$$\boldsymbol{d_S}_{,\text{parent}2} : \big[1\ \ 1\ \ 0\ \mid\ 0\ \ 1\ \mid\ 0\ \ 0\ \ 1\ \ 1\big].$$

It may be observed that the first random cut-point is between the third and fourth components, whereas the second random cut-point is between the fifth and sixth components. The respective substrings (*i.e.* the components between the two cut-points) correspond to the structural variables of the first and second hidden neurons within the second hidden layer, *i.e.* $s_1^{(2)}$ and $s_2^{(2)}$. In order to generate the corresponding offspring solutions (or network structures), the two substrings are interchanged, producing the two offspring encoded by

$$\boldsymbol{d_S}_{,\text{offspring}1} : \big[0\ \ 1\ \ 1\ \mid\ 0\ \ 1\ \mid\ 1\ \ 1\ \ 0\ \ 1\big],$$
$$\boldsymbol{d_S}_{,\text{offspring}2} : \big[1\ \ 1\ \ 0\ \mid\ 1\ \ 1\ \mid\ 0\ \ 0\ \ 1\ \ 1\big].$$

As mentioned earlier, this example relates specifically to the network structure, although the exact same procedure can be applied to the network weights $\boldsymbol{d_w}$ in (6.9) and the activation

[1] In this section, any reference to random selection assumes the use of a uniform distribution.

functions $\boldsymbol{d_G}$ in (6.10). The crossover probability p_c determines the likelihood of performing crossover individually in respect of each decision vector.

The other variation operator is that of mutation, which affects random alterations of a solution's gene values. This introduction of "new ideas" aids the search process in avoiding premature convergence by exploring new regions of the search space. Mutation is also probabilistic, induced by a mutation probability, denoted by p_m. Due to the inherent reliance on serendipity when performing mutation, low probability values are typically associated with this operator — high probabilities can cause a population degradation. Talbi [161] claimed that the nature of the decision variables (*i.e.* binary, discrete, continuous, or permutation-based) aids the decision regarding which mutation operator is most appropriate. In the case of binary decision variables, the *flip operator* is commonly used, *e.g.* a 0 is "flipped" to a 1 (or *vice versa*). When the decision variable is discrete in nature, a typical operator involves changing its value to another value in the allowable range. Continuous decision variables are usually mutated by means of the following operators: *Uniform random mutation*, *normally distributed mutation*, and *polynomial mutation*. Permutation-based mutation operators are also excluded from the discussion.

The flip operator is appropriately employed when mutating the network structure — attributable to the binary nature of the decision variables in $\boldsymbol{d_S}$. A simple example follows, which illustrates the working of this operator. Given the network structure in Figure 6.3 represented by the vector

$$\begin{bmatrix}0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1\end{bmatrix},$$

the flip operator randomly selects a component (hidden neuron) and flips it (activates or inactivates it). Accordingly, an example of this procedure is given by

$$\begin{bmatrix}0 & 1 & 1 & 0 & \underline{1} & 0 & 1 & 0 & 1\end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix}0 & 1 & 1 & 0 & \underline{0} & 0 & 1 & 0 & 1\end{bmatrix},$$

in which the flip corresponds to the second hidden neuron in the second hidden layer, *i.e.* the value of $s_2^{(2)}$ is changed from 0 to 1.

A different approach is necessitated when mutating the network weights. The vast increase in the decision variables pertaining to network weights presumably renders the change of a single weight insignificant, and so a different operator is employed. Montana and Davis [110] reported that within an ANN training context, the following weight mutation operator delivers promising results: A random hidden neuron or output neuron is selected and its incoming weights are mutated, *i.e.* replaced by random values from the relevant distribution (as discussed in §6.1.2). In order to facilitate its application to the proposed modelling approach, a slight modification is made — only active hidden neurons can be selected randomly, not just any hidden neuron. A simple example follows, which illustrates the working of this operator. If the same active hidden neuron (in the example above) is randomly selected to have its incoming weights mutated, then the weights $w_{20}^{(1)}$, $w_{21}^{(1)}$, $w_{22}^{(1)}$, and $w_{23}^{(1)}$ are randomly initialised accordingly. These changes may seem significant, but recall that the hypothetical network is small, comprising only a few weights; networks are typically notably wider, often comprising hundreds (if not thousands) of weights. In such a case, a change in the incoming weights of a single active hidden neuron (or output neuron) is not as significant and conforms to the notion of mutation being responsible for small changes.

The novelty of the modelling approach towards activation functions necessitates a novel approach towards the mutation thereof. The working of the proposed approach, which is similar to the weight mutation operator of Montana and Davis [110], is described as follows: A random active hidden neuron is first selected, which is then followed by the mutation of its corresponding slope variables, *i.e.* the slope variables are replaced by random values from the relevant distribution

(as discussed in §6.1.2). For example, if the same active hidden neuron (in the example above) is randomly selected to have its slope variables mutated, then $\alpha_2^{(2)}$ and $\beta_2^{(2)}$ are randomly initialised accordingly. The mutation probability p_m determines the likelihood of performing mutation individually in respect of the network structure, network weights, and activation functions — similar to crossover.

The discussion above elucidates the mechanisms related to the crossover and mutation operators employed by the NSGA-II so as to generate offspring solutions. Modified versions of existing crossover and mutation operators found in the evolutionary computation literature are employed — a necessity attributable to the novelty of the proposed modelling approach. The mechanisms of these novel operators allow for a versatile approach towards the generation of new and promising networks in respect of all three facets, *i.e.* network structure, network weights, and activation functions.

NSDE

The fundamental working of NSDE, including the mechanisms related to its evolutionary operators, was discussed in sufficient depth in §4.2.2. A brief overview is nevertheless provided here for the sake of completeness. The principal premise of NSDE is based on the notion of using vector differences to perturb a population of decision vectors. The proposed encoding scheme in §6.1.3 therefore provides an appropriate format for this sub-algorithm to perform its respective evolutionary operators effectively, *i.e.* by means of vector algebra. In a relatively recent SOTA survey, Das and Suganthan [26] recommend the use of the scheme $DE/\mathsf{rand}/2/\mathsf{bin}$, as discussed in §4.2.2. The corresponding mutation operator is expressed in (4.8), whereas the relevant crossover and replacement operators are expressed in (4.2) and (4.3), respectively.

The decision vectors related to both the network weights and the activation functions comprise real-valued decision variables, and so the mutation operator in (4.8) can be applied without further ado. The decision vector related to the network structure, on the other hand, is binary-valued, causing complications when performing this mutation operator — the application of (4.8) in respect of $\boldsymbol{d}^t_{\boldsymbol{S},j}$ produces a continuous-valued donor vector $\boldsymbol{v}^{t+1}_{\boldsymbol{S},j}$, thereby violating the binary domain constraint. Engelbrecht and Pampara [40], however, recommend two computationally efficient methods for converting continuous-valued decision variables into binary-valued decision variables in the context of DE. The first method, for which poor results were found during a separate qualitative pilot study performed with respect to the model formulated in Chapter 5, is based on the normalisation of the donor vector, *i.e.* linearly scaling its values to the range $[0, 1]$, which is followed by a probabilistic transformation to binary values. The second method, known as *binDE*, delivered superior results during the same pilot study and is, as a result, employed in this dissertation. Accordingly, the mutation operator (4.8) is applied in respect of $\boldsymbol{d}^t_{\boldsymbol{S},j}$ and produces a continuous-valued donor vector $\boldsymbol{v}^{t+1}_{\boldsymbol{S},j}$, which is then converted by means of the conversion process

$$v^{t+1}_{\boldsymbol{S},j,i} = \begin{cases} 1, & \text{if } \mathsf{rand}_i[0,1] < s(v^{t+1}_{\boldsymbol{S},j,i}), \qquad i \in \{1,\ldots,hm\}, \\ 0, & \text{otherwise,} \end{cases} \tag{6.11}$$

where $v^{t+1}_{\boldsymbol{S},j,i}$ denotes the structural variable $i \in \{1,\ldots,hm\}$ in the donor vector $j \in \{1,\ldots,M\}$. Furthermore, $s(v^{t+1}_{\boldsymbol{S},j,i})$ denotes the sigmoid function

$$s(v^{t+1}_{\boldsymbol{S},j,i}) = \frac{1}{1 + e^{(-v^{t+1}_{\boldsymbol{S},j,i})}},$$

which scales the decision variable to the range $[0, 1]$. This conversion method circumvents the problem encountered when applying the mutation operator (4.8) in respect of the network structure decision vector $\boldsymbol{d}^t_{\boldsymbol{S},j}$. The crossover operator (4.2) and replacement operators (4.3) can now be applied to generate the binary-valued trial vector $\boldsymbol{u}^{t+1}_{\boldsymbol{S},j}$ and binary-valued decision vector $\boldsymbol{d}^{t+1}_{\boldsymbol{S},j}$, respectively.

OMOPSO

The fundamental working of OMOPSO, including the mechanisms related to its evolutionary operators, was also discussed in sufficient detail in §4.2.3. A brief overview is nevertheless also provided here for the sake of completeness. The underlying notion of OMOPSO relates to the sharing of information between particles in a swarm, analogous to solutions (represented by decision vectors) in a population. The flight operators responsible for updating the position and velocity of the particles, as expressed in (4.9) and (4.10), respectively, are well suited to operating on solutions that are in vector format. The proposed solution scheme in §6.1.3 is once again well warranted as it allows for the effective application of these operators (*i.e.* also by means of vector algebra).

The decision vector related to the network structure presents the same challenge as in the case of NSDE. The operations in (4.9) and (4.10) produce a continuous-valued position (decision) vector $\boldsymbol{d}^{t+1}_{\boldsymbol{S},j}$, consequently violating the domain constraint. Fortunately, Kennedy and Eberhart [76] devised a method for circumventing this problem (in the context of PSO), which is called *binPSO*. The binDE method described in (6.11) is, in actual fact, based on this method and performs the same function. This method also delivered promising results during a separate qualitative pilot study performed with respect to the model formulated in Chapter 5, and so it is subsequently employed in this dissertation. Accordingly, the flight operator in (4.10) is first applied in respect of $\boldsymbol{d}^t_{\boldsymbol{S},j}$ and produces the continuous-valued velocity update $\boldsymbol{v}^{t+1}_{\boldsymbol{S},j}$. This is followed by the application of the flight operator in (4.9), which produces the continuous-valued position vector $\boldsymbol{d}^{t+1}_{\boldsymbol{S},j}$. This vector is converted by means of the conversion process

$$d^{t+1}_{\boldsymbol{S},j,i} = \begin{cases} 1, & \text{if } \mathsf{rand}_i[0,1] < s(d^{t+1}_{\boldsymbol{S},j,i}), \qquad i \in \{1,\ldots,hm\}, \\ 0, & \text{otherwise,} \end{cases} \tag{6.12}$$

where $d^{t+1}_{\boldsymbol{S},j,i}$ denotes the structural variable $i \in \{1,\ldots,hm\}$ in the position vector $j \in \{1,\ldots,M\}$. Again, $s(d^{t+1}_{\boldsymbol{S},j,i})$ denotes the sigmoid function

$$s(d^{t+1}_{\boldsymbol{S},j,i}) = \frac{1}{1+e^{(-d^{t+1}_{\boldsymbol{S},j,i})}}.$$

The conversion expressed in (6.12) circumvents the problems encountered when applying the flight operators (4.9) and (4.10) in the context of the network structure decision vector $\boldsymbol{d}^t_{\boldsymbol{S},j}$. Consequently, the binary-valued decision vector $\boldsymbol{d}^{t+1}_{\boldsymbol{S},j}$ is produced, which adheres to the domain constraint.

6.2 The test suite

In order to demonstrate the capabilities of the proposed BOHTA and gain pertinent insight into the dynamics of its sub-algorithms, a test suite of data sets was selected. Multiple data sets constitute this test suite, each of which induces a different problem instance of the BOP

formulated in §5.3 for which the BOHTA is tasked with finding networks that exhibit favourable performance with respect to memorisation and generalisation — *i.e.* performance in respect of the training data set and the testing data set, respectively. Furthermore, performance in respect of the validation data set forms a key part of both the stopping criterion and mitigation of overfitting — *i.e.* early stopping. Accordingly, each data set contained within the test suite comprises a training set $(\mathcal{X}, \mathcal{Y})$, a validation set $(\mathcal{X}^\dagger, \mathcal{Y}^\dagger)$ and a testing set $(\mathcal{X}^\ddagger, \mathcal{Y}^\ddagger)$, as described in §3.5.5 and illustrated graphically in Figure 5.6. Each data set therefore corresponds to three different sets of ordered pairs $(\mathcal{X}, \mathcal{Y})$, $(\mathcal{X}^\dagger, \mathcal{Y}^\dagger)$, and $(\mathcal{X}^\ddagger, \mathcal{Y}^\ddagger)$ for which the BOHTA is tasked with finding networks that best approximate the functional mapping from the presented input to the corresponding output.

The aim in this section is to provide a high-level overview of the data sets that constitute the test suite, thereby providing the reader with the necessary context and background information. Various generic, statistical, and information theoretic meta-features discussed in §3.9 are used to facilitate the exposition. A total of forty-nine diverse data sets, specific to classification problems, were selected from the literature. Regression problems were excluded from the test suite, as per the scope delimitations in §1.3. Recall from §3.6 that a data set comprises a certain number of input-output observations of the form $\{(\boldsymbol{x}^1, \boldsymbol{y}^1), (\boldsymbol{x}^2, \boldsymbol{y}^2), (\boldsymbol{x}^3, \boldsymbol{y}^3), \ldots\}$, where the input vector representing the independent variables or input features has, for example, the form $\boldsymbol{x}^1 = [x_1^1 \cdots x_i^1 \cdots x_n^1]$, and the target vector representing the dependent variables or target variables has, for example, the form $\boldsymbol{y}^1 = [y_1^1 \cdots y_k^1 \cdots y_o^1]$.

A summary of the generic meta-features (discussed in §3.9.1) pertaining to the data sets in the test suite, together with each corresponding domain, is presented in Table 6.1. The classification task can either be a binary classification problem or a multi-class classification problem. In the former case, only two classes are predicted, whereas in the latter case, three or more classes are predicted. The input data $\mathcal{X}$ comprise values for numerical and/or categorical independent variables. Numerical variables are continuous or discrete-valued, whereas categorical variables are nominal-valued or ordinal-valued. The criterion related to the nature of the output data $\mathcal{Y}$ is omitted, because of the fact that strictly classification problems are considered, and so all dependent variables are categorical.

TABLE 6.1: *A summary of the generic meta-features pertaining to the data sets in the test suite and their corresponding domains.*

Meta-feature	Domain
Classification task	{binary, multi-class}
Data set size	$\mathbb{N}$
Number of independent variables n	$\mathbb{N}$
Number of dependent variables o	$\mathbb{N}$
Nature of the input data	{num., cat., num. & cat.}
Number of numerical input features $n_{\text{num.}}$	$\mathbb{N}$
Number of categorical input features $n_{\text{cat.}}$	$\mathbb{N}$
Output-input ratio $o : n$	$\mathbb{R}$
Dimensionality $(n + o)$/data set size	$\mathbb{R}$

In addition, two statistical meta-features are considered. They are: Kurtosis and the Fisher-Pearson coefficient of skewness, as discussed in §3.9.2. Kurtosis reflects the extent to which the tails of the input data deviate (or differ) from those of a normal distribution. The skewness coefficient, on the other hand, is a measure of symmetry (or lack thereof) present within the data. The minimum, mean, and maximum kurtosis and skewness coefficient values are determined for

the input features (*i.e.* independent variables) constituting the data set. The final meta-feature considered stems from the field of information theory, *i.e.* entropy, and is a measure of uncertainty within a variable's possible outcomes. In the context of this dissertation, the mean entropy with respect to the target (dependent) variable is calculated. The domain corresponding to both the statistical and information theoretic meta-features correspond to the natural numbers domain $\mathbb{N}$.

The generic, statistical, and information theoretic features serve as diversity criteria which guide the data set selection process so as to ensure that the test suite employed in this dissertation is sufficiently diverse, which further facilitates an insightful and thorough analysis of the BOHTA performance. The meta-features are also central to the algorithmic performance prediction study conducted later in this dissertation. A summary of the test suite values corresponding to the generic meta-features is presented in Table 6.2, whereas a summary of the test suite values corresponding to the statistical and information theoretic meta-features is presented in Table 6.3. For the sake of brevity the name of each data set has been replaced by an abbreviated code, according to which, for example, "C1" refers to the first data set in the test suite. The original data set names that correspond to each of the codes can be found in Appendix A. The data sets were obtained from various sources in the literature [27, 30, 33, 120, 121].

6.2.1 Data pre-processing

Data pre-processing was performed using the *Orange: Data Mining Toolbox in Python* [30] and comprised two steps. The first pre-processing step is called *one-hot encoding* and was performed in respect of all nominal independent variables that are incorrectly recorded as ordinal — a consequence of the format in which the data are recorded. For each possible value that these nominal variables can assume, a corresponding binary variable was created. One-hot encoding prevents the training algorithm from being influenced or biased by incorrectly recorded ordinal variables. A consequence, however, is the potential introduction of collinearity in the input data, which can be addressed simply by removing one of the newly created binary variables in respect of each of the nominal variables. The categorical dependent variables were also one-hot encoded so as to facilitate use of the softmax layer (as shown in Figure 5.4). A binary variable was therefore associated with each possible dependent variable value. Consequently, the vector of probabilities produced by the softmax layer can now be compared directly from the corresponding binary-valued variables. Collinearity in the output data was not found to be a problem, and so no binary variables were removed.

The second pre-processing step involved the standardisation of the input data by rescaling each input feature to have a mean of zero and a standard deviation of one. This prevents the training algorithm from being biased by any input features that have significantly larger variances than others. The output data were not standardised as the one-hot encoding step already ensures that there are equal variances.

6.2.2 Random data sampling

Given the respective class distributions of the 49 data sets, an important issue related to random data sampling warrants further discussion. Recall that the main objective function $h_1(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$ is evaluated in respect of a random mini-batch $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$, where $(\hat{\mathcal{X}}, \hat{\mathcal{Y}}) \subset (\mathcal{X}, \mathcal{Y})$. The performance of the BOHTA can therefore greatly depend on the random sampling of $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$. If the BOHTA consistently samples mini-batches that are inaccurate reflections of $(\mathcal{X}, \mathcal{Y})$, then the algorithm will struggle to identify solutions (or networks) that best approximate the true underlying functional representation as it would be inhibited by a blatant lack of information. From the perspective

TABLE 6.2: *The values of the generic meta-features of the test suite.*

Data set	Task	Size	n	o	Input nature	$n_{\text{num.}}$	$n_{\text{cat.}}$	$o:n$	$\frac{n+o}{\text{size}}$
C1	binary	30 162	97	2	num./cat.	6	91	0.021	0.006
C2	binary	1 470	30	2	num./cat.	11	19	0.067	0.041
C3	binary	775	25	2	num.	25	0	0.080	0.065
C4	multi-class	186	79	3	num.	79	0	0.038	1.274
C5	binary	4 119	52	2	num./cat.	9	43	0.038	0.025
C6	binary	1 372	4	2	num.	4	0	0.500	0.006
C7	binary	116	9	2	num.	9	0	0.222	0.155
C8	binary	569	30	2	num.	30	0	0.067	0.105
C9	multi-class	106	9	6	num.	9	0	0.667	0.509
C10	multi-class	1 728	15	4	cat.	0	15	0.267	0.035
C11	binary	232	16	2	cat.	0	16	0.125	0.138
C12	multi-class	10 845	28	6	num.	28	0	0.214	0.015
C13	multi-class	358	155	6	cat.	0	155	0.039	2.598
C14	binary	496	152	2	num./cat.	1	151	0.013	0.613
C15	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C16	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C17	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C18	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C19	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C20	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C21	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C22	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C23	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C24	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C25	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C26	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C27	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C28	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C29	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C30	multi-class	1 885	12	7	num.	12	0	0.583	0.045
C31	binary	10 000	13	2	num.	13	0	0.154	0.003
C32	multi-class	194	61	6	num./cat.	10	51	0.098	1.887
C33	multi-class	523	27	4	num.	27	0	0.148	0.207
C34	binary	579	10	2	num./cat.	9	1	0.200	0.035
C35	multi-class	150	4	3	num.	4	0	0.750	0.080
C36	binary	1 163	20	2	num./cat.	13	7	0.100	0.034
C37	multi-class	731	234	4	num.	234	0	0.017	1.280
C38	multi-class	148	41	4	cat.	0	41	0.098	1.108
C39	binary	5 936	94	2	cat.	0	94	0.021	0.032
C40	multi-class	12 960	19	5	cat.	0	19	0.263	0.007
C41	binary	182	12	2	num.	12	0	0.167	0.132
C42	binary	1 055	41	2	num./cat.	38	3	0.049	0.078
C43	binary	250	12	2	cat.	0	12	0.167	0.096
C44	multi-class	210	7	3	num.	7	0	0.429	0.100
C45	binary	143	6	2	num.	6	0	0.333	0.084
C46	binary	2 201	5	2	cat.	0	5	0.400	0.005
C47	multi-class	846	18	4	num.	18	0	0.222	0.085
C48	multi-class	178	13	3	num.	13	0	0.231	0.219
C49	multi-class	101	16	7	num./cat.	0	16	0.438	1.109

TABLE 6.3: *The values of the statistical and information theoretic meta-features of the test suite.*

Data set	$\min(\xi)$	$\bar{\xi}$	$\max(\xi)$	$\min(\zeta)$	$\bar{\zeta}$	$\max(\zeta)$	$\bar{H}$
C1	−1.437	14.768	153.666	−0.751	2.106	11.903	0.81
C2	−1.835	0.24	3.936	−1.439	0.584	1.984	0.637
C3	−1.814	115.254	507.603	0.000	7.218	20.526	0.967
C4	−0.823	1.664	35.626	−5.058	−0.066	3.173	1.169
C5	−1.948	3.229	25.285	−1.076	0.644	4.023	0.498
C6	−1.953	−0.275	1.270	−1.022	−0.051	1.089	0.991
C7	−1.99	4.704	17.591	−0.211	1.609	3.812	0.992
C8	−1.727	7.507	49.209	−0.528	1.667	5.447	0.953
C9	−0.229	10.263	63.335	0.891	2.286	7.27	2.565
C10	−1.501	−1.431	−1.360	0.000	0.000	0.000	1.206
C11	−2.015	−1.738	0.663	−1.630	−0.112	0.657	0.997
C12	−1.485	−0.330	5.062	−2.423	0.205	1.328	1.468
C13	−1.406	2.352	22.845	−0.380	1.372	4.720	0.538
C14	−1.895	−0.353	2.340	−1.609	−0.024	0.973	0.980
C15	−2.002	4.444	58.781	−0.386	0.217	2.572	2.174
C16	−2.002	4.444	58.781	−0.386	0.217	2.572	2.157
C17	−2.002	4.444	58.781	−0.386	0.217	2.572	1.456
C18	−2.002	4.444	58.781	−0.386	0.217	2.572	2.152
C19	−2.002	4.444	58.781	−0.386	0.217	2.572	1.330
C20	−2.002	4.444	58.781	−0.386	0.217	2.572	2.687
C21	−2.002	4.444	58.781	−0.386	0.217	2.572	1.776
C22	−2.002	4.444	58.781	−0.386	0.217	2.572	1.981
C23	−2.002	4.444	58.781	−0.386	0.217	2.572	0.837
C24	−2.002	4.444	58.781	−0.386	0.217	2.572	2.036
C25	−2.002	4.444	58.781	−0.386	0.217	2.572	0.942
C26	−2.002	4.444	58.781	−0.386	0.217	2.572	1.186
C27	−2.002	4.444	58.781	−0.386	0.217	2.572	1.955
C28	−2.002	4.444	58.781	−0.386	0.217	2.572	1.384
C29	−2.002	4.444	58.781	−0.386	0.217	2.572	2.560
C30	−2.002	4.444	58.781	−0.386	0.217	2.572	1.196
C31	−1.200	−1.115	−0.389	−0.013	0.000	0.019	0.944
C32	−2.019	10.625	82.048	−1.425	1.902	8.57	2.489
C33	−0.199	5.731	19.765	−3.189	0.184	3.235	1.903
C34	−1.091	24.265	149.939	−1.209	2.659	10.512	0.862
C35	−1.402	−0.751	0.291	−0.274	0.067	0.334	1.585
C36	−2.000	122.955	900.954	−0.654	5.738	28.638	0.977
C37	−1.126	−0.241	2.061	−0.569	0.302	1.239	1.956
C38	−2.027	2.325	29.749	−1.443	0.737	5.442	1.228
C39	−1.988	48.172	737.624	−15.635	0.487	27.191	0.947
C40	−2.000	−1.523	−1.3.00	0.000	0.000	0.000	1.716
C41	−1.097	0.413	1.833	−0.642	−0.077	0.957	0.863
C42	−0.222	44.713	812.724	−1.762	3.799	26.846	0.922
C43	−1.452	−1.225	−1.070	−0.238	−0.04	0.103	0.985
C44	−1.507	−0.829	−0.067	−0.538	0.234	0.562	1.585
C45	−2.004	−0.069	1.392	−1.063	−0.452	0.285	0.996
C46	−1.686	4.518	15.282	−1.399	0.978	4.156	0.908
C47	−0.978	5.153	58.375	−0.226	1.042	6.778	1.999
C48	−1.323	−0.088	2.105	−0.307	0.333	1.098	1.567
C49	−1.990	−0.063	8.169	−1.707	0.401	3.163	2.391

of the training algorithm, the random mini-batch $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$ ought always be a true (or close to true) reflection of $(\mathcal{X}, \mathcal{Y})$ — and this will not be the case if training examples are just randomly sampled from the training set. Consequently, the BOHTA incorporates a random sampling approach that is more sophisticated than mere random sampling. During each evaluation of the main objective function, the random mini-batch is sampled in such a way that it accurately reflects the class distribution of the entire data set. For example, during every generation in the case of C48 (*i.e.* the Wine data set), $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$ is randomly sampled such that it reflects the class distribution of $(\mathcal{X}, \mathcal{Y})$, *i.e.* approximately 33.15% of Class-0, 39.89% of Class-1, and 26.97% of Class-2. The random mini-batch $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$ will therefore contain approximately 33.15% Class-0s, 39.89% Class-1s, and 26.97% Class-2s. In this way, the random mini-batch is more consistently a true reflection of the entire data set (or close to it), providing the BOHTA with an accurate measure of network performance and, as a result, guiding it towards discovering more promising networks.

6.3 Chapter summary

In this chapter, the solution methodology proposed in this dissertation for solving instances of the BOP in Chapter 5 was elucidated comprehensively. The chapter comprised two main sections, the first of which focussed on the most salient features of the BOHTA approach and was presented in §6.1. In particular, an overview was presented of the fundamental working of the BOHTA in §6.1.1, which focussed on its foundational source of inspiration (*i.e.* AMALGAM), as well as the key differences between these two approaches. This was followed in §6.1.2 by an exposition of the different initialisation procedures employed by the BOHTA in respect of the network structure, network weights, and activation functions. Next, the proposed encoding scheme was presented in §6.1.3, which served as a necessary precursor to an elucidation of the evolutionary operators employed by the NSGA-II, NSDE, and OMOPSO, as presented in §6.1.4.

The second main section focussed on the test suite selected for assessing the proposed solution methodology and was presented in §6.2. The aim of this test suite is to demonstrate the capabilities of the BOHTA and draw pertinent insight from its subsequent results in respect of the data. Accordingly, a diverse suite of data sets was selected. A discussion on the various generic, statistical, and information theoretic meta-features of the data sets in the test suite was presented which facilitated a description (on a high-level of abstraction) of the constituent data sets. This was followed in §6.2.1 by a discussion on the necessary data pre-processing steps, namely one-hot encoding and standardisation. Finally, an important matter related to the random sampling of data was discussed in §6.2.2.

Part III

Algorithmic Evaluation

CHAPTER 7

Algorithmic Parameter Evaluation

Contents

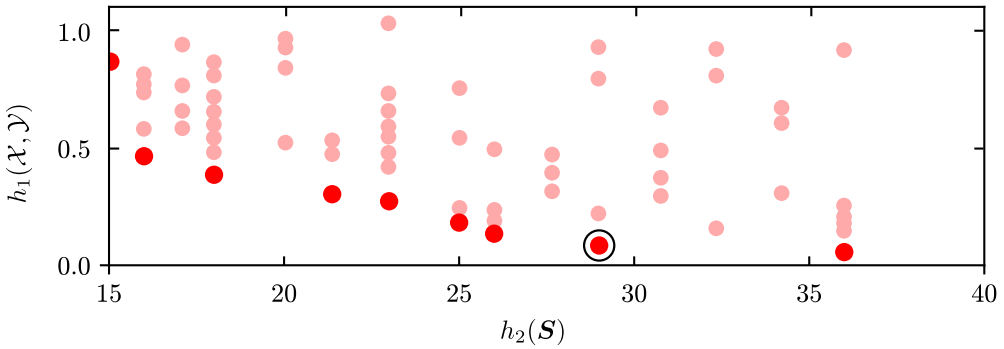
In this chapter, three algorithmic parameter evaluations are performed, the first of which focusses on the three sub-algorithms employed in the BOHTA, the second on the BOHTA itself, and the third on the gradient-based training algorithms. The chapter opens with a discussion on the method by which the performance of a training algorithm is assessed — a necessary precursor to the parameter evaluations. This is followed by a description of the experimental setup adopted, which includes a discussion on the parameters that are fixed throughout the subsequent experiments. The three algorithmic parameter evaluations then follow. The first of these is aimed at establishing good parameter values for the NSGA-II, NSDE, and OMOPSO. Next, good parameter values for the BOHTA are established. The final parameter evaluation focusses on finding good parameter values for SGD, RMSProp, and Adam. The chapter finally closes with a brief summary of the work included in the chapter.

7.1 Algorithmic performance assessment

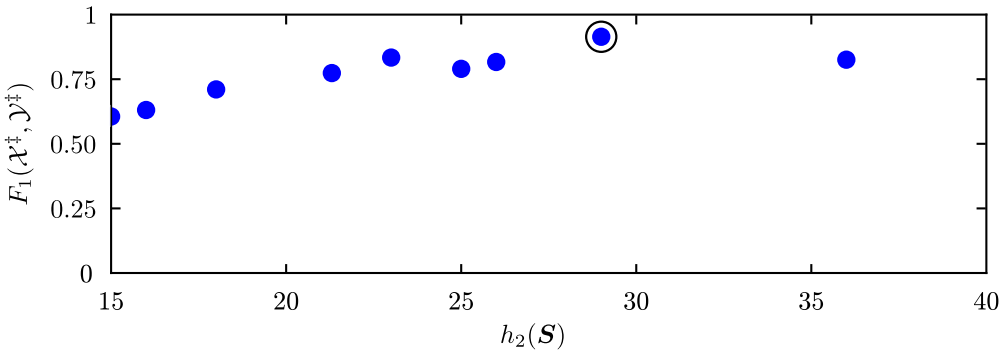
An important precursor to the parameter evaluation carried out in this chapter is a discussion on how the performance of a training algorithm is assessed. After each training run[1], a population of solutions, together with its corresponding approximate Pareto front, is returned by the training algorithm, as illustrated in the example presented in Figure 7.1(a). Although these solutions have been assessed in respect of the *entire* training set — from which the training performance $h_1(\mathcal{X}, \mathcal{Y})$ is obtained — a final unbiased performance estimation is required. The following approach is adopted in order to assess the performance of a training algorithm. The approximate Pareto front obtained at the end of the training run, as illustrated in Figure 7.1(a), is subsequently

[1] The length of such a run is determined by means of early stopping.

evaluated in respect of the independent test data set. Figure 7.1(b) illustrates the performance achieved by the solutions in the approximate Pareto front. The final performance of a training algorithm is assessed in terms of the best performing solution in the approximate Pareto front. This solution corresponds to the solution with the largest F_1-score, which is represented by the encircled solution in Figure 7.1. Evaluating the testing performance of the approximate Pareto front specifically in terms of the F_1-score is motivated by its greater prevalence in literature (relatively speaking), as discussed in §3.6. Recall from §5.3.1, that the main objective function in (5.13) is (to the best of the author's knowledge) novel. Therefore, reporting only on $h_1(\mathcal{X}^\ddagger, \mathcal{Y}^\ddagger)$ values inhibits the (future) dissemination of the findings in this dissertation — more specifically, the findings pertaining to the algorithmic performance results. The well-established reputation of the F_1-score performance metric provides sufficient justification for its selection as the final (unbiased) performance measure.



(a) Training performance of the population after the BOHTA terminates



(b) Testing performance of the final approximation front

FIGURE 7.1: *Performance assessment of the solutions generated by a training algorithm, where (a) represents the performance of solutions in the final population, in respect of the training set. Bright red circles represent solutions in the approximate Pareto front, whereas the semi-transparent red circles represent the dominated solutions. The vertical and horizontal axes indicate the main objective function in* (5.13) *evaluated with respect to the entire training set, and the regularising objective function in* (5.15)*, respectively. The final unbiased performance evaluation is presented in (b), where only the solutions in the approximate Pareto front are evaluated. The vertical axis here represents the F_1-score achieved, whereas the horizontal axis remains unchanged. The encircled solution is regarded as the best solution generated by the training algorithm as it achieves the largest F_1-score.*

Each training algorithm is, however, stochastic in nature, attributed to two factors: First, the fact that each algorithm starts with a population of randomly initialised solutions (*i.e.* random network structure, network weights, and activation functions) and, secondly, due to the probabilistic operation of the evolutionary operators (*e.g.* crossover, mutation, selection for reproduction, and

selection for replacement). As a result of this stochasticity, different approximation fronts are returned by an algorithm if it is applied multiple times to the same data set. A more representative indication of the average algorithmic performance and the variability in this performance can be obtained by executing multiple optimisation runs. By employing a fixed set of different random number generator seeds, along with a fixed set of random initial solutions for each data set, so-called *matched samples* are obtained. Appropriate statistical tests are then performed to determine, at a 5% level of statistical significance, whether there is a significant difference between any two (or more) samples under investigation, as discussed in §2.2. Box plots, also known as box-whisker plots, are presented to supplement the analyses and convey more information than the mere use of sample mean and standard deviation information [86]. The mean (indicated by a diamond), the median, the inter-quartile range, as well as the sample minimum and the sample maximum, are illustrated visually in box plots, and thus collectively provide a more comprehensive representation of the central tendency and spread of the data samples.

7.2 Experimental setup

There are a number of parameters for which values are determined *a priori* and which remain fixed during the subsequent experiments and analyses. The reasoning behind fixing these parameters beforehand is twofold: First, it limits the size of the experimental design, and secondly, these parameter choices are corroborated by empirical findings in the literature. The first of these parameters relate to the network structure. The mathematical model of Chapter 5 allows for a variable network size, *i.e.* a network that can change in terms of both width (number of active hidden neurons) and depth (number of non-redundant hidden layers), as discussed in §5.2.2. The parameters m and h determine an upper bound on the network size with respect to maximum width and maximum depth, respectively. Consequently, appropriate values for m and h afford enough computational capability to the network whilst limiting the computational burden experienced by the training algorithm. An unnecessarily large upper bound would most likely result in excessively long computation times (to reach convergence) or poor generalisation performance due to over-parameterised networks. According to one of the most commonly relied upon heuristics, the optimal size of a hidden layer is usually between the size of the input layer and that of the output layer [60], as mentioned in §3.1, (*i.e.* $m \in \{o, o+1, \ldots, n\}$ if the output layer is smaller than the input layer, or $m \in \{n, n+1, \ldots, o\}$ if the output layer is larger than the input layer). A conservative approach is followed when deciding on an appropriate upper bound value for m. Instead of assigning a value that is between the input- and output layer sizes (as in the case of the aforementioned heuristic), m is assigned a value twice the size of the maximum of these two values, *i.e.* $m = 2\max\{n, o\}$. This conservative approach allows for consideration of a sufficiently diverse set of possible network structures (both smaller than and larger than specified by the heuristic) which can be explored during the search process, but not so diverse such that an optimisation run is excessively long. Overfitting can, however, be a problem when networks comprise too many parameters. The regularising objective in (5.15) helps mitigate this problem by guiding the search process and favouring smaller networks. In addition, the early stopping criterion further aids in the mitigation of overfitting.

Recall from §3.1 that networks comprising two hidden layers are capable of approximating an arbitrary non-linear function and generating any complex decision region. When taking this and the relatively simple[2] nature of the data sets into account, the proposed experimental design limits network structures to three hidden layers, *i.e.* $h = 3$. The BOHTA can therefore produce

[2]The test suite comprises relatively simple classification problems. Complex problems within the domains of natural language processing or machine vision necessitate deep networks, comprising many more hidden layers, but are excluded from the current research scope.

networks comprising only one hidden layer more than prescribed by conventional wisdom. An investigation into the structural attributes of favourable networks produced by the BOHTA is performed later in this dissertation so as to provide insight into the number of hidden layers (and hidden neurons) that the BOHTA prescribes for the supervised learning task at hand — a (possibly revelatory) juxtaposition between conventional wisdom and BOHTA recommendations warrants further consideration. Three hidden layers may, arguably, restrain computational capability — especially so when compared with deep networks which comprise many more hidden layers. It is, however, conjectured that this potential computational deficit is mitigated by the conservative approach adopted with respect to the network width.

Recall from §5.3.1 that the training algorithm evaluates performance, *i.e.* the value of the main objective function in (5.13), in respect of a mini-batch of random training examples, denoted by $(\hat{\mathcal{X}}, \hat{\mathcal{Y}})$. The size of this random mini-batch, referred to as the batch size P, is another parameter that is determined beforehand and fixed throughout. In most cases $P \in \{8, 16, 32, \ldots\}$ (as discussed in §3.7.3) and the upper bound on P is problem-dependent. Due to the computationally expensive nature of a population-based metaheuristic solution approach, however, a batch size of 32 is chosen for mitigation purposes (*i.e.* to reduce the computational burden). A separate qualitative pilot study was carried out in which it was found that a batch size of $P = 32$ delivered promising results within a reasonable computation time, in respect of all the data sets.

7.3 Parameter evaluation

In this section, three algorithmic parameter evaluations are performed. The purpose of the first parameter evaluation is to fine-tune the performance of the three BOHTA sub-algorithms, namely the NSGA-II, NSDE, and OMOPSO. The aim in the second parameter evaluation is to find good parameter values for the BOHTA itself, which employs the sub-algorithms in conjunction with their suitable algorithmic parameter values. The third (and final) parameter evaluation focusses on the gradient-based algorithms under consideration — namely SGD, Adam, and RMSProp — and finding suitable values for their parameters.

The parameter values employed in current implementations of the NSGA-II, NSDE, and OMOPSO in the context of training of FNNs are arguably not suitable for (or capable of) providing good solutions to instances of the mathematical model of Chapter 5. There is a fundamental difference between the level of abstraction at which the optimisation problem of Chapter 5 is formulated and the approaches typically adopted in the literature. The novelty of both the problem at hand — represented by the mathematical model — and the proposed solution methodology (discussed in Chapter 6) necessitates the attainment of good sub-algorithm parameter values first, before the sub-algorithms may be employed by the BOHTA. In this way, the best computational capability is afforded to the sub-algorithms and, consequently, to the BOHTA itself. This approach facilitates the research aim of designing a training algorithm that raises the level of general applicability and delivers improved performance.

The two main parameters of the BOHTA which form the focal point of the second parameter evaluation are the minimum number of solutions a sub-algorithm can generate and the overall population size. Vrugt and Robinson [171] neither provided any justification for their selection of values for these parameters nor did they elucidate the impact that these parameter values have on algorithmic performance. The uncertainty surrounding these parameters is addressed in the second parameter evaluation.

The gradient-based training algorithms (*i.e.* SGD, RMSProp, and Adam) form the basis of the third parameter evaluation and also have a number of important parameters. Although these

training algorithms are well-established within the context of training FNNs, as stated in §3.7.3, an extensive parameter evaluation is still conducted for the sake of completeness. The BOHTA's sub-algorithms and the BOHTA itself are afforded the best computational capacity by assigning the parameters of the gradient-based training algorithms appropriate values. The gradient-based training algorithms ought therefore also to be subjected to the same level of scrutiny. The main parameters under investigation for the gradient-based training algorithms relate to the respective learning rates, exponential decay rates, and momentum.

All three parameter evaluations are performed in respect of the data sets C1–C40 in the test suite, described in §6.2. Data sets C41–C49 are excluded from the parameter evaluations and only used later in this dissertation for further validation of the robustness (*i.e.* meta-generalisation capabilities) of the BOHTA. Each parameter evaluation comprises thirty optimisation runs per data set. An extensive measure of "explicit" performance per data set is provided, facilitating an in-depth performance analysis. The findings of these evaluations allow for a more elucidating discourse when comparing and analysing the performance of the training algorithms. Statistical analyses are conducted so as to determine, at a 5% level of significance, the number of data sets for which each combination of parameter values outperforms all other parameter combinations[3].

7.3.1 Sub-algorithm parameter evaluation

For each sub-algorithm a good parameter combination is determined by means of a sensitivity analysis. In the case of the NSGA-II, the algorithm-specific parameters are the crossover probability p_c and the mutation probability p_m. In the case of NSDE, the relevant parameters are the crossover rate C_R and the amplification factor F. Lastly, in the case of OMOPSO, the relevant parameters are the inertia weight W, the learning factors C_1 and C_2, and the mutation probability p_m. For each of these parameters, three values, judged to be of a small, medium, and large magnitude, are considered. Multiple studies have been carried out with a view to acquire reasonably wide ranges for these parameter values. A summary of these parameter values is presented in Table 7.1.

TABLE 7.1: *Different parameter values, judged to be of a small, medium, and large magnitude, evaluated as part of the sub-algorithm parameter evaluation. Sources in the literature are included.*

		Parameter value			
Sub-algorithm	Parameter	Small	Medium	Large	References
NSGA-II	p_c	0.60	0.90	1.00	[81, 106, 171]
	p_m	0.02	0.05	0.10	[81, 110, 166]
NSDE	C_R	0.10	0.30	0.90	[70, 156]
	F	[0.20, 0.60]	[0.50, 1.00]	[0.60, 1.40]	[26, 156, 171]
OMOPSO	W	[0.10, 0.50]	1.00	[0.50, 1.50]	[70]
	C_1	1.00	1.50	2.00	[57, 70, 171]
	C_2	1.00	1.50	2.00	[57, 70, 171]
	p_m	0.02	0.05	0.10	[81, 110, 166]

The "general-purpose" parameter *population size*, which is common amongst each of the respective sub-algorithms, has been omitted from Table 7.1 for the sake of brevity. The inclusion of the population size parameter (in the parameter evaluation) aids the process of choosing an appropriate population size for the BOHTA implementation. Based on findings in the literature [70, 78, 110, 171], as well as a separate qualitative pilot study carried out by the author, three

[3]The term "parameter combination" is henceforth used to refer to a combination of parameter values.

different values, judged to be of a small, medium, and large magnitude, were settled upon. These values are 30, 50, and 100. Population size is denoted by N^i, for sub-algorithm $i \in \{G, D, P\}$, where G, D, and P denote the NSGA-II, NSDE, and OMOPSO, respectively. Recall from §6.1.1 that the BOHTA employs the notion of early stopping as termination criterion. The sub-algorithms also employ early stopping, ensuring a fair comparison between their separate algorithmic performances and that of the BOHTA. Consequently, the other "general-purpose" parameter typically considered within this context, *i.e. maximum number of generations*, is excluded from this parameter evaluation, because early stopping determines the maximum number of generations implicitly, as discussed in §6.1.1.

The different parameter combinations for the NSGA-II and NSDE are presented in Table 7.2, while in the case of OMOPSO, the different parameter combinations are presented in 7.3. The baseline of the sensitivity analysis represents the parameter combination in which each parameter is assigned its medium-magnitude value, *e.g.* in the case of the NSGA-II, the baseline is G1 (first row), comprising the parameter values $N^G = 50$, $p_c = 0.90$, and $p_m = 0.05$. For each of the other possible combinations, a single parameter is varied within its magnitude range, *i.e.* changed to either small or large while keeping the values of the other parameters at their medium-magnitude levels. The sensitivity of parameter value changes with respect to the baseline is therefore evaluated.

TABLE 7.2: *The different NSGA-II and NSDE parameter combinations.*

NSGA-II				**NSDE**			
Combination	N^G	p_c	p_m	Combination	N^D	C_R	F
G1	50	0.90	0.05	D1	50	0.30	[0.50, 1.00]
G2	30	0.90	0.05	D2	30	0.30	[0.50, 1.00]
G3	100	0.90	0.05	D3	100	0.30	[0.50, 1.00]
G4	50	0.60	0.05	D4	50	0.10	[0.50, 1.00]
G5	50	1.00	0.05	D5	50	0.90	[0.50, 1.00]
G6	50	0.90	0.02	D6	50	0.30	[0.20, 0.60]
G7	50	0.90	0.10	D7	50	0.30	[0.60, 1.40]

TABLE 7.3: *The different OMOPSO parameter combinations.*

OMOPSO					
Combination	N^P	W	C_1	C_2	p_m
P1	50	1.00	1.50	1.50	0.05
P2	30	1.00	1.50	1.50	0.05
P3	100	1.00	1.50	1.50	0.05
P4	50	[0.10, 0.50]	1.50	1.50	0.05
P5	50	[0.50, 1.50]	1.50	1.50	0.05
P6	50	1.00	1.00	1.50	0.05
P7	50	1.00	2.00	1.50	0.05
P8	50	1.00	1.50	1.00	0.05
P9	50	1.00	1.50	2.00	0.05
P10	50	1.00	1.50	1.50	0.02
P11	50	1.00	1.50	1.50	0.10

A visual, albeit cursory, analysis precedes an extensive statistical analysis. The box plots in Figures 7.2–7.16 provide a graphical summary of the results of the sensitivity and facilitate the preliminary analysis. Although a visual analysis of the algorithmic performance data (of which

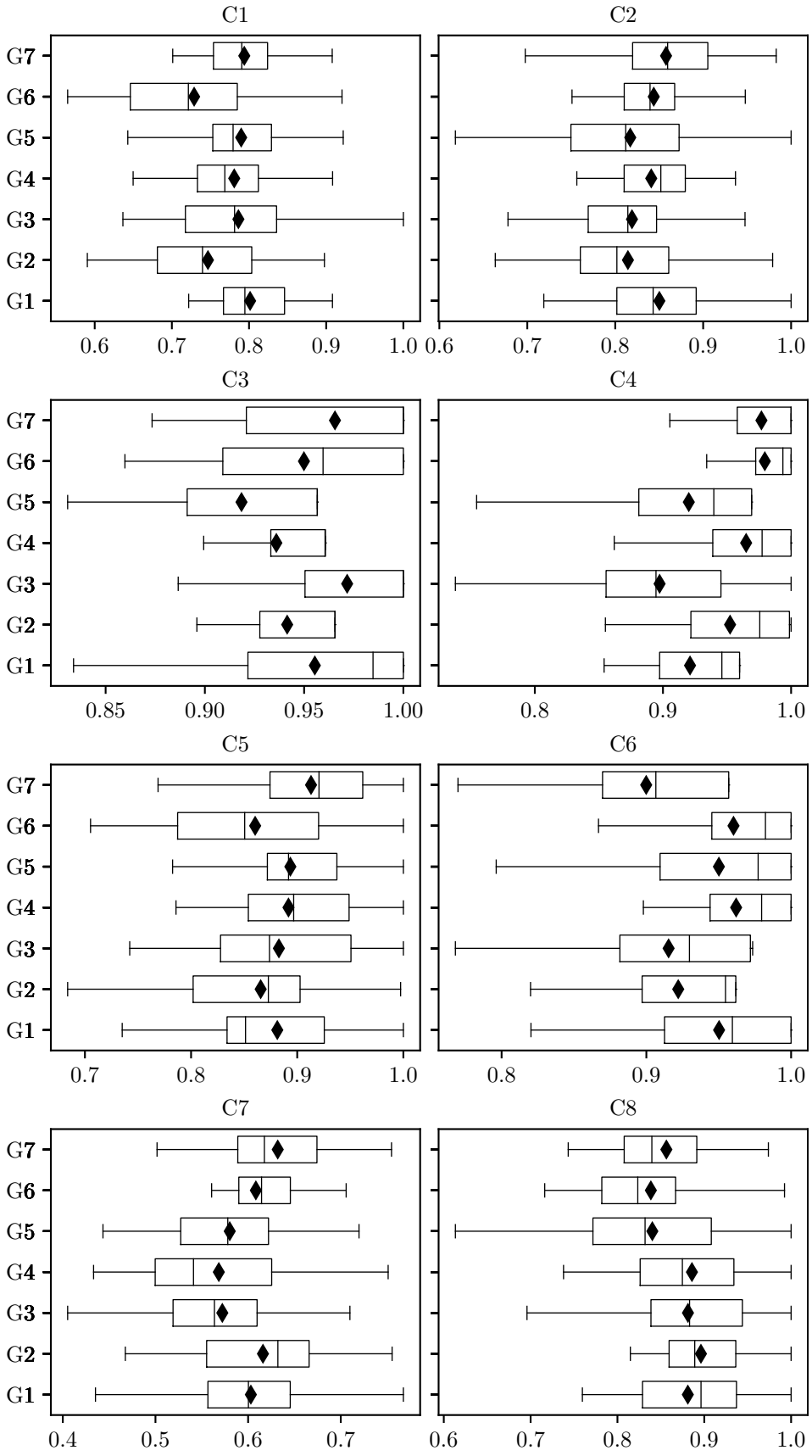


FIGURE 7.2: *Box plots of the F_1-scores achieved by each NSGA-II parameter combination (presented in Table 7.2) with respect to data sets C1–C8.*

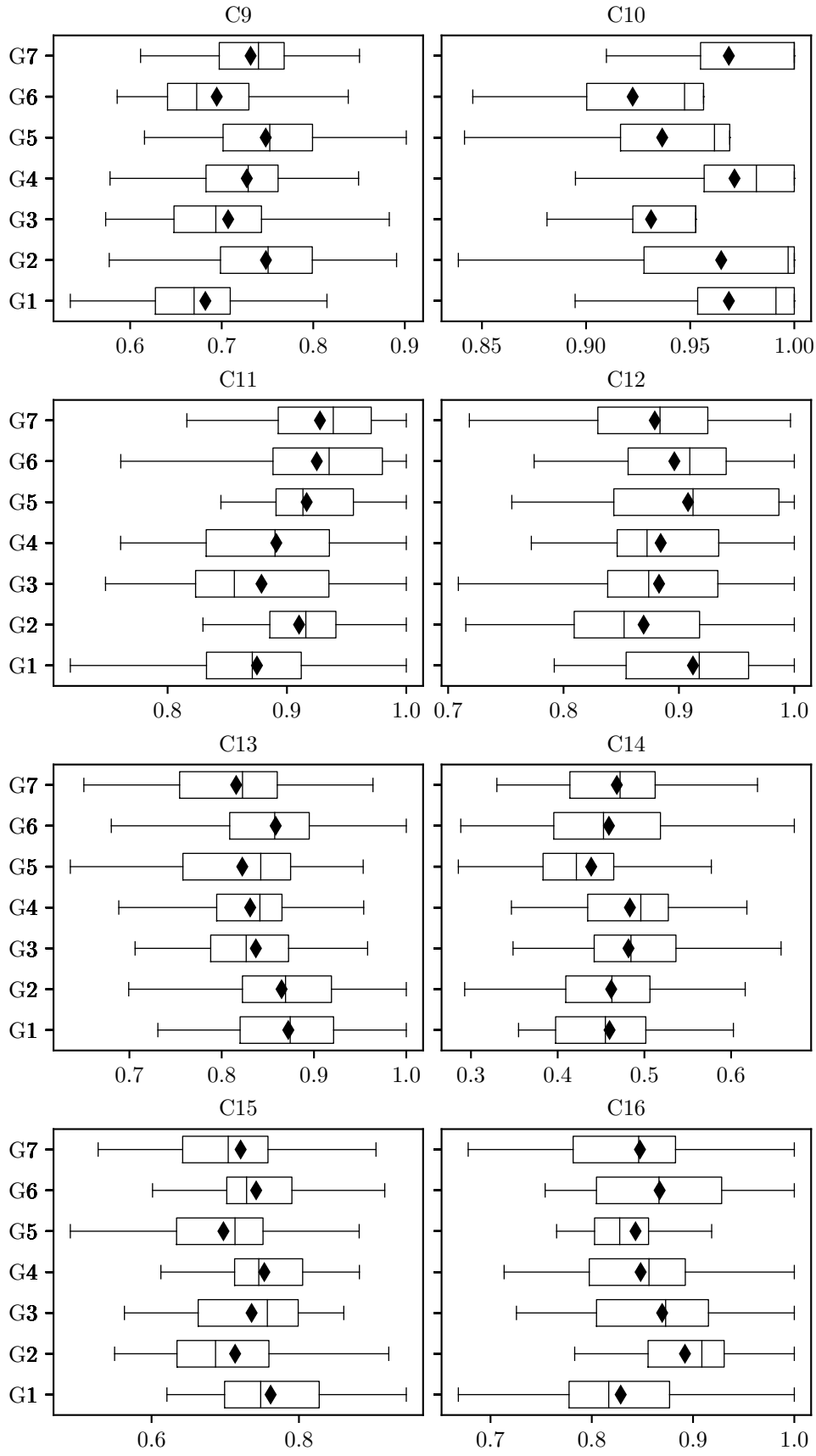


FIGURE 7.3: *Box plots of the F_1-scores achieved by each NSGA-II parameter combination (presented in Table 7.2) in respect of data sets C9–C16.*

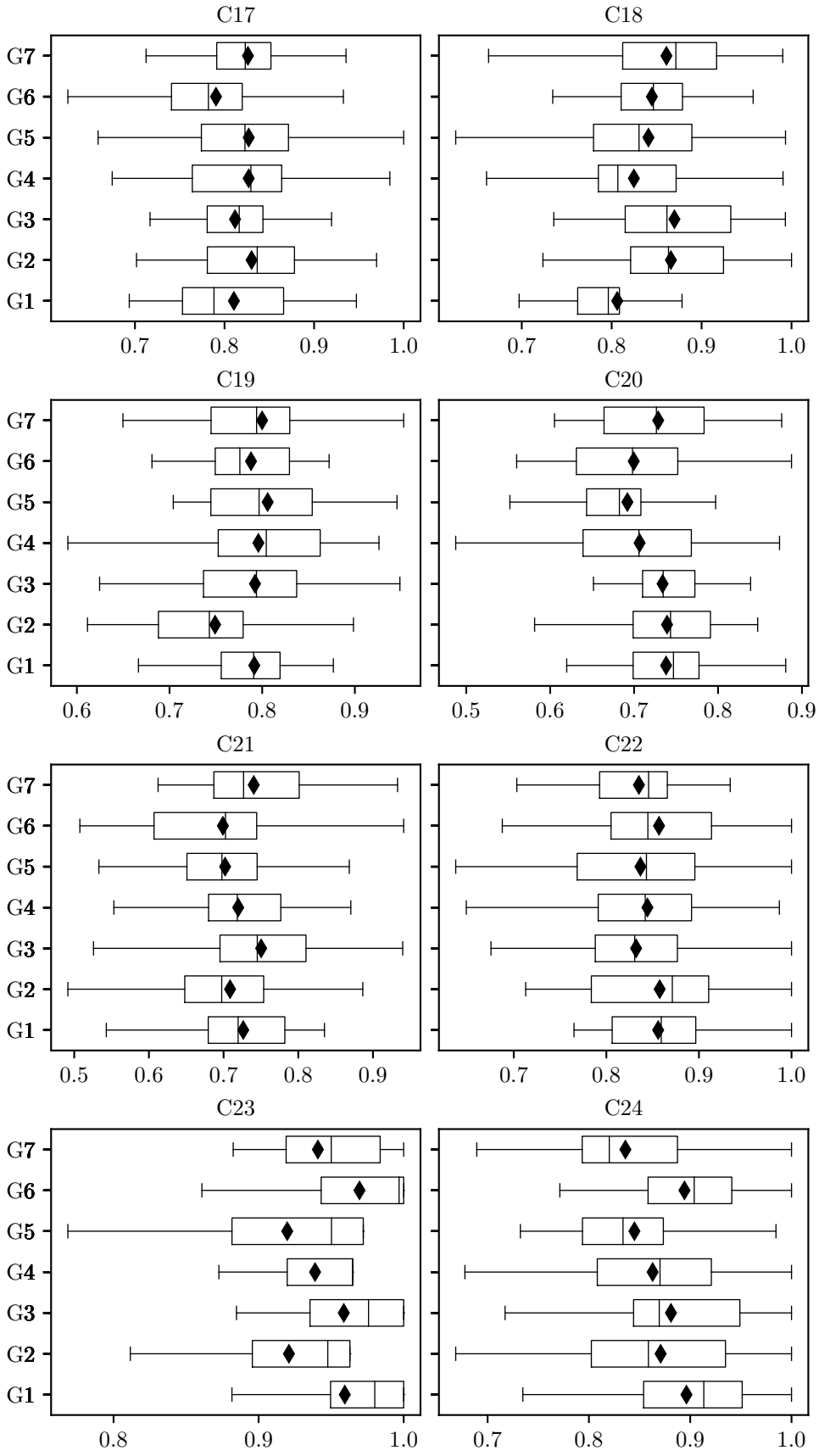


FIGURE 7.4: *Box plots of the F_1-scores achieved by each NSGA-II parameter combination (presented in Table 7.2) in respect of data sets C17–C24.*

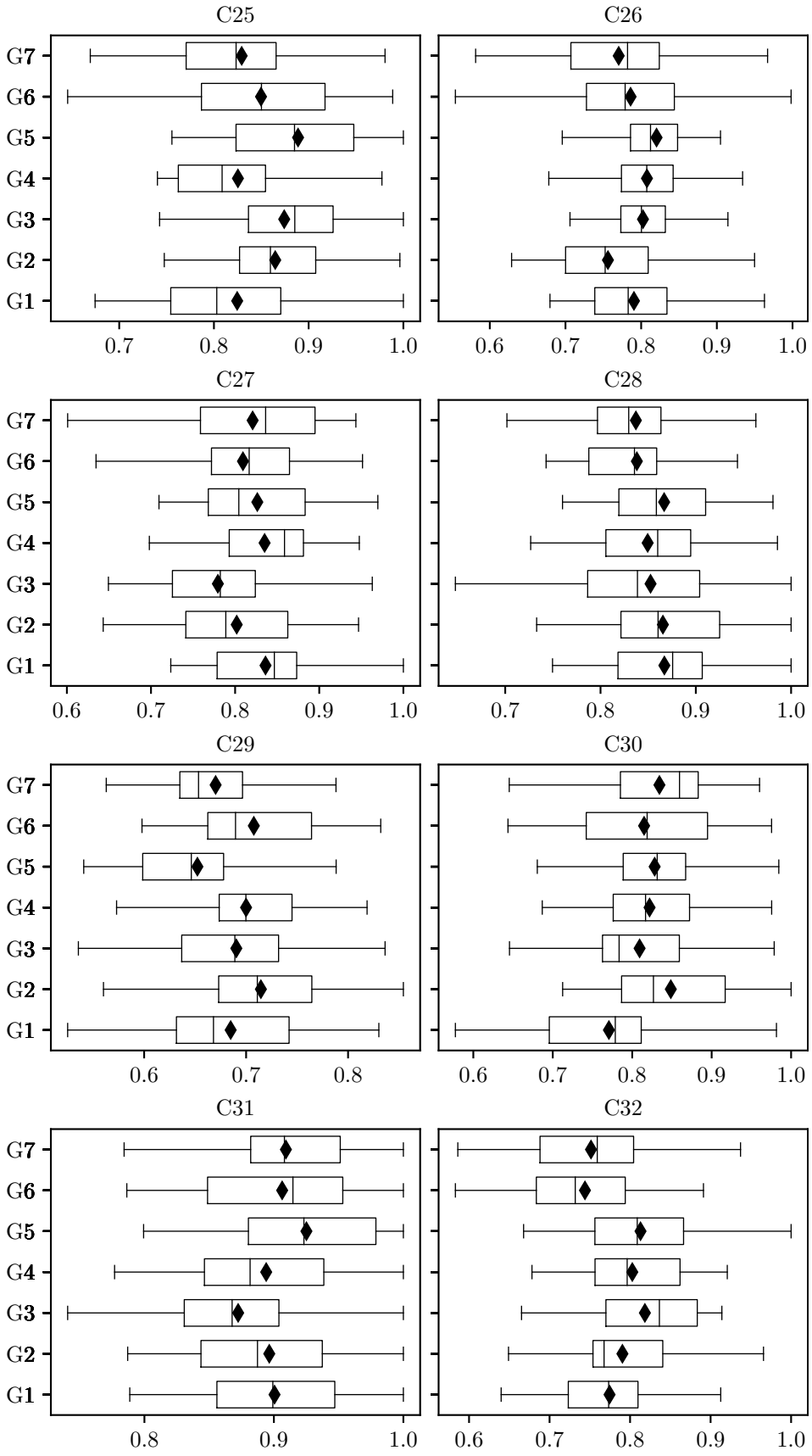


FIGURE 7.5: *Box plots of the F_1-scores achieved by each NSGA-II parameter combination (presented in Table 7.2) in respect of data sets C25–C32.*

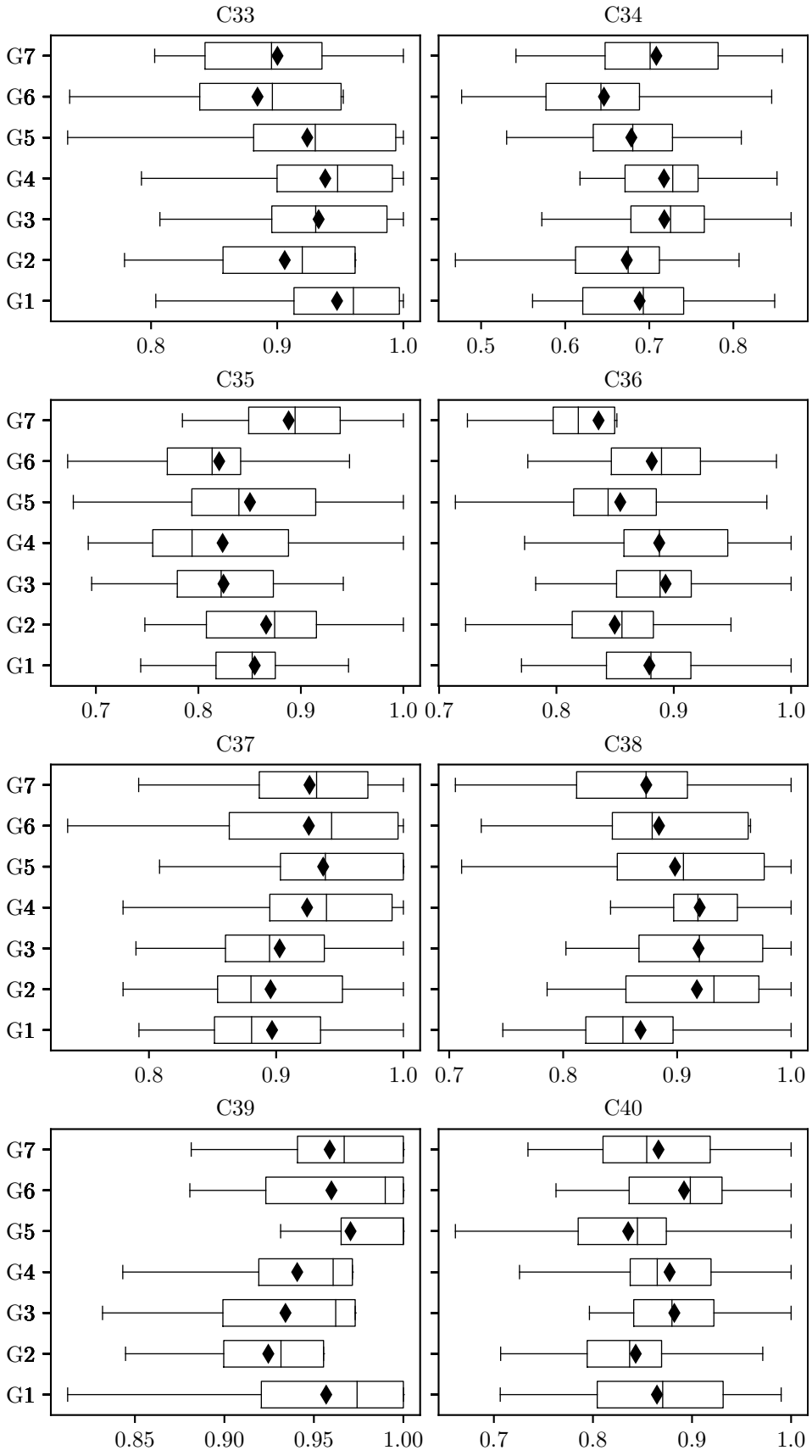


FIGURE 7.6: *Box plots of the F_1-scores achieved by each NSGA-II parameter combination (presented in Table 7.2) in respect of data sets C33–C40.*

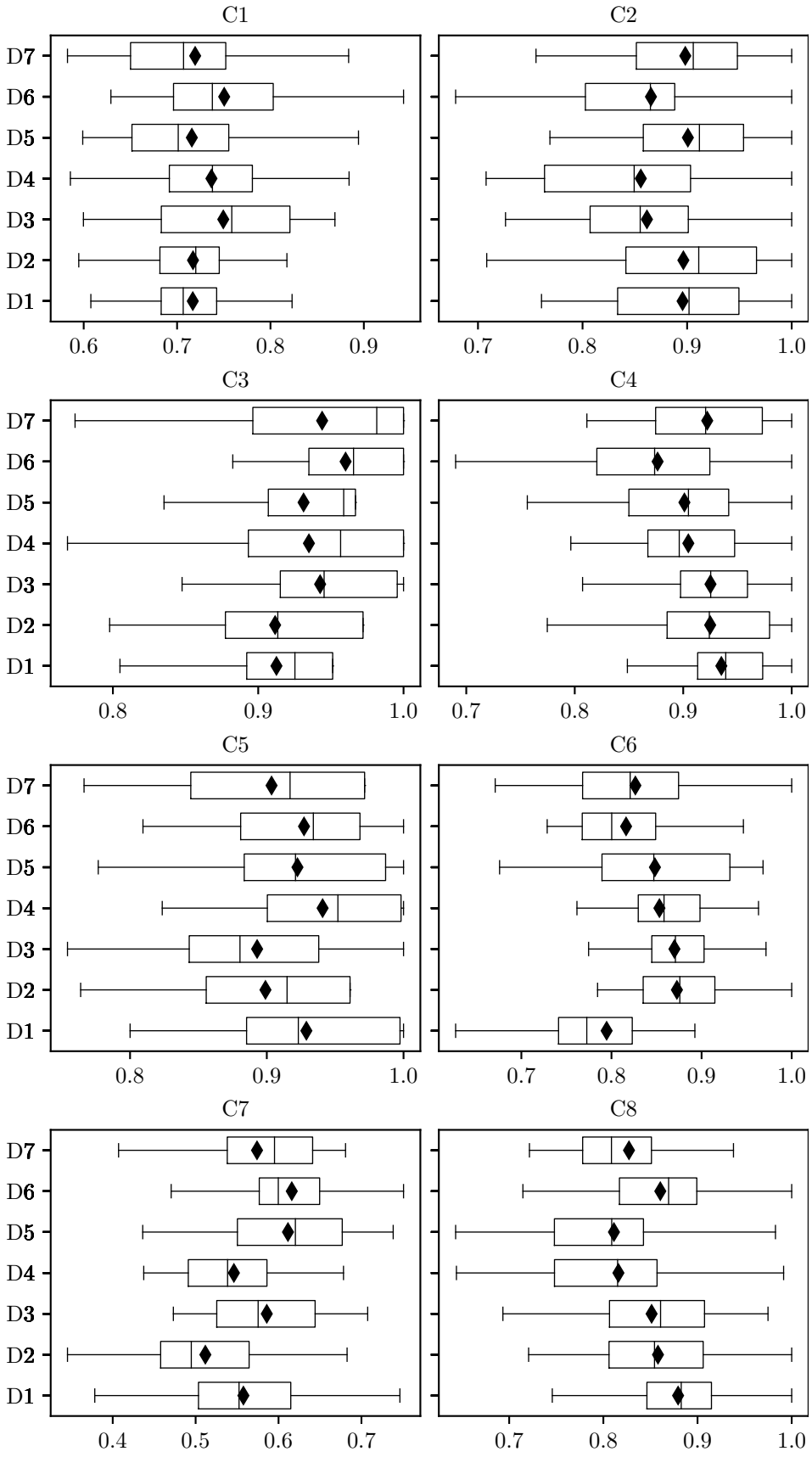


FIGURE 7.7: *Box plots of the F_1-scores achieved by each NSDE parameter combination (presented in Table 7.2) in respect of data sets C1–C8.*

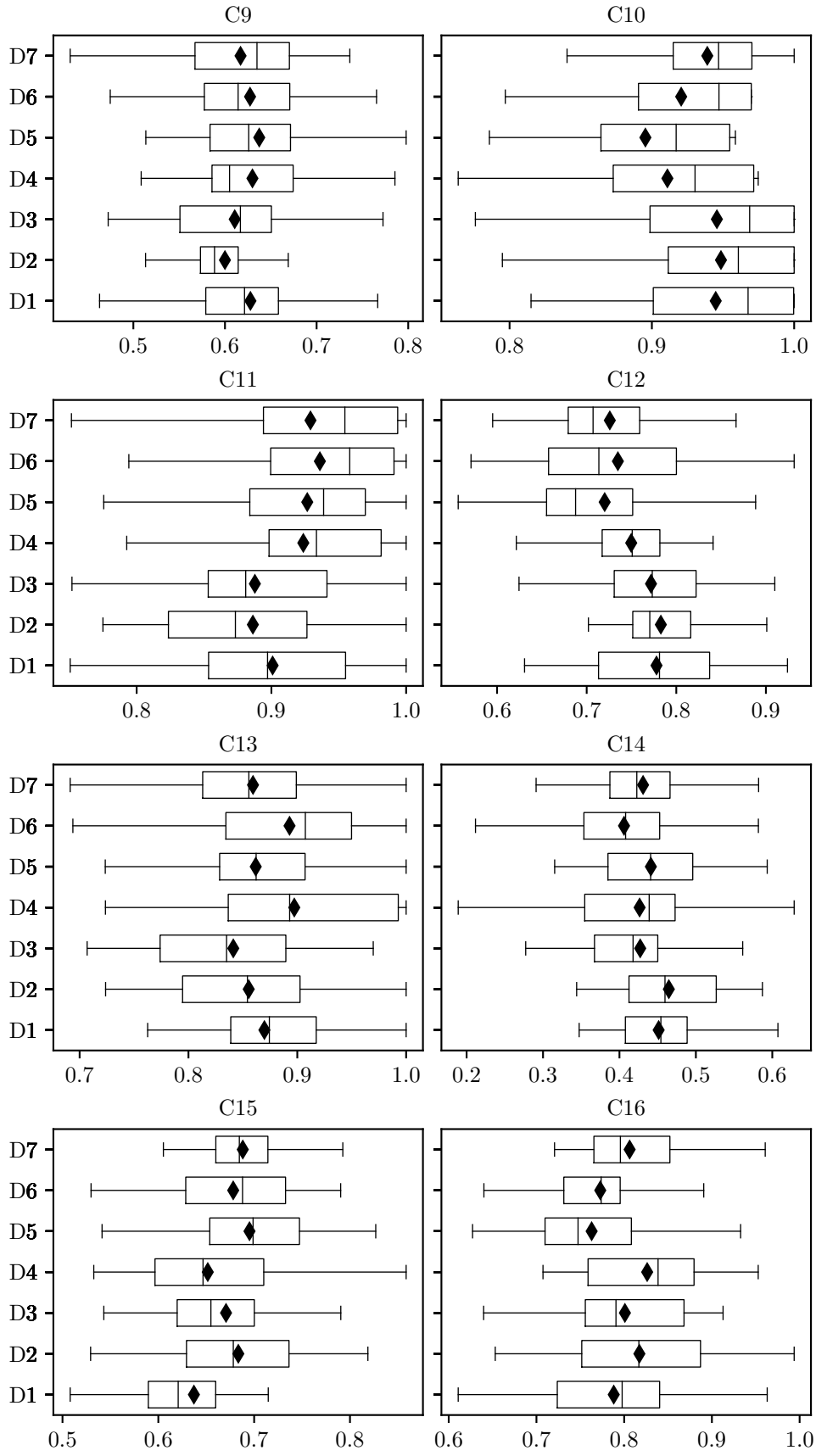


FIGURE 7.8: *Box plots of the F_1-scores achieved by each NSDE parameter combination (presented in Table 7.2) in respect of data sets C9–C16.*

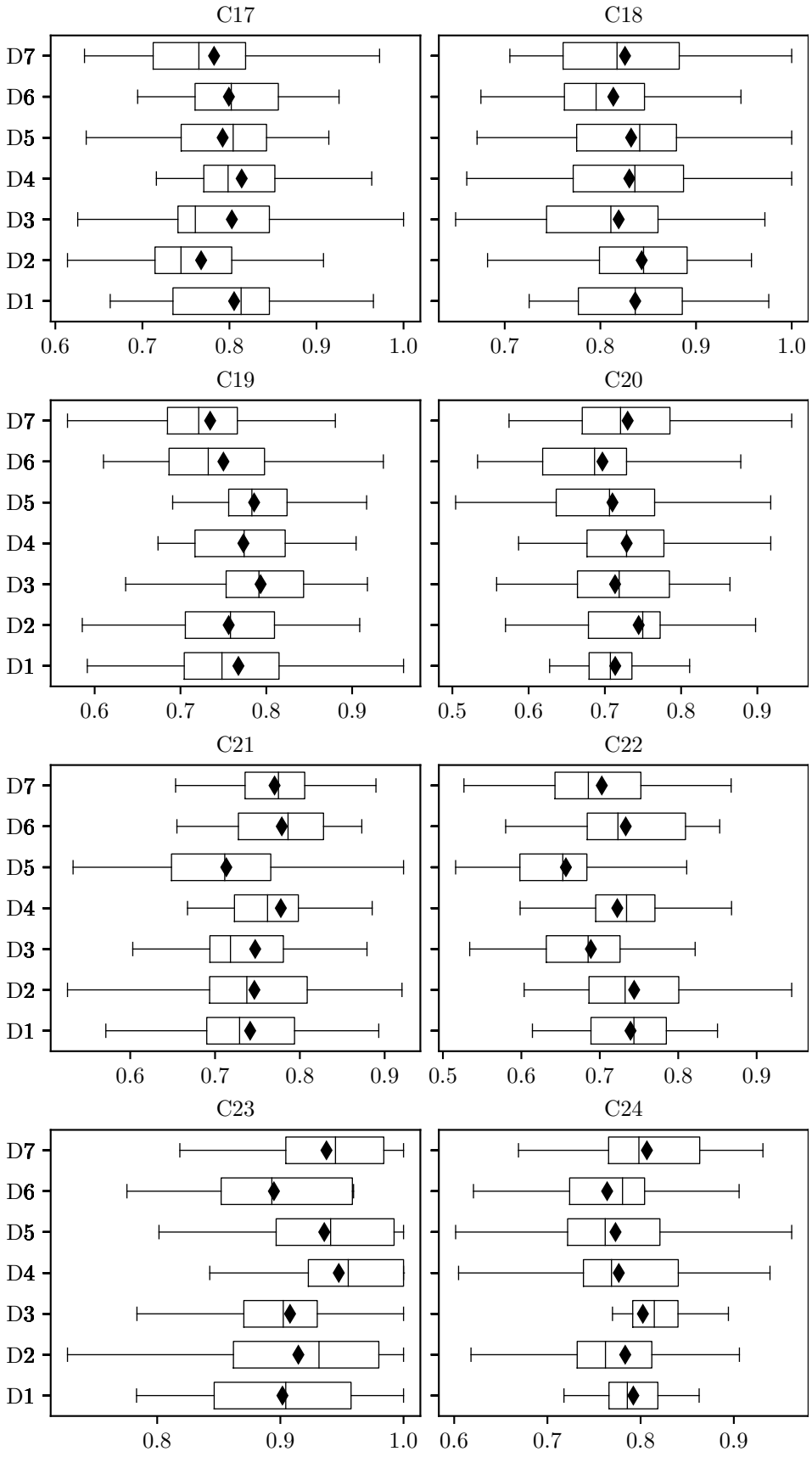


FIGURE 7.9: *Box plots of the F_1-scores achieved by each NSDE parameter combination (presented in Table 7.2) in respect of data sets C17–C24.*

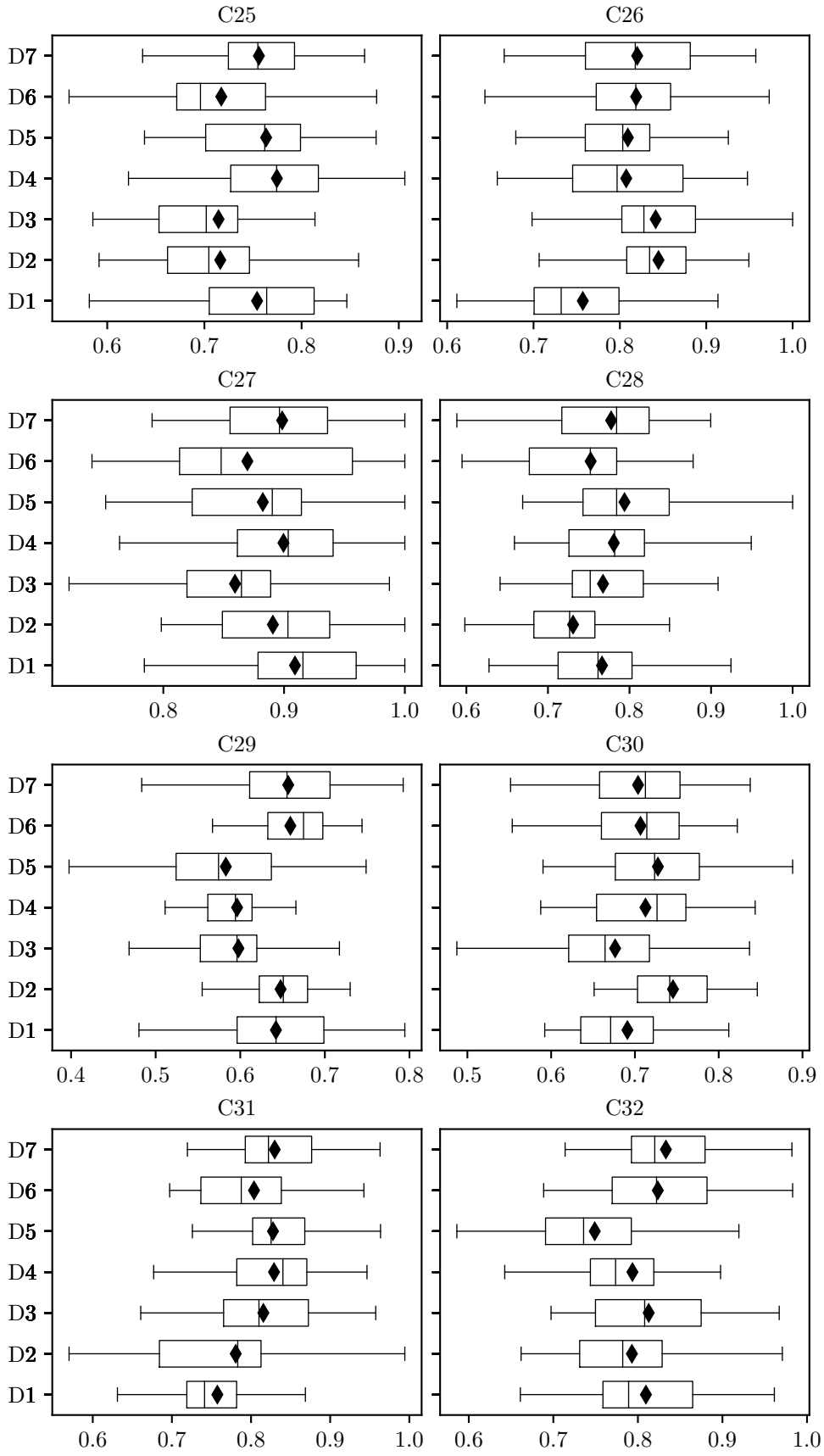


FIGURE 7.10: *Box plots of the F_1-scores achieved by each NSDE parameter combination (presented in Table 7.2) in respect of data sets C25–C32.*

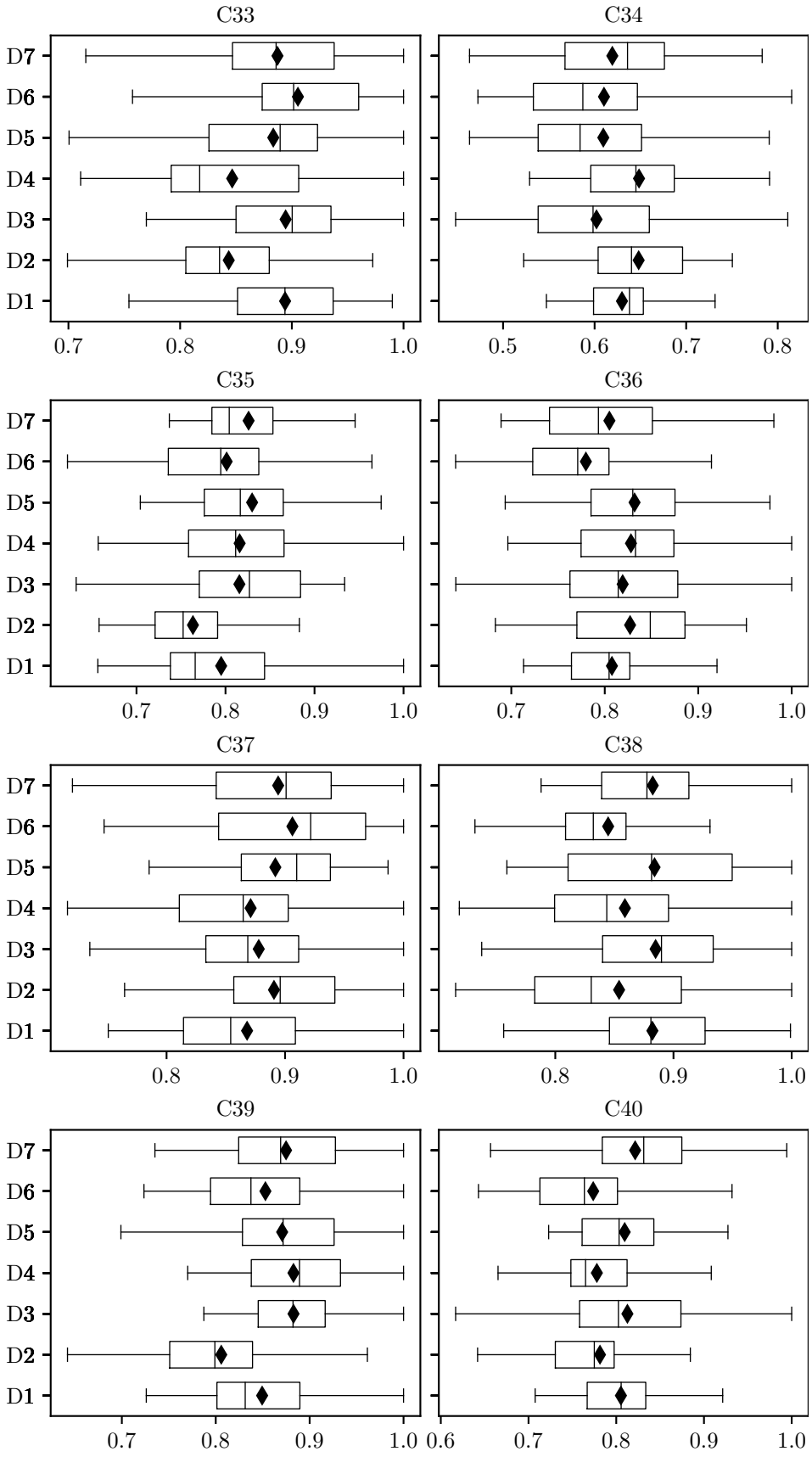


FIGURE 7.11: *Box plots of the F_1-scores achieved by each NSDE parameter combination (presented in Table 7.2) in respect of data sets C33–C40.*

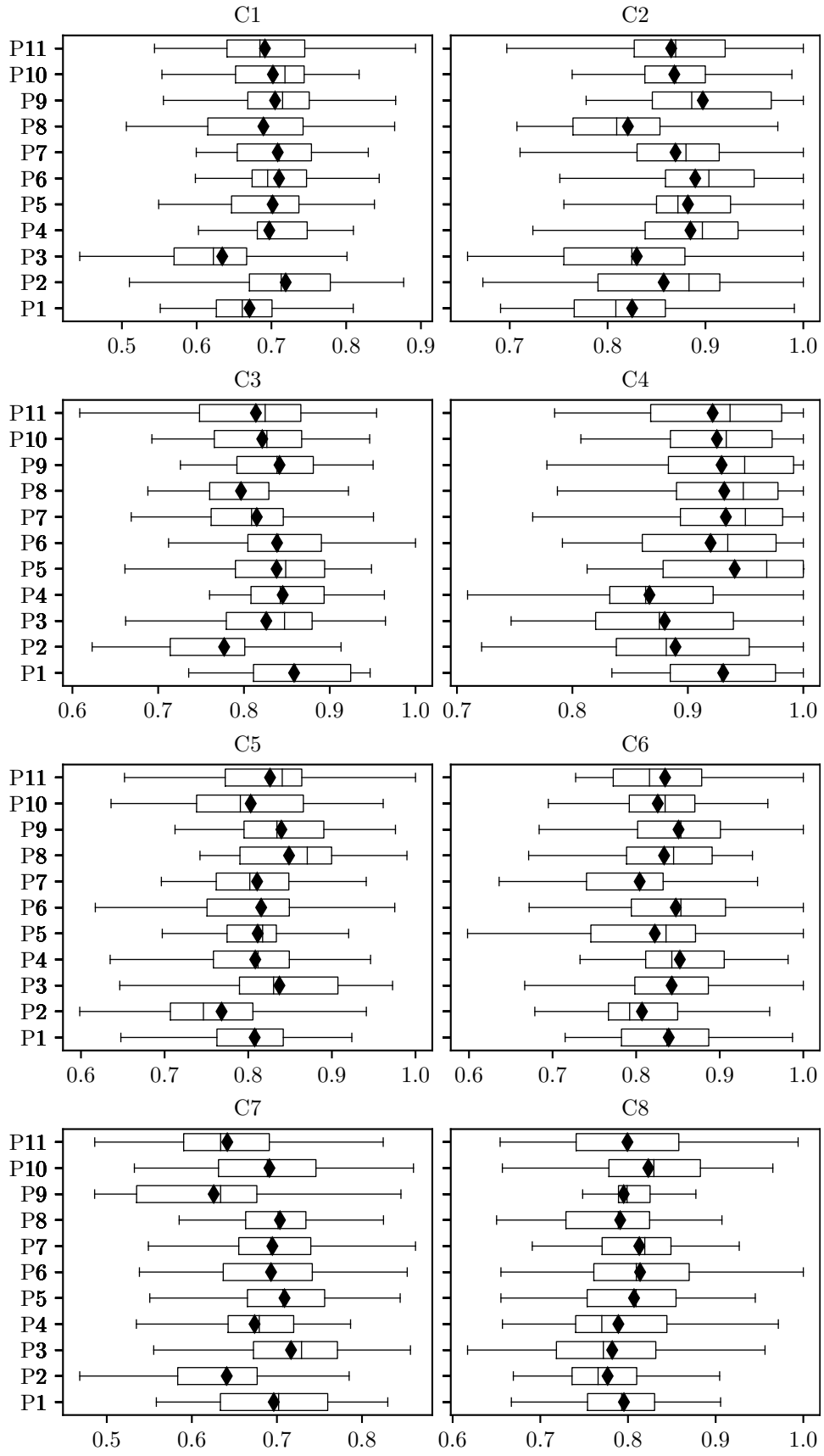


FIGURE 7.12: *Box plots of the F_1-scores achieved by each OMOPSO parameter combination (presented in Table 7.3) in respect of data sets C1–C8.*

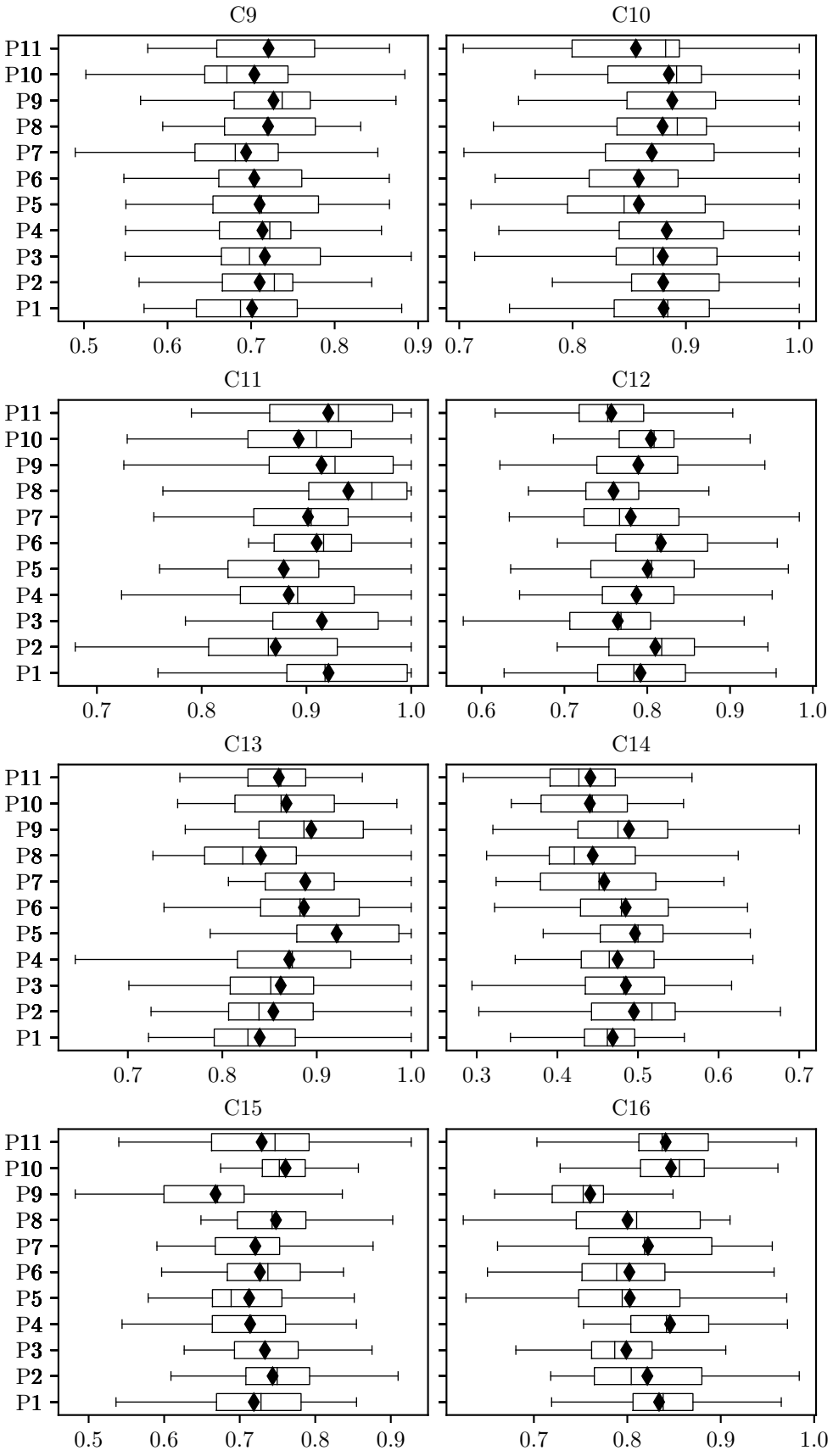


FIGURE 7.13: *Box plots of the F_1-scores achieved by each OMOPSO parameter combination (presented in Table 7.3) in respect of data sets C9–C16.*

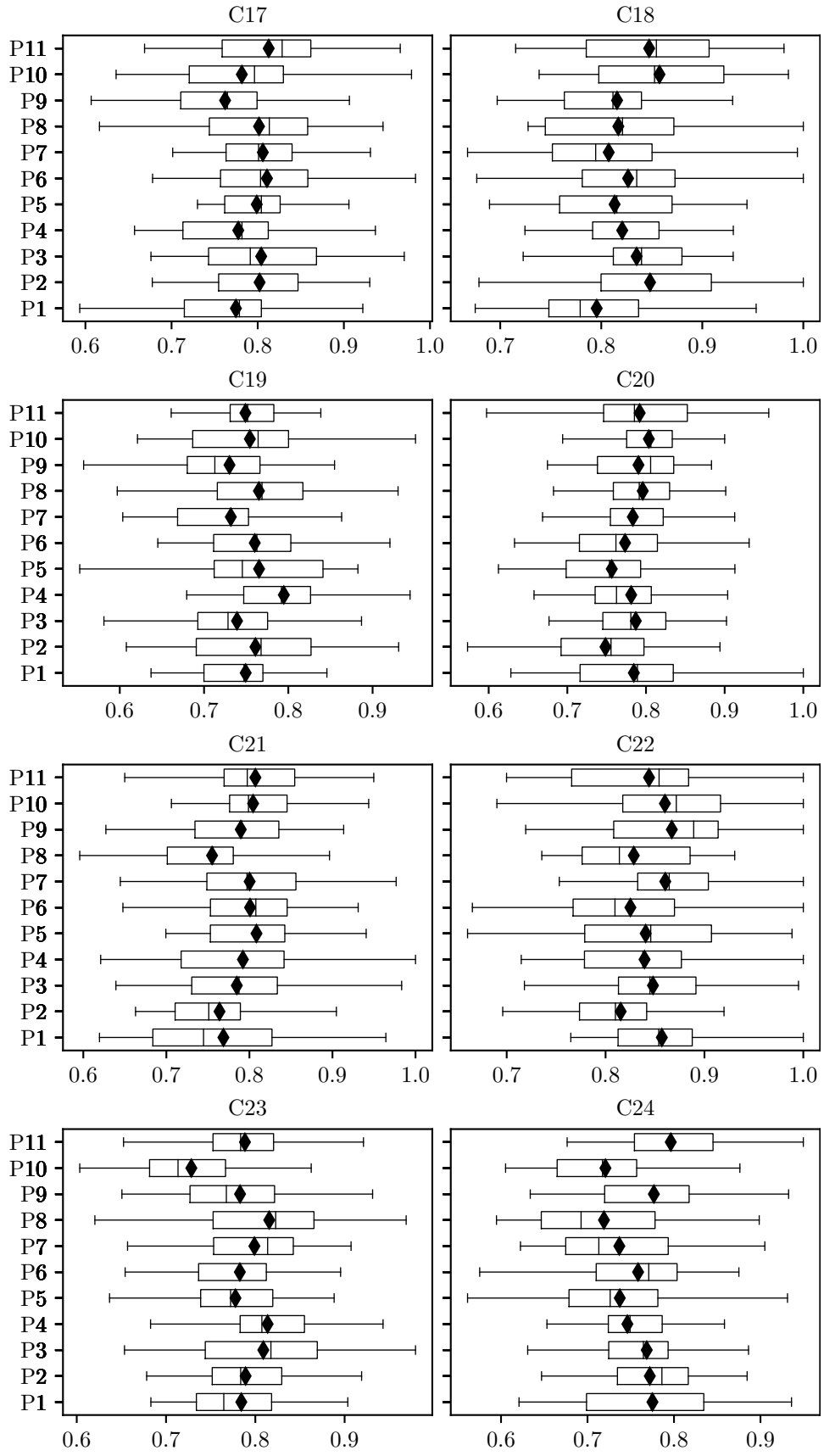


FIGURE 7.14: *Box plots of the F_1-scores achieved by each OMOPSO parameter combination (presented in Table 7.3) in respect of data sets C17–C24.*

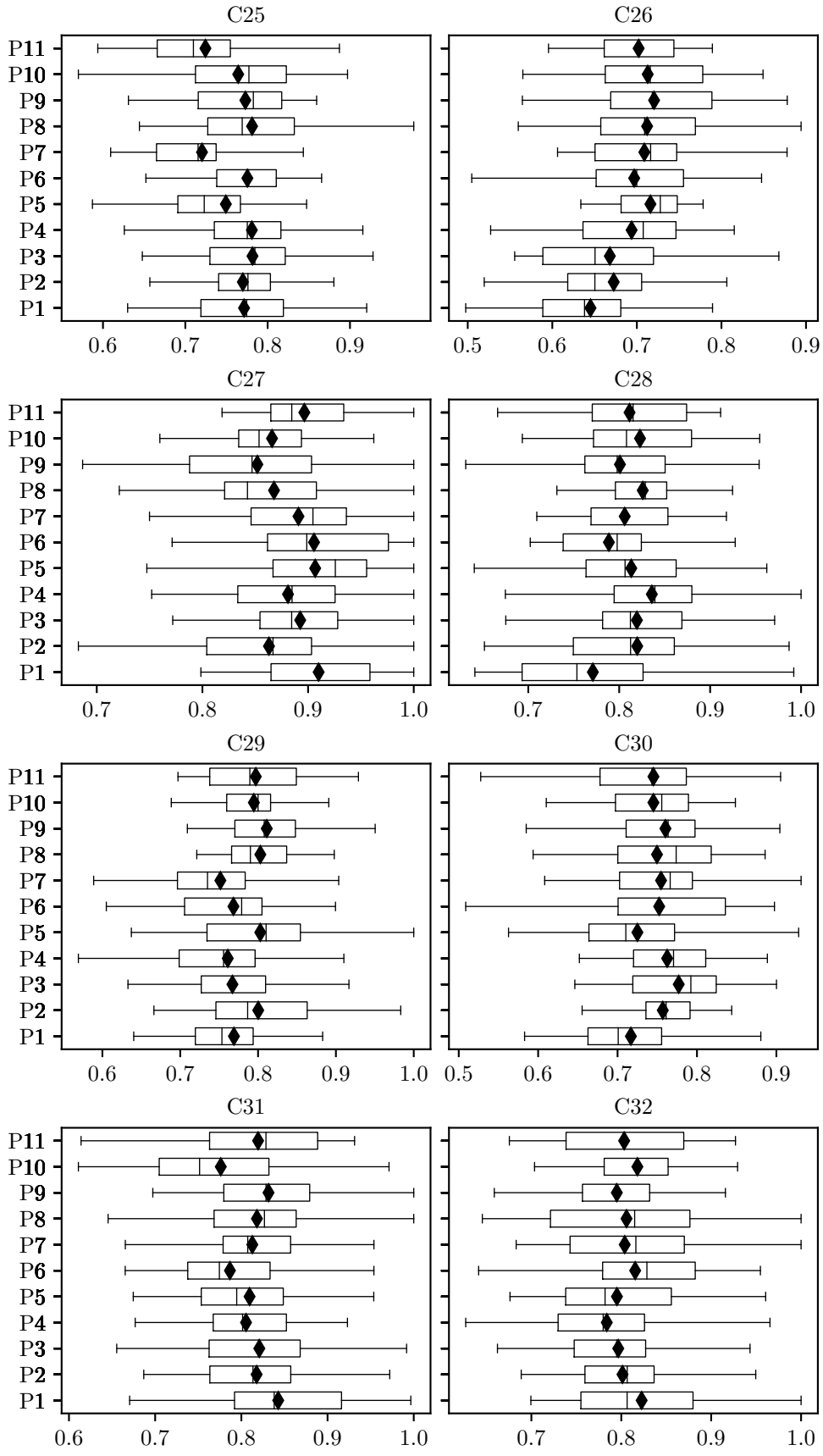


FIGURE 7.15: *Box plots of the F_1-scores achieved by each OMOPSO parameter combination (presented in Table 7.3) in respect of data sets C25–C32.*

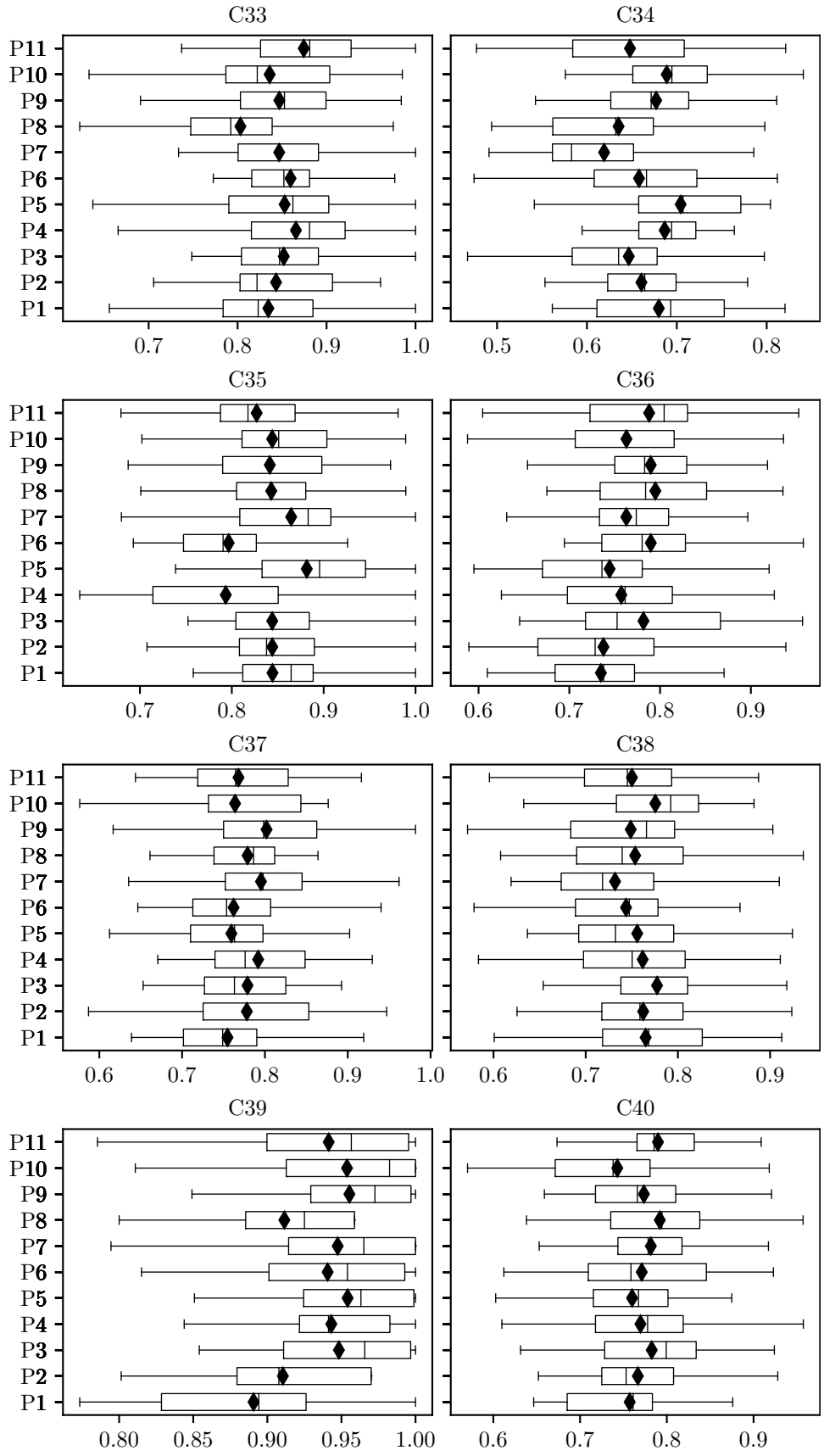


FIGURE 7.16: *Box plots of the F_1-scores achieved by each OMOPSO parameter combination (presented in Table 7.3) in respect of data sets C33–C40.*

there are many) does not provide the same level of insight as do statistical analyses, a few revelatory observations can still be made. It is apparent from the box plots that for numerous data sets, algorithmic performance is sensitive to the parameter combination employed — attributable to the notable variation amongst F_1-score sample means (and medians). For example, the sample means achieved by the different NSGA-II parameter combinations for data set C25 indicate a notable difference in performance between parameter combinations G1 and G5 — attributable to the increase in the crossover probability p_c from 0.90 to 1.00. Another example pertains to NSDE in respect of data set C31, for which an inertia weight of $[0.60, 1.40]$ (*i.e.* parameter combination D7) results in notably improved performance over an inertia weight of $[0.50, 1.00]$ (*i.e.* parameter combination D1). There are, however, a few data sets for which the corresponding algorithmic performance seems indistinguishable. For example, the sample means (and medians) achieved by the different NSDE parameter combinations for data set C20 indicate notably consistent performance amongst the different combinations. The different parameter combinations of OMOPSO in respect of data set C38 is another example of seemingly indistinguishable algorithmic performance. Overall, each of the three sub-algorithms demonstrates the same phenomenon — over numerous data sets, algorithmic performance is sensitive to the parameter combination employed.

Furthermore, when comparing the algorithmic performance achieved by each of the three sub-algorithms with one another it is apparent that the NSGA-II outperforms its counterparts. Consider, for example, the sample comprising the first eight data sets, *i.e.* C1–C8. The NSGA-II outperforms both NSDE and OMOPSO in respect of five data sets, *i.e.* C1, C3, C4, C6, and C8, whereas NSDE outperforms its counterparts in respect of only two data sets, *i.e.* C2 and C5. Finally, OMOPSO outperforms its counterparts in respect of only one data set, *i.e.* C7. While the limited sample size of eight data sets warrants some scepticism, the sample means and medians averaged across *all* parameter combinations and in respect of *all* data sets provide further empirical evidence of the NSGA-II's superiority, as summarised in Table 7.4. The algorithmic performances of NSDE and OMOPSO are notably similar within this, albeit highly aggregated, context.

TABLE 7.4: *Sample median and mean F_1-score performances achieved by the NSGA-II, NSDE, and OMOPSO (averaged across all parameter combinations and all data sets), denoted by $\bar{F}_1$.*

	$\bar{F}_1$-score	
	Median	Mean
NSGA-II	0.8157	0.7968
NSDE	0.7817	0.7644
OMOPSO	0.7736	0.7612

The cursory and (arguably) superficial analysis carried out in the above discussion does not elucidate the algorithmic nuances embedded within the considerable volume of algorithmic performance data available. Extensive statistical analyses are therefore performed to extract further insight. The Friedman test is performed in order first to determine whether a significant difference exists between at least two samples and then, if the Friedman test detects such a difference, the Nemenyi *post hoc* procedure is performed so as to identify specifically the pairs of samples in which the differences are present. The p-values obtained after performing the Friedman test on the respective samples for each sub-algorithm and data set are presented in Table 7.5. In the case of the NSGA-II, there are 22 data sets for which statistically significant performance differences exist between at least two of the samples at a 5% level of significance. Furthermore, in the case of NSDE and OMOPSO, there are 21 and 23 data sets, respectively, for which statistically significant performance differences exist between at least two of the samples at a 5% level of

significance. This finding corroborates the earlier empirical observation (in the box plots) that algorithmic performance is sensitive to the parameters chosen.

TABLE 7.5: *Friedman test p-values for each BOHTA sub-algorithm and for each data set. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Friedman test p-values						
Data set	NSGA-II	NSDE	OMOPSO	Data set	NSGA-II	NSDE	OMOPSO
C1	0.0025	0.4552	0.0077	C21	0.5716	0.1145	0.0566
C2	0.1528	0.1414	0.0005	C22	0.7894	0.0002	0.0494
C3	0	0.0004	0.0257	C23	0	0.0086	0.0071
C4	0	0.0455	0.0147	C24	0.0381	0.6191	0.0017
C5	0.3472	0.0209	0.0151	C25	0.0062	0.0007	0.0008
C6	0	0.0008	0.1654	C26	0.0537	0.0074	0.0011
C7	0.0099	0	0.0001	C27	0.3784	0.1711	0.0105
C8	0.0057	0.0019	0.5729	C28	0.4514	0.0445	0.1923
C9	0.0017	0.6325	0.8410	C29	0.0059	0.0007	0.0094
C10	0	0.0004	0.7994	C30	0.0458	0.0635	0.3168
C11	0.0271	0.0546	0.0240	C31	0.1872	0.0029	0.0895
C12	0.3064	0.0004	0.1609	C32	0.0068	0.0009	0.7122
C13	0.2602	0.1120	0.0001	C33	0.0005	0.0162	0.1550
C14	0.3562	0.3967	0.0114	C34	0.0017	0.2184	0.0085
C15	0.1754	0.0193	0.0045	C35	0.0014	0.0120	0.0008
C16	0.1698	0.0099	0	C36	0.0043	0.1721	0.0351
C17	0.1563	0.4949	0.5936	C37	0.3999	0.3522	0.1808
C18	0.0051	0.3936	0.0309	C38	0.0081	0.0562	0.4899
C19	0.1754	0.0938	0.0537	C39	0	0.0022	0
C20	0.0583	0.1496	0.1011	C40	0.0825	0.0589	0.2598

The Nemenyi procedure is consequently applied in respect of each sub-algorithm and data set for which the corresponding Friedman test p-value (presented in Table 7.5) is less than 0.05. The evaluations of the NSGA-II and NSDE each comprise seven parameter combinations, and so the Nemenyi procedure involves $\binom{7}{2} = 21$ pairwise significance tests. In the case of OMOPSO, there are eleven parameter combinations, and so the Nemenyi procedure involves $\binom{11}{2} = 55$ pairwise significance tests. Consider a data set for which statistical differences exist for each of the three sub-algorithms at a 5% level of significance, such as data set C3, for example. The corresponding Nemenyi test p-values for the NSGA-II, NSDE, OMOPSO are presented in Tables 7.6, 7.7, and 7.8, respectively.

TABLE 7.6: *Nemenyi test p-values for the NSGA-II in respect of data set C3. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Nemenyi test p-values						
	G1	G2	G3	G4	G5	G6	G7
G1	—	0.1066	0.1602	0.0232	0.0005	0.3241	0.5109
G2		—	0.0025	0.5109	0.0639	0.5303	0.0232
G3			—	0.0002	0	0.0168	0.4551
G4				—	0.232	0.1988	0.0034
G5					—	0.0131	0
G6						—	0.1003
G7							—

TABLE 7.7: *Nemenyi test p-values for the NSDE in respect of data set C3. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Nemenyi test p-values						
	D1	D2	D3	D4	D5	D6	D7
D1	—	0.5907	0.006	0.0365	0.0422	0.0002	0.0004
D2		—	0.0270	0.1202	0.1352	0.0015	0.0028
D3			—	0.5109	0.4733	0.3390	0.4372
D4				—	0.9523	0.1066	0.1515
D5					—	0.0943	0.1352
D6						—	0.8577
D7							—

TABLE 7.8: *Nemenyi test p-values for the OMOPSO in respect of data set C3. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Nemenyi test p-values										
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
P1	—	0.0011	0.1195	0.5334	0.3706	0.2758	0.0265	0.0040	0.4596	0.0734	0.0391
P2		—	0.0868	0.0081	0.0176	0.0293	0.2933	0.6971	0.0114	0.1391	0.2276
P3			—	0.3502	0.5081	0.6404	0.5081	0.1857	0.4137	0.8153	0.6128
P4				—	0.7853	0.6404	0.1105	0.0240	0.9070	0.2429	0.1498
P5					—	0.8457	0.1857	0.0471	0.8763	0.3706	0.2429
P6						—	0.2590	0.0734	0.7261	0.4835	0.3305
P7							—	0.5081	0.1391	0.6685	0.8763
P8								—	0.0323	0.2758	0.4137
P9									—	0.2933	0.1857
P10										—	0.7853
P11											—

The last step towards identifying a single best parameter combination involves the following procedure. For each data set and sub-algorithm, the combinations are first ranked according to sample median, thereafter the combinations that are statistically equivalent to the best performing combination, *i.e.* the combinations that are statistically indistinguishable according to the Nemenyi procedure, are identified. In the case of the aforementioned example data set C3, the ranked parameter combinations for the NSGA-II, NSDE, and OMOPSO, together with the respective statistical equivalents of the best performing combination, are presented in Tables 7.9, 7.10, and 7.11, respectively. In the case of the NSGA-II, the best performing combination (with respect to sample medians) is G3, for which performance is statistically indistinguishable from those for combinations G1 and G7. In the case of NSDE, parameter combination D7 results in the best sample median and its statistical equivalents are combinations D3, D4, D5, and D6. Finally, in the case of OMOPSO, the best performing parameter combination is P1, for which performance is statistically indistinguishable from those for combinations P3, P4, P5, P6, P9, and P10. The statistical equivalence classes for each of the three sub-algorithms can be corroborated simply by observing the Nemenyi test p-values in the relevant row and column of G3, D7, and P1 in Tables 7.6, 7.7, and 7.8, respectively. Finally, the single best parameter combination for each of the sub-algorithms can be identified by performing the procedure above in respect of all the data sets, *i.e.* C1–C40, and observing both the frequency with which a combination is ranked first and the frequency with which a combination is statistically equivalent to the combination that is ranked first. The parameter combination that achieves the highest frequency across all forty data sets is then selected as the best performing combination.

TABLE 7.9: *NSGA-II parameter combinations ranked in descending order of sample median in respect of data set C3. The parameter combinations that are statistically equivalent to the best performing combination (i.e. the combination with a rank of 1) are denoted by matching combination numbers in the column labelled "Stat. Eq."*

Rank	Combination	Median	Stat. Eq.
1	G3	1	G1,G7
2	G7	1	—
3	G1	0.9847	—
4	G2	0.9656	—
5	G4	0.9606	—
6	G6	0.9595	—
7	G5	0.9567	—

TABLE 7.10: *NSDE parameter combinations ranked in descending order of sample median in respect of data set C3. The parameter combinations that are statistically equivalent to the best performing combination (i.e. the combination with a rank of 1) are denoted by matching combination numbers in the column labelled "Stat. Eq."*

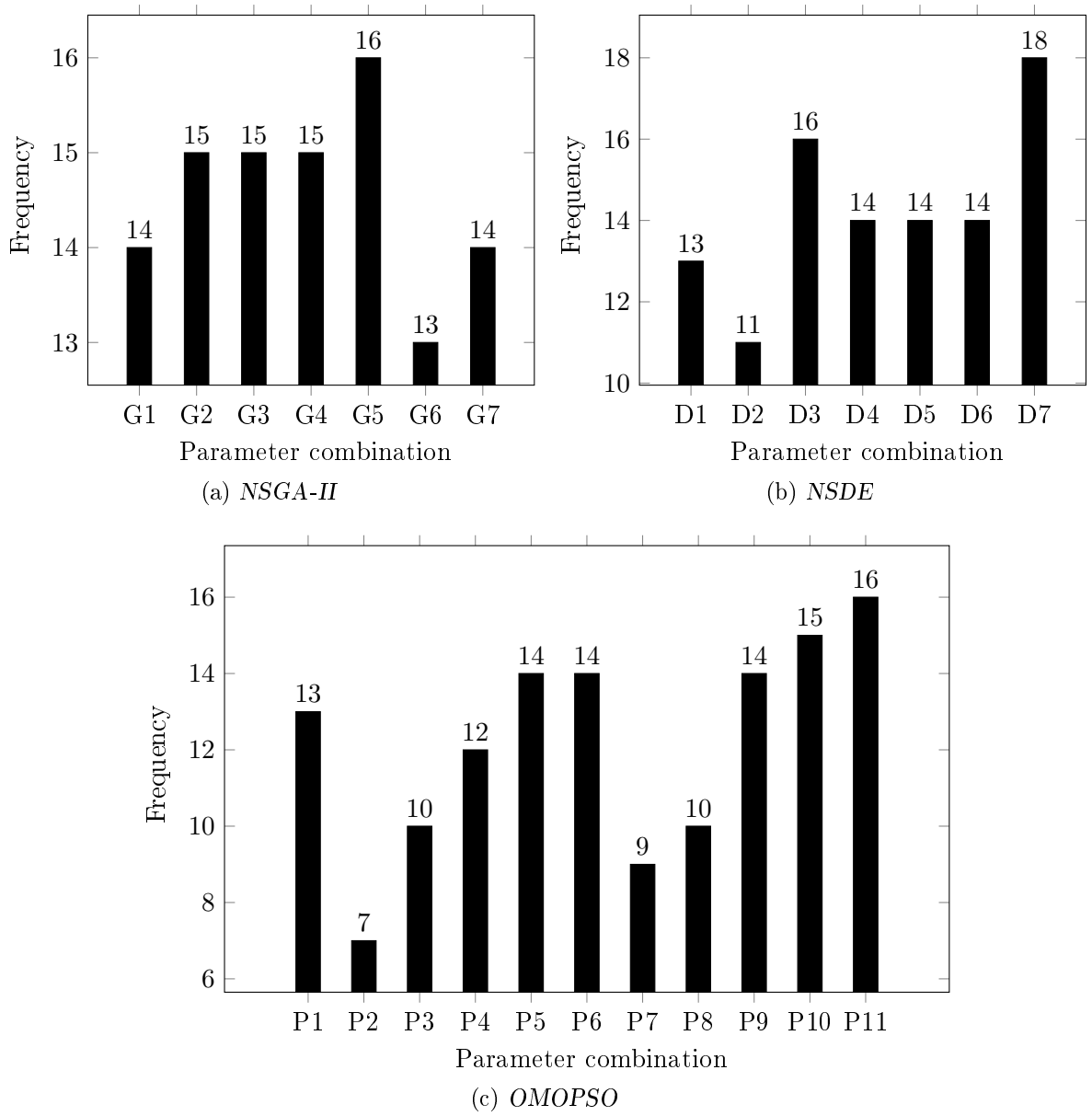
Rank	Combination	Median	Stat. Eq.
1	D7	0.9816	D3,D4,D5,D6
2	D6	0.9656	—
3	D5	0.9588	—
4	D4	0.9567	—
5	D3	0.9453	—
6	D1	0.9253	—
7	D2	0.9135	—

TABLE 7.11: *OMOPSO parameter combinations ranked in descending order of sample median in respect of data set C3. The parameter combinations that are statistically equivalent to the best performing combination (i.e. the combination with a rank of 1) are denoted by matching combination numbers in the column labelled "Stat. Eq."*

Rank	Combination	Median	Stat. Eq.
1	P1	0.8599	P3,P4,P5,P6,P9,P10
2	P5	0.8487	—
3	P3	0.8473	—
4	P4	0.8423	—
5	P6	0.8401	—
6	P9	0.8385	—
7	P10	0.8267	—
8	P11	0.8247	—
9	P7	0.8089	—
10	P8	0.7962	—
11	P2	0.7773	—

The respective frequencies for the different sub-algorithm parameter combinations are presented in Figure 7.17. In the case of the NSGA-II, parameter combination G5 outperforms its counterparts in terms of sample median. Based on this finding, it may be concluded that a large crossover probability of $p_c = 1.00$ leads to the most statistically significant performance improvement for this sub-algorithm. Overall, the difference in performance between the best performing

parameter combination G5 and the worst performing combination G6 is, surprisingly, not as significant — there are only three data sets for which G5 is superior when compared with G6. The NSGA-II can therefore be regarded as a robust training algorithm, exhibiting limited sensitivity to the choice of parameter values employed. Another interesting finding pertains to the impact of population size on performance (or the lack thereof) — when compared with NSDE and OMOPSO, a small population size of 30 (corresponding to D2 and P2) is evidently an inhibitor of performance in the case of the NSGA-II. The NSGA-II can therefore still perform satisfactorily even when restricted computationally.



(a) *NSGA-II* (b) *NSDE*

(c) *OMOPSO*

FIGURE 7.17: *The frequencies with which the parameter combinations of (a) the NSGA-II, (b) NSDE, and (c) OMOPSO are either ranked first or are statistically equivalent to the combination that is ranked first.*

In the case of NSDE, the best performing parameter combination in terms of sample median is D7. Based on this result, it may be inferred that a large amplification factor of $F \in [0.60, 1.40]$ results in the most statistically significant improvement over its counterparts. There is, however,

a marked increase in performance sensitivity to parameter changes when compared with the NSGA-II. There are, in fact, seven data sets for which the best performing combination D7 outperforms the worst performing combination D2 — this accounts for a third of the data sets for which statistical differences exist at a 5% level of significance.

Lastly, in the case of OMOPSO, the parameter combination P11 is identified as the best performer in terms of sample median. It may therefore be inferred that a large mutation probability of $p_m = 0.10$ leads to the most statistically significant improvement. When considering the performance variation between different parameter combinations, OMOPSO is evidently the sub-algorithm that exhibits the most sensitivity — a total of nine data sets account for the difference between the best performing combination P11 and the worst performing combination P2. Table 7.12 contains a summary of the final parameter values assigned to each sub-algorithm.

TABLE 7.12: *Best performing sub-algorithm parameter combinations identified.*

Sub-algorithm	Parameter	Value
NSGA-II	p_c	1.00
	p_m	0.05
NSDE	C_R	0.30
	F	[0.60, 1.40]
OMOPSO	W	1
	C_1	1.50
	C_2	1.50
	p_m	0.10

An argument can be made that the inferences drawn from the remaining parameter combinations which exhibit good performance for each sub-algorithm should also be incorporated. It is, however, unclear whether these combined "wholesale" changes would improve or worsen performance. In order to ascertain this, a full factorial experimental design is required, in which all possible parameter combinations are considered — a recommendation for future work. It should also be noted that the respective parameter improvements for the three sub-algorithms do not result in a significant change in performance. In the case of the NSGA-II, the largest F_1-score performance differential is 0.0821 in terms of sample mean, which occurs in respect of data set C4. In the case of NSDE, data set C7 accounts for the largest F_1-score performance differential of 0.1043 (also in terms of sample mean). Finally, in the case of OMOPSO, the most significant difference in F_1-score performance occurs for data set C15 where there is a difference of 0.0927 between the best and worst performing combinations. This, together with the fact that no statistical difference exists for many data sets (as summarised in Table 7.5), may suggest that the current range of sub-algorithm parameter values adopted in the literature (within the given optimisation context) do not allow for substantial changes in performance — greater insight may be acquired when performance varies considerably. This matter may therefore also be interesting to resolve as part of future work.

7.3.2 BOHTA parameter evaluation

After establishing good parameter values for the NSGA-II, NSDE, and OMOPSO, the focus now shifts to the BOHTA. To the best of the author's knowledge, the application of a hyperheuristic such as the BOHTA (or its muse, *i.e.* AMALGAM) to the problem at hand is an unaddressed matter in the literature. An algorithmic parameter evaluation is therefore performed so as to

identify appropriate values which afford the best computational capability to the BOHTA — to afford each sub-algorithm and, as a result, the BOHTA a reasonably good opportunity to perform at (or close to) its potential best. Two main parameters under investigation are: (1) The minimum number of solutions a sub-algorithm can generate, and (2) the overall population size. The findings of the sub-algorithm parameter evaluation in §7.3.1 facilitate the process of identifying good parameter values for the BOHTA.

Recall from §4.4 that Vrugt and Robinson [171] enforced (in the original implementation of AMALGAM) a constraint on the minimum number of solutions that any sub-algorithm can generate. The reasoning behind their inclusion of this constraint was to prevent any sub-algorithm from being entirely deactivated during the execution of the hyperheuristic. The notation employed to denote the population size of each sub-algorithm is modified so as to account for the fact that each sub-algorithm can have a varying population size during the execution of the BOHTA. Therefore, N_t^i denotes the population size of sub-algorithm $i \in \{G, D, P\}$ during generation t, which is subject to the condition $N_t^G + N_t^D + N_t^P = M$. The minimum constraint enforced by Vrugt and Robinson was $N_t^i \geq 5$, which, considering the adopted population size of $M = 100$, equates to 5% of the total population size. For the sake of brevity, let $\tilde{N}$ denote the minimum population size proportion imposed on N_t^i, *i.e.* $N_t^i \geq \lfloor \tilde{N} M \rfloor$. In the original implementation of AMALGAM, the value $\tilde{N} = 0.05$ was assumed, implying that $N_t^i \geq 0.05(100) = 5$. Vrugt and Robinson did not, however, provide any motivation for their choice of $\tilde{N} = 0.05$. Due to the obscurity surrounding this decision, a more in-depth analysis is warranted.

The subsequent parameter evaluation will therefore investigate which values of $\tilde{N}$ deliver good hyperheuristic performance. Three values, which are judged to be of a small, medium, and large magnitude, are again considered. The lower bound proportion of $\tilde{N} = 0.05$ employed by Vrugt and Robinson is arguably small considering the population size of $M = 100$. Allowing a sub-algorithm to generate only five solutions can be considered a stringent lower bound and, as a result, it is classified as the small-magnitude value in the subsequent parameter evaluation. The other two magnitude choices, *i.e.* medium- and large-magnitude, are $\tilde{N} = 0.10$ and $\tilde{N} = 0.15$, respectively, which are reasonably intuitive choices based on the premise that $\tilde{N} = 0.05$ is regarded as the small-magnitude value.

The other parameter under investigation is the population size of the BOHTA. Four values, which are judged to be of a small, medium, large and very large magnitude, are now considered. In the case of NSDE and OMOPSO, the performance of parameter combinations D3 and P3 (where $N_t^D = 100$ and $N_t^P = 100$, respectively) indicates that a large[4] population size leads to statistically significant performance improvements, as indicated in Figure 7.17. A reasonably good (large magnitude) value option for the subsequent evaluation is therefore $M = 100$. The process of choosing other appropriate parameter values is facilitated by the consideration of two contrasting conceptual scenarios. These extreme, yet plausible, scenarios are:

(1) **A single sub-algorithm dominates throughout:** Sub-algorithm $i \in \{G, D, P\}$ dominates (or outperforms) both sub-algorithms $j, k \in \{G, D, P\} \backslash \{i\}$ throughout the execution of the BOHTA. Sub-algorithm i therefore generates approximately $N_t^i = M(1 - 2\tilde{N})$ solutions, whereas sub-algorithm j and sub-algorithm k each generates approximately $N_t^j = N_t^k = \tilde{N} M$ solutions.

(2) **All three sub-algorithms perform equally throughout:** Sub-algorithms i, j, and k perform equally well, and so each generates approximately $N_t^i = N_t^j = N_t^k = M/3$ solutions.

[4] Large within the context of the NSGA-II, NSDE and OMOPSO literature.

By considering these two scenarios, a more informed decision can be made with respect to the choice of appropriate BOHTA population size values that should form part of the subsequent evaluation. A population size of $M = 100$ is deemed a reasonably good option based on the findings of the sub-algorithm parameter evaluations in the previous section. In the case of the first scenario, a population size of this magnitude together with a lower bound proportion of $\tilde{N} = 0.05$ should enable the dominant sub-algorithm to generate a population that is reasonably large, *i.e.* containing $M = 90$ solutions, so that performance is at (or close to) its best. For larger lower bound proportions, *i.e.* $\tilde{N} = 0.10$ and $\tilde{N} = 0.15$, it becomes less reasonable to expect that sufficient computational capability is afforded to the dominant sub-algorithm to perform at (or close to) its best. In order to address this issue, a larger population size should be considered and included in the evaluation.

The second scenario aids the process of choosing an even larger magnitude value. Given the premise of the second scenario, a population size of $M = 150$ is deemed sufficiently large as it allows all three sub-algorithms to generate $N_t^G = N_t^D = N_t^P = 50$ solutions, which can therefore potentially enable the BOHTA to perform at (or close to) its best — the performance achieved by the corresponding parameter combinations G1, D1, and P1 certainly warrants the consideration of $M = 150$ in the context of the BOHTA. Conversely, a population size of this magnitude should (in the case of the first scenario) also afford the dominant sub-algorithm with sufficient computational capability, even for larger lower bound proportions, *i.e.* $\tilde{N} = 0.10$ and $\tilde{N} = 0.15$.

A smaller magnitude value of $M = 30$ is also deemed a reasonably good option, based on the findings of the sub-algorithm parameter evaluation. Although the performance achieved by parameter combinations D2 and P2 (presented in Figure 7.17) indicates that a small population size of $N_t^D = 30$ and $N_t^P = 30$ is less favourable, both parameter combinations are still statistically indistinguishable in respect of a majority of the data sets. Furthermore, the parameter combination $G2$ is the second-best performing combination for the NSGA-II, demonstrating its versatility. A small population size of $M = 30$ warrants consideration in the context of the BOHTA — under the constrained circumstances the collaborative computational effort exerted by the three sub-algorithms ought to be revelatory. A natural and intuitive medium-magnitude value is $M = 65$, which represents the mid-point between the small and large values. Although this value represents a slight deviation from the norms established in the literature (as summarised in Table 7.1), the uncertainty surrounding the approach adopted by Vrugt and Robinson [171], together with the novelty of the problem at hand, certainly warrants this somewhat unconventional approach.

Table 7.13 contains a summary of the parameter values considered during the subsequent evaluation. The corresponding parameter value combinations are furthermore presented in Table 7.14. A full factorial design, as opposed to a sensitivity analysis, is performed because of the relatively small size of the experimental design (*i.e* only the two parameters M and $\tilde{N}$, comprising four and three possible values, respectively, are evaluated). Hence there are $4 \times 3 = 12$ different possible parameter combinations.

TABLE 7.13: *Different BOHTA parameter values considered. Population sizes judged to be small, medium, large, and very large are considered, whereas lower bound proportions judged to be small, medium, and large are considered.*

	Parameter value			
Parameter	Small	Medium	Large	Very-large
M	30	65	100	150
$\tilde{N}$	0.05	0.10	0.15	—

TABLE 7.14: *Different parameter combinations for the BOHTA parameter evaluation.*

Combination	M	$\tilde{N}$
B1	30	0.05
B2	65	0.05
B3	100	0.05
B4	150	0.05
B5	30	0.10
B6	65	0.10
B7	100	0.10
B8	150	0.10
B9	30	0.15
B10	65	0.15
B11	100	0.15
B12	150	0.15

The relative performance of each parameter combination is evaluated individually for each respective data set. Each sample, which corresponds to a parameter combination in Table 7.14, comprises thirty optimisation runs per data set, as was the case for the sub-algorithm parameter evaluation in §7.3.1. This approach allows for an extensive measure of explicit performance in respect of each data set, facilitating an insightful evaluation and subsequent analysis. The full set of results, containing the sample medians and sample means for each of the parameter combinations and data sets, can be found in Appendix B. A visual analysis once again precedes an extensive statistical analysis and is facilitated by the box plots in Figures 7.18–7.22. When comparing the box plots of the performance of the BOHTA with those of the individual sub-algorithms in Figures 7.2–7.16, it is apparent that the BOHTA achieves improved sample means and medians. Notable improvements with respect to both the lower and upper quartile ranges can also be observed. Furthermore, improved sample minima and maxima showcase the performance advantages of the BOHTA. When comparing the BOHTA F_1-score sample medians and sample means averaged across *all* parameter combinations and *all* data sets — *i.e.* 0.8525 and 0.8476, respectively — with those of the three sub-algorithms, the performance improvements presented in Table 7.15 are achieved. The BOHTA therefore outperforms (on average) each of the three sub-algorithms. Statistical analyses are, however, necessary in order to determine whether the performance improvements are statistically significant at a 5% level of significance, a matter addressed later in the dissertation.

TABLE 7.15: *Sample median and mean F_1-score improvements achieved by the BOHTA (averaged across all parameter combinations and all data sets), denoted by $\bar{F}_1$.*

	$\bar{F}_1$-score improvements	
	Median	Mean
NSGA-II	0.0368	0.0508
NSDE	0.0708	0.0832
OMOPSO	0.0789	0.0864

Based on the box plots, the performance of the BOHTA would seem to exhibit the same level of performance sensitivity to the parameter combination employed as did the constituent sub-algorithms. Consider, for example, data set C1. The sample mean (and median) differs notably between parameter combinations B1 and B11 — it may be inferred that the larger population size of $M = 100$ and the larger lower bound proportion of $\tilde{N} = 0.15$ result in a performance

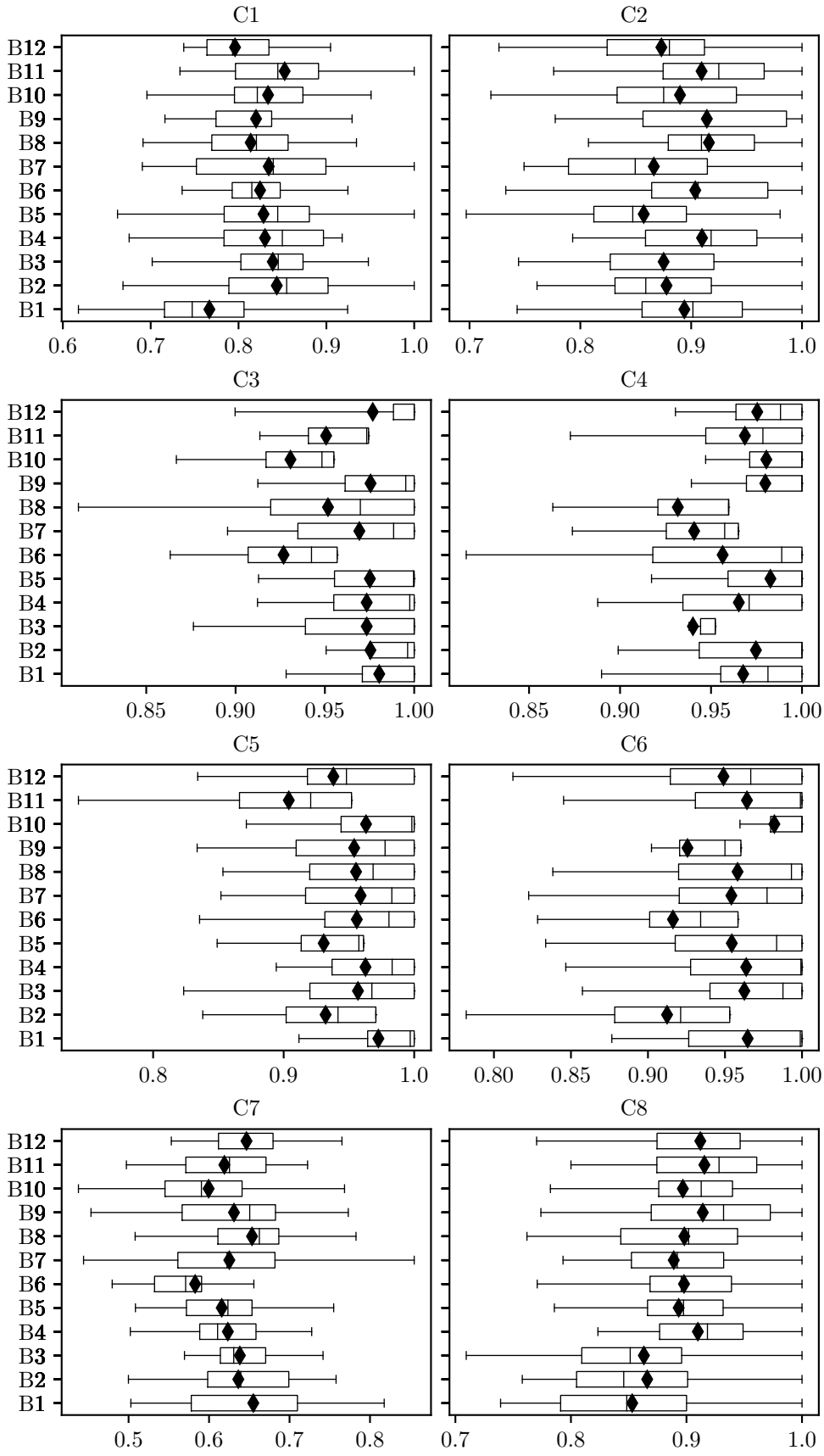


FIGURE 7.18: *Box plots of the F_1-scores achieved by each BOHTA parameter combination (presented in Table 7.14) with respect to data sets C1–C8.*

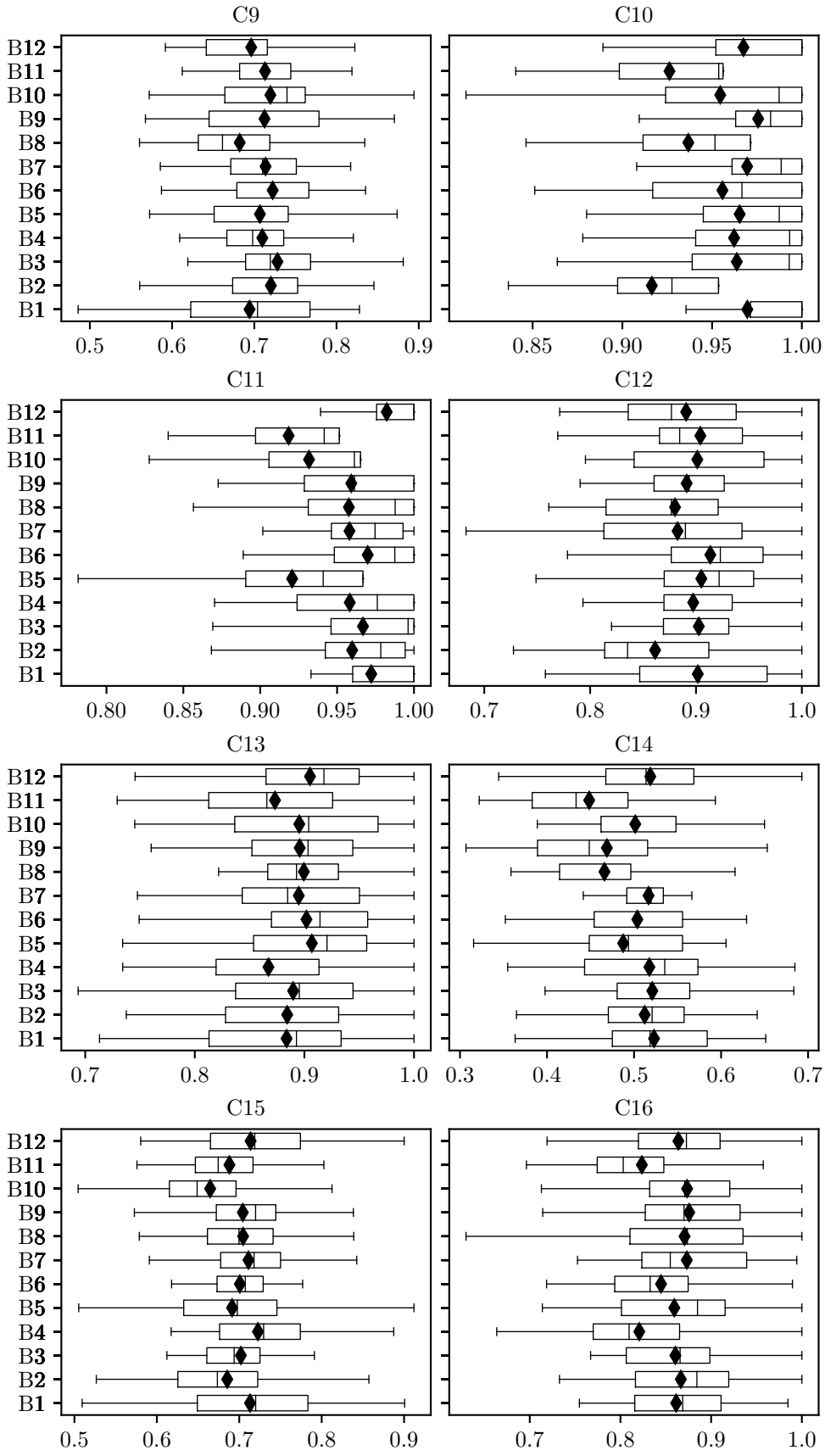


FIGURE 7.19: *Box plots of the F_1-scores achieved by each BOHTA parameter combination (presented in Table 7.14) in respect of data sets C9–C16.*

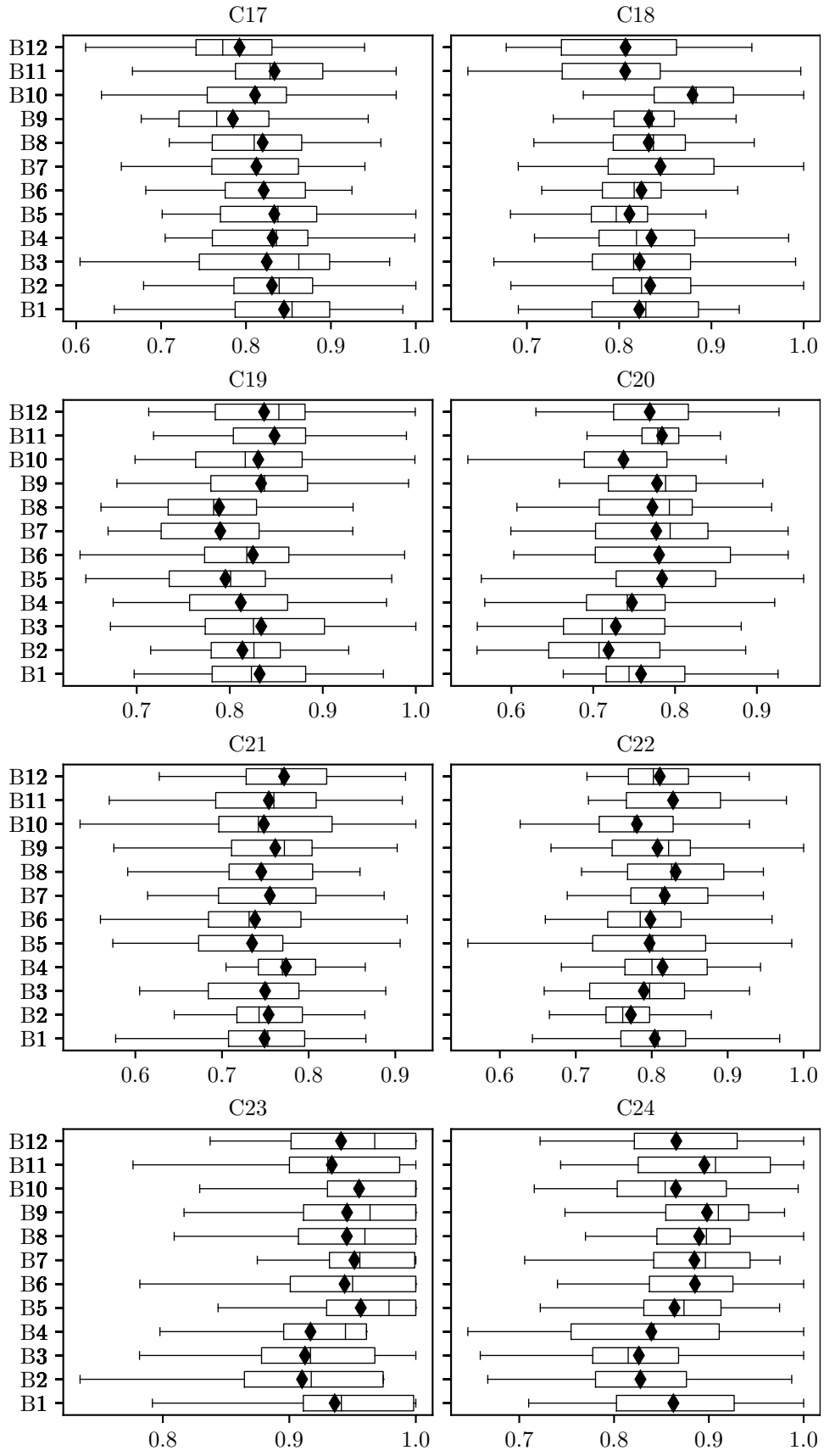


FIGURE 7.20: *Box plots of the F_1-scores achieved by each BOHTA parameter combination (presented in Table 7.14) in respect of data sets C17–C24.*

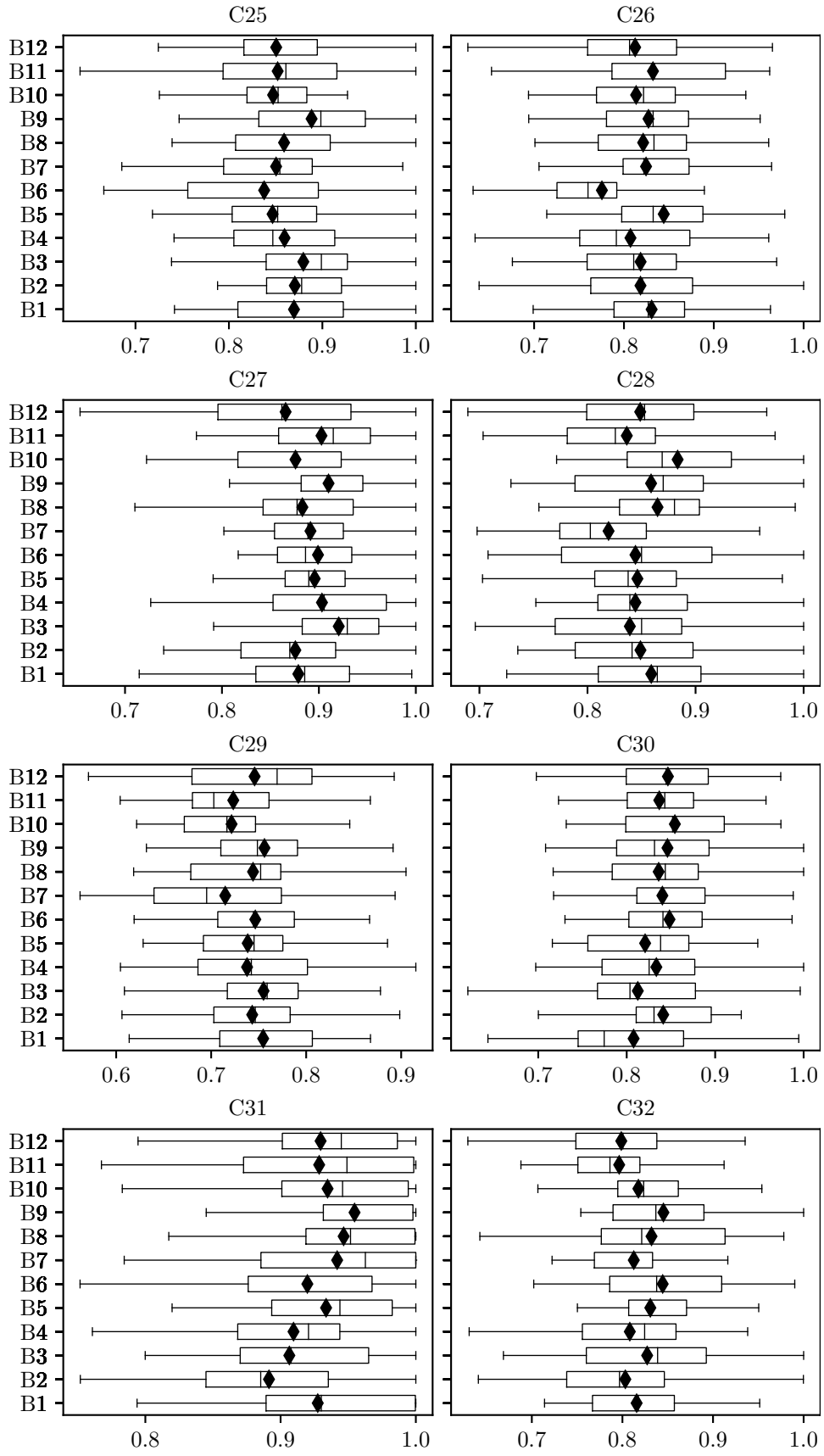


FIGURE 7.21: *Box plots of the F_1-scores achieved by each BOHTA parameter combination (presented in Table 7.14) in respect of data sets C25–C32.*

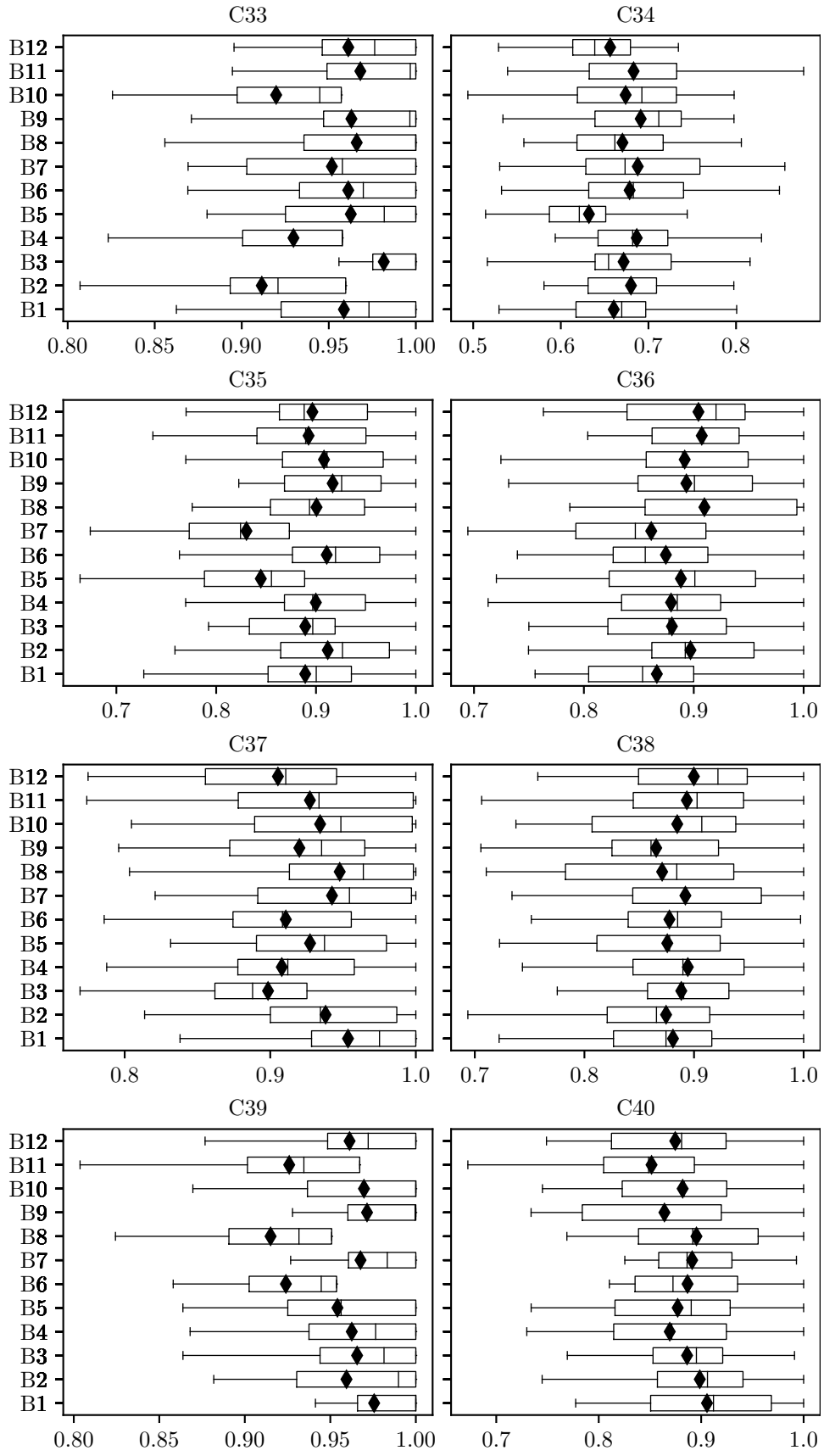


FIGURE 7.22: *Box plots of the F_1-scores achieved by each BOHTA parameter combination (presented in Table 7.14) in respect of data sets C33–C40.*

improvement. Another example of varying performance levels relates to data set C35, for which parameter combination B9 results in a considerable improvement in performance over combination B7 — interestingly, it may be inferred that the smaller population size of $M = 30$ is mitigated by the larger lower bound proportion of $\tilde{N} = 0.15$. The different parameter combinations under consideration (summarised in Table 7.14) therefore result in varying levels of performance which, consequently, facilitates the extraction of insight from the algorithmic performance data.

The same statistical procedure followed in §7.3.1 is also adopted in this context so as to determine which BOHTA parameter combinations result in statistically significant performance improvements. Accordingly, the Friedman test is performed in order first to determine whether a significant difference exists between at least two samples (*i.e.* parameter combinations) and then, if the Friedman test detects such a difference, the Nemenyi *post hoc* procedure is performed so as to identify the specific pairs of samples in which the differences are present. The p-values obtained upon performing the Friedman test on the respective samples for each data set are presented in Table 7.16. There are 21 data sets for which the Friedman test detected at least two samples that are statistically different at a 5% level of significance. This finding supports the earlier empirical observation (from the box plots) that algorithmic performance is sensitive to the parameters chosen in respect of numerous data sets. The lack of statistically significant differences for the remaining 19 data sets are corroborated by a visual scrutiny of the corresponding box plots in Figures 7.18–7.22. For example, data set C9 exhibits markedly consistent sample medians (and means) across the different parameter combinations. The BOHTA therefore exhibits a degree of robustness which is somewhat similar to the robustness observed in the three sub-algorithms.

TABLE 7.16: *Friedman test p-values for the BOHTA in respect of each data set. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

Friedman test p-values			
Data set	BOHTA	Data set	BOHTA
C1	0.0026	C21	0.9309
C2	0.0070	C22	0.0468
C3	0	C23	0.0037
C4	0	C24	0.0043
C5	0	C25	0.4449
C6	0	C26	0.0743
C7	0.0069	C27	0.1062
C8	0.0042	C28	0.1055
C9	0.3603	C29	0.2215
C10	0	C30	0.7492
C11	0	C31	0.0340
C12	0.1838	C32	0.1794
C13	0.3160	C33	0
C14	0.0021	C34	0.1575
C15	0.1326	C35	0.0004
C16	0.0244	C36	0.1110
C17	0.0842	C37	0.0549
C18	0.0124	C38	0.7004
C19	0.0458	C39	0
C20	0.0278	C40	0.3182

The Nemenyi procedure is consequently applied in respect of each data set for which the corresponding Friedman test p-value (presented in Table 7.16) is less than 0.05. There are twelve different parameter combinations, and so the Nemenyi procedure involves $\binom{12}{2} = 66$ pairwise significance tests. Consider, for example data set C1 for which, according to the Friedman test, a statistical difference exists between at least two samples at a 5% level of significance. The corresponding Nemenyi test p-values are presented in Table 7.17.

TABLE 7.17: *Nemenyi test p-values for the BOHTA in respect of data set C1. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Nemenyi test p-values											
	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
B1	—	0.0007	0.0007	0.0011	0.0021	0.0181	0.0014	0.0164	0.0135	0.0016	0.0001	0.2991
B2		—	0.9857	0.9003	0.7609	0.3075	0.8579	0.3248	0.3612	0.8160	0.5077	0.0190
B3			—	0.8861	0.7473	0.2991	0.8439	0.3161	0.3519	0.8021	0.5192	0.0181
B4				—	0.8579	0.3707	0.9572	0.3902	0.4309	0.9145	0.4309	0.0264
B5					—	0.4739	0.9003	0.4963	0.5427	0.9429	0.3337	0.0413
B6						—	0.4001	0.9714	0.9145	0.4309	0.0924	0.1852
B7							—	0.4205	0.4629	0.9572	0.4001	0.0303
B8								—	0.9429	0.4521	0.0995	0.1736
B9									—	0.4963	0.1152	0.1521
B10										—	0.3707	0.0346
B11											—	0.0026
B12												—

The best parameter combinations are established by the same procedure as that followed in §7.3.1. For each data set, the combinations are first ranked according to sample medians, and thereafter the combinations that are statistically equivalent to the best performing combination (*i.e.* the combinations that are statistically indistinguishable according to the Nemenyi procedure) are identified. In the case of the example data set C1, the ranked parameter combinations, together with the respective statistical equivalents of the best performing combination, are presented in Table 7.18.

TABLE 7.18: *BOHTA parameter combinations ranked in descending order of sample median in respect of data set C1. The parameter combinations that are statistically equivalent to the best performing combination (i.e. a combination with a rank of 1) are denoted by matching combination numbers in the column labelled "Stat. Eq."*

Rank	Combination	Median	Stat. Eq.
1	B2	0.8548	B3,B4,B5,B6,B7,B8,B9,B10,B11
2	B4	0.8499	—
3	B3	0.8452	—
4	B11	0.8447	—
5	B5	0.8446	—
6	B7	0.8395	—
7	B10	0.8214	—
8	B8	0.8203	—
9	B9	0.8188	—
10	B6	0.8151	—
11	B12	0.7932	—
12	B1	0.7472	—

The best performing parameter combination in terms of sample median is B2, for which the BOHTA performance is statistically indistinguishable from those for combinations B3, B4, B5, B6, B7, B8, B9, B10, and B11. The statistical equivalence class can be verified simply by observing the Nemenyi test p-values in the relevant row and column of combination B2 in Table 7.17. Finally, the relative algorithmic performance achieved by the different parameter combinations can be obtained by performing the procedure above in respect of all the data sets, *i.e.* C1–C40, and observing both the frequency with which a combination is ranked first and the frequency with which a combination is statistically equivalent to the combination that is ranked first. The respective frequencies for the different parameter combinations of the BOHTA are presented in Figure 7.23.

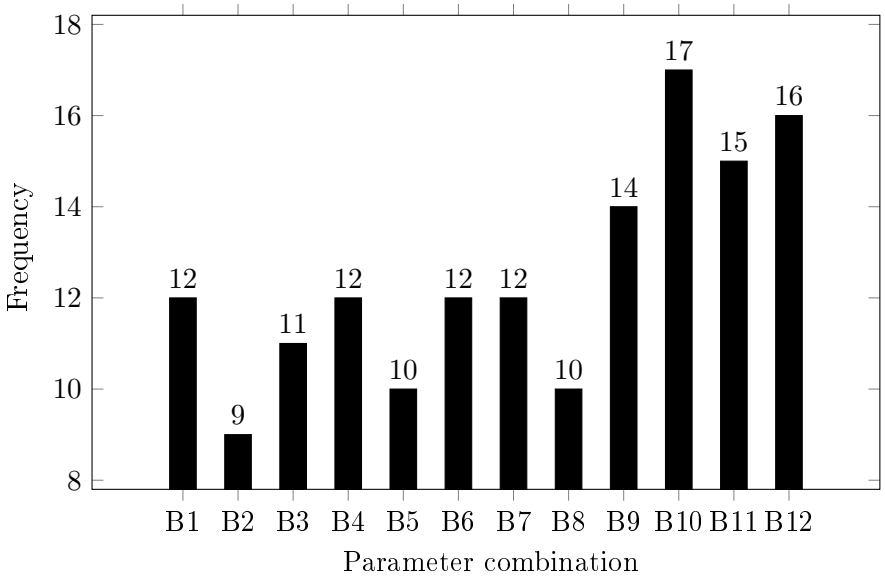
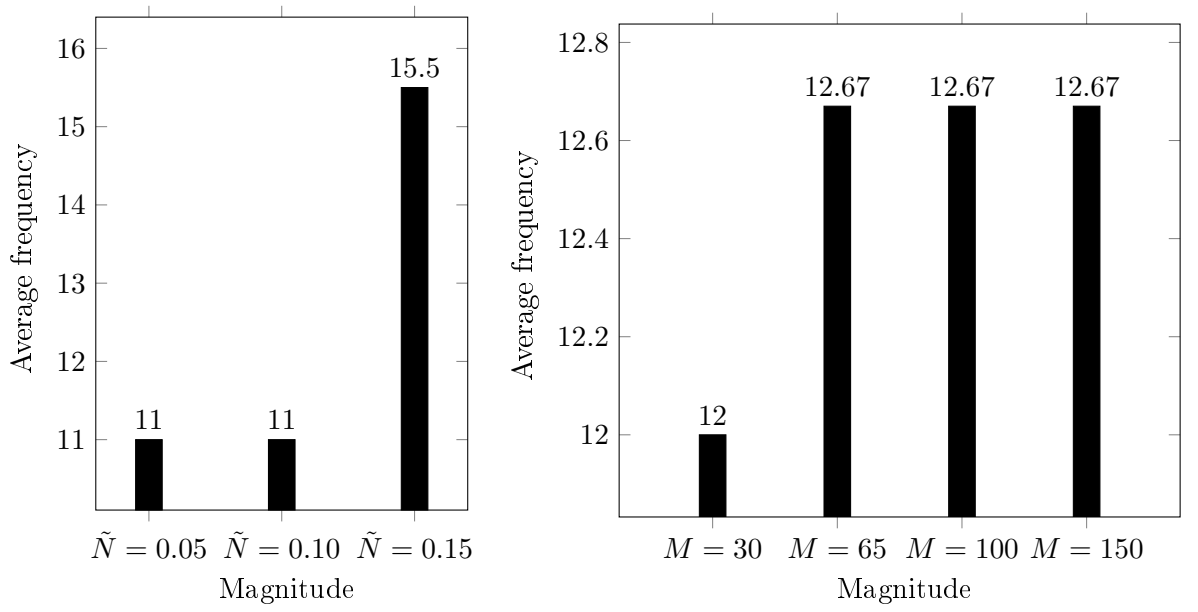


FIGURE 7.23: *The frequencies with which each BOHTA parameter combination is either ranked first or is statistically equivalent to the combination that is ranked first.*

Once again, each parameter combination is statistically indistinguishable in respect of a majority of the data sets. There is, however, a discernible difference between the best performing combination, *i.e.* B10, and the worst performing combination, *i.e.* B2 — a total of eight data sets separate these two combinations. Interestingly, parameter combinations B9–B12, which correspond to a large lower bound proportion of $\tilde{N} = 0.15$, represent the best performing combinations overall. The small and medium lower bound proportions of $\tilde{N} = 0.05$ and $\tilde{N} = 0.10$ each performs consistently worse. Figure 7.24(a) contains an aggregation of the average frequency for the different lower bound proportions. The performance bias towards the large value is perhaps an indication that the worst performing sub-algorithm(s) should not be penalised as severely, therefore allowing these sub-algorithm(s) to continue contributing in terms of generating more solutions.

Interestingly, the performance achieved by the BOHTA exhibits an insensitivity towards different population sizes, as shown in Figure 7.24(b). It may therefore be inferred that the BOHTA can achieve good performance even under computationally constrained circumstances. When comparing the performance of the BOHTA with those of the three sub-algorithms, performance is far less affected by a smaller population size — indicative of the robustness of the BOHTA. When comparing the performance achieved by the best BOHTA parameter combination that employs a small population size of $M = 30$, *i.e.* B9, with those of the NSGA-II, NSDE, and OMOPSO parameter combinations that employ the same population size, *i.e.* G2, D2, and P2, average F_1-score improvements ranging from of 0.0269 to 0.07 are achieved.



(a) *Lower bound proportion* (b) *Population size*

FIGURE 7.24: *The average frequencies with which each BOHTA parameter combination is either ranked first or is statistically equivalent to the combination that is ranked first with respect to the different values for (a) the lower bound proportion and (b) the population size.*

Recall that in the case of AMALGAM's original implementation a lower bound proportion of $\tilde{N} = 0.05$ and population size of $M = 100$ were employed, which corresponds to parameter combination B3 in this context. It is, however, apparent that numerous other parameter combinations result in statistically significant performance improvements when applying a similar hyperheuristic solution methodology in the context of training FNNs.

7.3.3 Gradient-based training algorithm parameter evaluation

Within the context of a gradient-based training approach, the objective (*i.e.* error or cost) function must be differentiable — the method of backpropagation necessitates differentiability in order to calculate the partial derivatives of the objective function with respect to the weights in the network, as discussed in §3.7.1. Without a differentiable objective function, the method of gradient descent cannot be used to calculate the contribution made by each weight to the objective function and therefore cannot adjust the network weights to ultimately minimise the objective function. Consequently, the objective function can only be expressed in terms of the network weights — parameters pertaining to the network structure and the activation function type[5] are therefore excluded. The level of abstraction at which optimisation transpires is correspondingly inhibited by such an approach — hence the proclaimed utility of a meta- and hyperheuristic based training approach throughout this dissertation. Nevertheless, in order to determine suitable parameter values for the gradient-based training algorithms, a number of parameters are fixed *a priori*. Recall from §7.2 that, as part of the experimental setup, upper bounds are imposed on both the network width and network depth — a necessity ascribed to the nature of the BOHTA. Gradient-based training algorithms, on the other hand, necessitate a different approach.

[5]The activation function slope parameters can, depending on the activation function at hand, still be included.

The first set of parameters that are fixed throughout this parameter evaluation relates to the network structure. Recall from §3.1 that, according to one of the most commonly relied upon heuristics, the optimal number of hidden neurons m should be chosen between the size of the input layer n and that of the output layer o [60]. Instead of assigning a value that is between the input- and output layer sizes (as in the case of the aforementioned heuristic), m is assigned as value the arithmetic mean of these two sizes, *i.e.* $m = (n+o)/2$. If this mean is fractional, then it is simply rounded to the nearest integer. Recall from §7.2 that the upper bound on the number of hidden neurons constituting networks generated by the BOHTA (and each of its three sub-algorithms) is $m = 2\max\{n, o\}$. Networks can therefore comprise more or fewer hidden neurons within the meta- and hyperheuristic context. It is, however, important to note that the BOHTA favours smaller network sizes (ascribed to the helper objective function). Therefore the upper bound on m is chosen so as to ensure that if a data set requires more neurons — defying conventional wisdom — the BOHTA can address this requirement accordingly by "switching" on more neurons (by means of the structural variables). Conversely, if fewer neurons are required, the BOHTA "switches" off neurons. It is also important to note that many of the heuristics that prescribe appropriate network sizes (discussed in §7.2) are based on empirical observations made within the context of gradient-based approaches. Setting the network width to the midpoint between n and o is more closely aligned with what is prescribed within a gradient-based context. In a separate qualitative pilot study, carried out by the author, it was found that setting $m = \max\{n, o\}$ within the gradient-based training context generally resulted in overfitting (*i.e.* good performance with respect to the training set but poor performance with respect to the testing set), especially so when compared with setting $m = (n+o)/2$. In the former case, over-parametrisation is evidently a problem. The aforementioned choice of fixing the number of hidden neurons to $m = (n+o)/2$ would therefore seem appropriate in this parameter evaluation.

The number of hidden layers, on the other hand, is fixed according to a widely adopted heuristic which states that two hidden layers are capable of approximating an arbitrary non-linear function and generating any complex decision region. Due to this heuristic also having been corroborated empirically within a gradient-based context, fixing the network depth at two hidden layers is deemed appropriate in this parameter evaluation. In another separate qualitative pilot study conducted by the author, it was found that one hidden layer generally resulted in comparatively poor overall performance (with respect to both the training set and testing set), whereas three hidden layers generally resulted in overfitting. Once again, the lack of a mechanism to adjust the network structure dynamically inhibits a gradient-based approach.

The other exclusion from the objective function relates to the type of network activation function employed. Akin to the network structure, the BOHTA determines appropriate network activation functions (including slope parameters) as part of its optimisation procedure — a characteristic attributable to the flexibility of the meta- and hyperheuristic-based training approach adopted. In the context of a gradient-based training approach, on the other hand, the type of activation function and, if applicable, the associated parameters (*i.e.* slope parameters) are fixed *a priori*. Recall from §5.2.3 that the bi-objective model's activation functional variables (5.7) are inspired by the PReLU activation function (3.6) due to its computational simplicity and favourable performance [59, 75]. Appropriately, the type of activation function employed in this parameter evaluation is the PReLU — an intuitive decision, especially when considering the fact that its favourable performance reported in the literature [59, 75] corresponds to studies conducted in the context of gradient-based approaches[6]. This finding, together with the similarity between the PReLU and the bi-objective model's activation functional variables, contributes towards the

[6]The author could not find conclusive evidence in the literature as to which activation functions result in favourable performance in the context of a meta- and hyperheuristic-based training.

suitability of using PReLU in this parameter evaluation. PReLU is, of course, accompanied by a slope parameter for negative input[7] only, as discussed in §3.2. One of two modelling approaches can be adopted in this parameter evaluation, they are:

(1) Define for all mh hidden neurons within the network the same PReLU activation function together with a single (shared) slope variable α for negative input, *i.e.* $\eta_j^{(f)} < 0$, where $j \in \{1, \ldots, m\}$ and $f \in \{1, \ldots, h\}$, or

(2) define for hidden neuron $j \in \{1, \ldots, m\}$ in hidden layer $f \in \{1, \ldots, h\}$ a PReLU activation function together with its own slope variable $\alpha_j^{(f)}$ for negative input, *i.e.* $\eta_j^{(f)} < 0$.

The latter approach is superior in terms of performance, as reported by He *et al.* [59] — the original authors of PReLU. Furthermore, the respective slope variables are learnt during the training process — the objective function is therefore expressed in terms of both the network weights as well as the respective slope variables and is subjected to the method of backpropagation. Each of the slope variables is initialised as $\alpha_j^{(f)} = 0.25$, as recommended by He *et al.* Furthermore, the performance achieved by PReLU is sensitive to the network weight initialisation procedure adopted. Fortunately, He *et al.* prescribed the following robust initialisation procedure for the PReLU activation function specifically: The (continuous) normal distribution, from which the real-valued random weight values are drawn, must have a mean of zero and a standard deviation of $\sqrt{2/\tilde{m}}$, where $\tilde{m}$ denotes the fan-in number, *i.e.* the number of connections entering the neuron. The weights associated with the biases are drawn from a normal distribution with a standard deviation of 1. Recall from §6.1.2 that the BOHTA adopts the same procedure — ascribed to the similarities between the two approaches in terms of activation functions.

In this parameter evaluation, the objective function to be minimised during the training stage, depends on the type of classification problem at hand. In the case of a binary classification problem, the negative log-likelihood cost of the *Bernoulli* distribution (3.20) (*i.e.* cross-entropy) is used, whereas in the case of a multi-class classification problem, the negative log-likelihood cost of the multinomial distribution (3.21) is used. As discussed in §3.6, gradient-based training algorithms perform notably better when these cost functions are used. The final algorithmic performances achieved by the networks trained by the gradient-based algorithms are also calculated using the F_1-score performance measure. This facilitates a meaningful algorithmic performance comparison between the BOHTA and the gradient-based algorithms. Furthermore, each of the parameter evaluations (*i.e.* for SGD, RMSProp, and Adam) comprises thirty optimisation runs per data set. Gradient-based algorithms are implemented using the open-source framework titled *Keras* [20].

Recall from §5.3.2 that the BOHTA incorporates a secondary, helper objective function to bias its search towards favouring smaller network structures — a form of regularisation to help mitigate overfitting. Popular methods of regularisation in a conventional gradient-based training paradigm include L^1 and L^2 parameter regularisation as well as early stopping, as discussed in §3.8. In order to ensure a fair performance comparison between the BOHTA and the gradient-based training algorithms, the latter should also incorporate regularisation. Towards this end, favourable results were found during a separate qualitative pilot study conducted by the author when using L^2 parameter regularisation as opposed to L^1 parameter regularisation. In addition, early stopping is also employed to help mitigate overfitting. The same patience parameter value adopted by the BOHTA approach (*i.e.* five) was found to deliver favourable results with respect to the gradient-based training algorithm.

[7] A slope parameter value of one is used for non-negative input.

After establishing which parameter values are fixed and assigning appropriate values to these parameters, the focus now turns to the parameters that form part of the parameter evaluation. A full factorial experimental design is conducted so as to determine good parameter combinations for the gradient-based training algorithms. A full factorial design, as opposed to a sensitivity analysis, is performed because of the relatively small size of the experimental design. In the case of SGD, the algorithm-specific parameters are the learning rate κ and momentum parameter μ. In the case of RMSProp, the relevant parameters include the learning rate κ and exponential decay rate ϱ. Lastly, in the case of Adam, the relevant parameters are the learning rate κ and the two exponential decay rates ϱ_1 and ϱ_2. For each of these parameters, three values, judged to be of a small, medium, and large magnitude, are considered. A summary of these parameter values is presented in Table 7.19. Note that the parameters ϱ (for RMSProp) and ϱ_1 (for Adam) are grouped together — a decision based on the fact that this exponential decay rate is common amongst both algorithms, as discussed in §3.7.3. The different parameter combinations for SGD, RMSProp, and Adam are presented in Table 7.20. Note that in the case of Adam, the exponential decay rate ϱ_1 is fixed throughout the nine different parameter combinations because of the prevalence of the value of $\varrho_1 = 0.9$ in the literature pertaining to Adam (sources are presented in Table 7.19).

TABLE 7.19: *Different parameter values, judged to be of a small, medium, and large magnitude, evaluated as part of the gradient-based training algorithm parameter evaluation. Sources from the literature are included.*

	Parameter value			
Parameter	Small	Medium	Large	References
κ	0.001	0.005	0.01	[4, 20, 53, 79, 140]
μ	0	0.50	0.90	[20, 59, 140, 159]
ϱ (or ϱ_1)	0.9000	0.9900	0.9990	[20, 79, 140]
ϱ_2	0.9900	0.9990	0.9999	[20, 34, 53, 79, 140]

TABLE 7.20: *The different SGD, RMSProp, and Adam parameter combinations considered.*

SGD			**RMSProp**			**Adam**			
Combination	κ	μ	Combination	κ	ϱ	Combination	κ	ϱ_1	ϱ_2
S1	0.001	0.00	R1	0.001	0.900	A1	0.001	0.9	0.9900
S2	0.005	0.00	R2	0.005	0.900	A2	0.005	0.9	0.9900
S3	0.010	0.00	R3	0.010	0.900	A3	0.010	0.9	0.9900
S4	0.001	0.50	R4	0.001	0.990	A4	0.001	0.9	0.9990
S5	0.005	0.50	R5	0.005	0.990	A5	0.005	0.9	0.9990
S6	0.010	0.50	R6	0.010	0.990	A6	0.010	0.9	0.9990
S7	0.001	0.99	R7	0.001	0.999	A7	0.001	0.9	0.9999
S8	0.005	0.99	R8	0.005	0.999	A8	0.005	0.9	0.9999
S9	0.010	0.99	R9	0.010	0.999	A9	0.010	0.9	0.9999

Based on the box plots in Figures 7.25–7.39, the performance of the three gradient-based training algorithms would seem to exhibit the same level of performance sensitivity to the parameter combination employed as did the BOHTA and its constituent sub-algorithms. Statistical analyses[8] are therefore conducted so as to determine which SGD, RMSProp, and Adam parameter combinations result in statistically significant performance improvements. Upon applying the

[8]The same non-parametric tests (*i.e.* the Friedman test and Nemenyi *post hoc* procedure) are employed in the context of gradient-based training algorithms. Luengo *et al.* [96] found that parametric tests are generally deemed less appropriate due to the extent of the violation of the *independence*, *normality*, and *heteroscedasticity* assumptions in the context of training ANNs by means of gradient-based approaches. The violation is less significant when employing non-parametric tests. Luengo *et al.* further stated that a separation of the testing

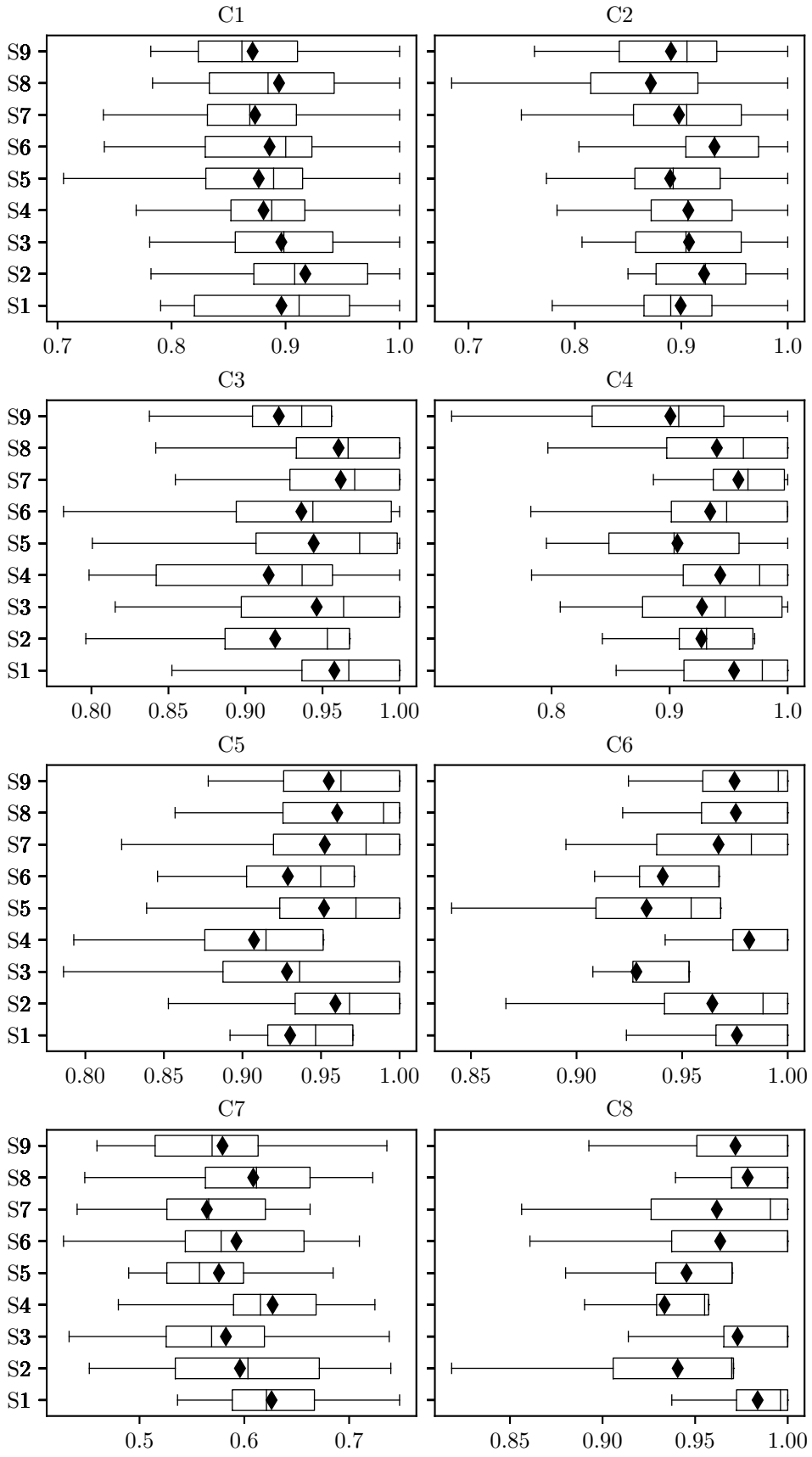


FIGURE 7.25: *Box plots of the F_1-scores achieved by each SGD parameter combination (presented in Table 7.20) with respect to data sets C1–C8.*

set from the training set mitigates the violation of the independence assumption — a convention adopted in this dissertation.

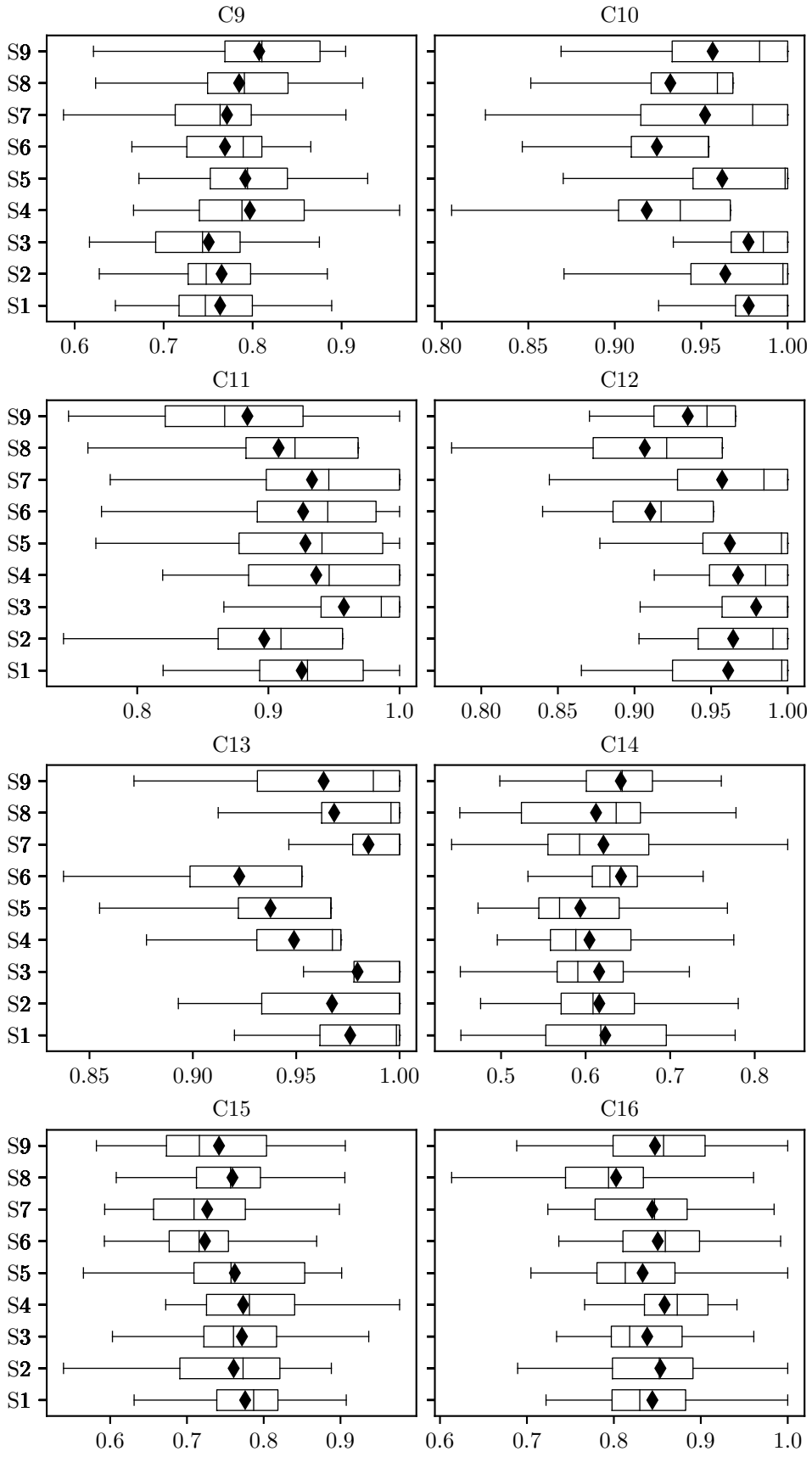


FIGURE 7.26: *Box plots of the F_1-scores achieved by each SGD parameter combination (presented in Table 7.20) in respect of data sets C9–C16.*

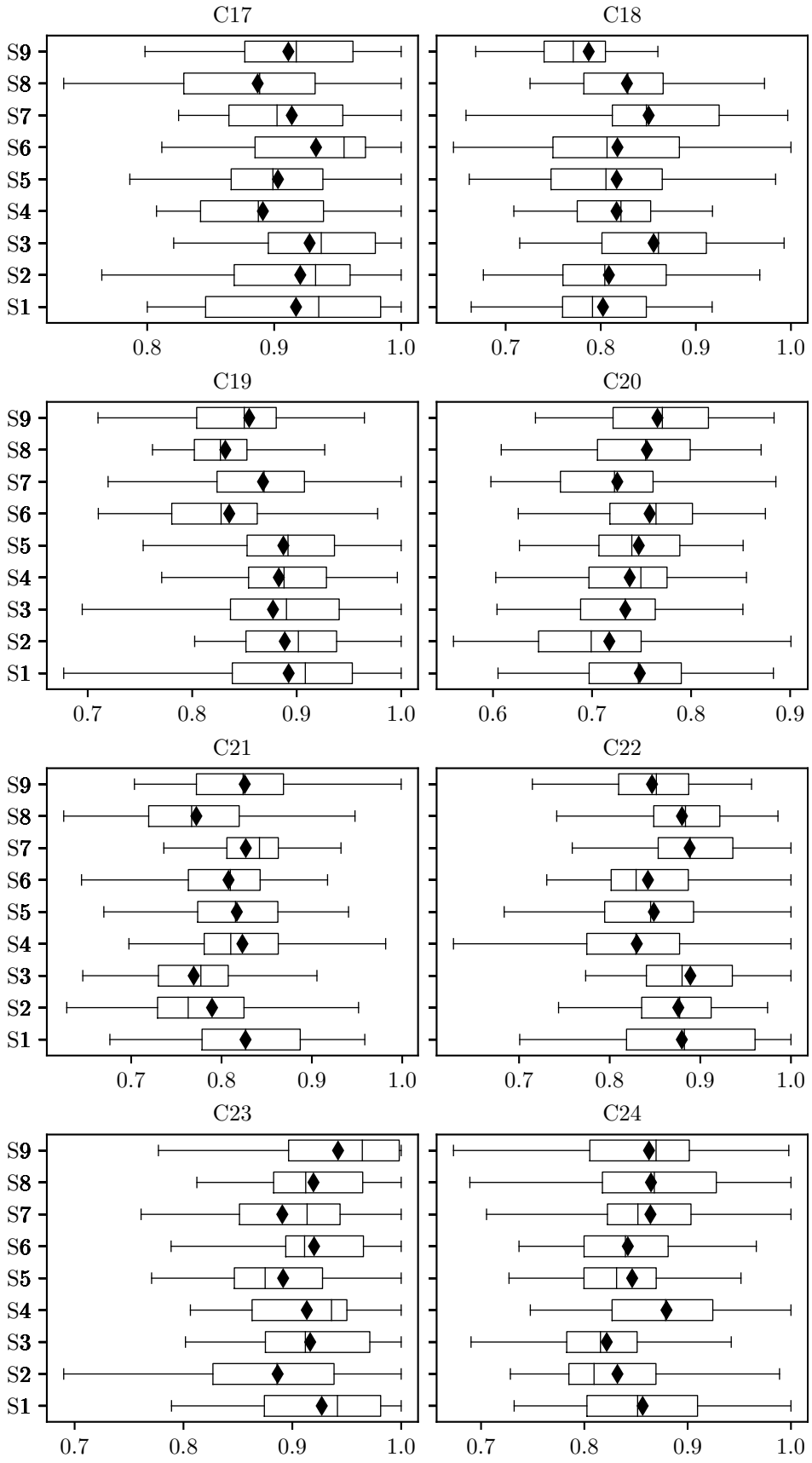


FIGURE 7.27: *Box plots of the F_1-scores achieved by each SGD parameter combination (presented in Table 7.20) in respect of data sets C17–C24.*

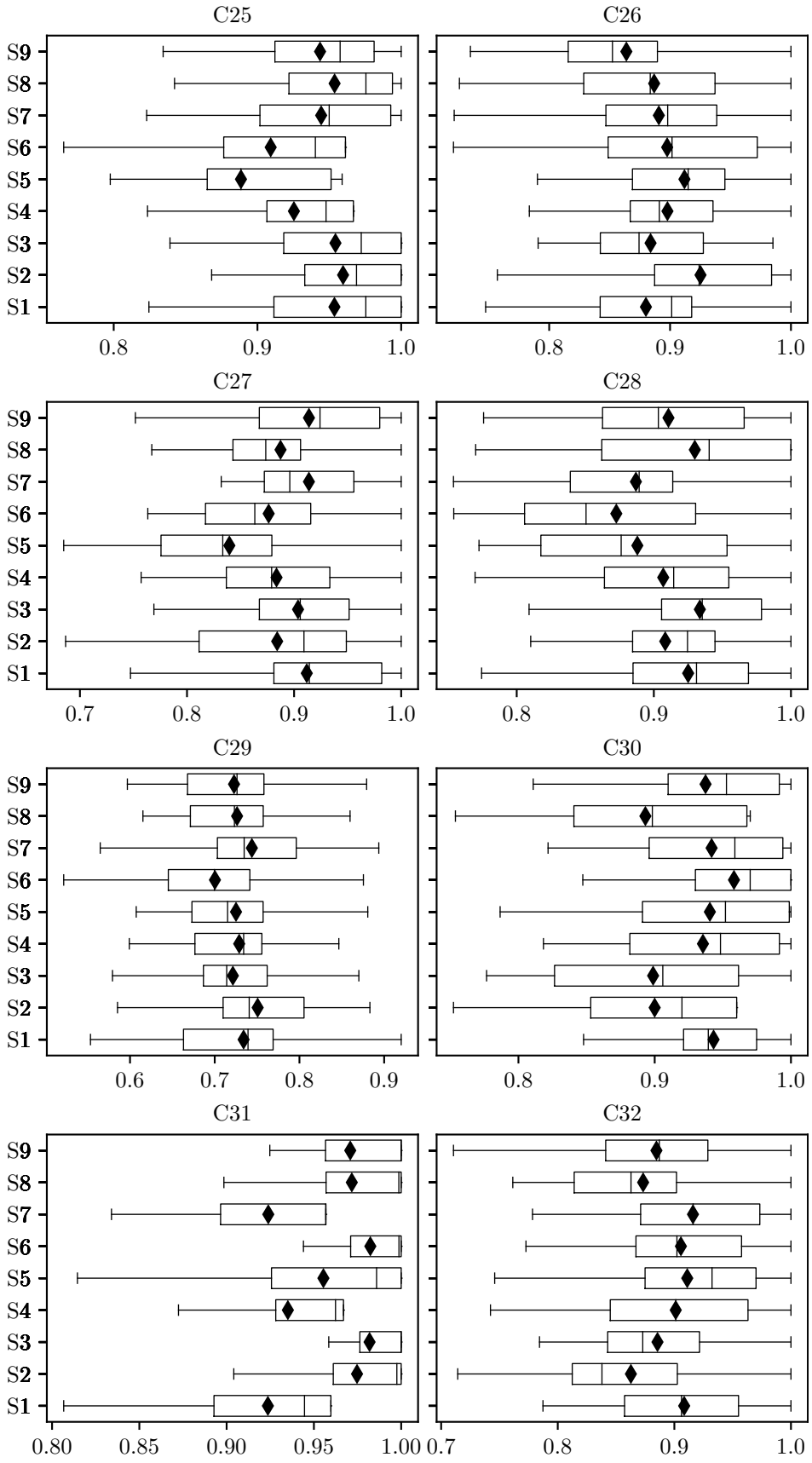


FIGURE 7.28: *Box plots of the F_1-scores achieved by each SGD parameter combination (presented in Table 7.20) in respect of data sets C25–C32.*

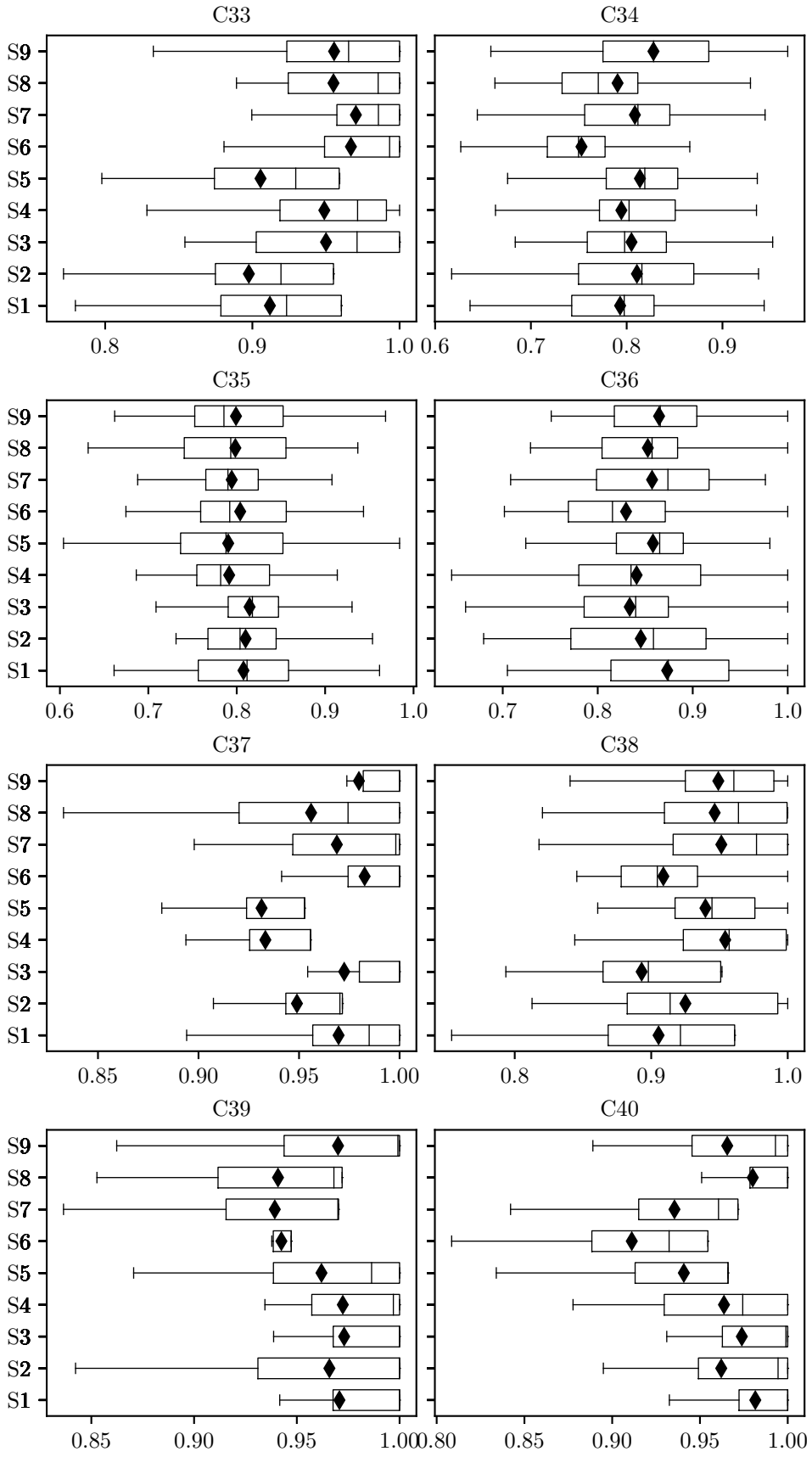


FIGURE 7.29: *Box plots of the F_1-scores achieved by each SGD parameter combination (presented in Table 7.20) in respect of data sets C33–C40.*

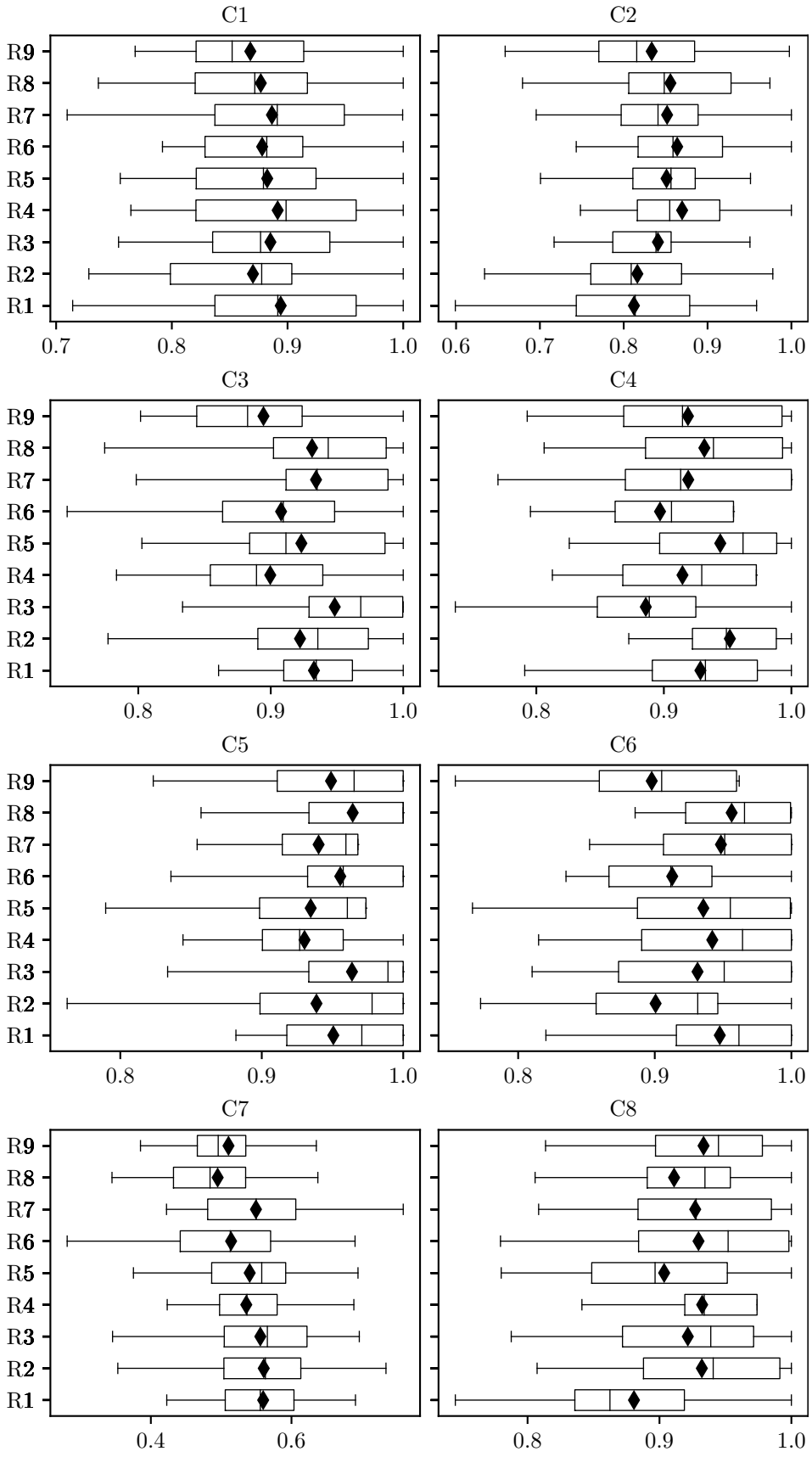


FIGURE 7.30: *Box plots of the F_1-scores achieved by each RMSProp parameter combination (presented in Table 7.20) in respect of data sets C1–C8.*

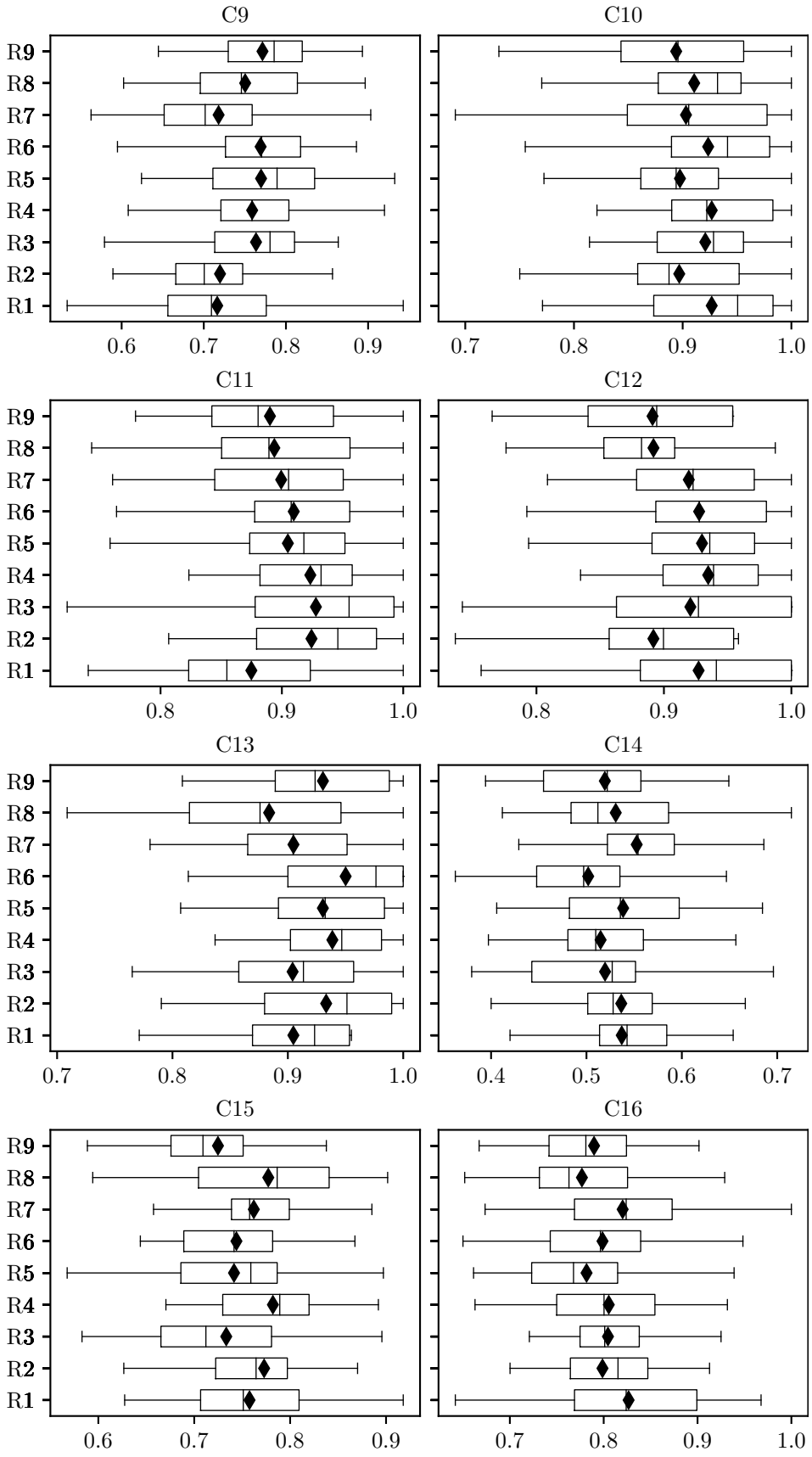


FIGURE 7.31: *Box plots of the F_1-scores achieved by each RMSProp parameter combination (presented in Table 7.20) in respect of data sets C9–C16.*

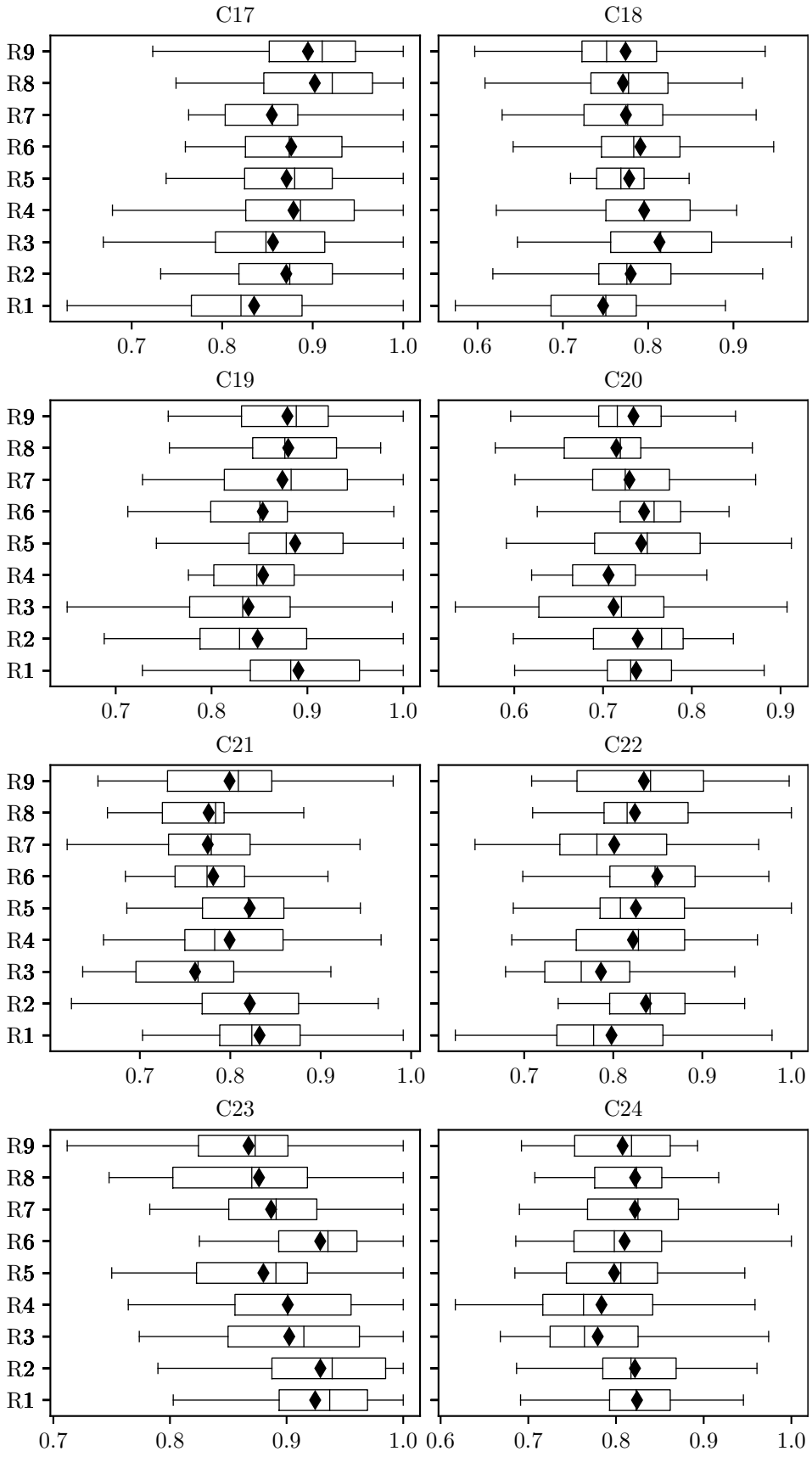


FIGURE 7.32: *Box plots of the F_1-scores achieved by each RMSProp parameter combination (presented in Table 7.20) in respect of data sets C17–C24.*

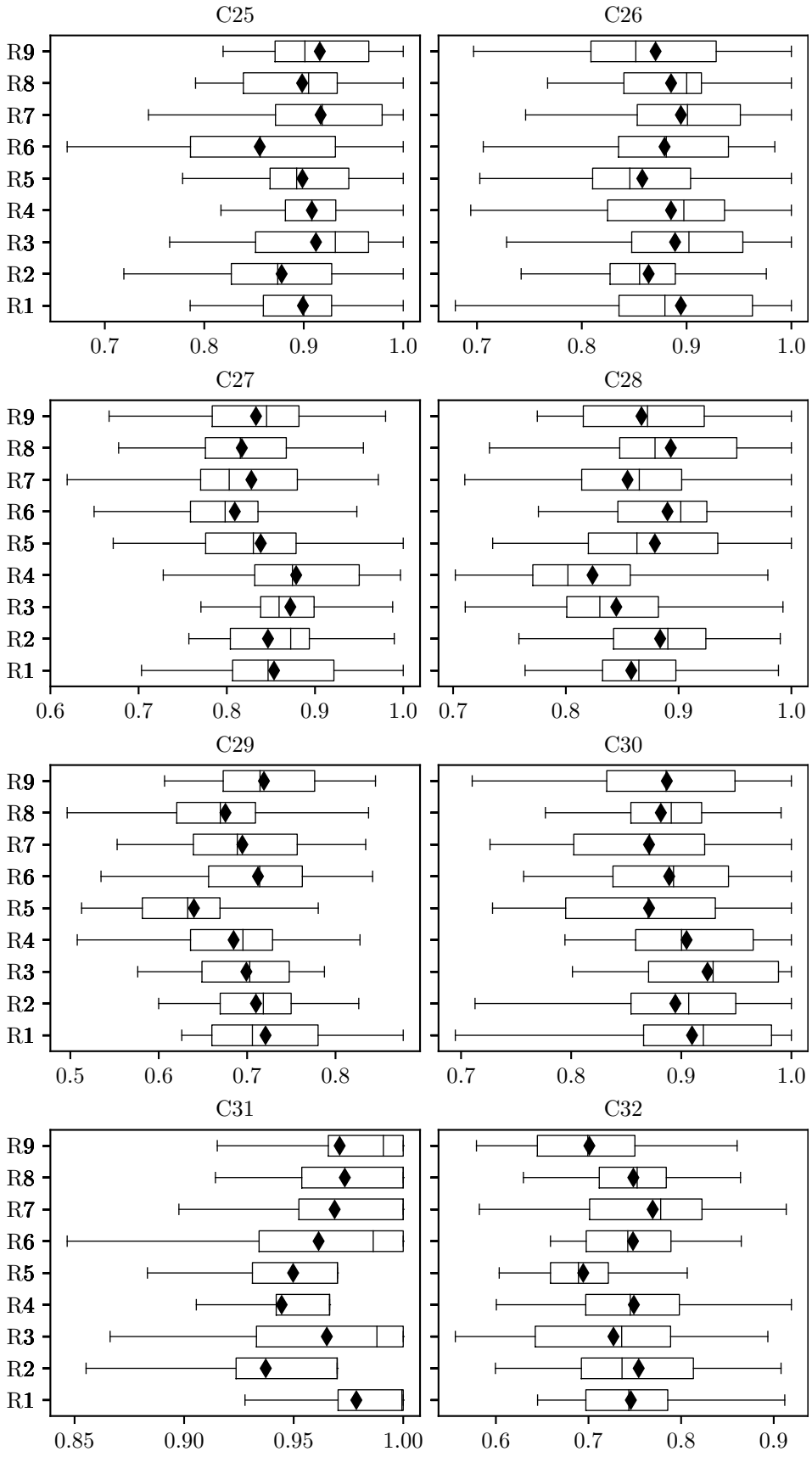


FIGURE 7.33: *Box plots of the F_1-scores achieved by each RMSProp parameter combination (presented in Table 7.20) in respect of data sets C25–C32.*

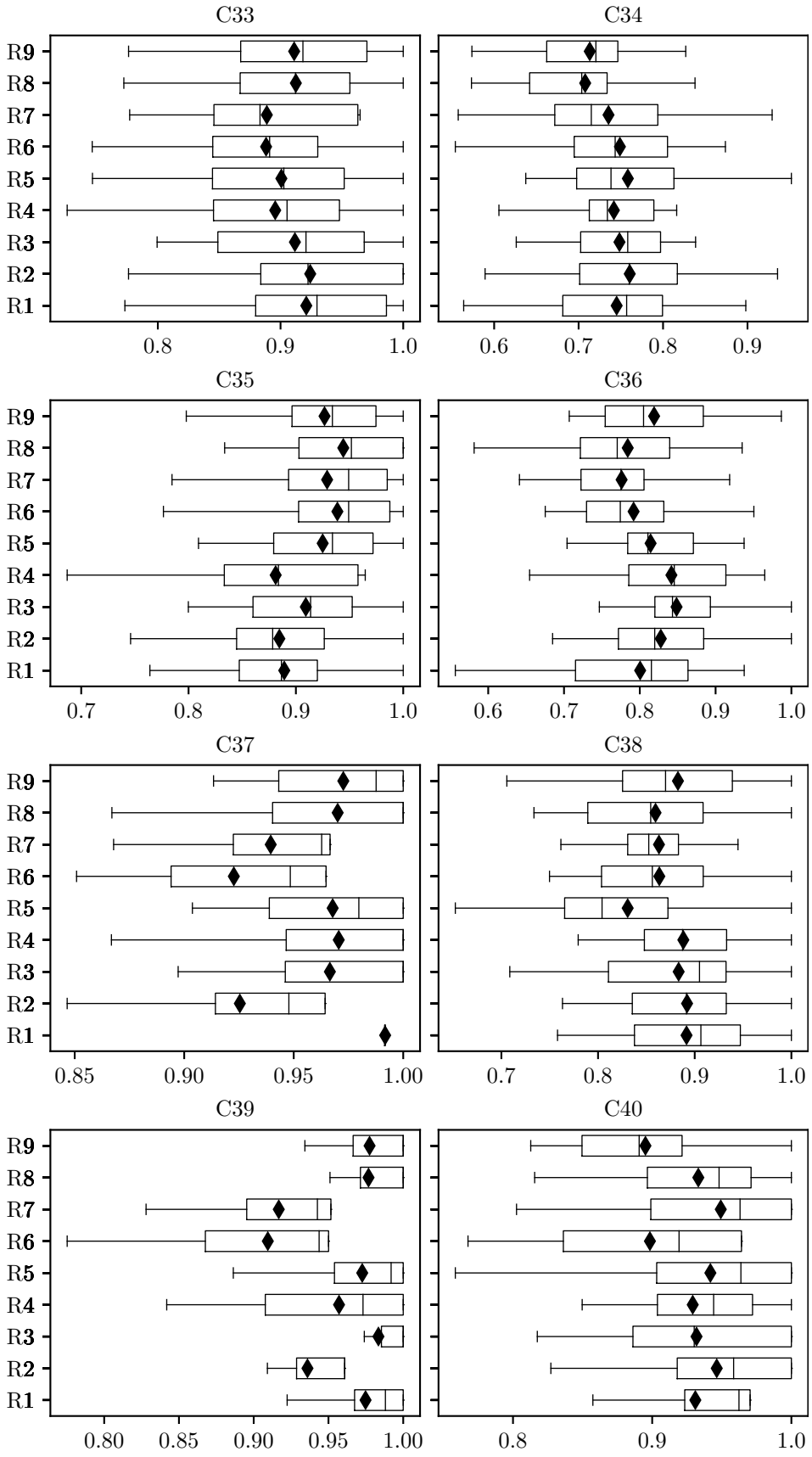


FIGURE 7.34: *Box plots of the F_1-scores achieved by each RMSProp parameter combination (presented in Table 7.20) in respect of data sets C33–C40.*

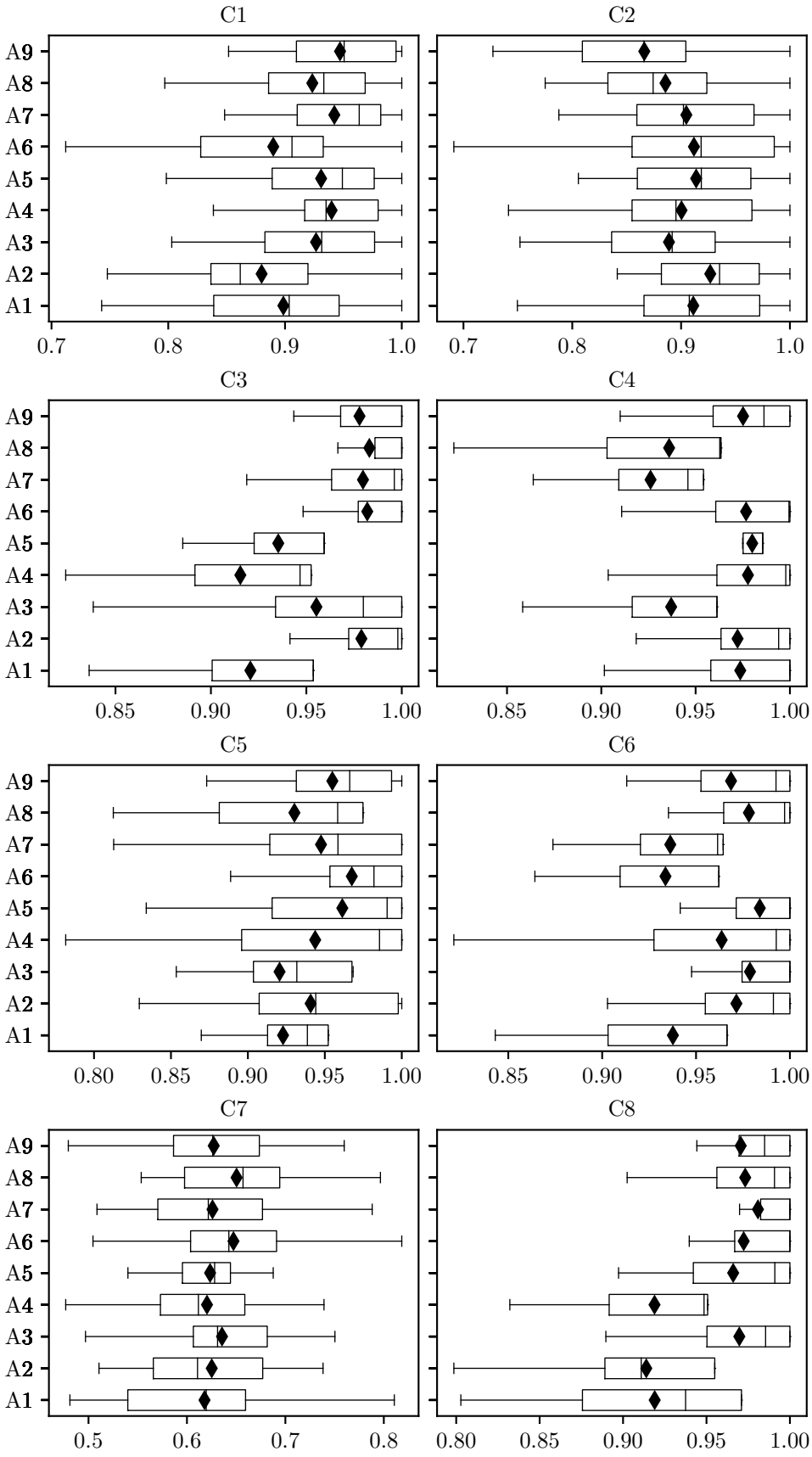


FIGURE 7.35: *Box plots of the F_1-scores achieved by each Adam parameter combination (presented in Table 7.20) in respect of data sets C1–C8.*

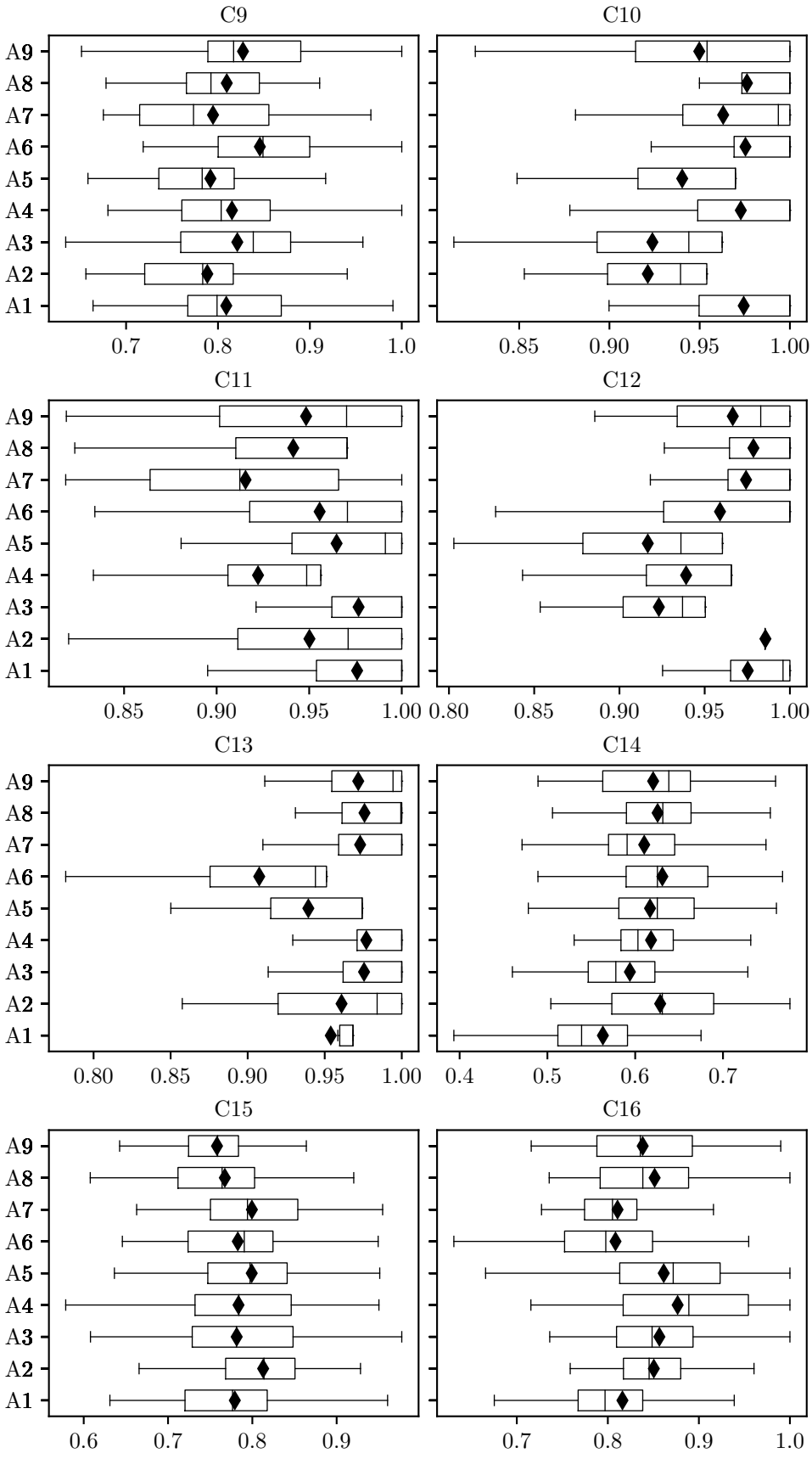


FIGURE 7.36: *Box plots of the F_1-scores achieved by each Adam parameter combination (presented in Table 7.20) in respect of data sets C9–C16.*

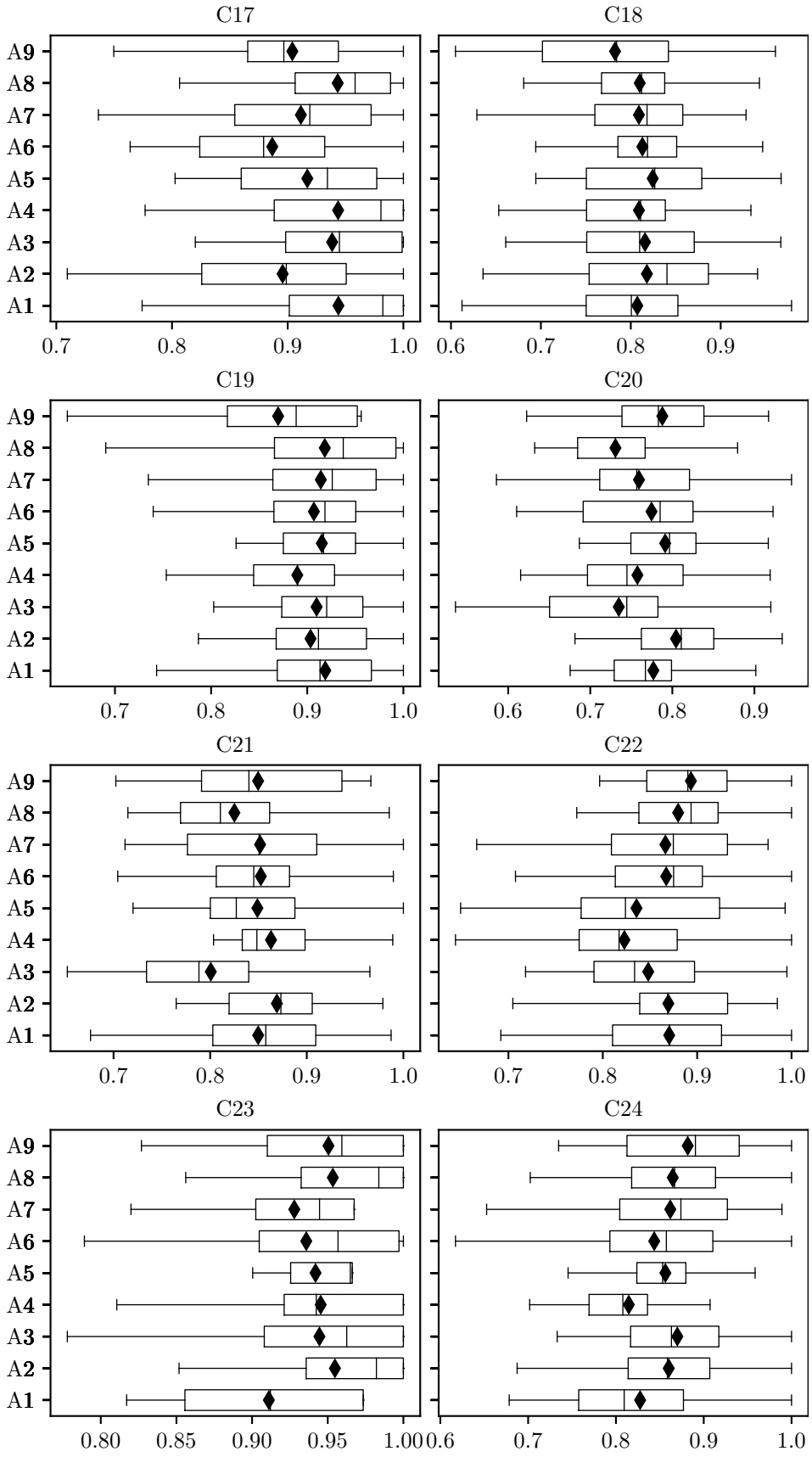


FIGURE 7.37: *Box plots of the F_1-scores achieved by each Adam parameter combination (presented in Table 7.20) in respect of data sets C17–C24.*

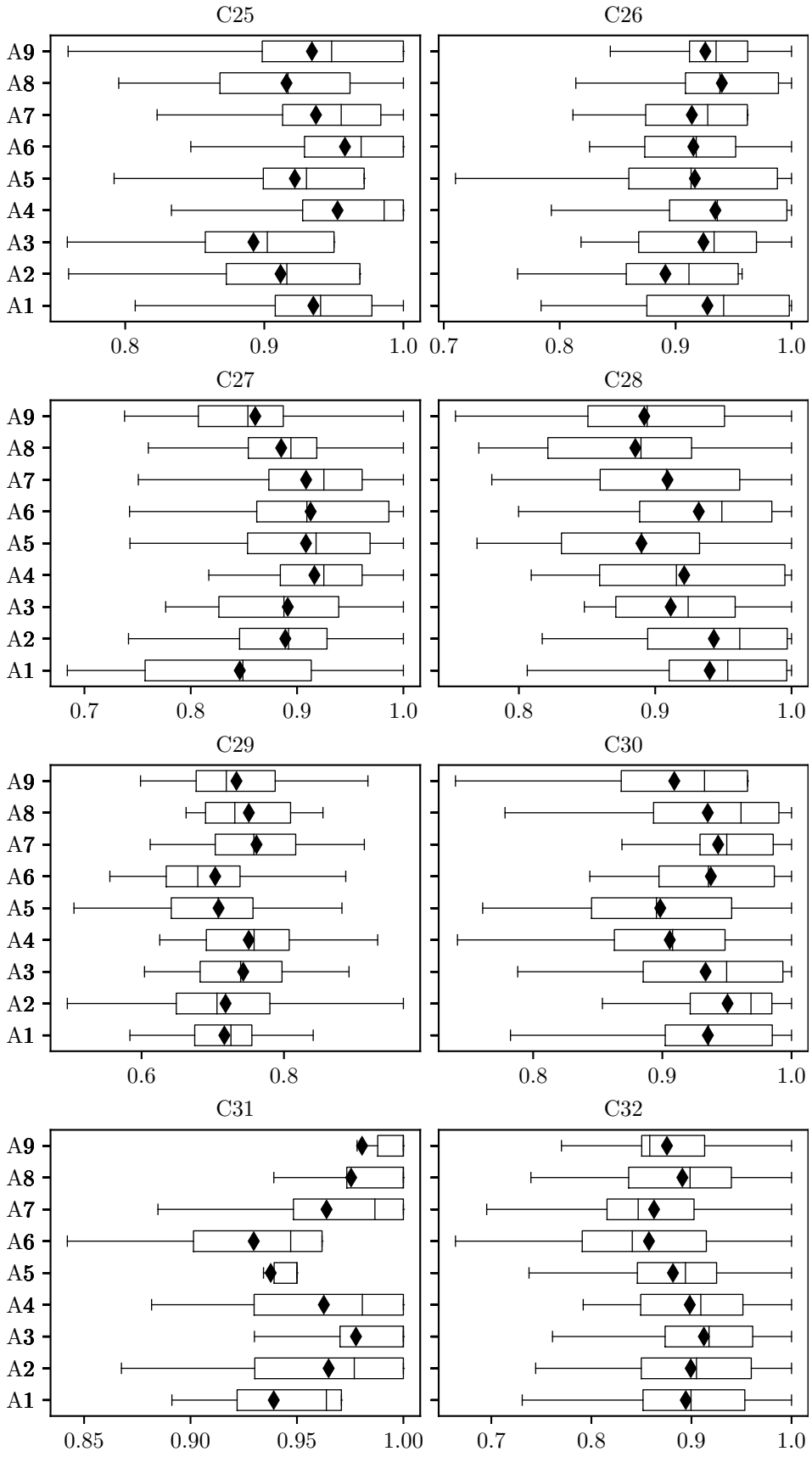


FIGURE 7.38: *Box plots of the F_1-scores achieved by each Adam parameter combination (presented in Table 7.20) in respect of data sets C25–C32.*

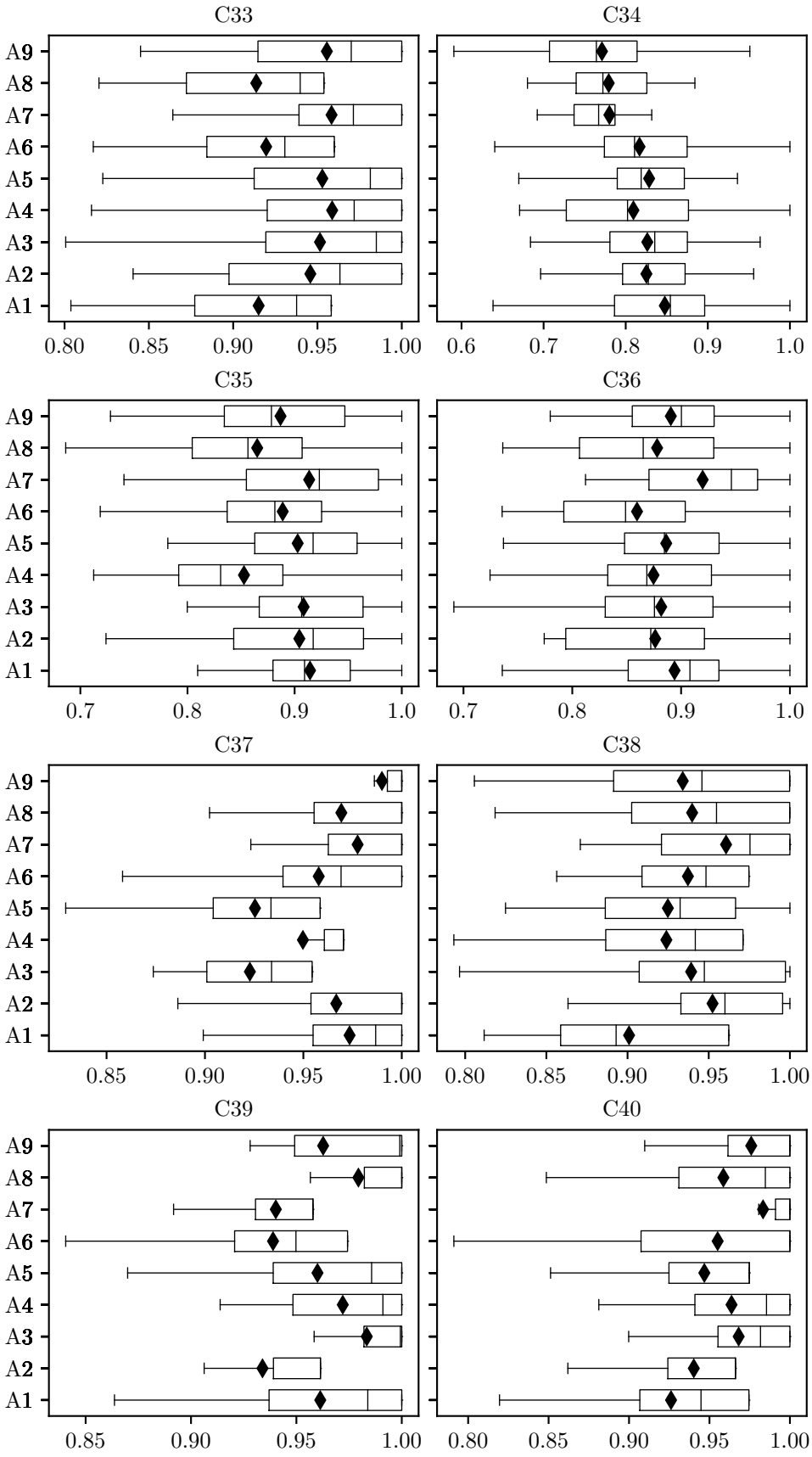


FIGURE 7.39: *Box plots of the F_1-scores achieved by each Adam parameter combination (presented in Table 7.20) in respect of data sets C33–C40.*

Friedman test to the results returned by each of these algorithms in respect of all the data sets, the corresponding p-values in Table 7.21 were computed. SGD (28 data sets) and Adam (27 data sets) represent the two algorithms for which the the Friedman test detected the most statistical differences between at least two samples (*i.e.* combinations) at a 5% level of significance. In the case of RMSProp, statistical differences were detected for only 20 data sets.

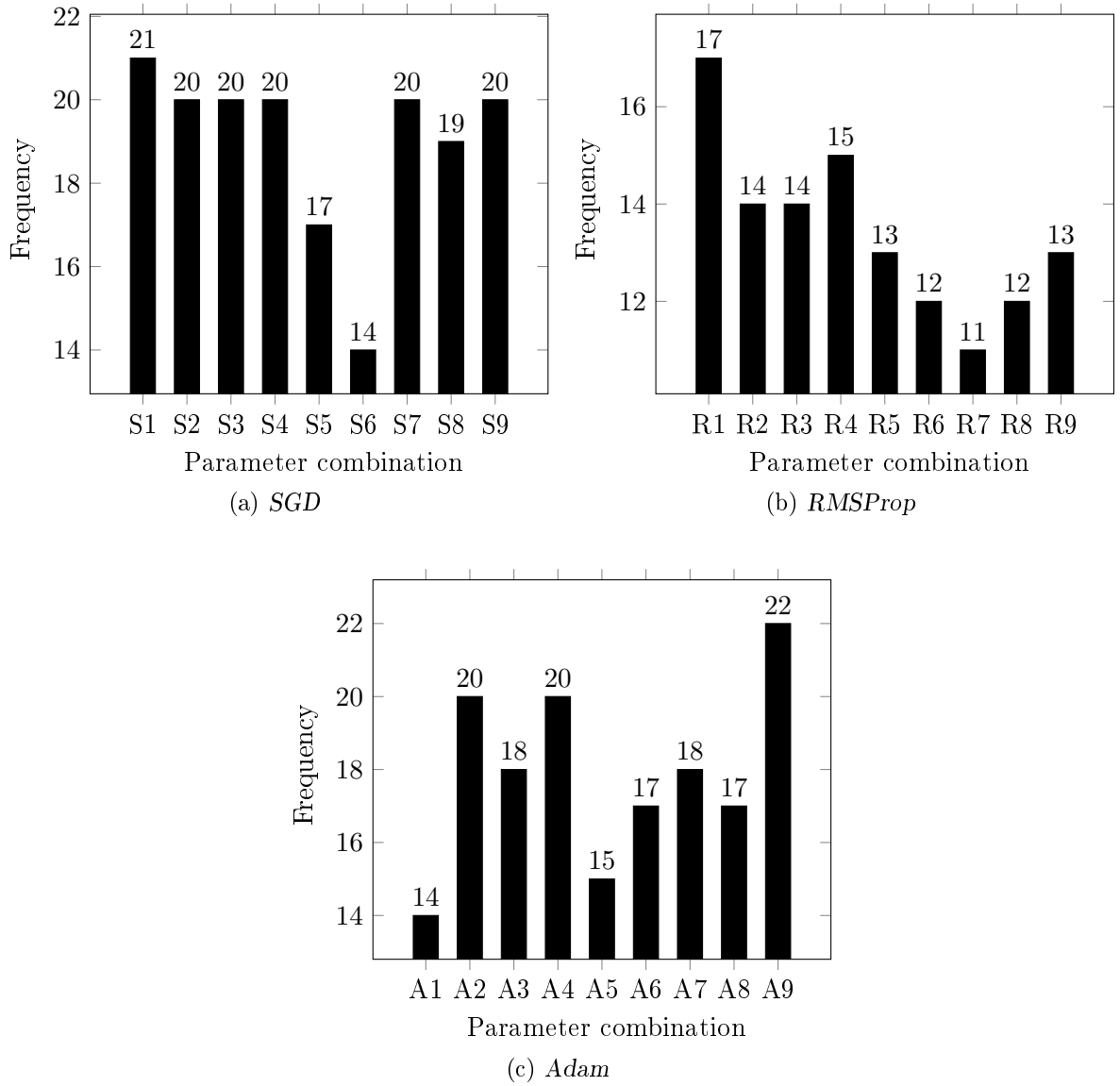
TABLE 7.21: *Friedman test p-values for SGD, RMSProp, and Adam in respect of each data set. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Friedman test p-values						
Data set	SGD	RMSProp	Adam	Data set	SGD	RMSProp	Adam
C1	0.2032	0.9843	0.0057	C21	0.0024	0.0130	0.043
C2	0.0821	0.1132	0.0586	C22	0.0527	0.0677	0.1212
C3	0.0005	0.0509	0	C23	0.0205	0.0036	0.0041
C4	0.0018	0.0035	0	C24	0.2363	0.1231	0.1305
C5	0	0.0063	0.0001	C25	0	0.0859	0
C6	0	0.0121	0	C26	0.0173	0.5703	0.1576
C7	0.0052	0.0352	0.6209	C27	0.0144	0.0142	0.0669
C8	0	0.0908	0	C28	0.0207	0.0026	0.0068
C9	0.0281	0.0343	0.0626	C29	0.5351	0.0017	0.0578
C10	0	0.1676	0	C30	0.0003	0.1497	0.0176
C11	0.0017	0.2542	0	C31	0	0	0
C12	0	0.0178	0	C32	0.1722	0.0015	0.0881
C13	0	0.0007	0	C33	0	0.2474	0
C14	0.3497	0.1019	0.0116	C34	0.0171	0.0984	0.0008
C15	0.0358	0.0435	0.1925	C35	0.9561	0.0005	0.0260
C16	0.1433	0.4289	0.0025	C36	0.4886	0.0122	0.1359
C17	0.0546	0.0776	0.0041	C37	0	0	0
C18	0.0497	0.0573	0.8537	C38	0	0.0514	0.0011
C19	0.0137	0.1184	0.3305	C39	0	0	0
C20	0.1493	0.1804	0.0038	C40	0	0.0012	0

The Nemenyi procedure was consequently applied to the results returned by SGD, RMSProp, and Adam in respect of each data set for which the corresponding Friedman test p-value (presented in Table 7.21) is less than 0.05. There are nine different parameter combinations for each gradient-based training algorithm, and so the Nemenyi procedure involves $\binom{9}{2} = 36$ pairwise significance tests. The best parameter combinations for SGD, RMSProp, and Adam are established by the same procedure as that followed in §7.3.1 and §7.3.2. For each data set, the combinations are first ranked according to sample medians, and thereafter the combinations that are statistically equivalent to the best performing combination (*i.e.* the combinations that are statistically indistinguishable according to the Nemenyi procedure) are identified. Both the frequency with which a combination is ranked first and the frequency with which a combination is statistically equivalent to the combination that is ranked first are used to obtain the relative algorithmic performance achieved by the different parameter combinations. The respective frequencies for the SGD, RMSProp, and Adam parameter combinations are presented in Figure 7.40.

The performance sensitivity observed in the box plots of Figures 7.25–7.39 coincides with the variability on display in the respective frequencies of Figure 7.40. In the case of SGD, a total of seven data sets separate the best performing combination (*i.e.* S1) from the worst perform-

ing combination (*i.e.* S6). Parameter combination S1 corresponds to a small learning rate of $\kappa = 0.001$ and, more interestingly, a momentum parameter value of $\mu = 0$. Accordingly, the nullification of momentum has resulted in the best overall performance. The success of incorporating momentum in gradient-based training algorithms has, however, been reported in the context of *deep* ANNs especially, as stated by Sutskever *et al.* [159] in their original implementation of momentum within this context — ANNs comprising at least seven hidden layers were considered in the study conducted by Sutskever *et al.* In contrast, only two hidden layers are considered in this parameter evaluation. Although parameter combination S1 is superior in terms of relative algorithmic performance, only one data set separates it from five other parameter combinations that do incorporate both medium and large momentum parameter values. Overall, performance is consistent, except in the cases of combinations S5 and S6. A large learning rate of $\kappa = 0.01$ and medium momentum parameter value of $\mu = 0.50$ corresponds to the worst algorithmic performance achieved by SGD.



(a) *SGD* (b) *RMSProp*

(c) *Adam*

FIGURE 7.40: *The frequencies with which the parameter combinations of (a) SGD, (b) RMSProp, and (c) Adam are either ranked first or are statistically equivalent to the combination that is ranked first.*

In respect of RMSProp, a total of six data sets separate the best performing combination (*i.e.* R1) from the worst performing combination (*i.e.* R7). Both a small learning rate of $\kappa = 0.001$ and small exponential decay rate of $\varrho = 0.9$ result in the best overall performance for RMSProp. Interestingly, a small learning rate of $\kappa = 0.001$, combined with a large exponential decay rate of $\varrho = 0.999$, results in the worst overall performance. Finally, in the case of Adam, the best performing combination (*i.e.* A9) and the worst performing combination (*i.e.* A1) are separated by eight data sets. This represents the most notable variation in performance. Accordingly, a large learning rate of $\kappa = 0.01$, together with a large secondary exponential decay rate of $\varrho_2 = 0.999$, results in the best overall performance by Adam. Conversely, a small learning rate of $\kappa = 0.001$, together with a small secondary exponential decay rate of $\varrho_2 = 0.9$, results in the worst overall performance. Table 7.22 contains a summary of the best SGD, RMSProp, and Adam parameter combinations.

TABLE 7.22: *Best performing SGD, RMSProp, and Adam parameter combinations identified.*

Algorithm	Parameter	Value
SGD	κ	0.001
	μ	0
RMSProp	κ	0.001
	ϱ	0.900
Adam	κ	0.001
	ϱ_1	0.900
	ϱ_2	0.990

7.4 Chapter summary

In this chapter, three extensive algorithmic parameter evaluations were performed. The main reasoning behind the necessity of the first two parameter evaluations is attributed to the novelty of both the mathematical model in Chapter 5 and the solution methodology in Chapter 6. The reasoning behind the necessity of the third parameter evaluation is to ensure a fair algorithmic performance comparison between the BOHTA and the gradient-based training algorithms later in the dissertation. An elucidation of the manner in which the training algorithms' performances were evaluated, was first presented in §7.1 — a necessary precursor to the parameter evaluations. This was followed in §7.2 by a discussion on the experimental setup, which focussed specifically on the parameters that remained fixed throughout the subsequent parameter evaluations.

In §7.3, the findings of the three algorithmic parameter evaluations were discussed. The first parameter evaluation focussed on the BOHTA sub-algorithms (*i.e.* the NSGA-II, NSDE, and OMOPSO) and was presented in §7.3.1. The aim was to find suitable algorithmic parameter values for these sub-algorithms, before their subsequent incorporation into the BOHTA. Sources from the literature were consulted in order to find appropriate parameter value ranges. Based on the findings of this parameter evaluation, the NSGA-II proved to be the most robust sub-algorithm, followed closely by NSDE and OMOPSO, both of which achieved similar performance. In the case of the NSGA-II, a large crossover probability delivered the best performance statistically, while in the case of NSDE, a large amplification factor clearly delivered superior performance. Lastly, in the case of OMOPSO, a large mutation probability delivered the best performance.

The second parameter evaluation, on the other hand, focussed on finding suitable algorithmic parameter values for the BOHTA as a whole and was presented in §7.3.2. The findings of the

sub-algorithm parameter evaluation, as well as a conceptual deliberation of different scenarios (or circumstances), were taken into account when identifying the different BOHTA parameter values. The findings of this parameter evaluation indicated that the BOHTA performs best with a large lower bound proportion, whereas the population size does not affect performance considerably.

The third, and final, parameter evaluation focussed on the three gradient-based training algorithms, *i.e.* SGD, RMSProp, and Adam, and was presented in §7.3.3. The imposition of differentiability required a number of amendments to the experimental setup so as to ensure a fair performance comparison with the BOHTA later in the dissertation. The findings of this parameter evaluation indicated that SGD favours the exclusion of momentum together with a small learning rate, while in the case of RMSProp, both a small learning rate and a small exponential decay rate deliver superior performance. Lastly, in the case of Adam, both a large learning rate and a large secondary exponential decay rate deliver the best performance.

CHAPTER 8

BOHTA Implementation

Contents

This chapter is devoted to a detailed investigation into the implementation of the BOHTA in the context of solving the problem instances contained within the test suite. The chapter opens with an in-depth algorithmic comparative study, with a focus on comparing the relative algorithmic performances of the NSGA-II, NSDE, OMOPSO with those of the different versions of the BOHTA. The meta-generalisation capabilities of the BOHTA are evaluated next in order to gain greater insight into the robustness of this hyperheuristic approach. The BOHTA is then compared with powerful gradient-based training algorithms, *i.e.* SGD, RMSProp, and Adam. Thereafter, the consolidation of the BOHTA and the best gradient-based training algorithm is investigated. This is followed by an investigation into algorithmic performance prediction during which the BOHTA is the focal point. A necessary precursor, however, is an analysis of the BOHTA's temporal dynamics — the behaviour of the constituent sub-algorithms is scrutinised so as to gain insight into the BOHTA's inner working. The structural attributes of favourable networks produced by the BOHTA are finally discussed before the chapter closes with a brief summary of the work included in the chapter.

8.1 BOHTA comparative study

After establishing suitable algorithmic parameter values for the NSGA-II, NSDE, OMOPSO, and the BOHTA in Chapter 7, the focus now turns to a comparative study of the relative performances of these algorithms. The aim in this comparative study is to determine whether a hyperheuristic can deliver both superior performance and enhanced levels of general applicability in the given optimisation context. The NSGA-II, NSDE, and OMOPSO are executed with the suitable parameter combinations ascertained during their respective algorithmic parameter evaluations. In §7.3.2, a cursory and notably aggregated overview of the relative algorithmic

performance achieved by the BOHTA (relative to three sub-algorithms) was provided — sample medians and sample means were averaged across *all* parameter combinations and *all* data sets. The evaluation of performance in this section, however, is performed on a markedly lower level of abstraction so as to gain greater insight into the BOHTA's performance characterisation.

In order to compare the relative performances of the four training algorithms, the Friedman test is first performed so as to determine whether there is a significant difference between the training algorithms at a 5% level of significance. Accordingly, two different Friedman tests are performed, depending on whether statistically significant differences were identified for the different BOHTA parameter combinations. A summary of the Friedman test p-values for the BOHTA in respect of each data set was presented in Table 7.16 — statistical differences exist in 22 (out of the 40) data sets at a 5% level of significance. The first Friedman test is therefore performed in respect of these 22 data sets and compares the performance of the best- and worst-performing versions of the BOHTA, denoted by BOHTA-B and BOHTA-W, respectively, with those of the three sub-algorithms. The best-performing version corresponds to the BOHTA parameter combination that achieved the best statistically significant performance improvement, whereas the worst-performing version corresponds, of course, to the parameter combination that achieved the worst in the same regard. Recall from §7.3.2 that statistical superiority corresponds to the BOHTA parameter combination that achieves the highest overall frequency of (1) data sets for which the corresponding sample median is statistically superior and the frequency of (2) data sets for which the parameter combination is statistically indistinguishable from the combination that is statistically superior in terms of sample median. The respective frequencies are shown in Figure 7.23. The BOHTA-B and BOHTA-W incarnations therefore correspond to parameter combinations B10 and B2, respectively, according to which both combinations employ a population size of $M = 65$. Furthermore, B10 employs a large lower bound proportion of $\tilde{N} = 0.15$, whereas B2 employs a small lower bound proportion of $\tilde{N} = 0.05$. The inclusion of the worst-performing version of the BOHTA facilitates a more in-depth algorithmic comparative study.

Table 8.1 contains the p-values for the first Friedman test. There are only three data sets for which statistical differences between at least two algorithms are not detected at a 5% level of significance. Note that the statistical differences detected in respect of the remaining data sets do not necessarily indicate whether the BOHTA delivers performance that is statistically different from those of the sub-algorithms as these differences can also be attributed to differences amongst the performances of the three sub-algorithms. The Nemenyi *post hoc* procedure is therefore performed so as to determine between which pairs of training algorithms statistically different performances (identified by the Friedman test) are present. Based on the findings of the Nemenyi procedure, the statistically significant performance improvements achieved by the BOHTA-W and BOHTA-B incarnations over their counterparts are shown in Table 8.1. The BOHTA-W delivers, on average, 0.0286, 0.0742, and 0.0785 F_1-score sample mean improvements over the NSGA-II, NSDE, and OMOPSO, respectively, when statistically significant differences are detected by the Nemenyi procedure between at least two sample medians. The BOHTA-B, on the other hand, delivers, on average, 0.0432, 0.0876, and 0.0909 F_1-score sample mean improvements over the NSGA-II, NSDE, and OMOPSO, respectively, when statistically significant differences are detected by the Nemenyi procedure between at least two sample medians. The box plots in Figure 8.1 visually illustrate the performance comparison of the three sub-algorithms as well as those of the best- and worst-performing versions of the BOHTA in respect of the data sets for which the improvements are most clearly evident. Sample maxima and minima are consistently superior when the performances of the two BOHTA versions are compared with those of the sub-algorithms. The narrow inter-quartile ranges in respect of numerous data sets are also indicative of the greater degree of performance consistency exhibited by the BOHTA.

The BOHTA-W is outperformed in respect of only two data sets, *i.e.* C6 and C22. It should be noted, however, that in these cases the BOHTA-W is never the worst overall performer — it either outperforms at least one of the sub-algorithms (in the case of C6) or is statistically indistinguishable from the worst performer (in the case of C22). The BOHTA-B, on the other hand, is statistically inferior to its counterparts in respect of only one data set, *i.e.* C22 — it is, once again, not the overall worst performer because it still outperforms the worst performing sub-algorithm, *i.e.* NSDE. Both incarnations of the BOHTA certainly exhibit robustness in terms of relative algorithmic performance and the consistency at which the improvements are achieved.

TABLE 8.1: *The p-values for the first Friedman test. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance. Only the data sets for which there exists a statistical difference between at least two BOHTA parameter combination sample medians are considered. The sub-algorithms are compared with the worst- and best-performing incarnations of the BOHTA, i.e. BOHTA-W and BOHTA-B, respectively. Values represent F_1-score sample mean differences. Values in parentheses indicate inferior performance, whereas omitted values indicate no statistical difference between the performances of the particular BOHTA incarnation and the sub-algorithms.*

	Friedman test	BOHTA-W *versus*			BOHTA-B *versus*		
	p-values	NSGA-II	NSDE	OMOPSO	NSGA-II	NSDE	OMOPSO
C1	0	—	0.1242	0.1522	—	0.1143	0.1423
C2	0.0032	0.0607	—	—	0.0728	—	—
C3	0	0.0570	—	0.1615	—	—	0.1168
C4	0	0.0545	0.0525	0.0532	0.0602	0.0583	0.0589
C5	0	—	—	0.1061	0.0696	0.0596	0.1370
C6	0	(0.0377)	0.0859	0.0778	—	0.1555	0.1474
C7	0.0086	0.0562	0.0623	—	—	—	—
C8	0	—	0.0388	0.0666	0.0567	0.0697	0.0975
C10	0	—	—	0.0605	—	—	0.0986
C11	0.1265	—	—	—	—	—	—
C14	0.0002	0.0735	0.0813	0.0713	0.0627	0.0705	0.0605
C16	0.0012	—	0.0603	—	0.0303	0.0673	0.0325
C18	0.0796	—	—	—	—	—	—
C19	0	—	0.0791	0.0646	—	0.0960	0.0816
C20	0.0120	—	—	—	—	—	—
C22	0	(0.0643)	—	(0.0715)	(0.0563)	0.0779	(0.0635)
C23	0	—	—	0.1217	0.0354	—	0.1668
C24	0.0778	—	—	—	—	—	—
C31	0	—	0.0615	0.0720	—	0.1048	0.1153
C33	0.0181	—	—	—	—	—	—
C35	0.0001	—	0.0858	0.0847	0.0577	0.0821	0.0810
C39	0	—	0.0846	—	—	0.0948	—

When comparing the respective frequencies with which each of the sub-algorithms and the two BOHTA versions are either ranked first (in terms of sample median) or are statistically equivalent to the algorithm that is ranked first, notable performance improvements are observed, as shown in Figure 8.2. The BOHTA-B delivers a 50% improvement over the best performing sub-algorithm, *i.e.* the NSGA-II, in terms of the aforementioned frequency. The improvements over the remaining sub-algorithms are even more consequential — the two BOHTA versions are superior in respect of more than double the data sets for which performance is not statistically indistinguishable. The versatility of the BOHTA is therefore, once again, showcased.

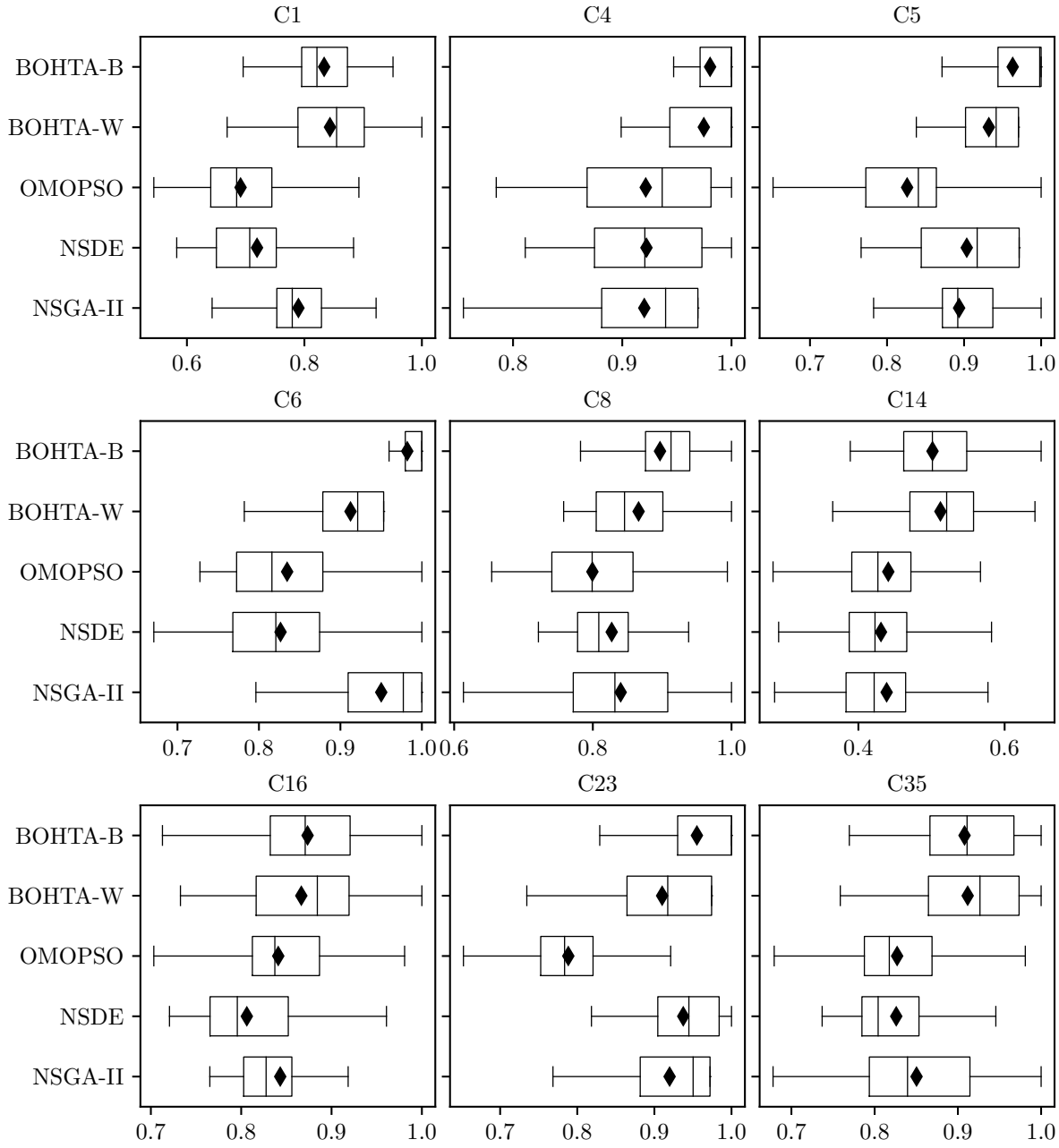


FIGURE 8.1: *Box plots of the data sets for which the BOHTA-W and the BOHTA-B algorithmic incarnations achieve the most notable F_1-score performance improvements.*

The second Friedman test, on the other hand, is performed in respect of the remaining 18 data sets for which no statistical difference exists between at least two BOHTA parameter combinations at a 5% level of significance (shown in Table 7.16). Furthermore, the three sub-algorithms are compared with only one version of the BOHTA — the lack of statistical differences deems this an appropriate course of action. The same best-performing BOHTA version considered in the first Friedman test is selected to form the basis of this comparison. Table 8.2 contains the p-values for the second Friedman test. There are only two data sets for which statistical differences between the performances of at least two algorithms are not detected at a 5% level of significance. Once again, the statistical differences detected in respect of the remaining data sets do not necessarily indicate whether the BOHTA delivers performance that is statistically different from those of the sub-algorithms — these differences can also be attributed to a difference in performance amongst the three sub-algorithms. The Nemenyi *post hoc* procedure is

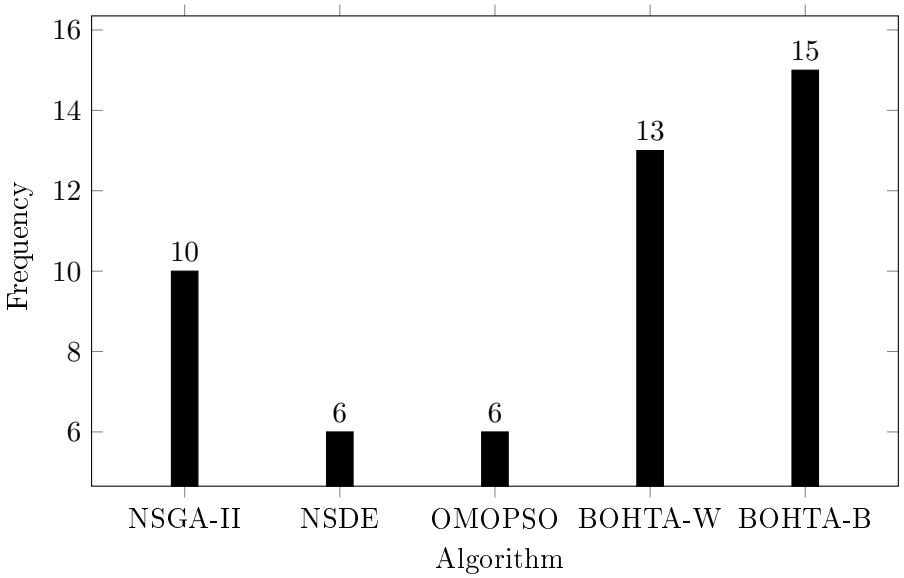


FIGURE 8.2: *The frequencies with which the sub-algorithms, BOHTA-W, and BOHTA-B are either ranked first or are else statistically equivalent to the algorithm that is ranked first by the first Friedman test.*

therefore performed so as to determine between which pairs of training algorithms statistically different performances (as identified by the Friedman test) are present. Based on the findings of the Nemenyi procedure, the statistically significant performance improvements achieved by the BOHTA-B over its counterparts are shown in Table 8.2. The BOHTA-B delivers, on average 0.0410, 0.0869, and 0.0715 F_1-score sample mean improvements over the NSGA-II, NSDE, and OMOPSO, respectively, when statistically significant differences are detected by the Nemenyi procedure between at least two sample medians.

The BOHTA-B exhibits statistically inferior performance in respect of only three data sets, *i.e.* C15, C21, and C29. It should be noted, however, that the BOHTA-B is outperformed by all three sub-algorithms in respect of only one of these data sets (in the case of C15) — this data set represents the only data set from the entire collection of forty data sets (*i.e.* C1–C40) for which the BOHTA is comprehensively outperformed. In the case of the remaining two data sets for which the performance achieved by the BOHTA is statistically inferior, the BOHTA-B is either statistically indistinguishable from the worst performers (in the case of C21) or outperforms at least one of the other sub-algorithms (in the case of C29). The BOHTA certainly exhibits robustness in terms of relative algorithmic performance and the consistency with which the improvements are achieved. Figure 8.3 shows the respective frequencies with which each of the sub-algorithms and the BOHTA-B are either ranked first (in terms of sample median) or are statistically equivalent to the algorithm that is ranked first.

Although the performance improvement is certainly less pronounced than in the case of the first Friedman test (and subsequent Nemenyi procedure), the BOHTA still delivers the best overall performance. This is further proof of the BOHTA's robustness. The performance advantage of the BOHTA is especially notable when its performance is compared with that of the best sub-algorithm, *i.e* the NSGA-II — a case could almost be made that the NSGA-II does not seem markedly inferior. The BOHTA-B, however, outperforms the NSGA-II in respect of 12 data sets, whereas the NSGA-II is only superior in respect of two data sets. The comparison is even more stark when considering NSDE and OMOPSO — the BOHTA outperforms NSDE and OMOPSO in respect of 21 and 22 data sets, respectively. The increased level of general applicability exhibited by the BOHTA is indisputable.

TABLE 8.2: *The p-values for the second Friedman test. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance. Only the data sets for which there does not exist a statistical difference between at least two BOHTA parameter combination sample medians are considered. The performance of the sub-algorithms are compared with that of the best-performing incarnation of the BOHTA, i.e. BOHTA-B. Values represent F_1-score sample mean differences. Values in parentheses indicate inferior performance, whereas omitted values indicate no statistical difference between the performances of the BOHTA incarnation and the sub-algorithms.*

	Friedman test	BOHTA-B *versus*		
	p-values	NSGA-II	NSDE	OMOPSO
C9	0	—	0.1026	—
C12	0	—	0.1755	0.1447
C13	0.0106	0.0730	—	—
C15	0.0135	(0.0329)	(0.0234)	(0.0644)
C17	0.2509	—	—	—
C21	0.0009	—	—	(0.0588)
C25	0	—	0.0912	0.1229
C26	0	—	—	0.1115
C27	0.0018	0.0495	—	—
C28	0	—	0.1059	0.0720
C29	0	0.0692	0.0648	(0.0756)
C30	0	—	0.1507	0.1092
C32	0.6395	—	—	—
C34	0.0355	—	0.0546	—
C36	0	—	0.0868	0.1038
C37	0	—	—	0.1661
C38	0	—	—	0.1345
C40	0.0001	0.0463	0.0604	0.0921

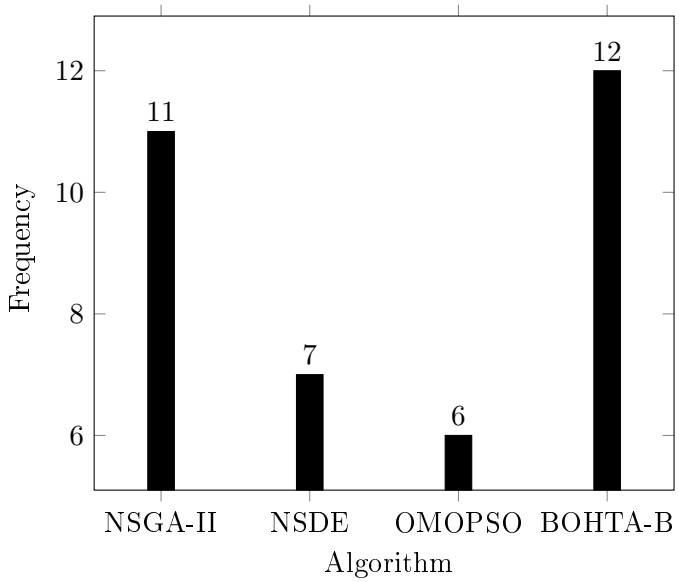


FIGURE 8.3: *The frequencies with which the sub-algorithms and BOHTA-B are either ranked first or are statistically equivalent to the algorithm that is ranked first for the second Friedman test.*

8.2 Meta-generalisation

Data sets C41–C49 are now employed to evaluate the BOHTA's meta-generalisation capabilities, *i.e.* to determine how well the BOHTA performs in respect of "unseen" data. The parameters of the BOHTA's sub-algorithms were fine-tuned in respect of the other forty data sets, and so insight into the BOHTA's robustness can be obtained by evaluating its performance in respect of unseen data sets, represented by data sets C41–C49 (described in §6.2). In order to perform this analysis, the NSGA-II, NSDE, OMOPSO, as well as the different versions of the BOHTA, are executed thirty times in respect of the nine data sets and their performances are recorded during each optimisation run. The box plots in Figure 8.4 supplement the following discourse. Upon initial inspection, the performance of the BOHTA would seem to exhibit the same level of performance sensitivity to the parameter combination employed as did the BOHTA in respect of data sets C1–C40 (*i.e.* in the parameter evaluation of §7.3). It is not entirely evident which parameter combinations result in superior performance with respect to the unseen data. Furthermore, a discernible observation relates to its improvement over those of the individual applications of the sub-algorithms, although the true extent thereof cannot be inferred from visual analyses of box plots alone. Statistical tests are therefore carried out so as to gain further insight into this matter.

The Friedman test is first applied to the results returned by each of these algorithms so as to determine whether there are statistically significant differences between the performances of at least two algorithms (at a 5% level of significance). The resulting Friedman test p-values in respect of data sets C41–C49 are shown in Table 8.3. Each p-value is less than 0.05, and so the Nemenyi *post hoc* procedure is performed next in respect of all nine data sets in order to identify those pairs of samples in which the differences are present.

TABLE 8.3: *Friedman test p-values for each BOHTA parameter combination and sub-algorithm in respect of data sets C41–C49. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Friedman test p-values
C41	0
C42	0.0199
C43	0
C44	0
C45	0.0026
C46	0
C47	0.0252
C48	0
C49	0

The results, which are summarised in Table 8.4, indicate the magnitudes of statistically significant performance improvements by the best algorithms in terms of F_1-score sample mean performance (relative to each of the other algorithms). Consider, for example, data set C41. The best algorithm is the NSGA-II — its sample median, however, is statistically indistinguishable from those of various BOHTA parameter combinations (B3, B4, B5, B6, B7, and B9). Improvements in respect of these statistically indistinguishable BOHTA incarnations are therefore omitted. The statistical difference (*i.e.* improvement) with respect to the other training algorithms are indicated in parentheses. It may be observed that none of the BOHTA parameter combinations are outperformed in respect of all nine data sets and that the performance of the BOHTA is statistically superior in a majority of the cases — excluding data sets C41 and C42, of course,

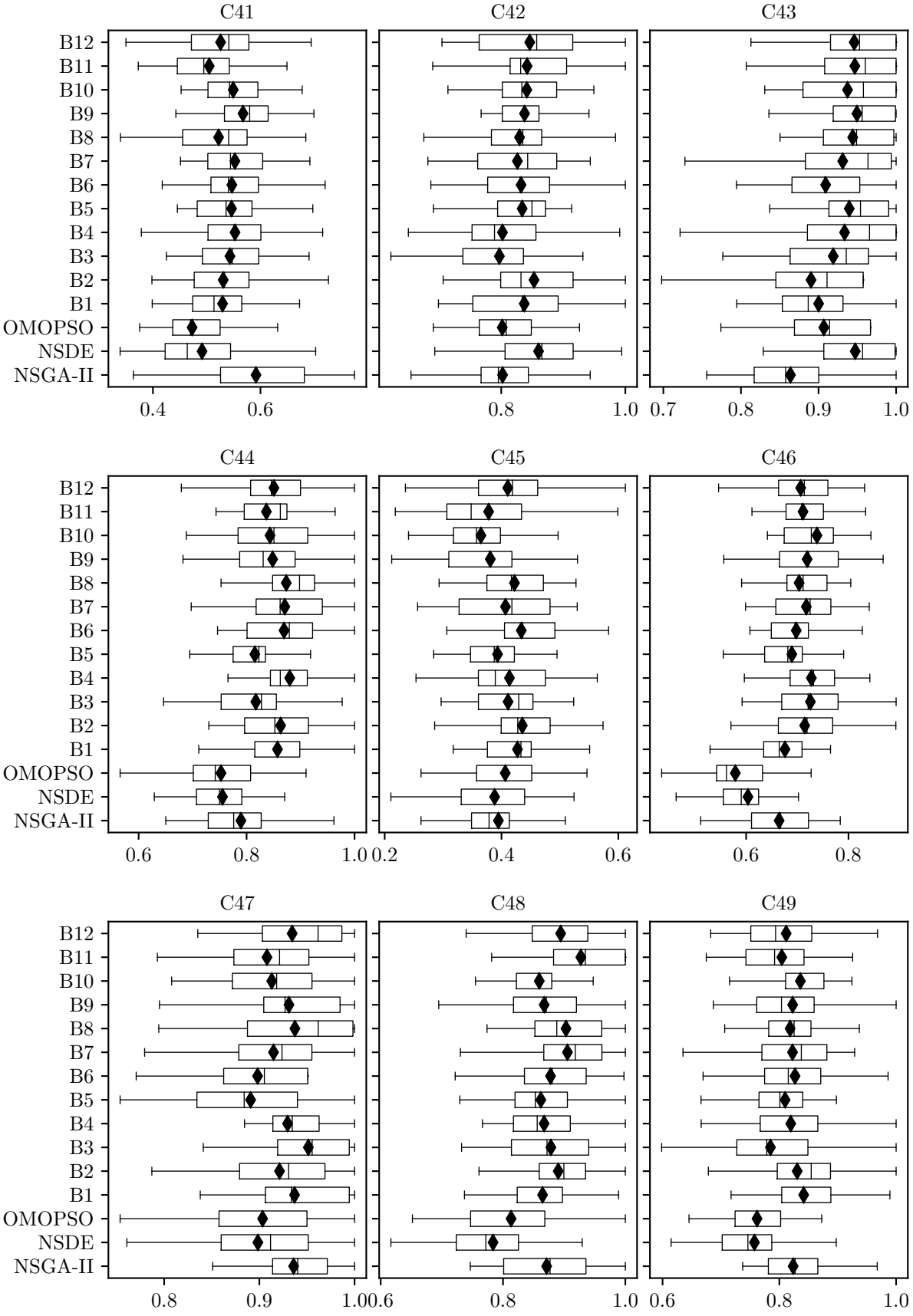


FIGURE 8.4: *Box plots of the F_1-scores achieved by the different BOHTA parameter combinations and the NSGA-II, NSDE and OMOPSO with respect to data sets C41–C49.*

TABLE 8.4: *Meta-generalisation performances achieved by the different BOHTA incarnations and the respective sub-algorithms (i.e. the NSGA-II, NSDE, and OMOPSO). Each value represents the difference in F_1-score sample mean performance relative to that of the best performing algorithm for the data set under consideration. Accordingly, a zero-valued entry corresponds to the best algorithm. Values in parentheses indicate inferior performance, whereas omitted values indicate no statistical difference between the performances of the BOHTA incarnations and the sub-algorithms.*

	C41	C42	C43	C44	C45	C46	C47	C48	C49
NSGA-II	0	(0.0580)	(0.0697)	(0.0840)	(0.0391)	(0.0629)	—	(0.0559)	—
NSDE	(0.1003)	0	—	(0.1181)	(0.0456)	(0.1238)	(0.0389)	(0.1436)	(0.0732)
OMOPSO	(0.1188)	(0.0585)	—	(0.1209)	—	(0.1485)	(0.0339)	(0.1141)	(0.0688)
B1	(0.0622)	—	(0.0334)	—	(0.0066)	(0.0515)	—	(0.0626)	—
B2	(0.0609)	—	(0.0432)	—	—	—	—	—	0
B3	—	(0.0631)	—	(0.0564)	—	—	—	(0.0490)	(0.0458)
B4	—	(0.0581)	0	—	—	0	—	(0.0600)	—
B5	—	—	—	(0.0583)	(0.0406)	—	(0.0465)	(0.0654)	—
B6	—	—	—	—	0	—	(0.0392)	(0.0496)	—
B7	—	—	—	—	—	—	—	—	—
B8	(0.0695)	—	—	0	—	—	0	—	—
B9	—	—	—	—	(0.0503)	—	—	(0.0600)	—
B10	(0.0424)	(0.0189)	—	—	(0.0691)	—	—	(0.0679)	—
B11	(0.0870)	—	—	—	—	—	—	0	(0.0263)
B12	(0.0658)	—	—	—	—	—	—	—	—

according to which the NSGA-II and NSDE are superior, although not exclusively so. Overall, the BOHTA exhibits markedly improved performance over the sub-algorithms. More specifically, average F_1-score sample mean improvements of 0.0623, 0.0905, and 0.0972 are delivered by the best BOHTA incarnation (in respect of the different data sets) over the NSGA-II, NSDE, and OMOPSO, respectively. The performance improvements achieved by the BOHTA are especially notable when comparing the respective frequencies with which each BOHTA parameter combination and sub-algorithm is either ranked first or is statistically equivalent to the combination that is ranked first (in respect of the nine data sets). Figure 8.5 contains the respective frequencies achieved by the different algorithms. It is apparent that the BOHTA outperforms the three sub-algorithms comprehensively. It may be inferred that the BOHTA is better equipped with generalising to (unseen) data sets — problems for which the constituent sub-algorithms are not assigned appropriate parameter values. The BOHTA's greater level of general applicability is therefore further corroborated.

Figure 8.6(a) contains an aggregation of the average frequency for the different lower bound proportions. Interestingly, parameter combinations B5–B8, which correspond to a medium lower bound proportion of $\tilde{N} = 0.10$, represent the best performing combinations overall, albeit marginally. The small and large lower bound proportions of $\tilde{N} = 0.05$ and $\tilde{N} = 0.15$ each performs consistently worse. This change in algorithmic preference is indicative of the necessity of a training algorithm that is able to adapt dynamically and adjust its working according to the problem at hand — the application of a hyperheuristic-based approach (such as AMALGAM and the BOHTA) would therefore seem well-warranted.

Another interesting observation relates to the greater bias towards large population sizes, as shown in Figure 8.6(b). A very large population size of $M = 100$ delivers markedly improved performance over a small population size of $M = 30$. It is conjectured that the unfamiliarity of the problems (*i.e.* data sets) at hand results in the BOHTA favouring greater computational capacity to offset the "unsuitability" of the constituent sub-algorithms. It should be noted, however, that medium and large population sizes of $M = 65$ and $M = 100$ exhibit performance levels that are only marginally inferior to a very large population size.

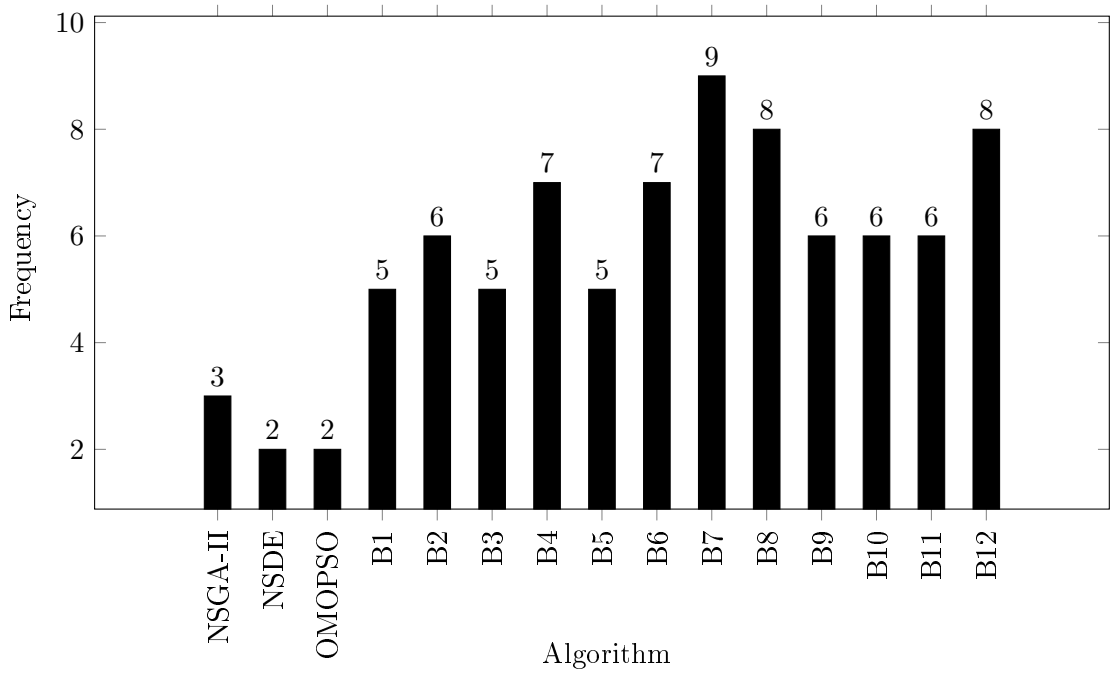
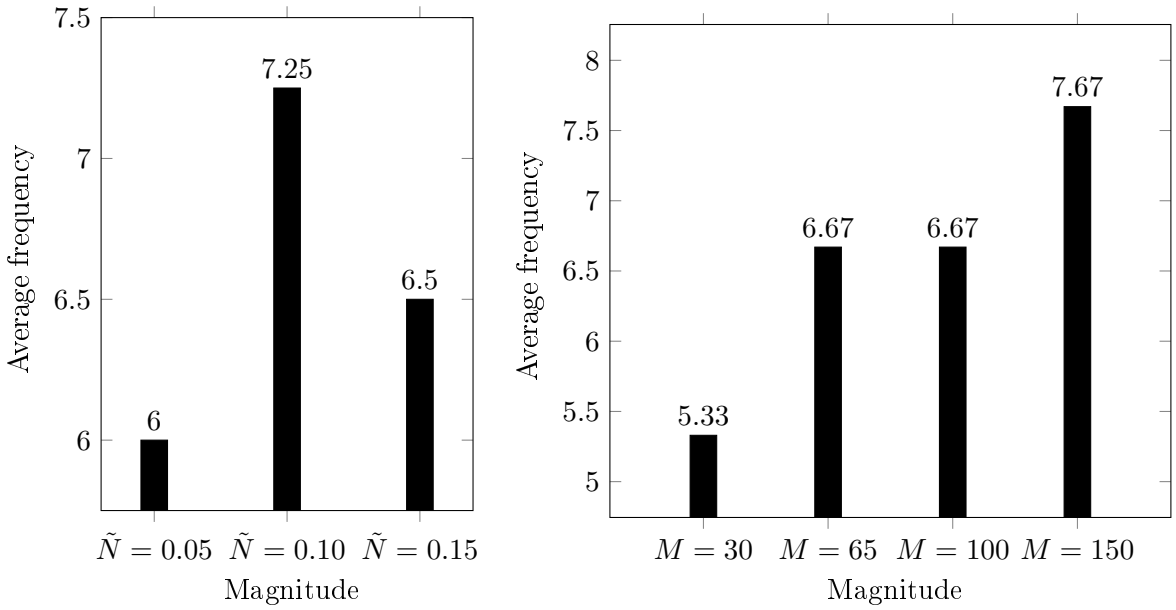


FIGURE 8.5: *The frequencies with which each BOHTA parameter combination and sub-algorithm is either ranked first or is statistically equivalent to the combination that is ranked first in respect of data sets C41–C49.*



(a) *Lower bound proportion* (b) *Population size*

FIGURE 8.6: *The average frequencies with which each BOHTA parameter combination is either ranked first or is statistically equivalent to the combination that is ranked first with respect to the different values for (a) the lower bound proportion and (b) the population size, in respect of data sets C41–C49.*

8.3 BOHTA and/or gradient-based training algorithms

The aim now turns to an investigation into how the performance of the BOHTA compares with those of the powerful gradient-based training algorithms considered in this dissertation, *i.e.* SGD, RMSProp, and Adam. Recall from §3.7 that the conventional approach towards training ANNs is gradient-based — SOTA performance in respect of numerous data sets serve as sufficient justification for its popularity in the ANN literature. A gradient-based approach is, of course, inhibited by the requirement of differentiability. Alternative gradient-free approaches, such as metaheuristics and hyperheuristics, circumvent this inhibition and facilitate specialised and nuanced ANN training algorithms. Optimisation can therefore transpire at an arbitrary level of abstraction, as showcased by the bi-objective model formulated in Chapter 5 and the solution methodology proposed in Chapter 6. The versatility and robustness of the BOHTA — the latter of which was verified thoroughly in §8.1 and §8.2 — warrant an investigation into the consolidation of the BOHTA and the paradigm of gradient-based training approaches. After comparing the algorithmic performance of the BOHTA with those of SGD, RMSProp, and Adam in §8.3.1, the focus then shifts to unifying the disparate approaches — *i.e.* combining the BOHTA with the best gradient-based training algorithm — in §8.3.2.

8.3.1 BOHTA *versus* gradient-based training algorithms

The best respective versions of the BOHTA, SGD, RMSProp, and Adam are selected to form the basis of the comparative study in this section. Based on the findings of each parameter evaluation in §7.3.2 and §7.3.3, parameter combination B10 is selected for the BOHTA, while parameter combinations S1, R1, and A9 are selected for SGD, RMSProp, and Adam, respectively. A visual analysis precedes an extensive statistical analysis and is facilitated by the box plots in Figures 8.7–8.11. Overall, the BOHTA would seem to achieve somewhat competitive performance, especially so in respect of data sets C4, C5, C18, C35, and C36 for which the BOHTA outperforms all of the gradient-based training algorithms in terms of the F_1-score sample mean, albeit empirically. Although performance is comparable, it must be noted that in the case of numerous data sets the BOHTA is comprehensively outperformed by SGD, RMSProp, and Adam. Consider, for example, data sets C1, C15, C21, and C34 — representative of the worst-case scenario. The difference in sample mean performance is substantial, albeit empirically, ranging from 0.0605 to 0.0929 in respect of this subset of data sets. Table 8.5 contains an aggregation of sample median and sample mean performance achieved by the BOHTA, SGD, RMSProp, and Adam in respect of *all* data sets. Interestingly, the average sample median and mean of the BOHTA and the worst performing gradient-based training algorithm, *i.e.* RMSProp, are notably similar. SGD and Adam are evidently the superior training algorithms — a finding that is not entirely surprising when considering the near-universal acclaim of gradient-based training algorithms. Extensive statistical analyses now follow to determine the true extent of the relative algorithmic performances.

TABLE 8.5: *Sample median and mean F_1-scores achieved by the BOHTA, SGD, RMSProp, and Adam (averaged across all data sets), denoted by $\bar{F}_1$.*

	$\bar{F}_1$-score performance	
	Median	Mean
BOHTA	0.8555	0.8484
SGD	0.8916	0.8828
RMSProp	0.8518	0.8493
ADAM	0.8964	0.8891

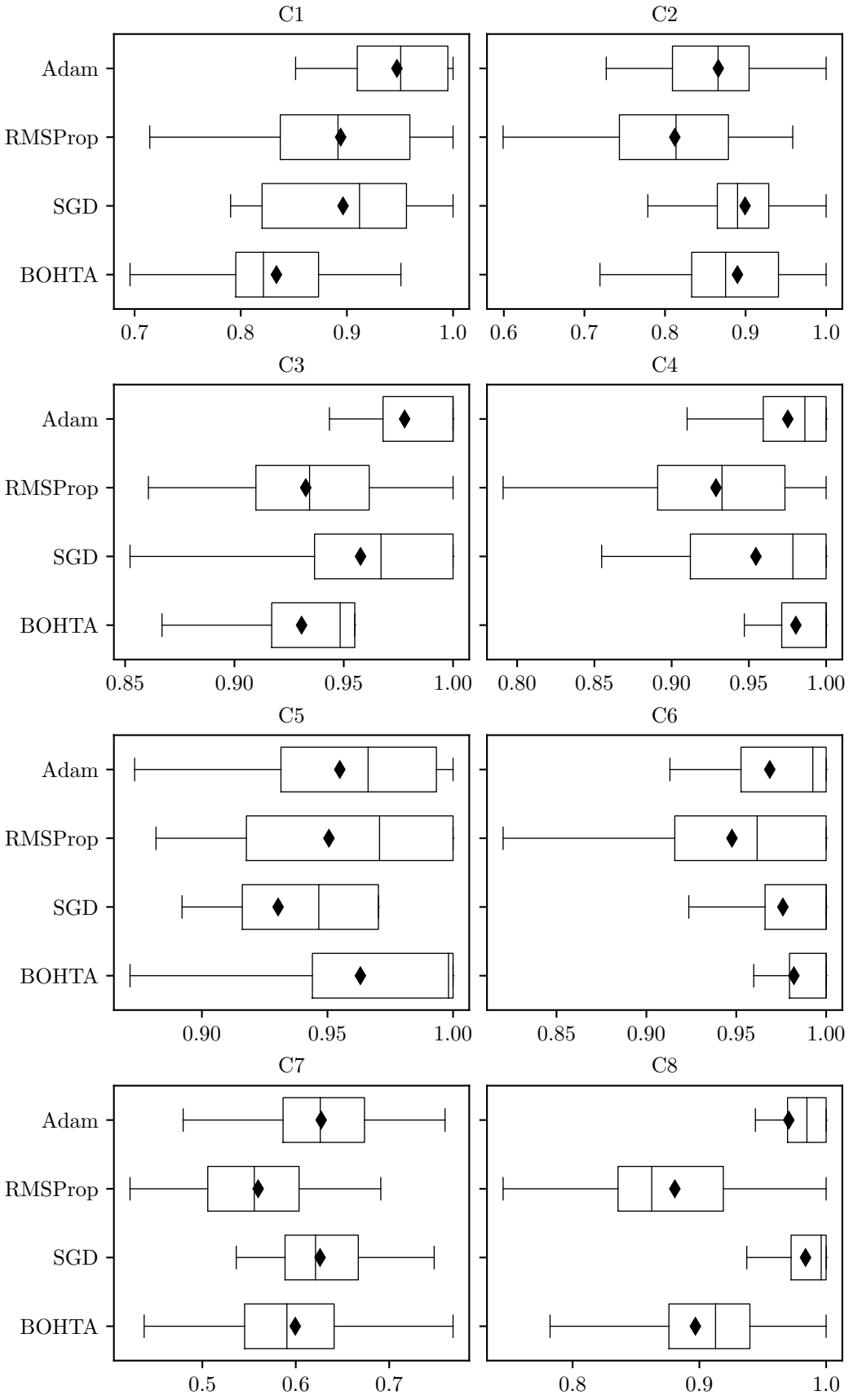


FIGURE 8.7: *Box plots of the F_1-scores achieved by the BOHTA, SGD, RMSProp, and Adam with respect to data sets C1–C8.*

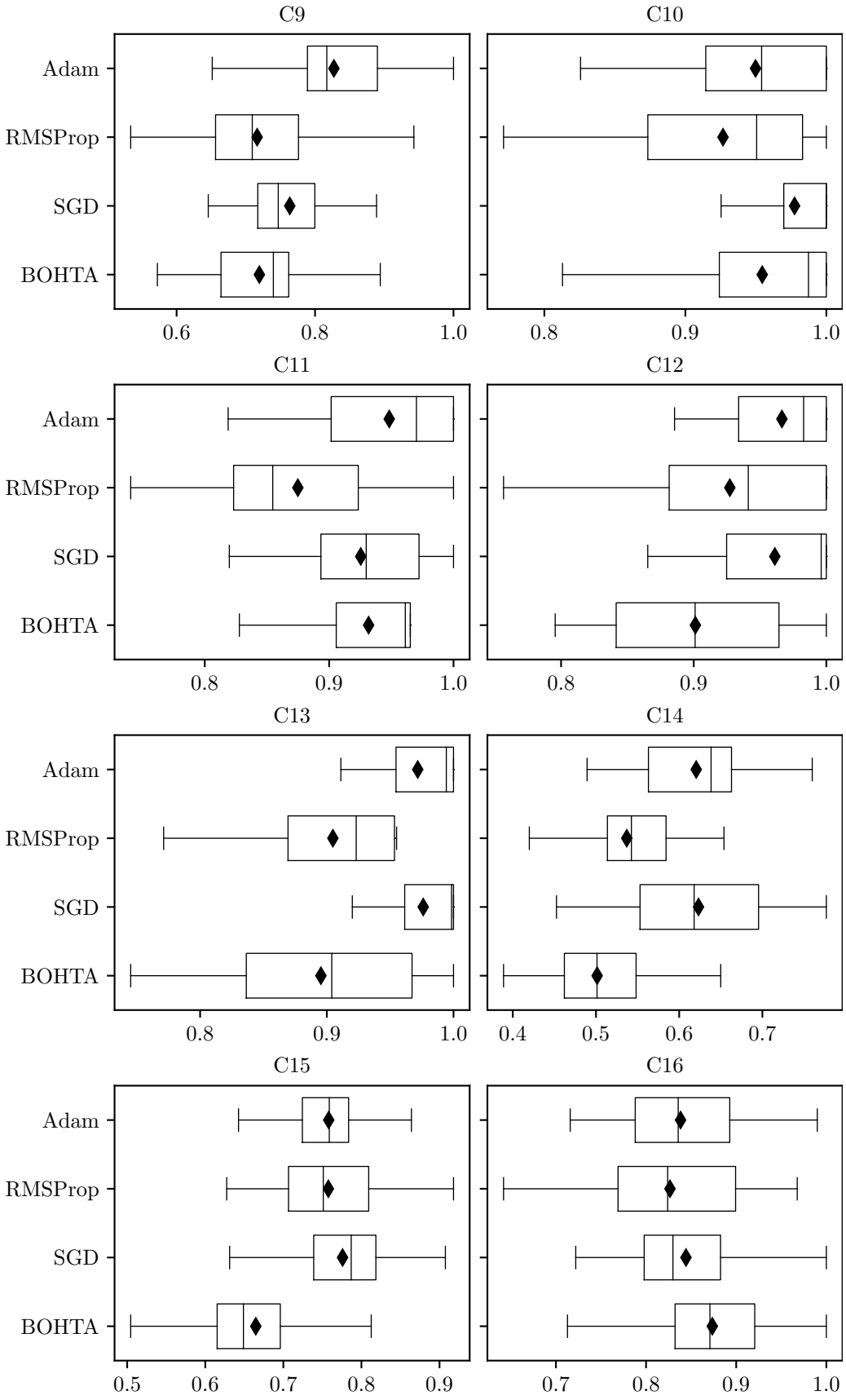


FIGURE 8.8: *Box plots of the F_1-scores achieved by the BOHTA, SGD, RMSProp, and Adam in respect of data sets C9–C16.*

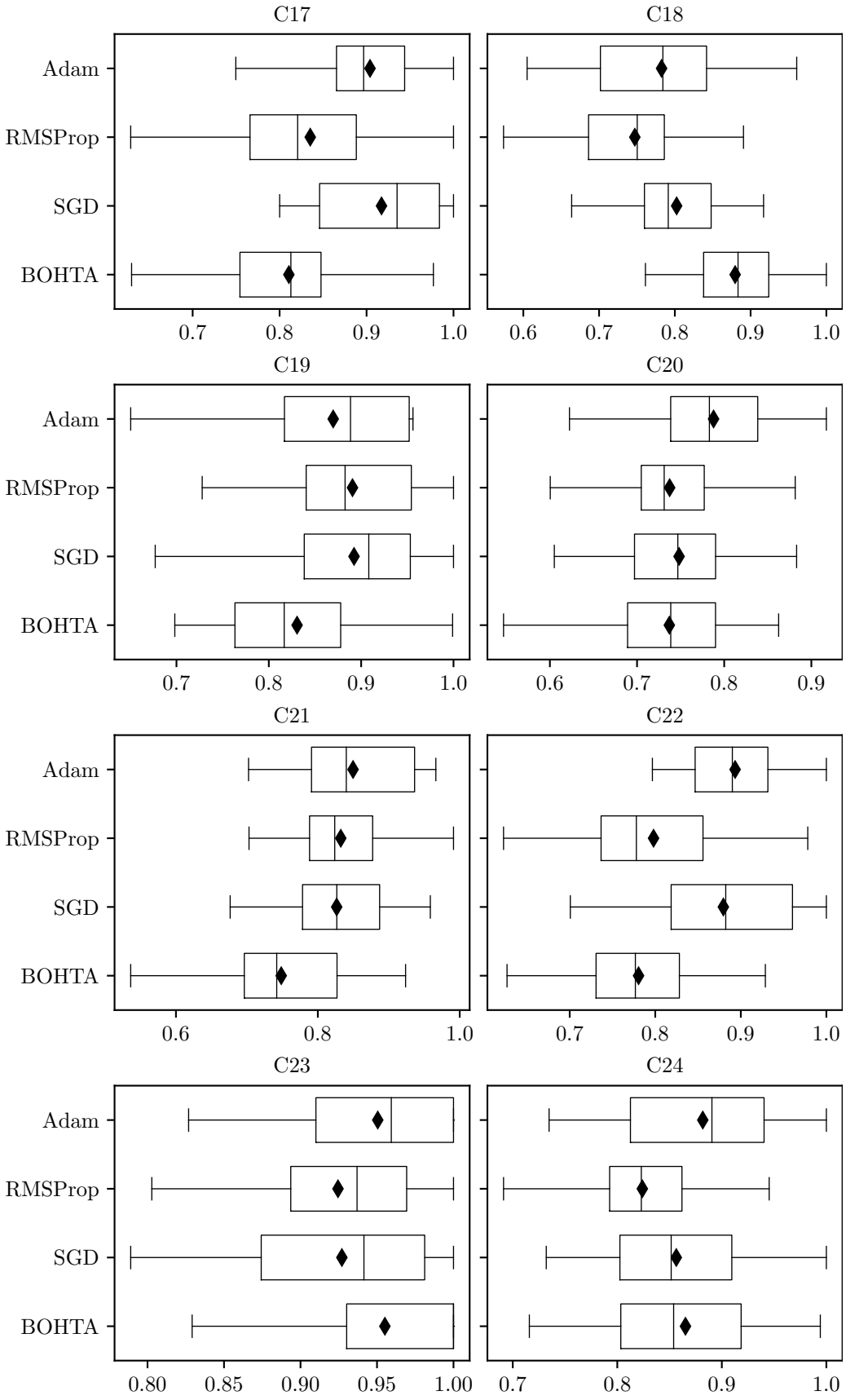


FIGURE 8.9: *Box plots of the F_1-scores achieved by the BOHTA, SGD, RMSProp, and Adam in respect of data sets C17–C24.*

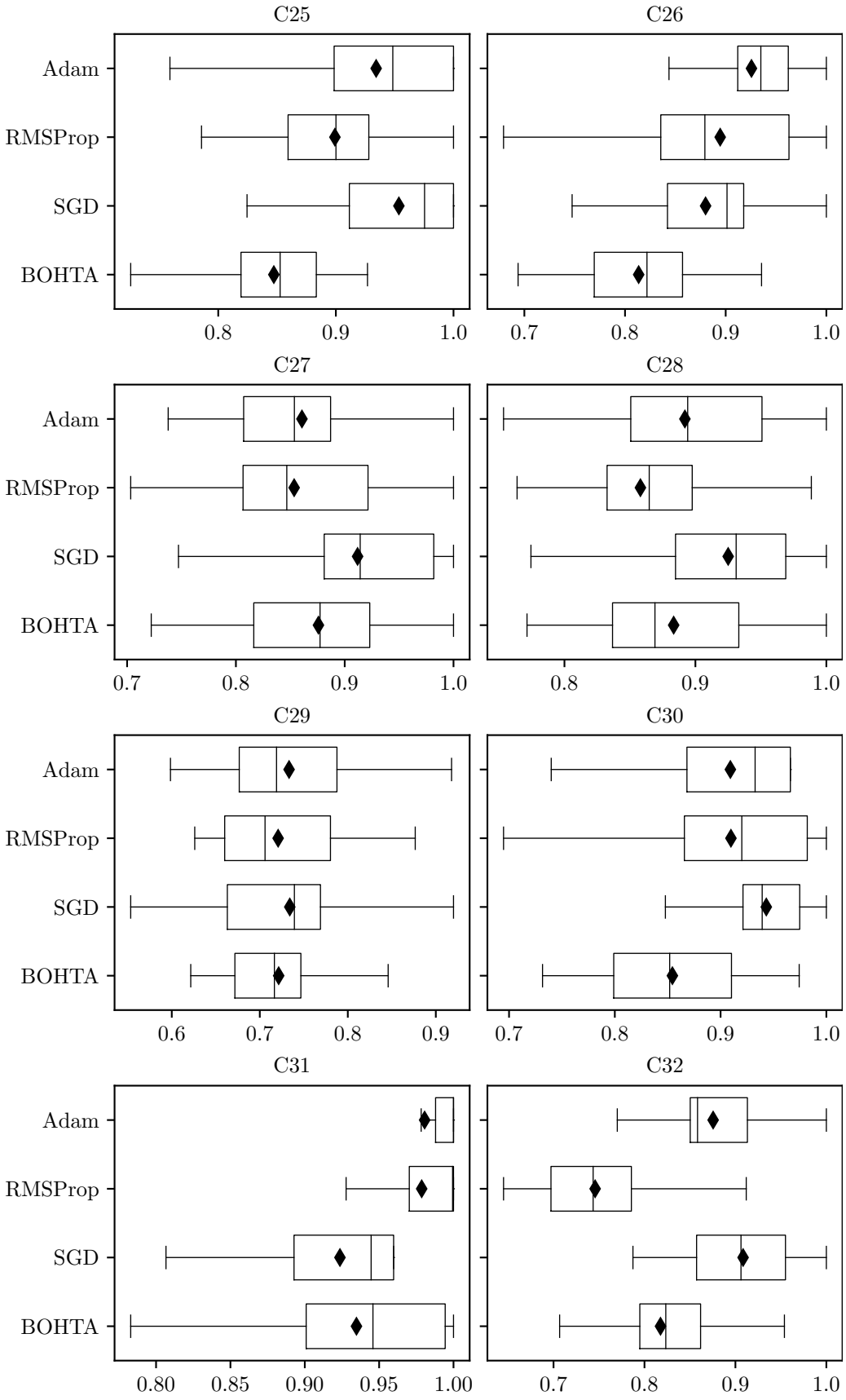


FIGURE 8.10: *Box plots of the F_1-scores achieved by the BOHTA, SGD, RMSProp, and Adam in respect of data sets C25–C32.*

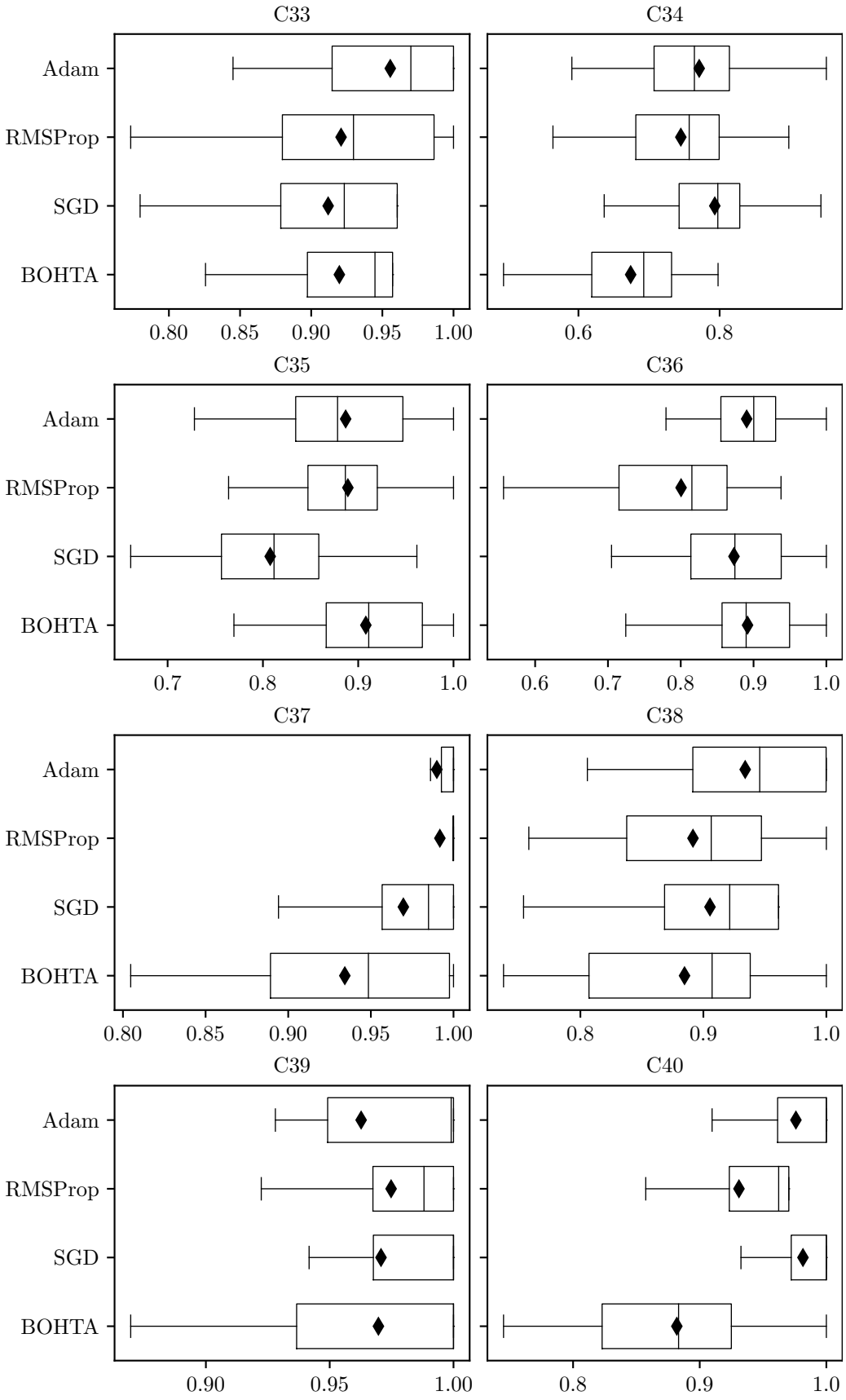


FIGURE 8.11: *Box plots of the F_1-scores achieved by the BOHTA, SGD, RMSProp, and Adam in respect of data sets C33–C40.*

The Friedman test is performed in order first to determine whether a significant difference exists between at least two samples (*i.e.* for the BOHTA, SGD, RMSProp, and Adam) at a 5% level of significance and then, if the Friedman test detects such a difference, the Nemenyi *post hoc* procedure is performed so as to identify the specific pairs of samples in which the differences are present. The p-values obtained upon applying the Friedman test to the results returned by each of these algorithms in respect of all the data sets are presented in Table 8.6. Statistical differences are detected in a vast majority of the data sets, *i.e.* 32 data sets contain a statistical difference. It may therefore be inferred that algorithmic performance varies markedly between the four training algorithms under consideration.

TABLE 8.6: *Friedman test p-values for the comparison between the sample medians of the BOHTA, SGD, RMSProp, and Adam in respect of data sets C1–C40. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Friedman test		Friedman test
Data set	p-value	Data set	p-value
C1	0	C21	0.0001
C2	0.0076	C22	0
C3	0	C23	0.0646
C4	0.0013	C24	0.0898
C5	0.0171	C25	0
C6	0.0534	C26	0
C7	0.0002	C27	0.0655
C8	0	C28	0.0009
C9	0.0001	C29	0.5468
C10	0.0142	C30	0
C11	0.0086	C31	0
C12	0.0129	C32	0
C13	0	C33	0.0076
C14	0	C34	0
C15	0	C35	0
C16	0.2305	C36	0.0013
C17	0	C37	0
C18	0	C38	0.0310
C19	0.0124	C39	0.6329
C20	0.4360	C40	0

The Nemenyi procedure is consequently applied in respect of each data set for which the corresponding Friedman test p-value is less than 0.05. The best overall training algorithm is established by the same procedure as that followed in the parameter evaluations in §7.3, the comparative study in §8.1, and the meta-generalisation analysis in §8.2. Accordingly, for each data set, the training algorithms are first ranked according to sample medians, and thereafter the algorithms that are statistically equivalent to the best performing algorithm (*i.e.* the algorithms that are statistically indistinguishable according to the Nemenyi procedure) are identified. Finally, the relative algorithmic performance achieved by the different algorithms is evaluated by calculating the frequency with which an algorithm is ranked first and the frequency with which an algorithm is statistically equivalent to the algorithm that is ranked first — these two frequencies are then summed. The respective frequencies for the different training algorithms are presented in Figure 8.12. Based on the resulting frequencies it would appear that the BOHTA is markedly inferior to the best performing gradient-based training algorithms — once again, an unsurprising finding. The BOHTA does, however, exhibit comparable performance when compared with RMSProp, *i.e.* the worst-performing gradient-based training algorithm.

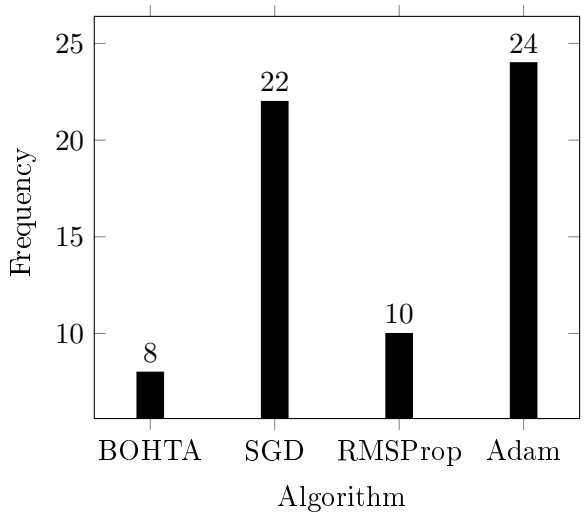


FIGURE 8.12: *The frequencies with which the BOHTA, SGD, RMSProp, and Adam are either ranked first or are statistically equivalent to the algorithm that is ranked first.*

8.3.2 Combining the BOHTA with gradient-based training algorithms

The utility of the BOHTA and the explicit performance of gradient-based training algorithms warrant an investigation into the consolidation of these two disparate approaches. Given the context of the BOHTA, two possible approaches may be adopted. The first approach is a so-called *post-optimisation* approach during which the weights of the networks (or solutions) in the approximate Pareto front — represented by the bright red circles in Figure 7.1 — are subjected to a gradient-based optimisation procedure. The second approach, on the other hand, is a so-called *intermediate-optimisation* approach during which the weights of the networks in the approximate Pareto front are subjected to a gradient-based optimisation procedure after each epoch.

A compromise must, however, be made in order to integrate the Keras framework into the working of the BOHTA. Networks constructed by the BOHTA can only be subjected to Keras' gradient-based optimisation procedures if their activation functions are computationally suitable. Accordingly, while activation functions containing slope parameters for negative input are permissible (as in the case of PReLU), activation functions that contain different slope parameters for negative and non-negative input, respectively, are not. The activation functional variables of the bi-objective model formulated in Chapter 5 violate the requirement imposed by the Keras framework — consequently, the activation functional variables are modified. In particular, the activation functional variable for hidden neuron $j \in \{1, \ldots, m\}$ in hidden layer $f \in \{1, \ldots, h\}$ is now taken as

$$g_j^{(f)}(\eta_j^{(f)}) = \begin{cases} \alpha_j^{(f)} \eta_j^{(f)}, & \eta_j^{(f)} < 0, \\ \eta_j^{(f)}, & \eta_j^{(f)} \geq 0, \end{cases} \tag{8.1}$$

where the slope parameter $\beta_j^{(f)}$ for non-negative input has been omitted. It is conjectured that the potential computational disadvantage resulting from this omission is mitigated by the now possible application of gradient-based approaches to the networks produced by the BOHTA. Investigating ways of integrating activation functions with negative and non-negative slope parameters in the Keras framework may be interesting to resolve as part of future work — a matter deemed outside the research scope of this dissertation.

In this investigation, the best version of the BOHTA is combined with the best gradient-based training algorithm, *i.e.* Adam (as determined in §8.3.1). In both a post-optimisation context and an intermediate-optimisation context, the respective implementations of the BOHTA and Adam are governed by (mostly) the same experimental setups[1] considered thus far, as described in §7.2 and §7.3.3. There is, however, one key difference which pertains to the stopping criterion employed by Adam within the intermediate-optimisation context. Adam (as well as SGD and RMSProp) has thus far been terminated according to the early stopping criterion together with a patience parameter value of five. Although this criterion may (most likely) have contributed towards Adam's superior algorithmic performance, its application in the context of intermediate-optimisation results in excessive computation time — a finding derived from a separate qualitative pilot study performed by the author. A smaller patience parameter value of two was found to mitigate the computational expenditure, resulting in training runs comparable with those of the original implementation of the BOHTA. Although reducing the patience parameter value may prove detrimental to the mitigation of overfitting, it is conjectured that the BOHTA's inclination towards smaller networks together with Adam's L^2 parameter regularisation, is a sufficient form of regularisation. In the case of the post-optimisation approach, on the other hand, Adam is only applied to the networks in the (final) approximate Pareto front — the original patience parameter value is therefore considered sufficient in terms of computational expenditure. Networks produced by the BOHTA are already of a high quality, and so convergence times are shorter when applying Adam to these networks.

The following notational convention is adopted in the subsequent discourse. In the case of the BOHTA's consolidation with Adam within an intermediate-optimisation context, the combined approach is denoted by BOHTA-I. In the case of the BOHTA's consolidation with Adam within a post-optimisation context, on the other hand, the approach is denoted by BOHTA-P.

The box plots in Figures 8.13–8.17 provide a graphical summary of the algorithmic performances achieved by Adam, the BOHTA-I, and the BOHTA-P. Overall, algorithmic performance seems to exhibit favourable tendencies when Adam is coupled with either the BOHTA-P or the BOHTA-I. Although Adam exhibits F_1-score sample mean performance improvements (albeit empirically) in respect of eight data sets, *i.e.* C9, C22, C24, C29, C31, C37, and C40, the general tendency favours the consolidated approach of the BOHTA together with Adam. In the case of the BOHTA-I, superior performance in terms of sample mean is observed in respect of a vast majority of data sets. In the case of the BOHTA-P, on the other hand, performance improvements over Adam is less considerable, but are still worth noting. Table 8.7 contains an aggregation of the sample median and sample mean performances achieved by Adam, the BOHTA-I, and the BOHTA-P in respect of *all* data sets. There are slight, yet discernible, differences in average F_1-score sample median and sample mean performances amongst the different algorithms. A more accurate reflection of relative algorithmic performance may be obtained by conducting extensive statistical analyses.

TABLE 8.7: *Sample median and mean F_1-scores achieved by Adam, the BOHTA-I, and the BOHTA-P (averaged across all data sets), denoted by $\bar{F}_1$.*

	$\bar{F}_1$-score performance	
	Median	Mean
Adam	0.8964	0.8891
BOHTA-I	0.9180	0.9093
BOHTA-P	0.9001	0.8923

[1] The experimental setups pertain to the network structure (for the BOHTA and Adam), batch size, activation function and network weight initialisation procedure employed in the gradient-based training context, as well as the error function and the regularisation method.

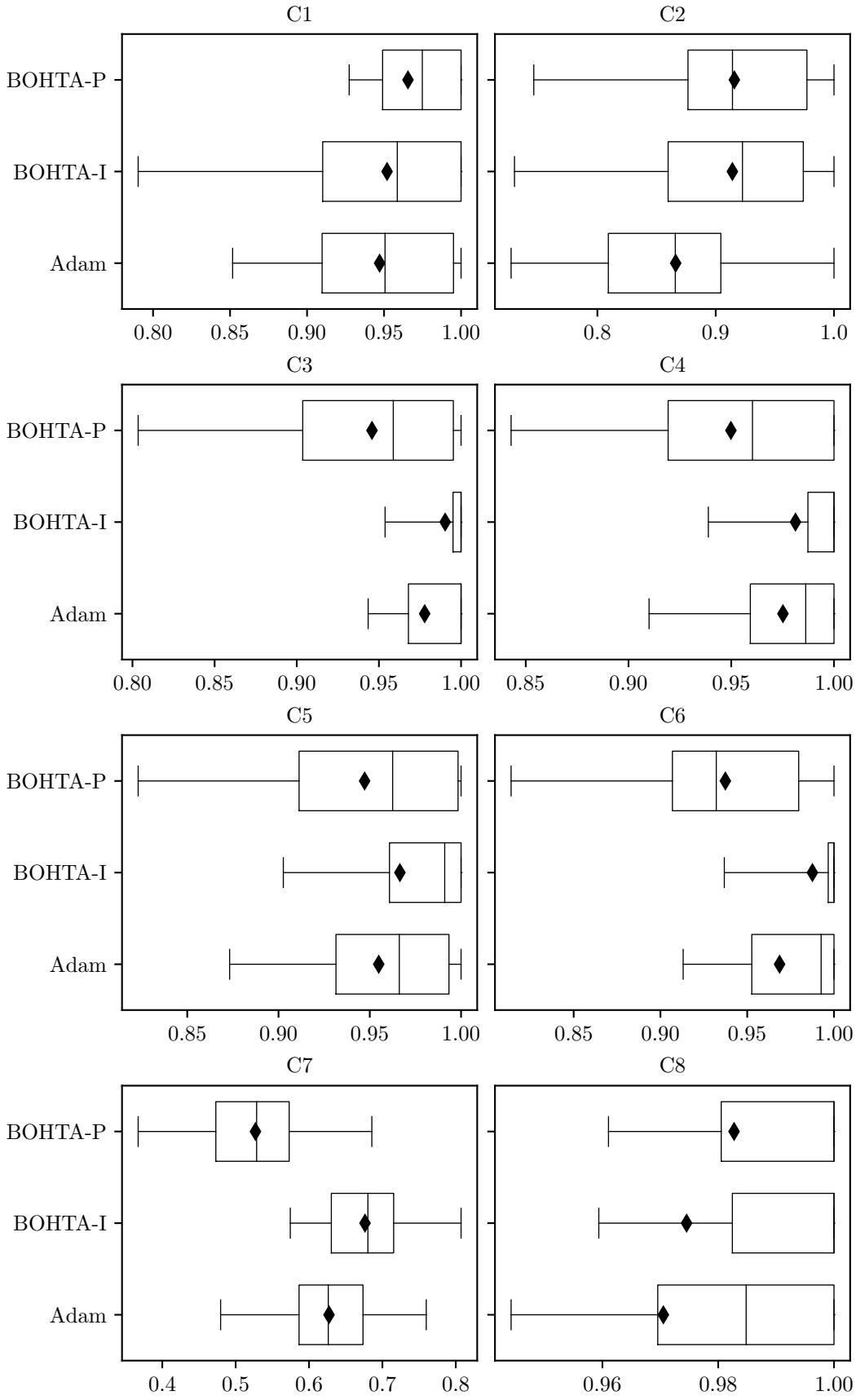


FIGURE 8.13: *Box plots of the F_1-scores achieved by the BOHTA-P, BOHTA-I, and Adam with respect to data sets C1–C8.*

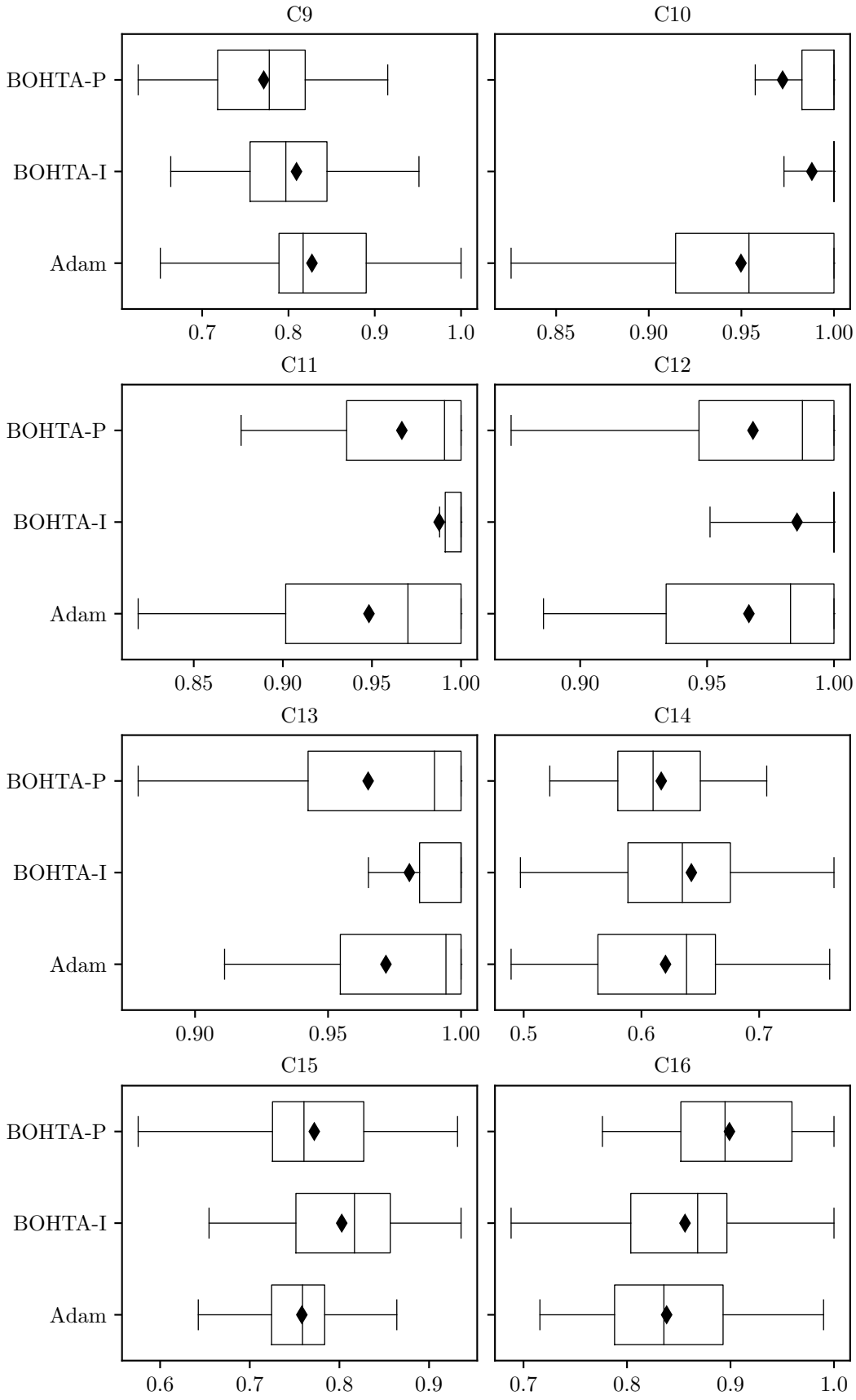


FIGURE 8.14: *Box plots of the F_1-scores achieved by the BOHTA-P, BOHTA-I, and Adam in respect of data sets C9–C16.*

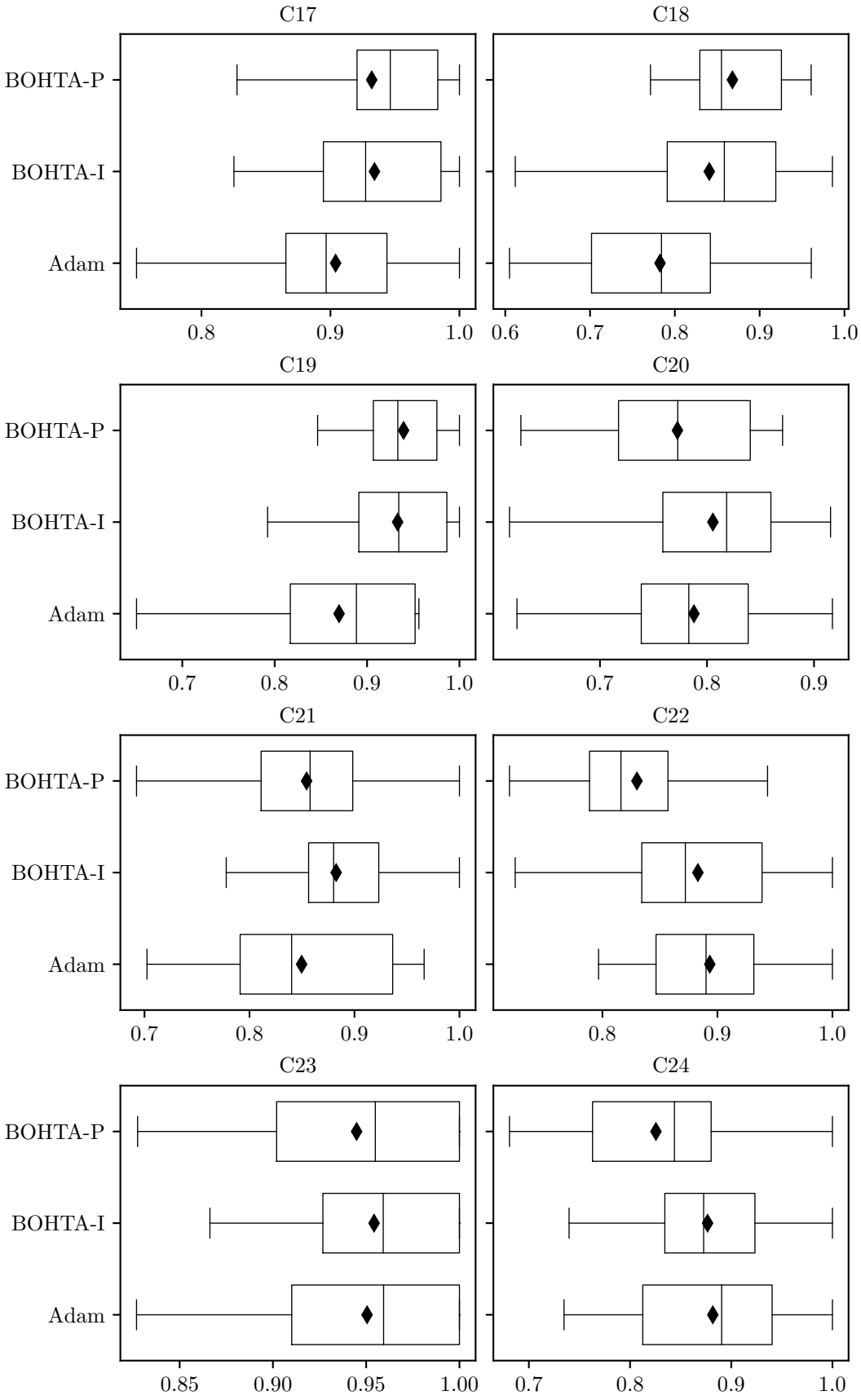


FIGURE 8.15: *Box plots of the F_1-scores achieved by the BOHTA-P, BOHTA-I, and Adam in respect of data sets C17–C24.*

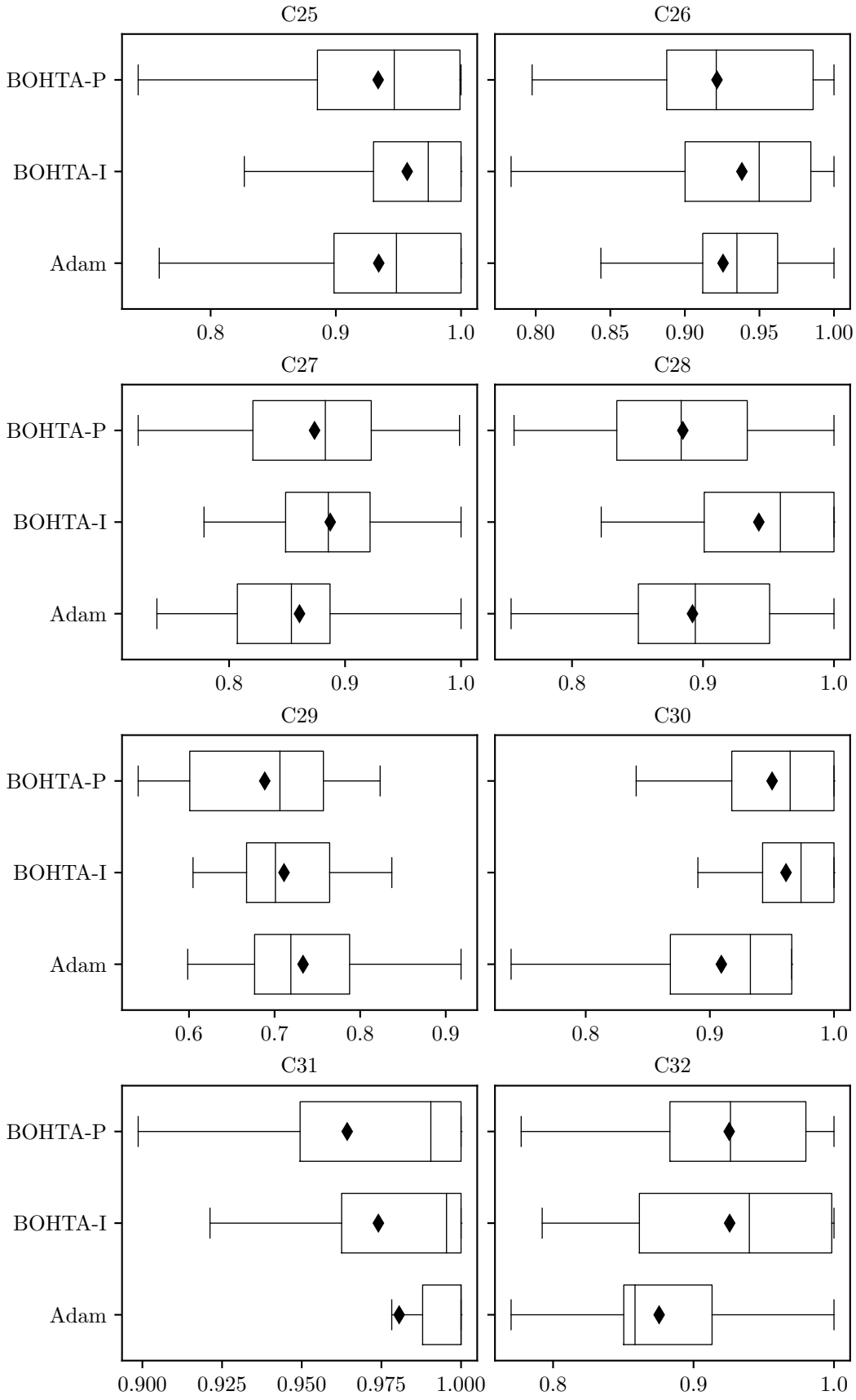


FIGURE 8.16: *Box plots of the F_1-scores achieved by the BOHTA-P, BOHTA-I, and Adam in respect of data sets C25–C32.*

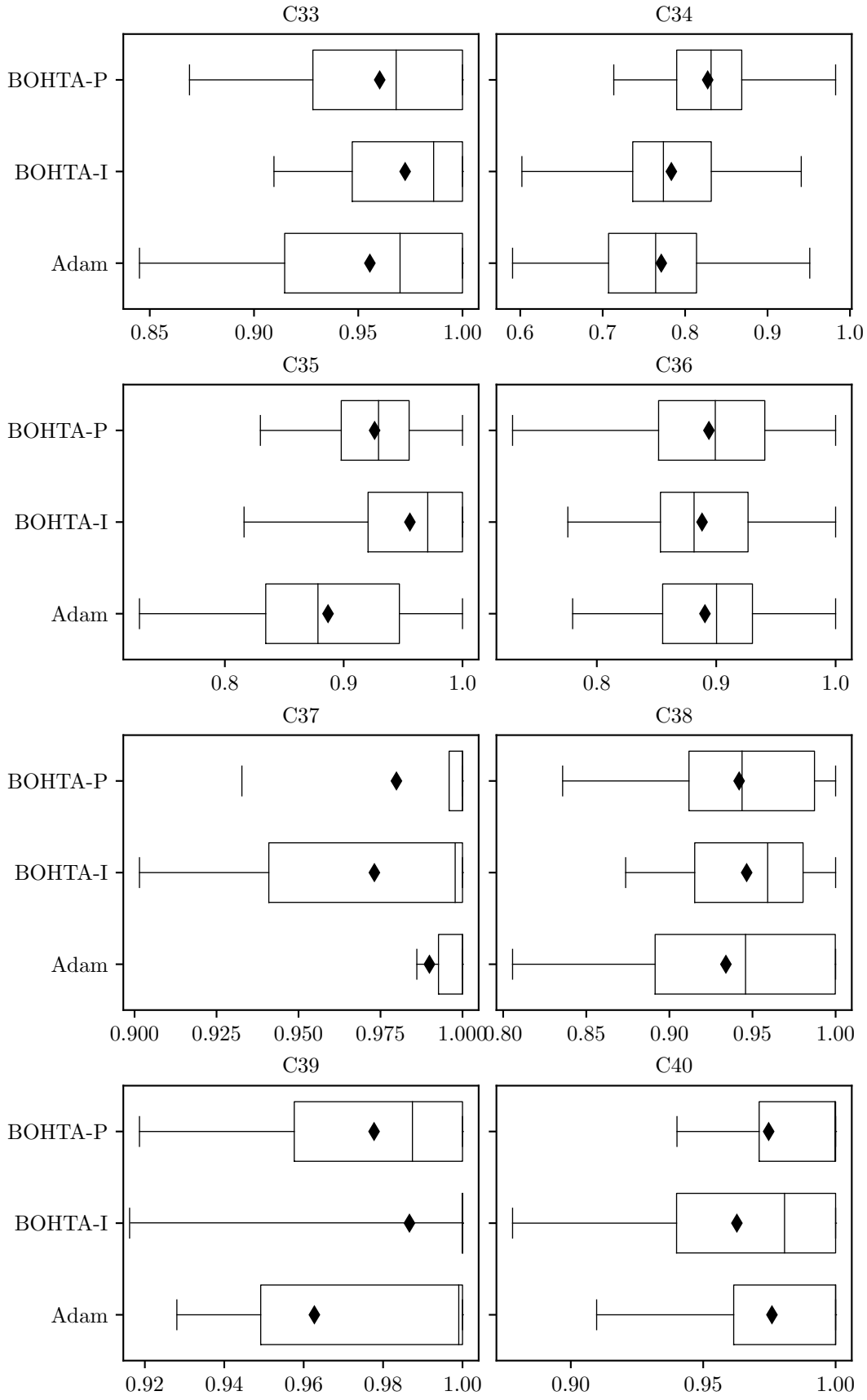


FIGURE 8.17: *Box plots of the F_1-scores achieved by the BOHTA-P, BOHTA-I, and Adam in respect of data sets C33–C40.*

The Friedman test is performed in order first to determine whether a significant difference exists between the performances of at least two samples (*i.e.* Adam, the BOHTA-I, and the BOHTA-P) at a 5% level of significance and then, if the Friedman test detects such a difference, the Nemenyi *post hoc* procedure is performed so as to identify the specific pairs of samples in which the differences are present. Table 8.8 contains the p-values obtained upon performing the Friedman test on the respective samples for each data set. Statistical differences are detected in only a few data sets, *i.e.* fourteen data sets contain a statistical difference at a 5% level of significance. Hence there is limited variation in performance between Adam, the BOHTA-I, and the BOHTA-P.

TABLE 8.8: *Friedman test p-values for the comparison between the sample medians of Adam, the BOHTA-P, and the BOHTA-I in respect of data sets C1–C40. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

	Friedman test		Friedman test
Data set	p-value	Data set	p-value
C1	0.1147	C21	0.0583
C2	0.0019	C22	0.0003
C3	0.0007	C23	0.9004
C4	0.0583	C24	0.0610
C5	0.1539	C25	0.4100
C6	0.0002	C26	0.8170
C7	0	C27	0.1500
C8	0.5255	C28	0.0022
C9	0.2418	C29	0.3061
C10	0.0175	C30	0.0024
C11	0.0057	C31	0.4759
C12	0.1890	C32	0.0195
C13	0.2418	C33	0.2985
C14	0.1219	C34	0.0696
C15	0.1219	C35	0.0002
C16	0.0366	C36	0.9683
C17	0.1890	C37	0.6092
C18	0.0002	C38	0.7411
C19	0.0005	C39	0.0696
C20	0.1177	C40	0.5255

The Nemenyi procedure is consequently applied in respect of each data set for which the corresponding Friedman test p-value is less than 0.05. The best overall training algorithm is established by the same procedure as that followed during the evaluation of the performance of the BOHTA when compared with those of the different gradient-based training algorithms in §8.3.1. Accordingly, for each data set, the training algorithms are first ranked according to sample medians, and thereafter the algorithms that are statistically equivalent to the best performing algorithm (*i.e.* the algorithms that are statistically indistinguishable according to the Nemenyi procedure) are identified. Finally, the relative algorithmic performances are determined by calculating the frequency with which an algorithm is ranked first and the frequency with which an algorithm is statistically equivalent to the algorithm that is ranked first. Figure 8.18 contains the respective frequencies for Adam, the BOHTA-I, and the BOHTA-P. It is apparent that both the BOHTA-I and the BOHTA-P outperform Adam in respect of the data sets for which performance is statistically different. Overall, the BOHTA-I is the best performing algorithm. It may also be inferred that the computational compromises made to facilitate the collaborative algorithmic approach has been mitigated satisfactorily.

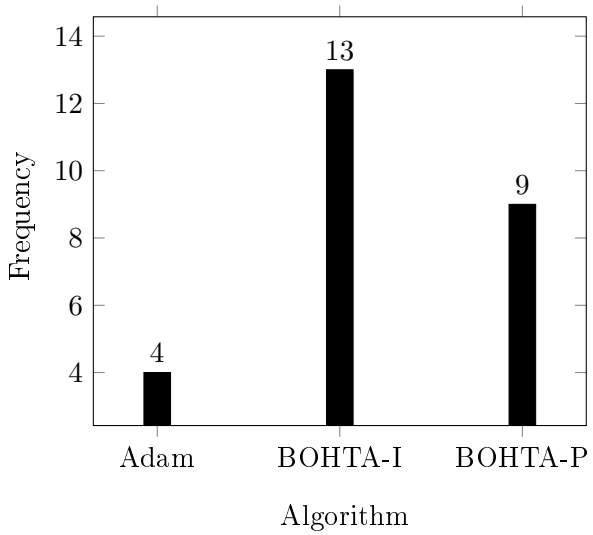


FIGURE 8.18: *The frequencies with which the Adam, the BOHTA-P, and the BOHTA-I are either ranked first or are statistically equivalent to the algorithm that is ranked first.*

8.4 Algorithmic performance prediction

Both the BOHTA parameter evaluation in §7.3.2 and the algorithmic comparative study in §8.1 alluded to some potential, albeit cursory, insight into the dynamics and operation of the BOHTA and its sub-algorithms. Appropriately, the aim in this section is, first, to consider these dynamics in more detail by conducting an analysis on a lower level of abstraction and, secondly, to apply the temporal algorithmic insight gained from this analysis towards predicting algorithmic performance — more specifically, to predict the performances achieved by the sub-algorithms constituting the BOHTA using the meta-features of a data set (discussed in §3.9.2). Statistical white-box learning algorithms form the basis of the algorithmic performance prediction. Recall from §2.2 that tree-based statistical learning algorithms, *e.g.* the CART and C4.5 decision tree algorithms, are both powerful and interpretable[2], therefore their consideration in the context of algorithmic performance prediction is well warranted.

The nature of the adopted solution methodology facilitates an investigation into the operation of the BOHTA and its sub-algorithms, allowing for the attainment of pertinent insight into the temporal dynamics of the three sub-algorithms during the BOHTA's optimisation procedure in respect of the different data sets. This niche subject matter has, to the best of the author's knowledge, not been addressed in a way similar to the subsequent exposition. The discourse focusses specifically on the best performing parameter combination of the BOHTA (found in §7.3.2) so as to investigate the factors that potentially contribute to the success of the constituent sub-algorithms and, of course, the BOHTA. To facilitate this analysis, samples are computed by executing the BOHTA thirty times in respect of each data set and recording the number of solutions generated by each sub-algorithm over each optimisation run (*i.e.* at each generation).

In order to gain further insight into the BOHTA's temporal dynamics, the samples of results are synthesised. Consider, for example, the BOHTA applied to data set C6 with a static stopping criterion according to which the optimisation procedure is terminated after the first 100 generations — the total number of generations is fixed only for the sake of visualisation and demonstration purposes. Recall from §6.1 that the BOHTA employs a dynamic stopping crite-

[2]Decision trees produce so-called rule formulations which elucidate the predictions made by the algorithms.

rion (*i.e.* early stopping), and the total number of generations before termination typically differs from one optimisation run to another because of the notable degree of stochasticity present within this optimisation context. Figure 8.19 contains a graphical illustration of the aggregation of values recorded during all thirty optimisation runs — *i.e.* the mean number of solutions generated is aggregated per generation, in respect of all thirty optimisation runs and plotted accordingly. The horizontal axis represents the normalised generation range, denoted by t_{norm}. The number of generations is normalised to the range $[0, 1]$ so as to facilitate a meaningful analysis of the temporal dynamics. It should be noted, however, that the entire optimisation run (not just the first 100 generations) is taken into account when determining *sub-algorithm superiority* (discussed later in this section).

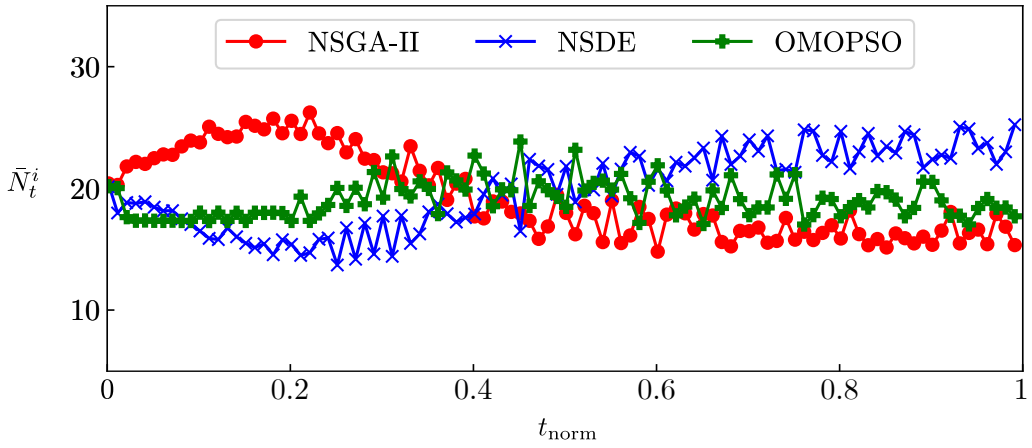


FIGURE 8.19: *The aggregated number of solutions generated by each sub-algorithm, calculated over all optimisation runs, in respect of data set C6. The horizontal axis represents the normalised generation range. Results for only the first 100 generations are shown. In this case, the BOHTA employed the best parameter combination, i.e. B10.*

The illustration in Figure 8.19 conveys the general tendencies exhibited by the sub-algorithms across the truncated optimisation process, thereby providing a representation of the temporal dynamics of the BOHTA. There are a few noteworthy insights to be gained from this illustration, the first of which relates to the strong performance of the NSGA-II at the start of the optimisation process. This can be attributed to the proficiency of its evolutionary operators in the context of global optimisation (*i.e.* exploration-based search). The key contributing factors are the large-valued crossover probability and medium-valued mutation probability (*i.e.* $p_c = 1.00$ and $p_m = 0.05$, respectively). There is, however, a notable change in the dynamics at $t_{\text{norm}} \approx 0.5$, where both NSDE and OMOPSO overtake the NSGA-II. For a brief period (at $t_{\text{norm}} \approx 0.4$), OMOPSO is the best overall performer, although it does exhibit diminishing performance gains mid-way through the optimisation process — the performance achieved by OMOPSO is only slightly better than that of the NSGA-II (towards the end). The crossover rate employed by NSDE is medium in magnitude (*i.e.* $C_R = 0.30$) and it is conjectured that this contributes to its utility in the context of an exploitation-based search and hence its success during the latter stages of the optimisation process. In the case of OMOPSO, the large mutation probability (*i.e.* $p_m = 0.1$) most likely contributes to its relatively strong performance during the early to mid-way stages of the optimisation procedure — when exploration is more important than exploitation. Overall, all three sub-algorithms appear to contribute somewhat equally to the optimisation process, although a more extensive investigation would be required to verify these arguably precursory findings. The transitory nature of the BOHTA's sub-algorithms (in respect of the three sub-algorithms) lends further vindication to the utility of a dynamic optimisation

procedure such as the BOHTA. It is evident that as the population of networks evolves over time, different evolutionary operators are required to improve upon the networks' parameters (*i.e.* network weights, structure, and activation functions) — a manifestation of the NFL theorem.

Interestingly, the temporal dynamics of the BOHTA (in respect of the data set C6) coincide with the findings of Vrugt and Robinson [171]. In the original implementation of AMALGAM, a BOP called ZDT4, which is a non-convex optimisation problem, constituted a central part of their discussion. Due to the non-convex nature of the current BOP at hand, a comparison with the findings of Vrugt and Robinson is considered natural. Accordingly, the reproduction success of AMALGAM's NSGA-II starts off strongly, and this is followed by a notable decrease mid-way through the optimisation process — a marked similarity with the observations in Figure 8.19. Furthermore, AMALGAM's DE starts off poorly, although a clear improvement in reproduction success is exhibited from mid-way onwards, which is, once again, similar to the observed dynamics of the BOHTA. Lastly, the reproduction success of AMALGAM's PSO also indicates comparatively weak performance — OMOPSO is, however, only marginally the weakest overall performer. The fluctuations (or oscillations) exhibited by the BOHTA's NSDE and OMOPSO (shown in Figure 8.19), furthermore, correspond to the fluctuating dynamics exhibited by AMALGAM's DE and PSO. The evident degree of similarity between the findings of the two respective studies are noteworthy, suggesting that AMALGAM's implementation in other optimisation contexts can provide additional insight into its application within the current context of training FNNs. In conclusion, Vrugt and Robinson declared that their findings provided numerical evidence of Wolpert and Macready's NFL theorem [178], claiming that it is "impossible to develop a single search algorithm that will always be superior to any other algorithm." The initial empirical findings of the analysis performed in this dissertation corroborate these claims, although in the optimisation context of training FNNs.

The preceding analyses were carried out at a high level of abstraction — *e.g.* Figure 8.19 illustrates an aggregation of the BOHTA's temporal dynamics. Additional insights can, however, be gained by investigating the samples on a notably lower level of abstraction. A micro-level representation is obtained by analysing the dynamics of individual samples (or optimisation runs). Figure 8.20 contains one such representation, in which the number of solutions generated during a single, arbitrary optimisation run is illustrated graphically. A noteworthy observation in this micro-level representation relates to the emulation of the tendencies manifested in the high-level representation of Figure 8.19. The two most discernible similarities are, first, the strong start by the NSGA-II and, secondly, its subjugation by NSDE from midway until the end. Another noteworthy observation in this micro-level representation relates to OMOPSO. Upon deeper inspection (facilitated by Figure 8.20), however, OMOPSO exhibits large increases (upward spikes) in its reproduction success between $t_{\text{norm}} \approx 0.4$ and $t_{\text{norm}} \approx 0.6$, resulting in a large contribution towards the generation of solutions. These micro-level fluctuations are, to a certain extent, corroborated by the findings of the aggregated dynamics in Figure 8.19, in which notable fluctuations are exhibited by OMOPSO during the middle parts of the optimisation process. Coincidentally, OMOPSO exhibits its sudden performance increase during the transitioning period of the NSGA-II and NSDE, which may suggest that disparate areas in the solution space are being explored.

The analysis of the BOHTA's temporal dynamics can be enhanced by investigating the number of generations during which a sub-algorithm generates, on average, the largest number of solutions. This measure provides an indication of the frequency with which a sub-algorithm exhibits superiority[3] relative to its counterparts. It should be noted that in all subsequent analyses the

[3]Superiority here means that a sub-algorithm is not outperformed by any other sub-algorithm in terms of number of solutions generated.

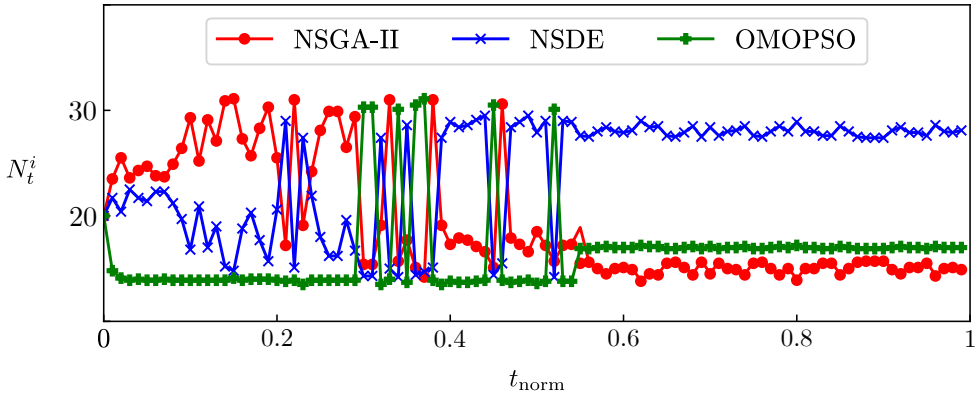


FIGURE 8.20: *The number of solutions generated by each sub-algorithm during one of the thirty optimisation runs (over the first 100 generations). This illustration represents a typical optimisation run in respect of the data set C6, where parameter combination B10 is employed by the BOHTA.*

entire optimisation run is considered (not just the first 100 generations). Figures 8.21 and 8.22 illustrate the aforementioned measure graphically, depicting the normalised mean number of generations during which each sub-algorithm generates the largest number of solutions, denoted by $\bar{t}_{\text{norm}}$, in respect of all data sets within the test suite. There are some interesting observations that emerge when analysing these temporal dynamics along with the findings of the algorithmic comparative study conducted in §8.1. The first noteworthy observation relates to data set C1. The performances achieved by the individual application of the three sub-algorithms — *i.e.* statistically significant performance improvements by the NSGA-II[4] over its counterparts (at a 5% level of significance) — led to the expectation that it ought to dominate during the optimisation process. The temporal dynamics of the BOHTA, however, indicate that OMOPSO delivers slightly improved reproduction success and superiority to those of the NSGA-II and even more significant improved reproduction success when compared with NSDE. This phenomenon can possibly be attributed to the suitability of OMOPSO's evolutionary operators in respect of this data set — *i.e.* the combined operation of the three sub-algorithms can result in the BOHTA operating in new (different) areas of the solution space, to which the OMOPSO (along with its evolutionary operators) is better suited.

A similar observation relates to data set C2. According to the findings of the algorithmic comparison, NSDE and OMOPSO delivers a statistically significant performance improvement over the NSGA-II, but its temporal dynamics exhibit surprising results. Accordingly, the NSGA-II generates, on average, considerably more solutions than NSDE and OMOPSO. This is another peculiar phenomenon and can, once again, be attributed to a lack of suitability of their evolutionary operators in the new context, *i.e.* the combined operation of the three sub-algorithms result in the BOHTA operating in new (different) areas of the solution space, to which the NSDE and OMOPSO (along with their evolutionary operators) are less suited.

The temporal dynamics of the BOHTA provide some fascinating insight into the working of the BOHTA and its constituent sub-algorithms within the context of training FNNs. These findings facilitate a greater understanding of the working of these algorithms and ought to be utilised and exploited so as to both elucidate the underpinnings of why certain sub-algorithms favour certain problems and to use this information to potentially improve the BOHTA's performance. Appropriately, the notion of algorithmic superiority forms the basis of the subsequent investigation into algorithmic performance prediction. The C4.5 decision tree algorithm[5] is applied

[4]The performance of the NSGA-II is statistically indistinguishable from that of the BOHTA which, in turn, is statistically superior to NSDE and OMOPSO.

[5]Described in §2.2.2 and implemented in [30] using default parameter settings.

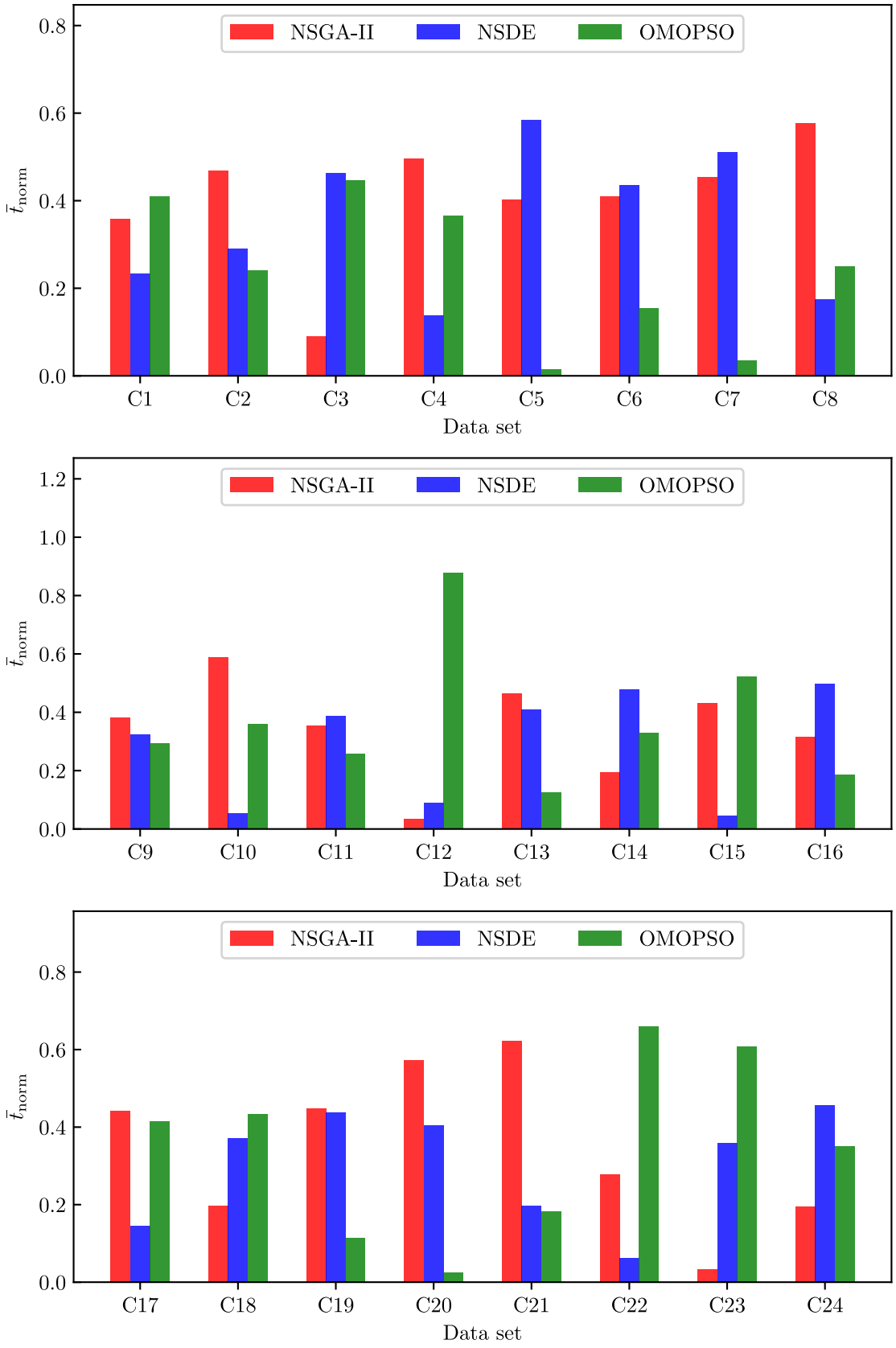


FIGURE 8.21: *The normalised mean frequency with which a sub-algorithm exhibits superiority relative to its counterparts in respect of data sets C1–C24. The values are averaged across all thirty optimisation runs.*

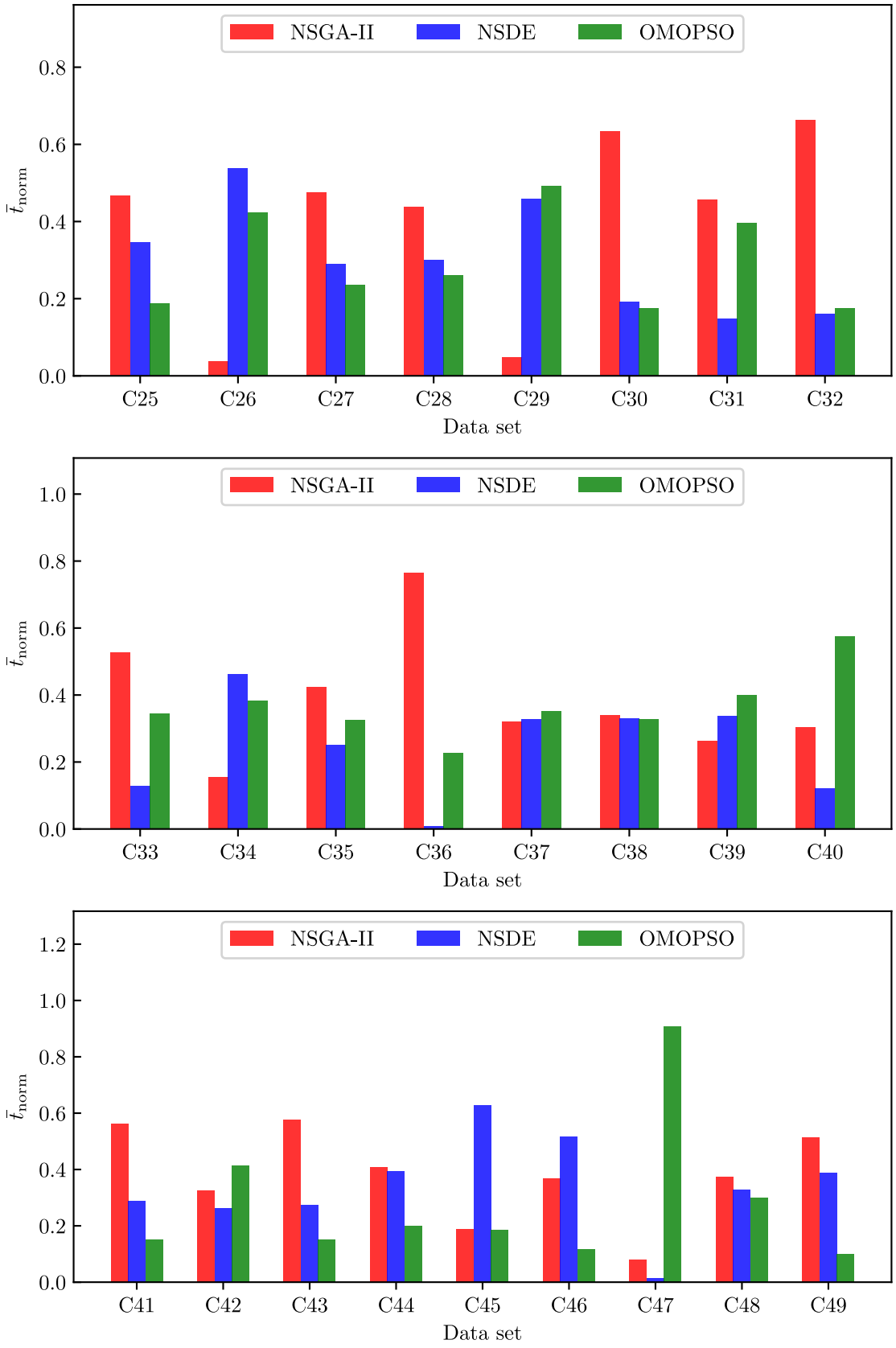


FIGURE 8.22: *The normalised mean frequency with which a sub-algorithm exhibits superiority relative to its counterparts in respect of data sets C25–C49. The values are averaged across all thirty optimisation runs.*

in respect of the meta-features of the test suite together with the corresponding sub-algorithm superiority findings. The aim is twofold: First, to investigate whether algorithmic performance (*i.e.* superiority) can be predicted in a supervised learning context according to which the independent variables are the meta-features of the data sets (presented in Table 6.2) while the target variable is the superior sub-algorithm of the BOHTA (summarised in Table 8.9). The supervised learning problem at hand is therefore a classification problem in which the task is to predict which BOHTA sub-algorithm will be superior overall in respect of a particular data set. Overall superiority simply corresponds to the sub-algorithm that achieves the greatest normalised mean frequency with which it is not outperformed by any other sub-algorithm in terms of solutions generated.

TABLE 8.9: *A summary of the normalised sub-algorithm superiority in respect of the entire test suite.*

	Normalised superiority				Normalised superiority		
Data set	NSGA-II	NSDE	OMOPSO	Data set	NSGA-II	NSDE	OMOPSO
C1	0.3580	0.2326	0.4094	C26	0.0381	0.5387	0.4232
C2	0.4694	0.2895	0.2411	C27	0.4755	0.2889	0.2356
C3	0.0902	0.4630	0.4468	C28	0.4387	0.3008	0.2606
C4	0.4967	0.1378	0.3655	C29	0.0482	0.4596	0.4922
C5	0.4015	0.5843	0.0142	C30	0.6335	0.1912	0.1753
C6	0.4105	0.4346	0.1549	C31	0.4560	0.1475	0.3965
C7	0.4536	0.5107	0.0356	C32	0.6632	0.1608	0.1760
C8	0.5761	0.1745	0.2494	C33	0.5274	0.1276	0.3450
C9	0.3823	0.3246	0.2931	C34	0.1551	0.4626	0.3823
C10	0.5880	0.0530	0.3590	C35	0.4240	0.2498	0.3262
C11	0.3551	0.3884	0.2566	C36	0.7639	0.0081	0.2280
C12	0.0334	0.0899	0.8768	C37	0.3211	0.3274	0.3515
C13	0.4643	0.4097	0.1260	C38	0.3403	0.3308	0.3289
C14	0.1931	0.4769	0.3300	C39	0.2637	0.3363	0.3999
C15	0.4323	0.0460	0.5217	C40	0.3029	0.1213	0.5758
C16	0.3157	0.4976	0.1868	C41	0.5620	0.2873	0.1507
C17	0.4417	0.1438	0.4145	C42	0.3238	0.2621	0.4140
C18	0.1961	0.3716	0.4323	C43	0.5753	0.2748	0.1499
C19	0.4471	0.4384	0.1145	C44	0.4075	0.3931	0.1994
C20	0.5717	0.4042	0.0241	C45	0.1875	0.6284	0.1842
C21	0.6219	0.1967	0.1815	C46	0.3679	0.5161	0.1160
C22	0.2780	0.0620	0.6600	C47	0.0792	0.0126	0.9082
C23	0.0328	0.3592	0.6080	C48	0.3745	0.3271	0.2983
C24	0.1939	0.4561	0.3500	C49	0.5134	0.3864	0.1003
C25	0.4661	0.3454	0.1885	—	—	—	—

The final data set (comprising meta-features and sub-algorithm superiority) which corresponds to the supervised learning problem of algorithmic performance prediction is shown in Table 8.10. This data set is henceforth referred to as the meta data set. There are, of course, a total of 49 instances (or observations), each of which corresponds to a different data set. The meta data set is further partitioned into a training and testing set which comprise approximately 80% and 20% of the data set instances, respectively. Coincidentally, this partitioning scheme results in the first forty data sets, *i.e.* C1–C40, forming part of the training set, while the testing set comprises the remaining nine data sets, *i.e.* C41–C49. A validation set is not considered within the context of this supervised classification task, because early stopping is not used. Finally,

TABLE 8.10: *Algorithmic performance prediction data set, also referred to as the meta data set.*

		Independent variables						**Dependent variable**
Data set	Task	Size	n	$\cdots$	$\bar{\zeta}$	max(ζ)	$\bar{H}$	Sub-algorithm superiority
C1	binary	30 162	97	$\cdots$	2.106	11.903	0.810	OMOPSO
C2	binary	1 470	30	$\cdots$	0.584	1.984	0.637	NSGA-II
C3	binary	775	25	$\cdots$	7.218	20.526	0.967	NSDE
C4	multi-class	186	79	$\cdots$	-0.066	3.173	1.169	NSGA-II
C5	binary	4 119	52	$\cdots$	0.644	4.023	0.498	NSDE
C6	binary	1 372	4	$\cdots$	-0.051	1.089	0.991	NSDE
C7	binary	116	9	$\cdots$	1.609	3.812	0.992	NSDE
C8	binary	569	30	$\cdots$	1.667	5.447	0.953	NSGA-II
C9	multi-class	106	9	$\cdots$	2.286	7.270	2.565	NSGA-II
C10	multi-class	1 728	15	$\cdots$	0.000	0.000	1.206	NSGA-II
C11	binary	232	16	$\cdots$	-0.112	0.657	0.997	NSDE
C12	multi-class	10 845	28	$\cdots$	0.205	1.328	1.468	OMOPSO
C13	multi-class	358	155	$\cdots$	1.372	4.720	0.538	NSGA-II
C14	binary	496	152	$\cdots$	-0.024	0.973	0.980	NSDE
C15	multi-class	1 885	12	$\cdots$	0.217	2.572	2.174	OMOPSO
C16	multi-class	1 885	12	$\cdots$	0.217	2.572	2.157	NSDE
C17	multi-class	1 885	12	$\cdots$	0.217	2.572	1.456	NSGA-II
C18	multi-class	1 885	12	$\cdots$	0.217	2.572	2.152	OMOPSO
C19	multi-class	1 885	12	$\cdots$	0.217	2.572	1.330	NSGA-II
C20	multi-class	1 885	12	$\cdots$	0.217	2.572	2.687	NSGA-II
C21	multi-class	1 885	12	$\cdots$	0.217	2.572	1.776	NSGA-II
C22	multi-class	1 885	12	$\cdots$	0.217	2.572	1.981	OMOPSO
C23	multi-class	1 885	12	$\cdots$	0.217	2.572	0.837	OMOPSO
C24	multi-class	1 885	12	$\cdots$	0.217	2.572	2.036	NSDE
C25	multi-class	1 885	12	$\cdots$	0.217	2.572	0.942	NSGA-II
C26	multi-class	1 885	12	$\cdots$	0.217	2.572	1.186	NSDE
C27	multi-class	1 885	12	$\cdots$	0.217	2.572	1.955	NSGA-II
C28	multi-class	1 885	12	$\cdots$	0.217	2.572	1.384	NSGA-II
C29	multi-class	1 885	12	$\cdots$	0.217	2.572	2.560	OMOPSO
C30	multi-class	1 885	12	$\cdots$	0.217	2.572	1.196	NSGA-II
C31	binary	10 000	13	$\cdots$	0.000	0.019	0.944	NSGA-II
C32	multi-class	194	61	$\cdots$	1.902	8.570	2.489	NSGA-II
C33	multi-class	523	27	$\cdots$	0.184	3.235	1.903	NSGA-II
C34	binary	579	10	$\cdots$	2.659	10.512	0.862	NSDE
C35	multi-class	150	4	$\cdots$	0.067	0.334	1.585	NSGA-II
C36	binary	1 163	20	$\cdots$	5.738	28.638	0.977	NSGA-II
C37	multi-class	731	234	$\cdots$	0.302	1.239	1.956	OMOPSO
C38	multi-class	148	41	$\cdots$	0.737	5.442	1.228	NSGA-II
C39	binary	5 936	94	$\cdots$	0.487	27.191	0.947	OMOPSO
C40	multi-class	12 960	19	$\cdots$	0.000	0.000	1.716	OMOPSO
C41	binary	182	12	$\cdots$	-0.077	0.957	0.863	NSGA-II
C42	binary	1 055	41	$\cdots$	3.799	26.846	0.922	OMOPSO
C43	binary	250	12	$\cdots$	-0.040	0.103	0.985	NSGA-II
C44	multi-class	210	7	$\cdots$	0.234	0.562	1.585	NSGA-II
C45	binary	143	6	$\cdots$	-0.452	0.285	0.996	NSDE
C46	binary	2 201	5	$\cdots$	0.978	4.156	0.908	NSDE
C47	multi-class	846	18	$\cdots$	1.042	6.778	1.999	OMOPSO
C48	multi-class	178	13	$\cdots$	0.333	1.098	1.567	NSGA-II
C49	multi-class	101	16	$\cdots$	0.401	3.163	2.391	NSGA-II

the meta data set is not pre-processed, which stands in contrast with the approach adopted in respect of data sets C1–C49 (discussed in §6.2.1). Data pre-processing was necessary in the context of training ANNs, because the nature of the presented data can have an adverse effect on ANN performance. In a qualitative pilot study performed by the author, it was found that the performance of the decision tree algorithm does not change (or improve) when pre-processing is applied. Consequently, the meta data set was not subjected to any pre-processing steps before the decision tree algorithm was applied.

The training and testing F_1-score performances achieved by the C4.5 decision tree algorithm are 0.820 and 0.770, respectively. A separate qualitative pilot study conducted by the author yielded no performance improvements when changing the default parameter values of the algorithm. The best (empirical) performance was achieved by inducing a binary tree (*i.e.* producing only two child nodes at each split) and setting the parameter value for *minimum number of instances in leaves*[6] to four. Based on both the training and testing performances achieved by the decision tree algorithm, it is apparent that the model possesses sufficient predictive capability. The prediction model is therefore able to infer appropriate rule formulations based on the input features constituting the meta data set (presented in Table 8.10) so as to make accurate predictions in respect of which sub-algorithm is expected to be superior.

The rules *learnt* by the C4.5 decision tree algorithm are presented in Figure 8.23. It may be inferred that the meta-feature contributing the most towards predicting which sub-algorithm is superior is the size of the data set. If the data set under consideration comprises more than 4 119 instances, then the decision tree algorithm predicts OMOPSO to be the superior algorithm. If, on the other hand, the data set comprises 4 119 instances or fewer, then the number of classes o (in conjunction with the data set size) should be used to predict which sub-algorithm will be superior. In the case of binary classification problems, the number of numerical features n_{num} can be used to help predict whether the superior algorithm will be the NSGA-II or NSDE. Accordingly, if there are nine numerical features or fewer, then the NSDE is predicted as superior, whereas if the number of numerical features is more than nine, then the NSGA-II is predicted to be superior. In the case of multi-class classification problems, on the other hand, the mean class entropy $\bar{H}$, together with the data set size, can help predict whether the superior algorithm is either the NSGA-II or OMOPSO. In the case of $\bar{H} \leq 1.955$, the superior sub-algorithm is (most likely) the NSGA-II. If, on the other hand, the mean class entropy satisfies the condition $1.955 < \bar{H} \leq 2.174$, then the superior sub-algorithm is predicted to be OMOPSO, while if $\bar{H} > 2.174$, then the NSGA-II is again predicted to be superior. The meta-features that underpin the aforementioned rule formulations are therefore instrumental in predicting sub-algorithm success.

An arguably simple investigation may be carried out to determine whether the "clairvoyance" afforded by the reasonably successful prediction model can be used to improve the performance of the BOHTA in respect of data sets C41–C49. Accordingly, an investigation is conducted into allocating greater (relative) computational capacity to the sub-algorithm that is predicted to exhibit superiority beforehand in respect of each of the data sets (shown in Table 8.10). Towards this end, lower bound proportions specific to the respective sub-algorithms are introduced and are denoted by $\tilde{N}^i$ for sub-algorithm $i \in \{G, D, P\}$, where G, D, and P denote the NSGA-II, NSDE, and OMOPSO, respectively. The following procedure is adopted when allocating greater computational capacity based on the inferences drawn from the algorithmic prediction model. Consider, for example, data set C41 for which the predictive model predicts that the NSGA-II will be superior. A very large lower bound proportion is therefore assigned to this sub-algorithm, *i.e.* $\tilde{N}^G = 0.20$, whereas the lower bound proportions of the remaining sub-algorithms remain

[6] The decision tree algorithm never constructs a split which would put less than the specified number of training examples into any of the branches.

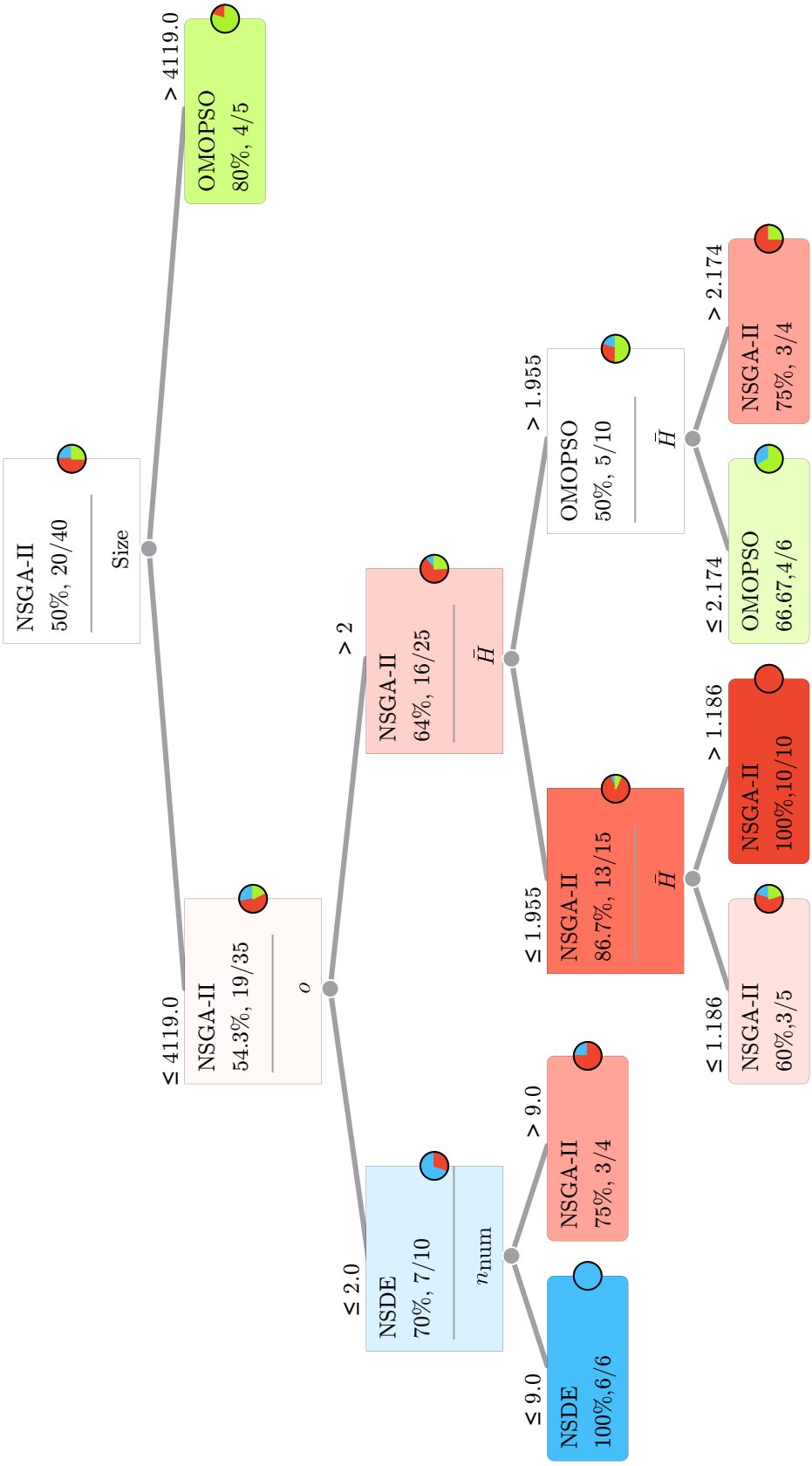


FIGURE 8.23: *Algorithmic performance prediction rule formulations produced by the C4.5 decision tree algorithm for the ternary classification problem with data as summarised in Table 8.10.*

unchanged, *i.e.* $\tilde{N}^G = 0.15$ and $\tilde{N}^G = 0.15$. The BOHTA using parameter combination B10 (which employs a universal lower bound proportion of $\tilde{N} = 0.15$) is subsequently compared with a new incarnation of the BOHTA according to which the sub-algorithm lower bound proportions are determined by means of the aforementioned predictive procedure. The Friedman test is again performed in respect of the nine data sets in order to determine for which data sets a statistical difference exists at a 5% level of significance. The Nemenyi *post hoc* procedure is not subsequently applied because there are only two algorithms under consideration — the Friedman test therefore already provides an indication of where statistical differences exist and where not. The resulting Friedman test p-values are shown in Table 8.11 together with the corresponding sample median and mean. The box plots in Figure 8.24 supplement this comparative study. Overall, the enhanced version of the BOHTA delivers statistically improved performance over the normal version of the BOHTA in respect of data sets C41–C49. On average, an improvement of 0.0576 in F_1-score sample mean performance is achieved — a showcase of the utility afforded by predicting algorithmic superiority and allocating computational resources accordingly.

TABLE 8.11: *Friedman test p-values together with sample medians and means achieved by the BOHTA (using parameter combination B10) and a version of the BOHTA enhanced by predictive modelling in respect of data sets C41–C49. A table entry less than 0.05 (indicated in red) denotes a difference at a 5% level of significance.*

Data set	Friedman test p-value	BOHTA Median	Enhanced BOHTA Median	BOHTA Mean	Enhanced BOHTA Mean
C41	0.0258	0.5424	0.6282	0.5493	0.6194
C42	0.7216	—	—	—	—
C43	0.1470	—	—	—	—
C44	0	0.8508	0.9791	0.8434	0.9654
C45	0.0081	0.3572	0.4174	0.3648	0.4384
C46	0.4746	—	—	—	—
C47	0.0425	0.9182	0.8767	0.9130	0.8696
C48	0.0020	0.8577	0.9231	0.8593	0.9248
C49	0.1470	—	—	—	—
Average		0.7053	0.7649	0.7059	0.7635

There are, however, a few limitations of the aforementioned pilot study. The fist limitation pertains to the size of the meta data set that formed the basis of the algorithmic performance prediction investigation, which comprises only 49 data sets (*i.e.* observations) — an arguably small sample size from which to draw inferences. The second limitation pertains to the prediction performance levels achieved by the C4.5 decision tree algorithm (*i.e.* training and testing F_1-score performances of 0.820 and 0.770, respectively). Although the performance achieved is certainly not dubious, the veracity of the prediction ought still to be accompanied by some scrutiny. The performance improvements (shown in Table 8.11) are certainly indicative of a sub-optimal predictive model — in the case of data set C47, the performance of the enhanced BOHTA was found to be statistically inferior at a 5% level of significance. This inferior performance can be ascribed to the predictive model incorrectly predicting OMOPSO as the superior sub-algorithm due to the fallible rule formulations. Nevertheless, the algorithmic benefit of providing the BOHTA with some computational foresight is certainly an approach that possesses promise and although the approach adopted is arguably rudimentary, its promising results warrant a more extensive investigation as part of future work.

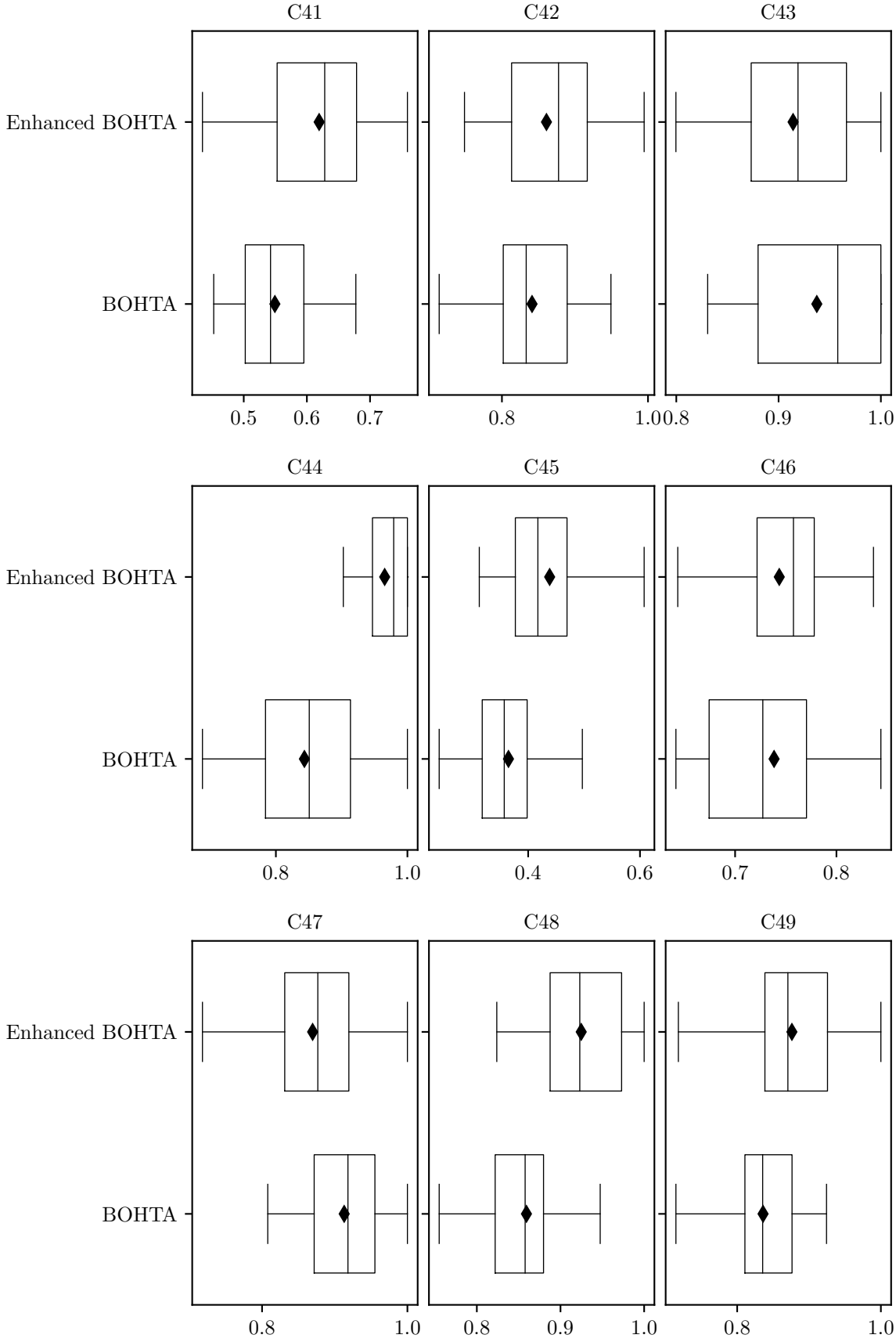


FIGURE 8.24: *Box plots of the F_1-scores achieved by the BOHTA using parameter combination B10 and the BOHTA enhanced by predictive modelling in respect of data sets C41–C49.*

8.5 Structural analysis

The discussion now turns to the structural attributes of the networks produced by the BOHTA. To facilitate this analysis, samples were computed by executing the BOHTA thirty times in respect of each data set and recording the network structures of solutions in the approximate Pareto front at the end of each optimisation run. The parameter combination employed by the BOHTA corresponds to the best performing combination found during the parameter evaluation in §7.3.2 — the intuition is to maximise the likelihood of obtaining the most favourable network structures. The resulting samples therefore contain valuable information about favourable network structures (characterised by their depths and widths) in respect of each data set.

Recall from the experimental setup in §7.2 that the network depth is limited to a maximum of $h = 3$ hidden layers. The reason for this is twofold: First, due to the relatively simple nature of the data sets in the test suite and, secondly, to limit the computational burden experienced by the training algorithm whilst still affording enough computational capability to the network. Accordingly, the minimum and maximum number of non-redundant hidden layers are zero and three, respectively. In the case of the network width, on the other hand, the upper bounds depend on the input- and output layer sizes (according to the commonly adopted heuristic suggested by Heaton [60]). Recall from §7.2 that the number of neurons per hidden layer is limited to at most $m = 2\max\{n, o\}$, *i.e.* the largest number of neurons per hidden layer is assigned a value twice the size of the maximum between the input and output layer sizes. Before the findings are scrutinised, emphasis is first placed on the multi-faceted nature of the problem at hand — the performance of the network structure depends also on the network weights and the activation functional variables. Consequently, the findings of this section only suggest (as opposed to guarantee) network structures that exhibit favourable performance in the given context. The network structures corresponding to the top performing solutions (in terms of F_1-scores) can be found in in Table 8.12.

It may be observed that the values of $\sum_j \bar{s}_j^{(1)}$, $\sum_j \bar{s}_j^{(2)}$, and $\sum_j \bar{s}_j^{(3)}$ are notably smaller than the upper bound m in respect of each data set, which strongly suggests that the helper objective $h_2(\boldsymbol{S})$ successfully guided the search process by favouring smaller networks. When comparing the promising network widths with the suggestions of the commonly adopted heuristic — more specifically, a conservative version of the commonly adopted heuristic (*i.e.* $\max\{n, o\}$) — it is apparent that a majority of the data sets require noticeably fewer neurons per hidden layer. Consider, for example, data set C37. The first two hidden layers require almost 60% fewer neurons each, while the third hidden layer requires almost 25% fewer neurons. Overall, many of the promising networks can be characterised by a network structure that is narrower than that prescribed by conventional wisdom. There are, however, data sets for which the number of hidden neurons prescribed is greater than what the heuristics recommends, *e.g.* data set C28 comprises 25% more neurons in its second hidden layer. Overall, the promising networks constructed by the BOHTA comprises, on average, between 20% and 30% fewer neurons per hidden layer.

The reasonably large standard deviations, on the other hand, indicate that a wide range of network widths (in the given range, of course) can deliver good performance in respect of these data sets — a finding that, together with the smaller network widths, further justifies the utility of a training approach that finds problem-specific network structures. Another interesting finding relates to the redundant hidden layers prescribed by the BOHTA. Consider, for example, data set C1. The best networks found by the BOHTA in respect of this data set comprise zero neurons in their third hidden layer. Interestingly, in the case of data set C30, only one non-redundant hidden layer is required by the best performing networks. Overall, the varying network depths provide further justification for the utility of a training approach that can find problem-specific network structures.

TABLE 8.12: *A summary of the network structures corresponding to the top solutions. The mean number of hidden neurons (rounded to the nearest integer) in the first, second, and third hidden layers are denoted by* $\sum_j \bar{s}_j^{(1)}$, $\sum_j \bar{s}_j^{(2)}$, *and* $\sum_j \bar{s}_j^{(3)}$, *respectively, where* $j \in \{1, \ldots, m\}$*; the respective standard deviations (rounded to the nearest integer) are also indicated. In addition, a conservative version of the commonly adopted heuristic (i.e.* $\max\{n, o\}$*) as well as the network width upper bound* m *are included.*

Data set	$\bar{F}_1$-score	$\sum_j \bar{s}_j^{(1)}$	$\sum_j \bar{s}_j^{(2)}$	$\sum_j \bar{s}_j^{(3)}$	$\max\{n, o\}$	m	Data set	$\bar{F}_1$-score	$\sum_j \bar{s}_j^{(1)}$	$\sum_j \bar{s}_j^{(2)}$	$\sum_j \bar{s}_j^{(3)}$	$\max\{n, o\}$	m
C1	0.9354	$37_{\pm 11}$	$100_{\pm 27}$	0	97	194	C26	0.9095	$8_{\pm 4}$	0	$10_{\pm 3}$	12	24
C2	1	$20_{\pm 10}$	0	$33_{\pm 8}$	30	60	C27	0.9882	$14_{\pm 3}$	$9_{\pm 3}$	$4_{\pm 2}$	12	24
C3	0.9550	$15_{\pm 7}$	$22_{\pm 10}$	$22_{\pm 7}$	25	50	C28	0.9832	0	$15_{\pm 4}$	$12_{\pm 6}$	12	24
C4	1	0	$41_{\pm 12}$	$34_{\pm 13}$	79	158	C29	0.8300	$4_{\pm 0}$	$6_{\pm 1}$	$15_{\pm 4}$	12	24
C5	1	$20_{\pm 8}$	$58_{\pm 11}$	0	52	104	C30	0.9616	0	$11_{\pm 3}$	0	12	24
C6	1	$1_{\pm 0}$	$2_{\pm 1}$	$5_{\pm 1}$	4	8	C31	1	$4_{\pm 0}$	$14_{\pm 5}$	$8_{\pm 4}$	13	26
C7	0.7303	0	$9_{\pm 3}$	$7_{\pm 1}$	9	18	C32	0.9236	$53_{\pm 19}$	$64_{\pm 27}$	$67_{\pm 24}$	61	122
C8	0.9859	$16_{\pm 8}$	$10_{\pm 5}$	$15_{\pm 5}$	30	60	C33	0.9572	$17_{\pm 4}$	$27_{\pm 6}$	$11_{\pm 2}$	27	54
C9	0.8200	$11_{\pm 2}$	$5_{\pm 1}$	0	9	18	C34	0.7688	$4_{\pm 1}$	$9_{\pm 4}$	$9_{\pm 3}$	10	20
C10	1	$13_{\pm 5}$	$13_{\pm 2}$	$8_{\pm 4}$	15	30	C35	1	$5_{\pm 1}$	0	$1_{\pm 0}$	4	8
C11	0.9653	0	$10_{\pm 5}$	$8_{\pm 4}$	16	32	C36	0.9850	0	$17_{\pm 3}$	$19_{\pm 9}$	20	40
C12	1	$17_{\pm 3}$	$25_{\pm 10}$	$14_{\pm 2}$	28	56	C37	1	$92_{\pm 35}$	$93_{\pm 25}$	$178_{\pm 75}$	234	468
C13	0.9964	$56_{\pm 19}$	0	$170_{\pm 75}$	155	310	C38	0.9960	0	$25_{\pm 6}$	$19_{\pm 9}$	41	82
C14	0.6116	$160_{\pm 52}$	$180_{\pm 39}$	$182_{\pm 64}$	152	304	C39	1	$49_{\pm 16}$	$40_{\pm 9}$	$115_{\pm 41}$	94	188
C15	0.7985	0	$9_{\pm 1}$	$7_{\pm 3}$	12	24	C40	0.9919	$10_{\pm 5}$	0	$13_{\pm 5}$	19	38
C16	0.9991	$7_{\pm 3}$	$9_{\pm 2}$	$14_{\pm 6}$	12	24	C41	0.6471	$8_{\pm 4}$	0	$4_{\pm 1}$	12	24
C17	0.9533	$9_{\pm 4}$	$14_{\pm 2}$	$4_{\pm 1}$	12	24	C42	0.9316	$39_{\pm 10}$	$16_{\pm 4}$	$35_{\pm 15}$	41	82
C18	0.9773	$11_{\pm 5}$	$8_{\pm 2}$	$15_{\pm 5}$	12	24	C43	1	$15_{\pm 5}$	$4_{\pm 0}$	$5_{\pm 2}$	12	24
C19	0.9470	$15_{\pm 3}$	$6_{\pm 1}$	$6_{\pm 1}$	12	24	C44	0.9584	0	$3_{\pm 1}$	$6_{\pm 2}$	7	14
C20	0.8491	$11_{\pm 4}$	$9_{\pm 1}$	$10_{\pm 5}$	12	24	C45	0.4945	$7_{\pm 1}$	$3_{\pm 1}$	$2_{\pm 1}$	6	12
C21	0.8808	$9_{\pm 1}$	$13_{\pm 5}$	0	12	24	C46	0.8863	$2_{\pm 1}$	$4_{\pm 0}$	$6_{\pm 1}$	5	10
C22	0.8872	$15_{\pm 7}$	$5_{\pm 2}$	$6_{\pm 2}$	12	24	C47	1	$8_{\pm 4}$	0	$14_{\pm 6}$	18	36
C23	1	$7_{\pm 2}$	$13_{\pm 5}$	$5_{\pm 1}$	12	24	C48	0.9696	$10_{\pm 5}$	$5_{\pm 1}$	$13_{\pm 2}$	13	26
C24	0.9640	$12_{\pm 2}$	0	$10_{\pm 2}$	12	24	C49	0.9120	$12_{\pm 6}$	$16_{\pm 6}$	$6_{\pm 3}$	16	32
C25	0.9471	$10_{\pm 2}$	$9_{\pm 2}$	$12_{\pm 3}$	12	24	—	—	—	—	—	—	—

The difference between the findings of the heuristic and those of the BOHTA may be attributed to the difference in modelling approaches, *i.e.* the heuristic is based on empirical findings obtained from the adoption of classical approaches during which only the network weights are trained whilst using standard activation functions such as the sigmoid and PReLU functions. The proposed modelling approach, however, introduces a piecewise linear activation that can model the neuron firing process on a notably lower level of abstraction, as discussed in §5.2.3 and §6.1.2. It is conjectured that the versatile nature of these activation functions enhances the network's capability to approximate multiple abstractions within the data set. Consequently, the network requires fewer hidden neurons and, as a result, fewer network weights in order to approximate the underlying functional representation. A more in-depth investigation is necessary to verify this supposition — another recommendation for future work.

8.6 Chapter summary

This chapter opened in §8.1 with an extensive algorithmic comparative study focussed on comparing the algorithmic performance of the BOHTA with those of its respective sub-algorithms (*i.e.* the NSGA-II, NSDE, and OMOPSO). The BOHTA exhibited both superior performance and greater general applicability in respect of the different data sets. This was followed by an investigation into the meta-generalisation capabilities of the BOHTA in §8.2 so as to evaluate the robustness of the BOHTA. Accordingly, the performance of the BOHTA was compared with its respective sub-algorithms in respect of unseen data sets (*i.e.* data sets C41–C49). The results from this evaluation indicated favourable performance improvements relative to the individual applications of the sub-algorithms.

In §8.3, the BOHTA was compared with SGD, RMSProp, and Adam, and this was followed by an investigation into the consolidation of the BOHTA with the best gradient-based training algorithm, *i.e.* Adam. In §8.3.1, it was reported that the BOHTA is outperformed comprehensively by SGD and NSDE. In §8.3.2, it was, however, reported that when the BOHTA was paired with Adam, notable performance improvements were achieved.

An exposition on the temporal dynamics exhibited by the BOHTA followed in §8.4 for the purpose of gaining deeper insight into its working and behaviour under different circumstances. Furthermore, an investigation into algorithmic performance prediction was carried out. Accordingly, the C4.5 decision tree algorithm was applied in respect of a meta data set which comprised different meta-features and sub-algorithm superiority in respect of each data set. Based on results obtained it was found that the C4.5 decision tree algorithm possessed reasonably good predictive performance. The utility of the predictive model was then incorporated into the working of the BOHTA and potentially noteworthy performance improvements were found.

Finally, an analysis of the structural attributes related to favourable networks produced by the BOHTA followed in §8.5. The results of this analysis indicated that a majority of the data sets required noticeably fewer neurons per hidden layer when comparing the promising networks produced by the BOHTA with that prescribed by conventional wisdom.

Part IV

Conclusion

CHAPTER 9

Summary and Conclusion

Contents

This closing chapter comprises two sections. In §9.1, a chapter-by-chapter overview of the research documented in this dissertation is provided. This is followed in §9.2 by an appraisal of the contributions made in this dissertation.

9.1 Dissertation contents

The introductory chapter of the dissertation, Chapter 1, opened in §1.1 with a general background in which the potential utility of training ANNs by means of a hyperheuristic optimisation approach (*i.e.* AMALGAM) was motivated. Thereafter, the problem considered in the dissertation was formally described in §1.2. The was followed in §1.3 by a delimitation of the dissertation scope to FNNs in a supervised learning paradigm, with a particular focus on classification problems. In addition, the optimisation approaches considered for inclusion in the proposed solution methodology were limited to those in the original implementation of AMALGAM. Algorithmic comparisons, on the other hand, were limited to the BOHTA, its constituent sub-algorithms, and the most popular gradient-based training algorithms. Next, the objectives pursued in the dissertation were outlined in §1.4. The first chapter closed in §1.5 with a detailed description of how the material in the remainder of the dissertation was organised into chapters and parts.

Apart from the above introductory chapter, this dissertation comprised a further nine chapters (partitioned into four distinct parts), a bibliography, and two appendices. Part I was a literature review in fulfilment of Objective I in §1.4, and consisted of three chapters. The first chapter of Part I, Chapter 2, was devoted to a review of the necessary mathematical and statistical prerequisites pertaining to the topic of this dissertation. The chapter opened in §2.1 with a discussion on important notions related to MOO, which included Pareto optimality, Pareto dominance, and Pareto rank. The FNSA and the crowding distance assignment algorithm, which formed an integral part of the BOHTA (and its constituent sub-algorithms), were also reviewed. This was followed in §2.2 by a discussion on important statistical preliminaries, which included inferential statistical testing and statistical learning algorithms. Non-parametric statistical procedures employed in the dissertation, *i.e.* the Friedman test along with the Nemenyi *post hoc* procedure, were discussed in §2.2.1. These statistical procedures facilitated the algorithmic parameter evaluation and the relative algorithmic performance comparison. Tree-based statistical learning algorithms were discussed in §2.2.2, which included a unified description of the most popular

decision tree algorithms, *i.e.* CART and C4.5, within the context of a classification prediction problem. Furthermore, prominent pruning techniques for improving predictive performance of the decision trees were discussed.

The second chapter of Part I, Chapter 3, contained a review of the pertinent literature related to ANNs. The reader was introduced to the principal concepts and terminology found in the ANN literature. This review facilitated a better understanding of the research reported in this dissertation. The chapter opened in §3.1 with a discussion on the fundamentals of ANNs. The inner working of an ANN was elucidated by means of a brief discourse on the operation of both biological and artificial neurons. The structure according to which neurons are interconnected was subsequently discussed, highlighting the uncertainty that accompanies its determination. The different activation functions that can be employed by neurons were discussed in §3.2, identifying the sigmoid and PReLU functions as the most ubiquitous in the academic literature. This was followed in §3.3 by an overview of three prominent ANN types, namely FFNs, recurrent neural networks, and convolutional neural networks. FNNs were selected to form the basis of discussion — the fundamental premise of training remains, by and large, the same amongst all of the major network types. Thereafter, the working of an FNN was elucidated in §3.4, which included a derivation of the mathematical expression of an SFNN. This was followed in §3.5 by an in-depth exposition of the important matter of network learning and training. This exposition included discussions on learning fundamentals, ML paradigms, data set types, weight initialisation and, most importantly, the derivation of a mathematical representation of supervised learning. A description of the different algorithmic performance measures was presented in §3.6, including the most popular error functions as well as the F_1-score, which accounts for class imbalance and, as a result, overcomes the accuracy paradox. This was followed in §3.7 by a detailed discussion on classical learning procedures and accompanying training algorithms. The influential backpropagation method was derived, and this was followed by an elucidation of prominent first-order training algorithms (SGD, RMSProp, and Adam). A discussion followed in §3.8 on regularisation — a method for reducing overfitting. The most popular regularisation techniques were briefly addressed. The domain of meta-learning was finally discussed in §3.9 with an emphasis on the most important generic, statistical, and information theoretic meta-features found in the literature.

The final chapter of Part I, Chapter 4, was a review of the pertinent literature related to metaheuristic and hyperheuristic optimisation techniques. Fundamental concepts and terminology in the literature were reviewed which facilitated a greater understanding of the work presented in this dissertation. The chapter opened in §4.1 with a brief discussion on the general notion of a metaheuristic approach towards solving optimisation problems. This was followed in §4.2 by a discussion on a powerful sub-field of metaheuristics known as evolutionary optimisation. The discussion included detailed descriptions of three key MOEAs, namely the NSGA-II, NSDE, and OMOPSO. These algorithms formed a fundamental part of the BOHTA proposed later in the dissertation. An overview of the relatively new and promising field of hyperheuristics followed in §4.3, with an emphasis specifically on hyperheuristics of a selective nature. AMALGAM was finally discussed in detail in §4.4, which included a description of its fundamental mechanisms (*i.e.* simultaneous multi-method search and self-adaptive offspring creation).

Part II of the dissertation consisted of a further two chapters and was focussed on a delineation of the modelling approach adopted. Chapter 5, the first of these two chapters, was devoted to a formulation of an appropriate mathematical model for concurrently training FNNs in respect of their network weights, network structure, and activation functions, for performing supervised learning, in fulfilment of Objective II. The chapter opened in §5.1 with an overview of the proposed model and was supplemented by a detailed graphical representation which enhanced the

model exposition. This was followed in §5.2 by detailed descriptions of the decision variables and model constraints. The three main classes of decision variables represented the network weights, the network structure, and its activation functions — indicative of the low level of abstraction at which optimisation transpires. Key notations and concepts were introduced and elucidated, the first of which related to the network structure. A redundancy condition was introduced which enabled the network structure to change (with respect to depth) and, along with the structural variables (responsible for changing the network width), provided the necessary means for the BOHTA to adjust the network structure dynamically when searching for favourable structures during the optimisation process. Another key concept pertained to the neuron-specific piecewise linear activation functions, which facilitated far greater freedom during the optimisation process, enabling the artificial neurons to emulate the firing process of biological neurons on a lower level of abstraction than that of the conventionally employed sigmoid and PReLU activation functions. The two objective functions considered as part of the bi-objective optimisation approach adopted in this dissertation towards maximising a novel performance measure and minimising network size were delineated in §5.3. An extensive mathematical derivation of the manner by which information signals propagate through the network (under different circumstances) accompanied this exposition. The main objective function, comprising both MAE and the F_1-score, represented the network performance measure and indicated how close the network's predictions were in respect of all the classes (dependent variables). A quantitative pilot study provided evidence of the computational efficiency afforded by the choice of performance measure in (5.13). The exposition of the main objective function included the data set partitioning convention (*i.e.* a 60%:20%:20% split) and the random sampling procedure employed in this dissertation. The secondary helper objective function, on the other hand, was employed to guide the search process by favouring smaller networks. The ultimate aim of this objective function was to address the problem of overfitting, hence its classification as a regularisation objective function. Conventional wisdom claims that smaller networks, compromising fewer parameters, are less inclined towards learning the noise in the data and therefore are more inclined to learn the true underlying functional representation. The secondary helper objective function ensured that the BOHTA avoided networks comprising too few parameters (*i.e.* excessively simple networks) as these networks exhibited underfitting and therefore performed poorly in respect of the evaluated random mini-batches. This bi-objective mathematical model is, to the best of the author's knowledge, a novel approach towards training FNNs.

The second chapter of Part II, Chapter 6, was devoted to a detailed description of the BOHTA and the test data suite employed for evaluation purposes. The chapter opened in §6.1 with a description of the most salient features of the BOHTA approach, in fulfilment of Objective III. In particular, the high-level working of the BOHTA was first described, highlighting the necessary modifications of AMALGAM. The stopping criterion employed by the BOHTA was discussed in detail, during which the notion of early stopping within the BOHTA optimisation context was elucidated. This high-level overview was followed by an exposition on the different initialisation procedures, in which novel mechanisms for initialising the network structure, network weights, and activation functions were outlined — a necessity ascribed to the novelty of the mathematical model itself. Thereafter, an appropriate solution encoding scheme was proposed and described extensively. The proposed encoding scheme facilitated the adoption of the different evolutionary operators of the sub-algorithms (in the context of the mathematical model) in fulfilment of Objective IV. The sub-algorithm evolutionary operators were modified versions of current operators found in the evolutionary computation literature. Novel modifications were made in order to facilitate their application in the given optimisation context. The test suite selected for assessing the proposed solution methodology was detailed in §6.2. The aim of the test suite was to facilitate a demonstration of the BOHTA's capabilities and draw pertinent insight into its

working, in fulfilment of Objective V. Accordingly, a diverse selection of data sets was selected and described through the lens of generic, statistical, and information theoretic meta-features which are central to the algorithmic performance prediction study conducted in this dissertation, in fulfilment of Objective VIII(c). The necessary data pre-processing steps (to which the data sets were subjected) were also described. These steps included one-hot encoding and standardisation, and were applied to each data set in fulfilment of Objective VI. An advanced random data sampling procedure was finally described which ensured that the main objective function sampled random mini-batches that were, to a reasonably large extent, an accurate reflection of the solution's true performance.

Part III of the dissertation consisted of a further two chapters and was devoted to algorithmic performance evaluation. In the first chapter of Part III, Chapter 7, three extensive algorithmic parameter evaluations were performed in fulfilment of Objective VII. An elucidation of the manner in which the training algorithms' performances were evaluated, was presented in §7.1 — a necessary precursor to the parameter evaluations. Accordingly, the final performance of a training algorithm was assessed in terms of the best performing solution in the approximate Pareto front (obtained after training), which corresponded to the solution with the largest F_1-score in respect of the independent testing set. A discussion on the experimental setup followed in §7.2 and focussed specifically on the parameters that remained fixed throughout the subsequent parameter evaluations. These parameters pertained to the network structure upper bounds (in respect of depth and width) as well as the mini-batch size. In §7.3, the findings of the three algorithmic parameter evaluations were discussed. The first parameter evaluation was presented in §7.3.1 and focussed on the BOHTA's sub-algorithms (*i.e.* the NSGA-II, NSDE, and OMOPSO). The main reasoning behind the necessity of this parameter evaluation was attributed to the novelty of both the mathematical model formulated and the solution methodology proposed — it was argued that current parameter values adopted in the literature were not suitable. The aim was therefore to find suitable algorithmic parameter values for these sub-algorithms, before their subsequent incorporation into the BOHTA. Sources in the literature were consulted in order to identify appropriate parameter value ranges. Insight into the performance impact of different parameter values (for the NSGA-II, NSDE, and OMOPSO) was gained in this novel optimisation context — a matter unaddressed in the literature. Based on the findings of this parameter evaluation, the NSGA-II proved to be the most robust sub-algorithm in terms of aggregated F_1-score sample mean, followed closely by NSDE and OMOPSO, both of which achieved similar performance. In the case of the NSGA-II, a large crossover probability delivered the best performance statistically, while in the case of NSDE, a large amplification factor clearly delivered superior performance. Lastly, in the case of OMOPSO, a large mutation probability delivered the best performance.

The second parameter evaluation was presented in §7.3.2 and focussed on identifying suitable algorithmic parameter values for the BOHTA as a whole. The findings of the sub-algorithm parameter evaluation, as well as a conceptual deliberation on different scenarios (or circumstances), were taken into account when the different BOHTA parameter values were conceived. The constructions of these conceptual scenarios allowed for valuable insight to be gained into the computational capability of the approach under different circumstances. The findings of this parameter evaluation indicated that the BOHTA, surprisingly, performed consistently regardless of the population size M, whereas variation of the lower bound proportion $\tilde{N}$ delivered more significant performance improvements. In particular, a large lower bound proportion of $\tilde{N} = 0.15$ resulted in a marked performance improvement even when the population size was small. It was therefore inferred that the BOHTA can achieve good performance even under computationally constrained circumstances. The BOHTA exhibited a convincing degree of robustness during this parameter evaluation.

The third, and final, parameter evaluation focussed on the three gradient-based training algorithms, *i.e.* SGD, RMSProp, and Adam, and was presented in §7.3.3. The imposition of differentiability required a number of amendments to the experimental setup so as to ensure a fair performance comparison with the BOHTA later in the dissertation. The findings of this parameter evaluation indicated that SGD favoured the exclusion of momentum together with a small learning rate, while in the case of RMSProp, both a small learning rate and a small exponential decay rate delivered superior performance. Lastly, in the case of Adam, both a large learning rate and a large secondary exponential decay rate delivered the best performance.

The second chapter of Part III, Chapter 8, was devoted to a detailed investigation into the implementation of the BOHTA in the context of solving the problem instances induced by the test suite. The chapter opened in §8.1 with an extensive algorithmic comparative study focussed on comparing the algorithmic performance of the BOHTA with those of its respective sub-algorithms (*i.e.* the NSGA-II, NSDE, and OMOPSO), in fulfilment of Objective VIII(a). The BOHTA exhibited both superior performance and greater general applicability in respect of the different data sets. This was followed by an investigation into the meta-generalisation capabilities of the BOHTA in §8.2 so as to evaluate the robustness of the BOHTA, in fulfilment of Objective VIII(e). Accordingly, the performance of the BOHTA was compared with its respective sub-algorithms in respect of unseen data sets (*i.e.* data sets C41–C49). The results of this evaluation indicated favourable performance improvements relative to the individual applications of the sub-algorithms. The BOHTA delivered networks of a higher-quality, whilst adapting well over the different data sets. The advantages associated with a hyperheuristic optimisation approach were therefore realised within the context of training FNNs. In §8.3, the BOHTA was compared with SGD, RMSProp, and Adam, in final fulfilment of Objective VIII(a), and this was followed by an investigation into the consolidation of the BOHTA with the best gradient-based training algorithm, *i.e.* Adam. In §8.3.1, it was reported that although the performance achieved by the BOHTA was comparable to that of RMSProp, it was outperformed comprehensively by SGD and NSDE. In §8.3.2, it was, however, reported that when the BOHTA was paired with Adam, notable performance improvements were achieved over the individual application of this gradient-based training algorithm.

An exposition on the temporal dynamics exhibited by the BOHTA followed in §8.4 for the purpose of gaining deeper insight into its working and behaviour under different circumstances, in fulfilment of Objective VIII(b). The analyses allowed for additional insight into the BOHTA's working and behaviour under different circumstances. An interesting finding pertained to notable similarities in temporal reproduction success (*i.e.* the change in reproduction success over time) between AMALGAM and the BOHTA in respect of a particular data set. Another interesting finding related to the comparison of algorithmic behaviour between the individual application of the BOHTA's sub-algorithms and their application in the context of the BOHTA. It was found that in the context of the BOHTA, the sub-algorithms performed markedly different, which provided, to some extent, an indication of the benefits associated with the cooperation of training algorithms — further vindication of a hyperheuristic optimisation approach. The insight gained from the temporal dynamics analysis was then used as part of an investigation into algorithmic performance prediction, in fulfilment of Objective VIII(c). Accordingly, the C4.5 decision tree algorithm was applied in respect of a meta-data set which comprised different meta-features and sub-algorithm superiority in respect of each data set. Based on results obtained it was found that the C4.5 decision tree algorithm possessed reasonably good predictive performance. Pertinent insight was gained from the rule formulations inherent to the generated decision tree — it was inferred that the meta-feature contributing the most towards predicting which sub-algorithm would be superior is the size of the data set. The number of classes o constituting the classification problem together with the number of numerical features n_{num} and mean class entropy $\bar{H}$ were also

identified as meta-features that contribute notably towards predicting sub-algorithm superiority. The utility of the predictive model was then incorporated into the working of the BOHTA and noteworthy performance improvements were found.

The structural attributes of favourable networks produced by the BOHTA were finally analysed in §8.5, in fulfilment of Objective VIII(d). The first noteworthy finding of this analysis was that the network widths were significantly smaller than the upper bounds for a majority of the data sets — an indication that the regularisation objective function achieved its intended goal of favouring smaller networks, reducing, as a result, the prevalence of overfitting. It was found that the favourable networks produced by the BOHTA were, in most cases, notably narrower than those recommended by a widely adopted heuristic. It was consequently argued that the versatile piecewise linear activation functions permitted networks to comprise fewer hidden neurons (therefore fewer weights) and still be capable of delivering reasonably good performance.

9.2 Appraisal of dissertation contributions

The main contributions of this dissertation are eleven-fold. This section contains a documentation and appraisal of these contributions.

Contribution 1 *The establishment of a formal mathematical model representing the training of FNNs in respect of their network weights, network structure, and activation functions concurrently.*

The proposal of a new approach towards the training of FNNs (on the specified level of abstraction) requires a formal mathematical model and an exact delineation of its inner working in an unambiguous manner so as to avoid any misinterpretation by future researchers. The model, containing all the decision variables, constraints, and objective functions, was presented in its entirety in Chapter 5. An extensive exposition of the manner according to which information signals propagate through the network under all possible circumstances was also presented in §5.3.1. This exposition was essential in deriving the main objective function in (5.13) and delineating the novel and salient components of the modelling approach — more specifically, the structural variables (in conjunction with the redundancy conditions) and the activation functional variables. This extensive, unambiguous, and generic formulation facilitates the application of other metaheuristic and hyperheuristic optimisation approaches in respect of FNN training.

Contribution 2 *The proposal and design of a novel modelling approach towards the dynamic adjustment of an FNN structure in respect of both width and depth.*

The modelling approach of the network structure in §5.2.2 enabled the BOHTA to adjust the network size dynamically during the optimisation process. The structural switching-variables in (5.4) are responsible for adjusting the network width (*i.e.* the number of active hidden neurons), whereas the redundancy condition in (5.5) is responsible for adjusting the network depth (*i.e.* the number of non-redundant hidden layers). This modelling approach facilitated the incorporation of the regularising objective function in (5.15) which guided the search to favour smaller networks so as to avoid, to an extent, overfitting. The findings in §8.5 indicated that the favourable networks produced by the BOHTA comprised significantly fewer hidden neurons than the network width upper bounds employed, which suggests the successful working of the regularising objective function. The network structural variables were only subjected to the binary-valued domain constraint — networks were therefore permitted to comprise zero active hidden neurons whilst still possessing reasonably sufficient computational capability (as expressed in (5.6)). The computational burden associated with constraint handling techniques was therefore circumvented.

The dynamic nature of the proposed modelling approach also facilitates further analyses to be carried out — given sufficient computational resources the proposed modelling approach can be employed in the context of deep ANNs comprising any arbitrary number of hidden layers with a view to solve difficult supervised (or unsupervised) learning problems in domains such as image vision and natural language processing.

Contribution 3 *The proposal and design of a novel approach towards modelling the emulation of biological neurons on a lower level of abstraction.*

The modelling approach pertaining to activation functions in §5.2.3 enabled the hidden neurons within the network to emulate the firing process of biological neurons on a lower level of abstraction. This enhanced level of emulation was facilitated by the incorporation of slope variables for both negative and non-negative input in respect of each hidden neuron in every hidden layer. Both excitatory and inhibitory relationships within the data sets presented for training purposes could therefore be modelled in a continuous fashion. These neuron-specific piecewise linear activation functions represented a markedly different approach to those typically adopted in the literature. The findings in §8.5 indicated that the BOHTA converged to networks comprising significantly fewer hidden neurons than recommended by a commonly relied upon, albeit conservative, heuristic. The versatile nature of the activation functions adopted enhanced the network's capability to approximate multiple abstractions within a data set. Consequently, favourable networks comprised fewer hidden neurons and, as a result, fewer network weights in order to approximate the underlying functional representation.

Contribution 4 *The modification and furtherance of the AMALGAM hyperheuristic within the context of training FNNs.*

The application of AMALGAM towards the training of FNNs in respect of their network weights, network structure, and activation functions has, to the best of author's knowledge, not been attempted in the literature. Several key modifications were therefore proposed and outlined in Chapter 6 so as to further the seminal research of Vrugt and Robinson [171]. Superior alternatives to some of AMALGAM's sub-algorithms were propounded and included improved versions of DE and PSO (*i.e.* NSDE and OMOPSO). These two sub-algorithms, together with the much-celebrated NSGA-II, constituted the foundational working of the BOHTA. The novelty of the mathematical model presupposed supplementary modifications to the solution methodology, the first of which related to the initialisation of random solutions (or networks). Suitable initialisation procedures in respect of the network structure, network weights, and activation functions were detailed in §6.1.2, accommodating and complementing the dynamic nature of the modelling approach. A versatile encoding scheme was also proposed in §6.1.3. The proposed solution representation format greatly facilitated the application of the sub-algorithm evolutionary operators, as well as the implementation of other EAs, in potential follow-up work. Pertinent operators found in the evolutionary computation literature were contextualised in §6.1.4. Furthermore, suitable conversion processes in respect of NSDE and OMOPSO were recommended so as to account for the binary-valued nature of the structural variables.

Contribution 5 *The establishment of an appropriate test bed in order to demonstrate the capabilities of a hyperheuristic training approach.*

Several publicly available data sets were synthesised into a comprehensive test suite in §6.2. The selection of these data sets was based on diversity criteria (*i.e.* generic, statistical, and information theoretic meta-features), thereby ensuring that a heterogeneous test bed underpinned the analyses performed in this dissertation. The test suite comprised forty-nine data sets which represents, to the best of the author's knowledge, the largest test bed subjected to any meta-

heuristic (such as the NSGA-II, NSDE, and OMOPSO) or hyperheuristic approach (such as the BOHTA) within the context of training ANNs with respect to their network weights, structure, and activation functions. A deeper insight into the performance and behaviour of the BOHTA (and its sub-algorithms) was consequently gained. The publicly available nature of these data sets encourages future work in a similar vein and facilitates possible inferences that can be drawn from the implementation of comparable hyperheuristic training approaches.

Contribution 6 *An evaluation of algorithmic performance in respect of several parameter value combinations for the NSGA-II, NSDE, OMOPSO, and the BOHTA.*

The novelty of both the mathematical model and proposed solution methodology necessitated an extensive algorithmic parameter evaluation so as to ascertain suitable BOHTA sub-algorithm parameter values. A comprehensive overview of the corresponding results was presented in §7.3.1 and provided new insight into the performance impact of the different sub-algorithm parameter combinations — valuable contextual information that could possibly be used by future researchers. The findings of the sub-algorithm parameter evaluation, along with the conceptual deliberation of different scenarios, aided in the selection of informative parameter values for the BOHTA parameter evaluation in §7.3.2. Pertinent insight was gained into the performance impact of different values for the population size M and lower bound $\tilde{N}$ — a matter that Vrugt and Robinson [171] failed to address in their original implementation of AMALGAM.

Contribution 7 *An evaluation of algorithmic performance in respect of several parameter value combinations for SGD, RMSProp, and Adam.*

An extensive parameter evaluation was conducted in §7.3.3, focussing on SGD, RMSProp, and Adam, and facilitating a fair and robust algorithmic performance comparison. Various sources in the literature were consulted so as to find appropriate parameter value ranges to consider during the parameter evaluation. Necessary and appropriate modifications were also made to the experimental setup in order to further facilitate the performance comparison. The modifications were based on standard (best) practice in the literature — the parameter evaluation therefore represents a standardised reference that could possibly be used by future researchers. The findings of the parameter evaluations also provides pertinent insight into algorithmic performance when employing different parameter values.

Contribution 8 *An evaluation of the extent to which a hyperheuristic training approach can deliver higher-quality solutions, whilst enhancing the level of general applicability, when compared with the individual application of the constituent sub-algorithms.*

A detailed comparative study, underpinned by appropriate statistical testing, was carried out in §8.1 which provided convincing evidence of the potential algorithmic performance gains that can be expected when a hyperheuristic approach, such as the BOHTA, is adopted. The traditional advantages associated with a hyperheuristic optimisation approach were evidently realised — both higher-quality solutions and a greater level of general applicability were achieved. The improved meta-generalisation capabilities of the BOHTA were also validated in §8.2, showcasing the extent of performance gains that can be expected when adopting a hyperheuristic approach (such as the BOHTA) in the context of training FNNs.

Contribution 9 *The proposal of a novel approach towards consolidating the BOHTA and a gradient-based training algorithm.*

The successful consolidation of the BOHTA with the best performing gradient-based training algorithm, *i.e.* Adam, was demonstrated in §8.3. Although a modification to the bi-objective model was first necessitated, the flexibility of the modelling approach facilitated this modifi-

cation. Furthermore, the potential computational drawback associated with this modification was mitigated, as evidenced by the performance improvements achieved during the consolidated approach over Adam. Two different approaches were proposed, namely (1) a post-optimisation approach and (2) an intermediate-optimisation approach, and both exhibited statistically superior performance. The consolidation of an optimisation approach such as the BOHTA with a gradient-based training algorithm is, to the best of the author's knowledge, novel.

Contribution 10 *An investigation into algorithmic performance prediction based on the temporal dynamics of the BOHTA.*

An in-depth analysis of the BOHTA's temporal dynamics was performed in §8.4, offering new insight into the behaviour of the EAs adopted in the context of multi-method FNN training — a matter unaddressed in the literature. One of the pertinent findings pertained to the marked differences in respect of the temporal reproduction success (*i.e.* the change in reproduction success over time) between the different data sets and between the individual application of the sub-algorithms and their application as part of the BOHTA. These findings provided a circumstantial and qualitative measure of the NFL theorem manifesting itself in the given optimisation context. The insight gained from the BOHTA's temporal dynamics was then used as part of an attempt at algorithmic performance prediction — accordingly, the supervised learning problem of predicting sub-algorithm superiority based on meta-features was investigated. The C4.5 decision tree algorithm was employed for this purpose and exhibited reasonably good predictive capability. Deeper insight into the different meta-features that contribute towards the relative success and failure of sub-algorithms was gained — the white-box nature of the decision tree algorithm facilitated the extraction of this insight. The foresight afforded by the predictive model was also successfully used to improve the working of the BOHTA — notable performance improvements were achieved. Both the insight gained from the temporal dynamics analysis and the success of the algorithmic performance prediction study are, to the best of the author's knowledge, novel contributions to the field of ANN training.

Contribution 11 *The delineation and contextualisation of early stopping within an MOO context.*

The notion of early stopping is a dynamic and effective stopping criterion employed in conventional ANN training contexts in which gradient-based training algorithms are employed. The proposed implementation of early stopping in this dissertation addresses a gap in the literature which pertains to a lack of explanatory and detailed implementations of early stopping in unconventional ANN training environments, *e.g.* meta- and hyperheuristics (such as AMALGAM and the BOHTA) within an MOO context. The elucidation of early stopping in §6.1 was sufficiently generic to facilitate a better understanding of the manner in which this approach can be modified to successfully incorporate the notions of MOO (such as Pareto dominance) into its working.

CHAPTER 10

Future Work

This final chapter contains suggestions for seven avenues of further investigation as possible follow-up work on the contributions of this dissertation. In each case, the suggestion is stated formally and then elaborated upon and motivated briefly.

Suggestion 1 *Consider further improvements to the high-level working of the BOHTA.*

The BOHTA's emulation of AMALGAM results in the inheritance of some of its shortcomings. Raad [127] reported that a notable drawback of the AMALGAM method can be attributed to its selection for replacement operator (employed by the much-celebrated NSGA-II). A recommended improvement, according to Raad *et al.* [128], is the adoption of the selection strategy used by the *strength Pareto evolutionary algorithm 2* [187], which delivered superior results in the context of water distribution systems design. Furthermore, Schlünz [145] pointed out that another drawback relates to the measure of a sub-algorithm's reproductive success in (4.11). It was argued that this measure does not explicitly take the quality of fitness improvement into account — a sub-algorithm that produces few, high-quality solutions is not rewarded adequately, compared with a sub-algorithm that produces many, low-quality solutions. The temporal dynamics of the sub-algorithms in respect of various data sets (discussed in §8.4) are symptomatic of this very phenomenon. Schlünz recommended selection strategies based on performance indicators such as the hypervolume- and R2-indicators. For example, the reward scheme can incorporate the contribution of a successful offspring solution to the hypervolume of the population. Future endeavours may therefore aim to include these potentially worthwhile improvements.

Suggestion 2 *Perform a more in-depth algorithmic parameter evaluation in respect of the NSGA-II, NSDE, and OMOPSO.*

The findings of the sub-algorithm parameter evaluation in §7.3.1 indicated a marked degree of insensitivity towards parameter value changes in the NSGA-II, NSDE, and OMOPSO. It would therefore be interesting to conduct a full factorial experimental design (as opposed to a sensitivity analysis) in order to evaluate the performance impact in respect of significantly more parameter value combinations. Furthermore, the full factorial design can include wider parameter value ranges as well as five qualitative intervals (as opposed to three, *i.e.* small, medium, and large). An anticipated trade-off, however, relates to the increased computation time that will be required for the sub-algorithm parameter evaluation.

Suggestion 3 *Investigate a more extensive parameterisation approach.*

The level of abstraction at which network optimisation transpires can be lowered even further by assigning sub-algorithm parameters to each facet of the network. That is, in the case of the NSGA-II, distinct crossover and mutation probabilities may be assigned to the respective

decision vectors of the network weights, network structure, and activation functions, as described in §6.1.3. Similarly, in the case of NSDE and OMOPSO, the crossover rate, amplification factor, inertia weight, learning factors, and mutation probability may be assigned individually to each facet. It is expected that the performance of the sub-algorithms may be fine-tuned even further in this manner, although at the cost of a significantly higher computation time, during the sub-algorithm parameter evaluations.

Suggestion 4 *Perform a more in-depth analysis of the structural attributes of the networks produced by the BOHTA and its constituent sub-algorithms.*

The structural analysis performed in §8.5 can be extended in various respects, the first of which pertains to the comparison of favourable networks produced by the BOHTA to a more extensive collection of heuristic techniques, which stands in contrast to the approach adopted in this dissertation where the commonly relied upon heuristic by Heaton [60] formed the basis of comparison. Another natural extension relates to the comparison of favourable network structures to unfavourable network structures, providing insight into the unfavourable characteristics exhibited by poorly performing networks. In addition, the insight thus gained may provide decision support to future researchers with respect to network structures that ought to be avoided in respect of the different data sets. Finally, a detailed comparative study may be conducted into favourable network structures produced by the BOHTA and the individual sub-algorithms. Valuable insight may thus be gained into the preference for certain network structures in respect of the different training algorithms. The notion of algorithmic performance prediction can also be extended to structural attributes — a decision tree algorithm (or any other white-box statistical learning algorithm) can be applied to the supervised learning task of predicting whether a certain network structure will result in good algorithmic performance. The attributes of good network structures can therefore be learnt.

Suggestion 5 *Enlarge the scope of the ANN type considered.*

The scope of ANN types trained by the BOHTA considered in this dissertation was limited to the prominent network type of FNNs. It is suggested that this scope delimitation be relaxed in order to consider other network types, such as convolutional neural networks, recurrent neural networks, and *transformers* [168]. The fundamental premise of training remains, by and large, the same, but the size and nature of the data sets can differ notably when compared with the problems traditionally solved by FNNs. It should therefore be interesting to adopt the BOHTA approach within these contexts and subsequently analyse both the algorithmic performance and behaviour in respect of different data sets related to image classification and natural language processing.

Suggestion 6 *Investigate other statistical and machine learning algorithms for predicting algorithmic performance.*

The predictive performance achieved by the decision tree algorithm was reasonably good, *i.e.* training and testing F_1-score performances of 0.820 and 0.770, respectively (discussed in §8.4). It was inferred that the sub-optimal predictive capability contributed to the inferior (or not statistically superior) performance achieved by the enhanced BOHTA in respect of a few data sets. Consequently, an investigation into other statistical or machine learning algorithms (both white-box and black-box) may therefore lead to more improved performance. Algorithms such as the *random forest* algorithm [13], *support vector machines* [23], and even ANNs may be considered for this purpose.

Suggestion 7 *Investigate different BOHTA sub-algorithm configurations.*

The BOHTA employed three sub-algorithms, *i.e.* the NSGA-II, NSDE, and OMOPSO, throughout the various analyses performed in this dissertation. It should be interesting to investigate the additional inclusion of other population-based metaheuristics, *e.g. evolution strategies* [161], *evolutionary programming* [47], *genetic programming* [84], and *ant colony optimisation* [35], to name but a few. Furthermore, single solution-based metaheuristics, *e.g.* the method of *simulated annealing* [80], *tabu search* [51], and the *variable neighbourhood search* method [108], to name but a few, may be considered as part of either an intermediate-optimisation approach or a post-optimisation approach. The number of sub-algorithms can also vary from one optimisation run to another — the BOHTA does not always have to use the same k sub-algorithms. The flexibility of the BOHTA allows for various configurations with which to experiment.

References

[1] Alexandros K & Melanie H, 2001, *Model selection via meta-learning: A comparative study*, International Journal on Artificial Intelligence Tools, **10(4)**, pp. 525–554.

[2] Armstrong JS & Collopy F, 1992, *Error measures for generalizing about forecasting methods: Empirical comparisons*, International Journal of Forecasting, **8(1)**, pp. 69–80.

[3] Bebis G & Georgiopoulos M, 1994, *Feed-forward neural networks*, IEEE Potentials, **13(4)**, pp. 27–31.

[4] Bengio Y, 2012, *Practical recommendations for gradient-based training of deep architectures*, pp. 437–478 in Montavon G, Orr GB & Müller KR (Eds), *Neural networks: Tricks of the trade*, Springer, Berlin.

[5] Berger JO, 1985, *Statistical decision theory and bayesian analysis*, 2nd Edition, Springer, New York (NY).

[6] Bishop CM, 1995, *Neural networks for pattern recognition*, Clarendon Press, Oxford.

[7] Bishop CM, 2006, *Pattern recognition and machine learning*, Springer, New York (NY).

[8] Bottou L, 2012, *Stochastic gradient descent tricks*, pp. 421–436 in Montavon G, Orr GB & Müller KR (Eds), *Neural networks: Tricks of the trade*, Springer, Berlin.

[9] Bottou L, Curtis FE & Nocedal J, 2018, *Optimization methods for large-scale machine learning*, SIAM Review, **60(2)**, pp. 223–311.

[10] Boussaïd I, Lepagnot J & Siarry P, 2013, *A survey on optimization metaheuristics*, Information Sciences, **237**, pp. 82–117.

[11] Branke J, Deb K, Miettinen K & Slowiński R, 2008, *Multiobjective optimization: Interactive and evolutionary approaches*, Springer, New York (NY).

[12] Breiman L, Friedman JH, Olshen RA & Stone CJ, 1984, *Classification and regression trees*, Wadsworth International Group, Belmont (CA).

[13] Breiman L, 2001, *Random forests*, Machine Learning, **45(1)**, pp. 5–32.

[14] Brownlee J, 2014, *Classification accuracy is not enough: More performance measures you can use*, [Online], [Cited March 2018], Available from `https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/`.

[15] Brownlee J, 2017, *What is the difference between test and validation datasets?*, [Online], [Cited October 2017], Available from `https://machinelearningmastery.com/difference-test-validation-datasets/`.

[16] Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Ozcan E & Qu R, 2013, *Hyper-heuristics: A survey of the state of the art*, Journal of the Operational Research Society, **64(12)**, pp. 1695–1724.

[17] Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E & Woodward JR, 2010, *A classification of hyper-heuristic approaches*, 2nd Edition, Springer, New York (NY).

[18] Castiello C, Castellano G & Fanelli AM, 2005, *Meta-data: Characterization of input features for meta-learning*, Proceedings of the International Conference on Modeling Decisions for Artificial Intelligence, Tsukuba, pp. 457–468.

[19] Chandrasekar R, 2014, *Elementary? Question answering, IBM's Watson, and the Jeopardy! challenge*, Resonance, **19(3)**, pp. 222–241.

[20] Chollet F, 2015, *Keras*, [Online], [Cited February 2019], Available from `https://keras.io`.

[21] Coello Coello CA, Lamont GB & Van Veldhuizen DA, 2007, *Evolutionary algorithms for solving multi-objective problems*, Springer, New York (NY).

[22] Copeland M, 2016, *What's the difference between artificial intelligence, machine learning, and deep learning?*, [Online], [Cited March 2017], Available from `https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/`.

[23] Cortes C & Vapnik V, 1995, *Support-vector networks*, Machine Learning, **20(3)**, pp. 273–297.

[24] Dahl GE, Yu D, Deng L & Acero A, 2012, *Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition*, IEEE Transactions on Audio, Speech, and Language Processing, **20(1)**, pp. 30–42.

[25] Darwin C, 1859, *On the origins of species by means of natural selection*, John Murray, London.

[26] Das S & Suganthan PN, 2011, *Differential evolution: A survey of the state-of-the-art*, IEEE Transactions on Evolutionary Computation, **15(1)**, pp. 4–31.

[27] Dawson RJM, 1995, *The "unusual episode" data revisited*, Journal of Statistics Education, **3(3)**, pp. 1–9.

[28] Deb K, Pratap A, Agarwal S & Meyarivan T, 2002, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, IEEE Transactions on Evolutionary Computation, **6(2)**, pp. 182–197.

[29] DeCarlo LT, 1997, *On the meaning and use of kurtosis*, Psychological Methods, **2(3)**, pp. 292–307.

[30] Demšar J, Curk T, Erjavec A, Gorup Č, Hočevar T, Milutinovič M, Možina M, Polajnar M, Toplak M, Starič A, Štajdohar M, Umek L, Žagar L, Žbontar J, Žitnik M & Zupan B, 2013, *Orange: Data mining toolbox in Python*, Journal of Machine Learning Research, **14**, pp. 2349–2353.

[31] Derrac J, García S, Molina D & Herrera F, 2011, *A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms*, Swarm and Evolutionary Computation, **1(1)**, pp. 3–18.

[32] Devabhaktuni VK, Yagoub MC & Zhang QJ, 2001, *A robust algorithm for automatic development of neural-network models for microwave applications*, IEEE Transactions on Microwave Theory and Techniques, **49(12)**, pp. 2282–2291.

[33] Dheeru D & Karra Taniskidou E, 2010, *UCI machine learning repository*, [Online], [Cited March 2017], Available from `http://archive.ics.uci.edu/ml`.

[34] Dozat T, 2016, *Incorporating Nesterov momentum into Adam*, Proceedings of the 6th International Conference on Learning Representations, San Juan, pp. 1–4.

[35] Drigo M, 1996, *The ant system: Optimization by a colony of cooperating agents*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), **26(1)**, pp. 1–13.

[36] Duchi J, Hazan E & Singer Y, 2011, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research, **12**, pp. 2121–2159.

[37] Durillo JJ, García-Nieto J, Nebro AJ, Coello Coello CA, Luna F & Alba E, 2009, *Multi-objective particle swarm optimizers: An experimental comparison*, Proceedings of the International Conference on Evolutionary Multi-criterion Optimization, Berlin, pp. 495–509.

[38] Durillo JJ, Nebro AJ, Luna F, Coello Coello CA & Alba E, 2010, *Convergence speed in multi-objective metaheuristics: Efficiency criteria and empirical study*, International Journal for Numerical Methods in Engineering, **84(11)**, pp. 1344–1375.

[39] Eberhart R & Kennedy J, 1995, *A new optimizer using particle swarm theory*, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, pp. 39–43.

[40] Engelbrecht AP & Pampara G, 2007, *Binary differential evolution strategies*, Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore, pp. 1942–1947.

[41] Fausett LV, 1994, *Fundamentals of neural networks: Architectures, algorithms and applications*, 1st Edition, Prentice-Hall, Englewood Cliffs (NJ).

[42] Feo TA & Resende MG, 1989, *A probabilistic heuristic for a computationally difficult set covering problem*, Operations Research Letters, **8(2)**, pp. 67–71.

[43] Filchenkov A & Pendryak A, 2015, *Datasets meta-feature description for recommending feature selection algorithm*, Proceedings of the 2015 Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT), St Petersburg, pp. 11–18.

[44] Fletcher R, 2013, *Practical methods of optimization*, 2nd Edition, John Wiley & Sons, New York (NY).

[45] Fletcher R & Reeves CM, 1964, *Function minimization by conjugate gradients*, The Computer Journal, **7(2)**, pp. 149–154.

[46] Flood G, 2016, *Deep learning with a directed acyclic graph structure for segmentation and classification of prostate cancer*, PhD dissertation, Lund University, Lund.

[47] Fogel LJ, Owens AJ & Walsh MJ, 1966, *Artificial intelligence through simulated evolution*, John Wiley & Sons, New York (NY).

[48] Gkanogiannis A & Kalamboukis T, 2008, *A novel supervised learning algorithm and its use for spam detection in social bookmarking systems*, Proceedings of the 2008 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD), Antwerp, pp. 13–21.

[49] Glorot X & Bengio Y, 2010, *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, Sardinia, pp. 249–256.

[50] Glorot X, Bordes A & Bengio Y, 2011, *Deep sparse rectifier neural networks*, Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale (FL), pp. 315–323.

[51] Glover F, 1986, *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, **13(5)**, pp. 533–549.

[52] Glover FW & Kochenberger GA, 2006, *Handbook of metaheuristics*, Springer, New York (NY).

[53] Goodfellow I, Bengio Y & Courville A, 2016, *Deep learning*, MIT Press, Cambridge (MA).

[54] Gudise VG & Venayagamoorthy GK, 2003, *Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks*, Proceedings of the 2003 IEEE Swarm Intelligence Symposium, Indianapolis (IN), pp. 110–117.

[55] Haario H, Saksman E & Tamminen J, 2001, *An adaptive Metropolis algorithm*, Bernoulli Society for Mathematical Statistics and Probability, **7(2)**, pp. 223–242.

[56] Hagan MT & Menhaj MB, 1994, *Training feedforward networks with the Marquardt algorithm*, IEEE Transactions on Neural Networks, **5(6)**, pp. 989–993.

[57] Hasanipanah M, Noorian-Bidgoli M, Armaghani DJ & Khamesi H, 2016, *Feasibility of PSO-ANN model for predicting surface settlement caused by tunneling*, Engineering with Computers, **32(4)**, pp. 705–715.

[58] Haykin S, 1998, *Neural networks: A comprehensive foundation*, 2nd Edition, Prentice Hall PTR, Upper Saddle River (NJ).

[59] He K, Zhang X, Ren S & Sun J, 2015, *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*, Proceedings of the 2015 IEEE International Conference on Computer Vision, Santiago, pp. 1026–1034.

[60] Heaton J, 2008, *Introduction to neural networks with Java*, 2nd Edition, Heaton Research, Incorporated, New York (NY).

[61] Hochberg Y & Tamhane AC, 1987, *Multiple comparison procedures*, John Wiley & Sons, New York (NY).

[62] Holland J & Goldberg D, 1989, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Reading (MA).

[63] Holland JH, 1975, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, University of Michigan Press, Ann Arbor (MI).

[64] Hollander M, Wolfe DA & Chicken E, 2013, *Nonparametric statistical methods*, John Wiley & Sons, New York (NY).

[65] Hong WC, 2008, *Rainfall forecasting by technological machine learning models*, Applied Mathematics and Computation, **200(1)**, pp. 41–57.

[66] Hornik K, Stinchcombe M & White H, 1990, *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks*, Neural Networks, **3(5)**, pp. 551–560.

[67] Hüsken M, Jin Y & Sendhoff B, 2005, *Structure optimization of neural networks for evolutionary design optimization*, Soft Computing — A Fusion of Foundations, Methodologies and Applications, **9(1)**, pp. 21–28.

[68] Hutter F, Babic D, Hoos HH & Hu AJ, 2007, *Boosting verification by automatic tuning of decision procedures*, Proceedings of the 7th International Conference on Formal Methods in Computer Aided Design, Austin (TX), pp. 27–34.

[69] Ilonen J, Kamarainen JK & Lampinen J, 2003, *Differential evolution training algorithm for feed-forward neural networks*, Neural Processing Letters, **17(1)**, pp. 93–105.

[70] Iorio AW & Li X, 2004, *Solving rotated multi-objective optimization problems using differential evolution*, Proceedings of the Australasian Joint Conference on Artificial Intelligence, Cairns, pp. 861–872.

[71] James G, Witten D, Hastie T & Tibshirani R, 2013, *An introduction to statistical learning*, 6th Edition, Springer, New York (NY).

[72] Jinli M & Zhiyi S, 2000, *Application of combined neural networks in nonlinear function approximation*, Proceedings of the 3rd World Congress on Intelligent Control and Automation, St Louis (MO), pp. 839–841.

[73] Joachims T, 1998, *Text categorization with support vector machines: Learning with many relevant features*, Proceedings of the European Conference on Machine Learning, Berlin, pp. 137–142.

[74] Joanes D & Gill C, 1998, *Comparing measures of sample skewness and kurtosis*, Journal of the Royal Statistical Society: Series D (The Statistician), **47(1)**, pp. 183–189.

[75] Karpathy A, 2017, *CS231n convolutional neural networks for visual recognition*, [Online], [Cited April 2017], Available from `http://cs231n.github.io/neural-networks-1/`.

[76] Kennedy J & Eberhart RC, 1997, *A discrete binary version of the particle swarm algorithm*, Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Orlando (FL), pp. 4104–4108.

[77] Keskar NS, Mudigere D, Nocedal J, Smelyanskiy M & Tang PTP, 2017, *On large-batch training for deep learning: Generalization gap and sharp minima*, Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, pp. 1–16.

[78] Khan K & Sahai A, 2012, *A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context*, International Journal of Intelligent Systems and Applications, **4(7)**, pp. 23–29.

[79] Kingma DP & Ba J, 2015, *Adam: A method for stochastic optimization*, Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), San Diego (CA), pp. 256–273.

[80] Kirkpatrick S, Gelatt CD & Vecchi MP, 1983, *Optimization by simulated annealing*, Science Magazine, **220(4598)**, pp. 671–680.

[81] Kitano H, 1990, *Empirical studies on the speed of convergence of neural network training using genetic algorithms*, Proceedings of the 8th National Conference on Artificial Intelligence, Cambridge (MA), pp. 789–795.

[82] Knowles JD, Thiele L & Zitzler E, 2006, *The performance assessment of stochastic multiobjective optimizers*, ETH Zurich, Zurich.

[83] Kotthoff L, Gent IP & Miguel I, 2011, *A preliminary evaluation of machine learning in algorithm selection for search problems*, Proceedings of the 4th Annual Symposium on Combinatorial Search, Barcelona, pp. 84–91.

[84] Koza JR, 1994, *Genetic programming as a means for programming computers by natural selection*, Statistics and Computing, **4(2)**, pp. 87–112.

[85] Krose B & Van der Smagt P, 1996, *An introduction to neural networks*, 8th Edition, The University of Amsterdam, Amsterdam.

[86] Krzywinski M & Altman N, 2014, *Points of significance: Visualizing samples with box plots*, Nature Methods, **11(2)**, pp. 119–120.

[87] Kuhn M & Johnson K, 2013, *Applied predictive modeling*, Springer, New York (NY).

[88] Kwok TY & Yeung DY, 1997, *Constructive algorithms for structure learning in feed-forward neural networks for regression problems*, IEEE Transactions on Neural Networks, **8(3)**, pp. 630–645.

[89] Laumanns M, Thiele L, Deb K & Zitzler E, 2002, *Combining convergence and diversity in evolutionary multiobjective optimization*, Evolutionary Computation, **10(3)**, pp. 263–282.

[90] Lawrence S, Tsoi AC & Back AD, 1996, *Function approximation with neural networks and local methods: Bias, variance and smoothness*, Proceedings of the 7th Australian Conference on Neural Networks, Canberra, pp. 16–21.

[91] LeCun Y, Bottou L, Orr GB & Müller KR, 2012, *Efficient backprop*, pp. 9–48 in Montavon G, Orr GB & Müller KR (Eds), *Neural networks: Tricks of the trade*, Springer, Berlin.

[92] LeCun Y, Kavukcuoglu K & Farabet C, 2010, *Convolutional networks and applications in vision*, Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS), Paris, pp. 253–256.

[93] LeCun Y, Touresky D, Hinton G & Sejnowski T, 1988, *A theoretical framework for back-propagation*, Proceedings of the 1988 Connectionist Models Summer School, Pittsburgh (PA), pp. 21–28.

[94] Liu DC & Nocedal J, 1989, *On the limited memory BFGS method for large scale optimization*, Mathematical Programming, **45(1-3)**, pp. 503–528.

[95] Loh W, 2011, *Classification and regression trees*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, **1(1)**, pp. 14–23.

[96] Luengo J, Garcia S & Herrera F, 2009, *A study on the use of statistical tests for experimentation with neural networks: Analysis of parametric test conditions and non-parametric tests*, Expert Systems with Applications, **36(4)**, pp. 7798–7808.

[97] Marquardt DW, 1963, *An algorithm for least-squares estimation of nonlinear parameters*, Journal of the Society for Industrial and Applied Mathematics, **11(2)**, pp. 431–441.

[98] Masters T & Land W, 1997, *A new training algorithm for the general regression neural network*, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Orlando (FL), pp. 1990–1994.

[99] Mastin L, 2010, *Neurons & synapses*, [Online], [Cited March 2017], Available from `http://www.human-memory.net/brain_neurons.html`.

[100] MathWorks, 2017, *Choose a multilayer neural network training function*, [Online], [Cited May 2017], Available from `https://www.mathworks.com/help/nnet/ug/choose-a-multilayer-neural-network-training-function.html`.

[101] Mehrotra K, Mohan C & Ranka S, 1997, *Elements of artificial neural networks*, MIT Press, Cambridge (MA).

[102] Mendel J & McLaren R, 1970, *8 Reinforcement-learning control and pattern recognition systems*, Mathematics in Science and Engineering, **66**, pp. 287–318.

[103] Mezura-Montes E, Reyes-Sierra M & Coello Coello CA, 2008, *Multi-objective optimization using differential evolution: A survey of the state-of-the-art*, Springer, New York (NY).

[104] Michie D, 1994, *Machine learning, neural and statistical classification*, Technometrics, Alexandria (VA).

[105] Miettinen K, 1999, *Nonlinear multiobjective optimization*, Springer Science & Business Media, Berlin.

[106] Miller GF, Todd PM & Hegde SU, 1989, *Designing neural networks using genetic algorithms*, Proceedings of the International Conference on Genetic Algorithms, East Lansing (MI), pp. 379–384.

[107] Minsky M & Papert S, 1969, *Perceptrons*, MIT Press, Cambridge (MA).

[108] Mladenovic N, 1995, *A variable neighborhood algorithm — A new metaheuristic for combinatorial optimization*, Proceedings of the 1995 Optimization Days, Montrèal, pp. 433–458.

[109] Møller MF, 1993, *A scaled conjugate gradient algorithm for fast supervised learning*, Neural Networks, **6(4)**, pp. 525–533.

[110] Montana DJ & Davis L, 1989, *Training feedforward neural networks using genetic algorithms*, Proceedings of the 11$^{\text{th}}$ International Joint Conference on Artificial Intelligence (IJCAI), Detroit (MI), pp. 762–767.

[111] Munoz A, 2014, *Machine learning and optimization*, [Online], [Cited February 2018], Available from `https://cims.nyu.edu/~munoz/files/ml_optimization.pdf`.

[112] Musliu N & Schwengerer M, 2013, *Algorithm selection for the graph coloring problem*, Proceedings of the International Conference on Learning and Intelligent Optimization, Catania, pp. 389–403.

[113] Narula SC & Wellington JF, 1982, *The minimum sum of absolute errors regression: A state of the art survey*, International Statistical Review, **50**, pp. 317–326.

[114] Nesterov Y, 1983, *A method of solving a convex programming problem with convergence rate* $\mathcal{O}(1/k^2)$, Soviet Mathematics Doklady, **27(2)**, pp. 372–376.

[115] Nesterov Y, 2013, *Introductory lectures on convex optimization: A basic course*, Springer, New York (NY).

[116] Nielsen M, 2017, *Neural networks and deep learning*, [Online], [Cited April 2017], Available from `http://neuralnetworksanddeeplearning.com/`.

[117] Oke S, 2008, *A literature review on artificial intelligence*, International Journal of Information and Management Sciences, **19(4)**, pp. 535–570.

[118] Osman IH & Kelly JP, 1996, *Meta-heuristics: An overview*, Springer, New York (NY).

[119] Osman IH & Laporte G, 1996, *Metaheuristics: A bibliography*, Annals of Operational Research, **63**, pp. 513–628.

[120] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M & Duchesnay E, 2011, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, **12**, pp. 2825–2830.

[121] Peng C, Chiappini F, Kaščáková S, Danulot M, Sandt C, Samuel D, Dumas P, Guettier C & Le Naour F, 2015, *Vibrational signatures to discriminate liver steatosis grades*, Analyst, **140(4)**, pp. 1107–1118.

[122] Petrowski JDA & Taillard PSE, 2006, *Metaheuristics for hard optimization*, Springer, New York (NY).

[123] Pfanzagl J, 1994, *Parametric statistical theory*, Walter de Gruyter, Berlin.

[124] Portilla JM, 2017, *Python for data science and machine learning bootcamp*, [Online], [Cited January 2020], Available from `udemy.com/python-for-data-science-and-machine-learning-bootcamp`.

[125] Prechelt L, 1998, *Early stopping — but when?*, pp. 55–69 in Montavon G, Orr GB & Müller KR (Eds), *Neural networks: Tricks of the trade*, Springer, Berlin.

[126] Quinlan JR, 1984, *C4.5: Programs for machine learning*, Morgan Kaufmann Publishers, San Mateo (CA).

[127] Raad DN, 2011, *Multi-objective optimisation of water distribution systems design using metaheuristics*, PhD dissertation, Stellenbosch University, Stellenbosch.

[128] Raad D, Sinske A & Van Vuuren JH, 2011, *Water distribution systems design optimisation using metaheuristics and hyperheuristics*, ORiON, **27(1)**, pp. 17–43.

[129] Reed R, 1993, *Pruning algorithms — A survey*, IEEE Transactions on Neural Networks, **4(5)**, pp. 740–747.

[130] Ribiere EPG, 1969, *Notes on convergence of directions conjugated*, Francaise Informat Recherche Operatinelle, **16**, pp. 35–43.

[131] Riedmiller M, 1994, *Advanced supervised learning in multi-layer perceptrons from backpropagation to adaptive learning algorithms*, Computer Standards and Interfaces, **16(3)**, pp. 265–278.

[132] Ripley BD, 2007, *Pattern recognition and neural networks*, Cambridge University Press, New York (NY).

[133] Rodríguez-Fdez I, Canosa A, Mucientes M & Bugarín A, 2015, *STAC: A web platform for the comparison of algorithms using statistical tests*, Proceedings of the 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Istanbul, pp. 1–8.

[134] Rokach L & Maimon O, 2005, *Decision trees*, pp. 165–192 in Rokach L & Maimon O (Eds), *Data mining and knowledge discovery handbook*, Springer, Boston (MA).

[135] Rosasco L, De Vito E, Caponnetto A, Piana M & Verri A, 2004, *Are loss functions all the same?*, Neural Computation, **16(5)**, pp. 1063–1076.

[136] Rosenblatt F, 2000, *A probabilistic model for information storage and organization in the brain*, Artificial Intelligence: Critical Concepts, **2(6)**, pp. 398.

[137] Ross P, 2005, *Search methodologies: Introductory tutorials in optimization and decision support*, Springer, New York (NY).

[138] Roth S, Gepperth A & Igel C, 2006, *Multi-objective neural network optimization for visual object detection*, Springer, New York (NY).

[139] Rouse M, 2016, *AI (Artificial Intelligence)*, [Online], [Cited April 2017], Available from `http://searchcio.techtarget.com/definition/AI`.

[140] Ruder S, 2016, *An overview of gradient descent optimization algorithms*, [Online], [Cited May 2018], Available from `https://http://ruder.io/optimizing-gradient-descent/`.

[141] Rumelhart DE, McClelland JL & Group PR, 1987, *Parallel distributed processing*, MIT Press, Cambridge (MA).

[142] Russell S & Norvig P, 2009, *Artificial intelligence: A modern approach*, 3rd Edition, Pearson Education, London.

[143] Salerno J, 1997, *Using the particle swarm optimization technique to train a recurrent neural model*, Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence, Newport Beach (CA), pp. 45–49.

[144] SAS, 2010, *Machine learning: What it is & why it matters*, [Online], [Cited April 2017], Available from `https://www.sas.com/en_za/insights/analytics/machine-learning.html`.

[145] Schlünz EB, 2016, *Multiobjective in-core fuel management optimisation for nuclear research reactors*, PhD dissertation, Stellenbosch University, Stellenbosch.

[146] Schmidhuber J, 2015, *Deep learning in neural networks: An overview*, Neural Networks, **61**, pp. 85–117.

[147] Sebastiani F, 2002, *Machine learning in automated text categorization*, ACM Computing Surveys, **34(1)**, pp. 1–47.

[148] Shah H, 2011, *Turing's misunderstood imitation game and IBM's Watson success*, Proceedings of the 2nd Symposium in the Artificial Intelligence and Simulation of Behaviour (AISB) Conference, York, pp. 1–5.

[149] Shi Y & Eberhart RC, 1999, *Empirical study of particle swarm optimization*, Proceedings of the IEEE Congress on Evolutionary computation, Washington (DC), pp. 1945–1950.

[150] Sierra MR & Coello Coello CA, 2005, *Improving PSO-based multi-objective optimization using crowding, mutation and dominance*, Proceedings of the 2005 International Conference on Evolutionary Multi-Criterion Optimization, Guanajuato, pp. 505–519.

[151] Simon D, 2013, *Evolutionary optimization algorithms*, John Wiley & Sons, New York (NY).

[152] Smith-Miles K, Baatar D, Wreford B & Lewis R, 2014, *Towards objective measures of algorithm performance across instance space*, Computers and Operations Research, **45**, pp. 12–24.

[153] Smith BR, 1997, *Neural network enhancement of closed-loop controllers for ill-modeled systems with unknown non-linearities*, PhD dissertation, Virginia Polytechnic Institute and State University, Blacksburg (VA).

[154] Sörensen K & Glover F, 2013, *Encyclopedia of operations research and management science*, 3rd Edition, Springer, New York (NY).

[155] Srivastava N, Hinton G, Krizhevsky A, Sutskever I & Salakhutdinov R, 2014, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research, **15(1)**, pp. 1929–1958.

[156] Storn R, 1996, *On the usage of differential evolution for function optimization*, Proceedings of the 1996 Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS), Berkeley (CA), pp. 519–523.

[157] Storn R & Price K, 1997, *Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces*, Journal of Global Optimization, **11(4)**, pp. 341–359.

[158] Stützle T, 1998, *Local search algorithms for combinatorial problems*, PhD dissertation, Darmstadt University of Technology, Darmstadt.

[159] Sutskever I, Martens J, Dahl G & Hinton G, 2013, *On the importance of initialization and momentum in deep learning*, Proceedings of the 30th International Conference on Machine Learning, Atlanta (GA), pp. 1139–1147.

[160] Sutton RS & Barto AG, 1998, *Reinforcement learning: An introduction*, MIT Press, Cambridge (MA).

[161] Talbi EG, 2009, *Metaheuristics: From design to implementation*, John Wiley & Sons, New York (NY).

[162] Tamura S & Tateishi M, 1997, *Capabilities of a four-layered feedforward neural network: Four layers versus three*, IEEE Transactions on Neural Networks, **8(2)**, pp. 251–255.

[163] Thompson C, 2011, *What is IBM's Watson*, [Online], [Cited June 2017], Available from `http://www.nytimes.com/2010/06/20/magazine/20Computer-t.html`.

[164] Tieleman T & Hinton G, 2012, *COURSERA: Neural networks for machine learning*, [Online], [Cited February 2019], Available from `http://archive.ics.uci.edu/ml`.

[165] Trafalis TB & Ince H, 2000, *Support vector machine for regression and applications to financial forecasting*, Proceedings of the 2000 IEEE-INNS-ENNS International Joint Conference on Neural Networks, Como, pp. 348–353.

[166] Tsai JT, Chou JH & Liu TK, 2006, *Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm*, IEEE Transactions on Neural Networks, **17(1)**, pp. 69–80.

[167] Valverde-Albacete FJ & Peláez-Moreno C, 2014, *100% Classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox*, PloS One, **9(1)**, pp. 1–10.

[168] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L & Polosukhin I, 2017, *Attention is all you need*, Proceedings of the 30th Annual Conference on Advances in Neural Information Processing Systems, Long Beach (CA), pp. 5998–6008.

[169] Vilalta R & Drissi Y, 2002, *A perspective view and survey of meta-learning*, Artificial Intelligence Review, **18(2)**, pp. 77–95.

[170] Voudouris C, 1997, *Guided local search for combinatorial optimisation problems*, PhD dissertation, University of Essex, Colchester.

[171] Vrugt JA & Robinson BA, 2007, *Improved evolutionary optimization from genetically adaptive multimethod search*, National Academy of Sciences, **104(3)**, pp. 708–711.

[172] Wang FY, Zhang JJ, Zheng X, Wang X, Yuan Y, Dai X, Zhang J & Yang L, 2016, *Where does AlphaGo go: From Church-Turing thesis to AlphaGo thesis and beyond*, IEEE/CAA Journal of Automatica Sinica, **3(2)**, pp. 113–120.

[173] Wang J, Li L, Niu D & Tan Z, 2012, *An annual load forecasting model based on support vector regression with differential evolution algorithm*, Applied Energy, **94**, pp. 65–70.

[174] Whiteley D, 1988, *Applying genetic algorithms to neural network problems*, Neural Networks: Design Techniques and Tools, **1**, pp. 230–244.

[175] Whitley D, 2014, *Blind no more: Constant time non-random improving moves and exponentially powerful recombination*, Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, Vancouver, pp. 559–580.

[176] Wilson AC, Roelofs R, Stern M, Srebro N & Recht B, 2017, *The marginal value of adaptive gradient methods in machine learning*, Proceedings of the 30th Annual Conference on Advances in Neural Information Processing Systems, Long Beach (CA), pp. 4151–4161.

[177] Winograd S & Cowan JD, 1963, *Reliable computation in the presence of noise*, 1st Edition, MIT Press, Cambridge (MA).

[178] Wolpert DH & Macready WG, 1997, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation, **1(1)**, pp. 67–82.

[179] Yao X & Liu Y, 1997, *A new evolutionary system for evolving artificial neural networks*, IEEE Transactions on Neural Networks, **8(3)**, pp. 694–713.

[180] Zabashta A, Smetannikov I & Filchenkov A, 2015, *Study on meta-learning approach application in rank aggregation algorithm selection*, Proceedings of the 2015 International Workshop on Meta-Learning and Algorithm Selection, Porto, pp. 115–116.

[181] Zaremba W, Sutskever I & Vinyals O, 2015, *Recurrent neural network regularization*, Proceedings of the 3rd International Conference on Learning Representations, San Diego (CA), pp. 1–8.

[182] Zelinka I & Lampinen J, 1999, *Evolutionary learning algorithms for neural networks*, Proceedings of the 5th International Conference on Soft Computing, Honolulu (HI), pp. 410–414.

[183] Zhang C, Shao H & Li Y, 2000, *Particle swarm optimisation for evolving artificial neural network*, Proceedings of the 2000 IEEE International Conference on Systems, Management and Cybernetics, Nashville (TN), pp. 2487–2490.

[184] Zhang QJ, Gupta KC & Devabhaktuni VK, 2003, *Artificial neural networks for RF and microwave design — From theory to practice*, IEEE Transactions on Microwave Theory and Techniques, **51(4)**, pp. 1339–1350.

[185] Zhou A, Qu BY, Li H, Zhao SZ, Suganthan PN & Zhang Q, 2011, *Multiobjective evolutionary algorithms: A survey of the state of the art*, Swarm and Evolutionary Computation, **1(1)**, pp. 32–49.

[186] Zhu X, 2011, *Semi-supervised learning*, Springer, New York (NY).

[187] Zitzler E, Laumanns M & Thiele L, 2001, *SPEA2: Improving the strength Pareto evolutionary algorithm*, (Unpublished) Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich.

APPENDIX A

Original Data Set Names

This appendix contains the original names of the different data sets constituting the test suite considered in this dissertation. An abbreviated code was used as replacement for each data set name for the sake of brevity. The original data set names that correspond to each of the codes can be found in Table A.1. The data sets were obtained from various sources in the literature [27, 30, 33, 120, 121].

TABLE A.1: *Original names of the data sets constituting test suite in this dissertation.*

Data set name	Data set code
Adult	C1
Attrition	C2
Audit	C3
Baker's yeast	C4
Bank marketing	C5
Banknote authentication	C6
Breast cancer Coimbra	C7
Breast cancer Wisconsin	C8
Breast tissue	C9
Car evaluation	C10
Congressional voting records	C11
Crowdsourced mapping	C12
Dermatology	C13
Dresses	C14
Drug consumption (alcohol)	C15
Drug consumption (amphetamine)	C16
Drug consumption (amyl nitrite)	C17
Drug consumption (benzodiazepine)	C18
Drug consumption (caffeine)	C19
Drug consumption (cannabis)	C20
Drug consumption (chocolate)	C21
Drug consumption (cocaine)	C22
Drug consumption (crack)	C23
Drug consumption (ecstasy)	C24
Drug consumption (heroin)	C25
Drug consumption (ketamine)	C26
Drug consumption (lysergic acid diethylamide)	C27
Drug consumption (methamphetamine)	C28
Drug consumption (nicotine)	C29
Drug consumption (volatile substance abuse)	C30
Electrical grid stability	C31
Flags	C32
Forest type mapping	C33
Indian liver patient	C34
Iris	C35
Kickstarter projects	C36
Liver spectroscopy	C37
Lymphography	C38
Mushroom	C39
Nursery	C40
Planning relax	C41
QSAR biodegradation	C42
Qualitative bankruptcy	C43
Seeds	C44
Somerville happiness	C45
Titanic	C46
Vehicle silhouettes	C47
Wine	C48
Zoo	C49

APPENDIX B

Additional Parameter Evaluation Results

This appendix contains additional results obtained during the second parameter evaluation described in Chapter 7 which focussed on determining good parameter values for the BOHTA. The results documented in this appendix were omitted from §7.3.2 so as to enhance the exposition of the main text.

After determining good parameter values for the respective sub-algorithms in §7.3.1, another parameter evaluation was performed in §7.3.2 to determine good parameter values for the BOHTA. The parameter values under investigation were presented in Table 7.14, which were subjected to a full factorial design performance comparison (as opposed to a sensitivity analysis performance comparison). For this parameter evaluation, thirty optimisation runs were obtained per data set. The resulting samples provide an extensive representation of the performance achieved by the parameter combinations, which facilitates a more in-depth analysis of the BOHTA. The F_1-score sample median and mean achieved by the different parameter combinations in respect of data sets C1–C40 are presented in Tables B.1 and B.2, respectively.

TABLE B.1: *BOHTA parameter evaluation sample medians in respect of data sets C1–C40.*

	Parameter combination sample median											
Data set	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
C1	0.7472	0.8548	0.8452	0.8499	0.8446	0.8151	0.8395	0.8203	0.8188	0.8214	0.8447	0.7932
C2	0.9016	0.8590	0.8753	0.9180	0.8473	0.9062	0.8495	0.9093	0.9134	0.8752	0.9249	0.8804
C3	1	0.9963	1	0.9975	0.9996	0.9425	0.9885	0.9698	0.9953	0.9484	0.9734	1
C4	0.9813	1	0.9523	0.9709	1	0.9889	0.9575	0.9597	1	1	0.9785	0.9882
C5	0.9970	0.9416	0.9676	0.9831	0.9576	0.9807	0.9829	0.9685	0.9777	0.9982	0.9207	0.9481
C6	0.9989	0.9212	0.9876	0.9993	0.9835	0.9342	0.9773	0.9931	0.9500	1	0.9990	0.9667
C7	0.6565	0.6394	0.6307	0.6106	0.6235	0.5709	0.6224	0.6625	0.6505	0.5905	0.6254	0.6455
C8	0.8483	0.8457	0.8513	0.9181	0.8974	0.8955	0.8920	0.9019	0.9321	0.9128	0.9282	0.9131
C9	0.7039	0.7214	0.7195	0.6980	0.7082	0.7207	0.7101	0.6613	0.7123	0.7398	0.7123	0.6965
C10	1	0.9276	0.9930	0.9932	0.9874	0.9666	0.9885	0.9516	0.9826	0.9873	0.9537	1
C11	1	0.9784	0.9962	0.9761	0.9410	0.9875	0.9747	0.9878	0.9613	0.9612	0.9416	1
C12	0.9034	0.8354	0.9037	0.8960	0.9219	0.9230	0.8900	0.8767	0.8935	0.9011	0.8846	0.8768
C13	0.8927	0.8837	0.8954	0.8682	0.9206	0.9142	0.8847	0.8928	0.9034	0.9039	0.8658	0.9178
C14	0.5184	0.5207	0.5187	0.5353	0.4935	0.5049	0.5196	0.4649	0.4485	0.5012	0.4334	0.5138
C15	0.7199	0.6734	0.6938	0.7295	0.6977	0.7073	0.7179	0.6997	0.7198	0.6489	0.6744	0.7188
C16	0.8686	0.8843	0.8657	0.8096	0.8849	0.8328	0.8549	0.8737	0.8702	0.8708	0.8031	0.8727
C17	0.8543	0.8391	0.8622	0.8358	0.8377	0.8230	0.8144	0.8096	0.7656	0.8130	0.8284	0.7726
C18	0.8289	0.8243	0.8157	0.8188	0.7969	0.8163	0.8430	0.8371	0.8358	0.8834	0.8051	0.8089
C19	0.8233	0.8260	0.8255	0.8119	0.8009	0.8183	0.7885	0.7827	0.8374	0.8167	0.8479	0.8528
C20	0.7438	0.7071	0.7109	0.7417	0.7840	0.7810	0.7940	0.7931	0.7885	0.7387	0.7791	0.7684
C21	0.7529	0.7428	0.7517	0.7696	0.7338	0.7313	0.7566	0.7438	0.7720	0.7421	0.7599	0.7685
C22	0.8083	0.7618	0.7972	0.8003	0.8012	0.7848	0.8132	0.8261	0.8222	0.7768	0.8287	0.8021
C23	0.9411	0.9173	0.9167	0.9444	0.9788	0.9500	0.9557	0.9597	0.9638	1	0.9305	0.9675
C24	0.8629	0.8264	0.8149	0.8422	0.8738	0.8871	0.8964	0.8974	0.9100	0.8539	0.9070	0.8655
C25	0.8708	0.8778	0.8989	0.8470	0.8522	0.8361	0.8545	0.8600	0.8984	0.8526	0.8610	0.8500
C26	0.8270	0.8205	0.8109	0.7915	0.8325	0.7600	0.8222	0.8334	0.8324	0.8218	0.8338	0.8062
C27	0.8854	0.8700	0.9293	0.9055	0.8896	0.8862	0.8887	0.8775	0.9084	0.8773	0.9148	0.8618
C28	0.8646	0.8412	0.8501	0.8392	0.8376	0.8500	0.8026	0.8805	0.8701	0.8691	0.8257	0.8528
C29	0.7558	0.7464	0.7589	0.7423	0.7451	0.7446	0.6952	0.7521	0.7487	0.7166	0.7028	0.7694
C30	0.7745	0.8309	0.8037	0.8254	0.8383	0.8410	0.8401	0.8436	0.8314	0.8519	0.8431	0.8445
C31	0.9301	0.8853	0.9078	0.9206	0.9439	0.9203	0.9627	0.9517	0.9549	0.9458	0.9491	0.9449
C32	0.8175	0.7968	0.8388	0.8245	0.8310	0.8378	0.8134	0.8213	0.8369	0.8234	0.7862	0.8005
C33	0.9731	0.9208	1	0.9578	0.9819	0.9698	0.9578	1	0.9966	0.9448	0.9968	0.9764
C34	0.6695	0.6809	0.6545	0.6820	0.6211	0.6828	0.6733	0.6617	0.7118	0.6925	0.6817	0.6387
C35	0.9002	0.9265	0.8967	0.8967	0.8554	0.9196	0.8244	0.8935	0.9257	0.9110	0.8896	0.8881
C36	0.8534	0.8923	0.8774	0.8850	0.9010	0.8559	0.8468	0.9098	0.9006	0.8898	0.9087	0.9203
C37	0.9751	0.9344	0.8879	0.9121	0.9373	0.9084	0.9544	0.9639	0.9353	0.9485	0.9335	0.9106
C38	0.8744	0.8657	0.8906	0.8899	0.8777	0.8850	0.8918	0.8843	0.8608	0.9071	0.9028	0.9218
C39	1	0.9899	0.9814	0.9765	0.9563	0.9447	0.9833	0.9317	0.9996	1	0.9345	0.9721
C40	0.9120	0.9061	0.8953	0.8713	0.8903	0.8725	0.8862	0.8918	0.8653	0.8833	0.8487	0.8809

TABLE B.2: *BOHTA parameter evaluation sample means in respect of data sets C1–C40.*

	Parameter combination sample mean											
Data set	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
C1	0.7668	0.8435	0.8390	0.8301	0.8285	0.8245	0.8344	0.8139	0.8199	0.8336	0.8526	0.7960
C2	0.8939	0.8778	0.8752	0.9098	0.8572	0.9037	0.8663	0.9160	0.9141	0.8899	0.9094	0.8731
C3	0.9804	0.9755	0.9734	0.9734	0.9752	0.9269	0.9693	0.9517	0.9756	0.9307	0.9507	0.9768
C4	0.9677	0.9747	0.9401	0.9653	0.9826	0.9564	0.9407	0.9318	0.9798	0.9804	0.9686	0.9754
C5	0.9725	0.9321	0.9569	0.9627	0.9307	0.9560	0.9589	0.9554	0.9540	0.9631	0.9039	0.9381
C6	0.9647	0.9124	0.9626	0.9638	0.9544	0.9162	0.9541	0.9582	0.9257	0.9821	0.9643	0.9490
C7	0.6550	0.6364	0.6383	0.6234	0.6158	0.5827	0.6252	0.6534	0.6310	0.5995	0.6190	0.6465
C8	0.8530	0.8660	0.8631	0.9099	0.8933	0.8979	0.8890	0.8982	0.9141	0.8969	0.9155	0.9120
C9	0.6942	0.7201	0.7281	0.7097	0.7068	0.7223	0.7137	0.6822	0.7124	0.7195	0.7130	0.6962
C10	0.9696	0.9164	0.9638	0.9622	0.9653	0.9558	0.9695	0.9368	0.9757	0.9545	0.9262	0.9675
C11	0.9722	0.9598	0.9668	0.9582	0.9208	0.9699	0.9581	0.9576	0.9592	0.9317	0.9185	0.9823
C12	0.9017	0.8614	0.9026	0.8974	0.9051	0.9135	0.8826	0.8802	0.8912	0.9013	0.9041	0.8907
C13	0.8838	0.8843	0.8898	0.8672	0.9068	0.9019	0.8949	0.8995	0.8957	0.8953	0.8732	0.9051
C14	0.5230	0.5122	0.5210	0.5176	0.4875	0.5038	0.5166	0.4660	0.4688	0.5014	0.4485	0.5187
C15	0.7131	0.6854	0.7019	0.7227	0.6913	0.7007	0.7112	0.7046	0.7043	0.6647	0.6880	0.7137
C16	0.8614	0.8665	0.8605	0.8208	0.8594	0.8447	0.8731	0.8707	0.8758	0.8735	0.8236	0.8638
C17	0.8448	0.8306	0.8245	0.8313	0.8333	0.8209	0.8123	0.8196	0.7847	0.8107	0.8336	0.7924
C18	0.8218	0.8337	0.8223	0.8349	0.8112	0.8242	0.8448	0.8320	0.8325	0.8796	0.8066	0.8070
C19	0.8321	0.8136	0.8338	0.8118	0.7954	0.8249	0.7898	0.7885	0.8337	0.8306	0.8481	0.8369
C20	0.7584	0.7187	0.7276	0.7473	0.7842	0.7805	0.7771	0.7723	0.7780	0.7370	0.7841	0.7691
C21	0.7491	0.7538	0.7497	0.7739	0.7347	0.7382	0.7552	0.7454	0.7615	0.7484	0.7541	0.7717
C22	0.8040	0.7725	0.7896	0.8143	0.7971	0.7983	0.8171	0.8315	0.8076	0.7805	0.8281	0.8106
C23	0.9359	0.9101	0.9124	0.9167	0.9565	0.9437	0.9514	0.9455	0.9456	0.9551	0.9336	0.9409
C24	0.8626	0.8278	0.8262	0.8393	0.8638	0.8854	0.8848	0.8898	0.8981	0.8652	0.8952	0.8656
C25	0.8696	0.8705	0.8796	0.8595	0.8467	0.8377	0.8505	0.8590	0.8885	0.8473	0.8521	0.8506
C26	0.8308	0.8184	0.8185	0.8072	0.8443	0.7754	0.8246	0.8214	0.8273	0.8136	0.8323	0.8124
C27	0.8788	0.8757	0.9205	0.9032	0.8957	0.8991	0.8913	0.8830	0.9099	0.8759	0.9027	0.8657
C28	0.8590	0.8490	0.8392	0.8443	0.8462	0.8444	0.8195	0.8648	0.8588	0.8834	0.8364	0.8488
C29	0.7547	0.7432	0.7551	0.7378	0.7385	0.7464	0.7148	0.7441	0.7561	0.7215	0.7233	0.7457
C30	0.8078	0.8413	0.8125	0.8335	0.8208	0.8484	0.8403	0.8361	0.8461	0.8544	0.8368	0.8466
C31	0.9275	0.8914	0.9065	0.9096	0.9336	0.9198	0.9418	0.9466	0.9547	0.9347	0.9285	0.9296
C32	0.8157	0.8032	0.8273	0.8082	0.8308	0.8444	0.8125	0.8321	0.8452	0.8176	0.7964	0.7987
C33	0.9587	0.9115	0.9816	0.9297	0.9626	0.9612	0.9518	0.9660	0.9630	0.9198	0.9681	0.9612
C34	0.6604	0.6800	0.6717	0.6867	0.6321	0.6785	0.6878	0.6704	0.6912	0.6740	0.6831	0.6563
C35	0.8892	0.9118	0.8893	0.8998	0.8447	0.9108	0.8303	0.9005	0.9167	0.9080	0.8925	0.8964
C36	0.8664	0.8970	0.8801	0.8793	0.8883	0.8746	0.8613	0.9097	0.8933	0.8917	0.9073	0.9043
C37	0.9534	0.9380	0.8985	0.9079	0.9273	0.9107	0.9424	0.9477	0.9200	0.9342	0.9272	0.9053
C38	0.8808	0.8745	0.8884	0.8944	0.8756	0.8775	0.8921	0.8709	0.8655	0.8847	0.8935	0.8999
C39	0.9756	0.9594	0.9657	0.9625	0.9542	0.9240	0.9677	0.9151	0.9715	0.9697	0.9260	0.9613
C40	0.9057	0.8986	0.8862	0.8693	0.8771	0.8867	0.8912	0.8954	0.8641	0.8819	0.8516	0.8747