Investigating hyperheuristics for solving bi-objective simulation optimisation problems

by

Leanne Nigrini

Thesis presented in fulfilment of the requirements for the degree of
Master of Engineering (Industrial Engineering)
in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof JF Bekker
Co-supervisor: Dr GS Nel

March 2023

Declaration

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

March 2023

Abstract

The *investigation* and *exploration* of search and optimisation methodologies are crucial research areas. Take for example the potential impact of an effective and computationally efficient decision support methodology, it could be the difference between life and death in healthcare scheduling. Schedule too few doctors and patients could die; schedule long work-hours and doctors could make fatal mistakes due to fatigue. *Simulation optimisation* is typically used to approximately solve large and complex problems that cannot be solved by exact methods. In addition, the need for better simulation optimisation approaches are further motivated by the combinatorial relationship that results in significant search spaces. One of the biggest problems that researchers face with metaheuristic approaches is the lack of *general applicability* and the high number of hyperparameter combinations that algorithms have and a lack of insight on how to choose them. This is due to the fact that the performance of metaheuristics greatly depends on the type of problem being optimised, as supported by the *no free lunch* (NFL) theorem. Accordingly, each optimisation algorithm has its strengths and weaknesses when it comes to exploring the search space.

Hyperheuristics propose to compensate, to some extent, for the weaknesses of the individual low-level heuristics (LLHs) by method of *algorithmic cooperation*, creating ensemble algorithms that are more generally applicable, *i.e.* can solve a larger range of problems than the individual LLHs are capable of solving. In this study, two hyperheuristic approaches are developed, one for population-based search and the second for single-solution based search, and are assessed using five discete-event dynamic stochastic bi-objective simulation optimisation problems. The hyperheuristics as well as their individual LLHs are implemented and assessed in Tecnomatix Plant Simulation. In addition, an algorithmic parameter study is presented for the respective LLHs to determine good hyperparameter combinations and possibly infer insights from the complex interaction.

Furthermore, due to the dynamic and stochastic nature of simulation models, there exists a sufficient number of observations per solution that need to be evaluated to be able to construct suitable narrow confidence intervals. This renders simulation optimisation computationally expensive and for that reason a pilot study is conducted to determine the feasibility of an ANN as metamodel to screen out solutions that are predicted to be of low-quality, thereby reducing the number of computationally expensive evaluations that need to be made by the simulation model.

The statement that hyperheuristics perform better (or at least similar) to its individual LLHs does not hold true for the population-based hyperheuristic. The statement, however, holds true for the single-solution based hyperheuristic. It can be concluded that both hyperheuristics failed to exhibit superior performance and did not indicate favourable performance improvements relative to *all* the individual applications of the LLHs. Furthermore, the novel pilot study provided valuable insights pertaining to the complex interaction within an ANN, however, the study could not conclude whether or not an ANN metamodel is a feasible solution to enhance the simulation optimisation process. The study does provide valuable insights which could inspire further research.

Opsomming

Die ondersoek en verkenning van optimeringmetodes word beskou as 'n kritieke navorsings gebied. Simulasie-optimeringsmetodes word dikwels gebruik om benaderde oplossings te vind vir komplekse probleme wat onoplosbaar is deur presiese metodes. Die behoefte vir beter optimeringsmetodes word verder gemotiveer deur kombinatoriese verwantskappe wat lei tot beduidende groot besluitnemingsruimtes. Een van die grootste tekortkominge in navorsing in hierdie gebied is die gebrek aan metaheuristieke wat oor die algemeen toepaslik is asook die groot aantal algoritme parameters wat kundigheid verg om van te kies. Dit is as gevolg van die metaheuristieke se sterk en swakpunte, in terme van die besluitnemingsruimte verken, wat lei tot die metaheuristiek se vermoë om sekere probleme goed te kan benader en ander nie.

Hiperheuristieke is voorgestel met die doel om vir die swakpunte van die enkele metaheuristieke of heuristieke komponente te kompenseer deur middel van algoritme samewerking om gevolglik 'n groter verskeidenheid van optimeringsprobleme te kan benader as voorheen. Twee hiperheuristieke was voorgestel en ontwikkel in hierdie studie. Die een volg 'n enkel-oplossing benadering, terwyl die ander 'n populasie-oplossing benadering volg. Die uitvoer van die voorgestelde hiperheuristieke was in Tecnomatix Plant Simulation verrig en geevalueer op vyf tweedoelige stogastiese en dinamies simulasie optimeringsprobleme. Daarby, was 'n omvattende parameterevaluarings studie uitgevoer met die doel om afleidings te kan maak oor die komplekse interaksies wat plaasvind tussen die parameters.

As gevolg van die stogastiese en dinamies aard van die simulasie-optimeringsprobleme moet daar 'n sekere aantal observasies per oplossing geevalueer word om sodoende nou vertrouensintervalle te bewerkstellig. Dit is as gevolg van die statistiese steekproefneming dat die produksielopies tydrowend is. Gevolglik is 'n ondersoek ingestel om te bepaal of 'n kunsmatige neurale netwerk as 'n metamodel die simulasie optimeringsoekproses kan ondersteun.

Die opvatting dat hiperheuristieke se prestasie beter of ten minste dieselfde is as sy individuele algoritmes, geld nie vir die populasie-oplossing hiperheuristiek nie. Die verklaring geld wel vir die enkel-oplossing hiperheuristiek wat ten minste so goed gedoen het of beter as al sy individuele heuristieke. Om af te sluit, die beslissing wat gevind was in die studie is dat die hiperheuristieke nie beter presteer het in vergelyking met al hulle individuele algoritmes nie. Die uiteinde van die ondersoek kon nie oortuigend tot 'n beslissing kom nie en daarom moet verdere ondersoek ingestel word. Waardevolle insigte is wel verkry uit die studie en kan gebruik work om verdere navorsing te inspireer.

Acknowledgements

The author wishes to acknowledge the following people and institutions for their various contributions towards the completion of this work: I am grateful to the following people who supported me toward completing this research project:

- Derrick van Schalkwyk, my husband, for his continuous love and support.
- Professor James Bekker, my supervisor and role model, for his guidance and wisdom throughout this study. Providing a strategic level guidance, always helping me see the bigger picture and helping me identify the intermediate steps that I needed to take to get where I wanted to go.
- Doctor Gerrit Stephanus Nel, my co-supervisor, for helping me dream big but also helping me scope my project, otherwise I would have never finished.
- My parents, for teaching me to always finish what I started.
- St Patrick Nice, for lending me three of his computers, which allowed me to run approximately 19 500 simulation runs and thereby complete my masters on time.

Table of Contents

List of Acronyms

AI	Artificial Intelligence
AMALGAM	A Multi-algorithm, Genetically Adaptive Multi-objective
ANN	Artificial Neural Network
ANOVA	Analysis of Variance
BAP	Buffer-allocation Problem
BAP5	Buffer-allocation Problem: five machines
BAP10	Buffer-allocation Problem: 10 machines
BAP16	Non-linear Buffer-allocation Problem: 16 machines
COP	Combinatorial Optimisation Problem
CNN	Convolutional Neural Network
DBMOSA	Dominance-based Multi-objective Simulated Annealing
DES	Discrete-event Simulation
DNN	Deep Neural Network
EA	Evolutionary Algorithm
ELU	Exponential Linear Units
FNN	Feedforward Neural Network
GA	Genetic Algorithm
GD	Great Deluge
GRASP	Greedy Randomised Adaptive Search Procedure
GLS	Guided Local Search
HA	Hyperarea
HC	Hill Climbing
ILS	Iterated Local Search
IP	(s, S) Inventory Problem
LLH	Low-level Heuristic
LReLU	Leaky Rectified Linear Units
MAE	Mean Absolute Error
ML	Machine Learning

MOO	Multi-objective Optimisation
MOOCEM	Multi-objective Cross-entropy Method
MOOP	Multi-objective Optimisation Problem
MOP	Multi-objective Problem
MOSA	Multi-objective Simulated Annealing
MOSO	Multi-objective Simulation Optimisation
MSE	Mean Squared Error
MTTR	Mean Time to Repairs
NFL	No Free Lunch
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OMP	Open Mine Problem
PReLU	Parametric Rectified Linear Units
PSO	Particle Swarm Optimisation
RBM	Restricted Boltzman Machine
ReLU	Rectified Linear Units
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
RNS	Random Number Seed
SA	Simulated Annealing
SeLU	Scaled Exponential Linear Units
SO	Simulation Optimisation
SOOP	Single Objective Optimisation Problem
SGD	Stochastic Gradient Descent
SS	Scatter Search
TA	Threshold Accepting
TPS	Tecnomatix Plant Simulation
TS	Tabu Search
TSP	Travelling Salesman Problem
WWO	Water Wave Optimisation

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

This chapter serves as an introduction to the research assignment and is formalised by research objectives, a proposed research methodology and scope. Finally, the structure of the document is discussed.

1.1 Research background

The *investigation* and *exploration* of search and optimisation methodologies are crucial research areas of decision support systems, which can be applied across industry, commerce, science, healthcare and military. Take for example the potential impact of an effective and computationally efficient decision support methodology, it could be the difference between life and death in healthcare scheduling. Schedule too few doctors and patients could die; schedule too long working hours and doctors could make fatal mistakes due to fatigue [80].

Many real-world problems are too complex to be modelled and solved by exact methods. These complex real-world problems often have no convenient mathematical representation, linear or non-linear. A subset of these problems consists of *dynamic*, *discrete*, *stochastic* problems, representing systems that evolve over time, for example a production line or a traffic intersection. Simulation is a markedly powerful and widely adopted technique for analysing and studying these complex systems [20, 256].

Attempts to use exact methods such as linear programming, non-linear programming or integer programming require several *simplifications* of assumptions and abstractions of the problem under study, resulting in two inadequate outcomes. The *first* outcome is where an optimal solution is found, but for a model that does do not adequately represent the real system, or the *second* outcome according to which the developed model is an adequate representation of the real system, but for which only low-quality solutions can be obtained [156]. Thus creating the need for *approximate* methods.

Simulation optimisation (SO) can be used to approximately solve these complex problems without the need for simplifying assumptions or abstractions. In the context of *simulation*, it is not necessary to formulate the objective function mathematically, since the simulation model (or black-box[1]) becomes the objective function, subjected to some optimisation approach [80]. Although simulation is a powerful alternative to solving complex problems, it must be noted that there is an embedded abstraction error when approximating real-world problems.

[1]A system where the inputs and outputs are known, without any knowledge of the systems inner workings.

Furthermore, complex optimisation problems, typically have significantly large search spaces, rendering them computationally intractable for most optimisation procedures. These problems are classified as *combinatorial optimisation problems* (COPs), *i.e.* the decision variables can assume a large number of values, subject to numerous and intricate constraints, which, taken together, can result in large, complex *search spaces* that are difficult to traverse by optimisation procedures. The combinatorial nature of large-scale problems serve as motivation for the use of *metaheuristics* and is considered a widely adopted simulation optimisation approach based on treating the simulation model as a black-box function evaluator [5].

One of the biggest problems that researchers face with metaheuristic approaches is the lack of *general applicability* and *metaheuristic selection*. This is due to the fact that the performance of metaheuristics greatly depends on the type of problem being optimised. This is where the *No Free Lunch* (NFL) theorem originated as each metaheuristic has its strengths and weaknesses when it comes to searching the search space [80],

> The ***NFL*** theorem states that, no search algorithm, irrespective of the performance measures used, performs better than (or worse) than another based on its average performance over all optimisation problems [257].

Researchers therefore tend to search for algorithms that are considered to be *generally applicable*, *i.e.* algorithms that can perform satisfactory on a subset of optimisation problems or specific problem domain. The problem, however, is that a vast majority of problems do not have an *exploitable* structure that can be used to characterise them into a subset of properties, and consequently one cannot design an effective algorithm for a subset of problems with unknown properties. The application of *hyperheuristics* to the field of optimisation, specifically simulation optimisation, is motivated by the NFL theorem, as they circumvent and mitigate (to an extent) the issue of characterising and designing problem-specific algorithms, by instead operating in the algorithm search space (first) before operating on the solution space, as illustrated in Figure 1.1 [80, 213].

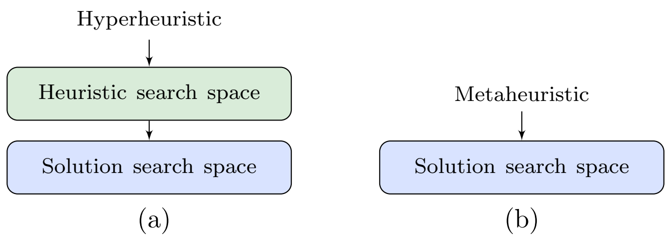


Figure 1.1: *Illustration of the search space of (a) hyperheuristics, and (b) metaheuristics, adapted from [93].*

Hyperheuristics can be used to construct an ensemble of algorithms by combining known heuristics in ways that enable each to compensate, to some degree, for the weaknesses of others by method of *algorithmic cooperation*, thereby solving a larger range of problems than their individual counterparts are capable of solving. Hyperheuristics are commonly referred to as "*heuristics to choose heuristics*" in the context of combinatorial optimisation [36], importantly, unlike most metaheuristic applications, hyperheuristics work with the *search spaces of heuristics* and not the *search spaces of solutions* and can therefore be used to construct a more general-purpose optimisation procedure. The NFL theorem still applies, the only difference is that a larger subset of problems can be solved using hyperheuristics [80].

It is important to note that optimisation procedures can be classified as either *single solution-based* search, which operate iteratively on a single candidate solution, or *population-based* search, which operate iteratively on a population of candidate solutions [93]. In this study both metaheuristics and hyperheuristics are classified as such. An example of a single-solution based metaheuristic (or S-metaheuristic) is the *simulated annealing* (SA) [143] algorithm and a population-based metaheuristic (or P-metaheuristic) is the *genetic algorithm* (GA) [120].

Due to the dynamic and stochastic nature of simulation models, it is required that a sufficient number of observations per solution be evaluated. Consequently, the simulation optimisation process should intelligently search the search space as to minimise its computationally expensive nature [20]. Metaheuristic solution methodologies often use *metamodels* to screen out solutions that are predicted to be of low-quality when compared with the incumbent solution, *i.e.* they reduce the number of solutions to be evaluated by filtering out low-quality solutions. The simulation optimisation process can therefore be enhanced by combining heuristic procedures with metamodels to search the solution space less *exhaustively* and more *intelligently* by eliminating low-quality solutions from the search space. Note that quality is based on the performance measure being optimised, for example solution dominance [5].

Some metamodel procedures use *artificial neural networks* (ANNs), to build metamodels that aid the metaheuristic search procedure. The ANN uses predefined rules to filter out the solutions that are potentially low-quality solutions, thereby mitigating computationally expensive evaluations of the black box evaluator [5, 156]. The ANN is trained using the decision variable values together with their corresponding objective function values obtained during the search, over many iterations, as datapoints. The trained ANN then becomes the metamodel that evaluates solutions, if the solutions are not considered to be low-quality solutions, then the simulation model may evaluate the solution. The metamodel is less computationally expensive to evaluate than the simulation model.

The hyperheuristic approach to simulation optimisation is an attempt to create a more generally applicable optimisation tool, whereby decreasing the effects of the NFL theorem. Additionally, an ANN metamodel filter attempts to accelerate the simulation optimisation process, thereby decreasing the effects of computationally expensive problems.

1.2 Problem description and research assignment

From the research background given, it should be apparent that there is a lack of general-purpose (context-independent) optimisation tools, specifically within the field of simulation optimisation. Therefore, the aim of this thesis is twofold. The first aim is to create a general-purpose bi-objective simulation optimisation procedure using a hyperheuristic framework, to successfully explore and approximately solve a subset of five simulation optimisation problems. The second is to conduct a pilot study to determine whether or not an ANN metamodel is a feasible solution to enhance the simulation optimisation process. The procedure is to be implemented in Tecnomatix, thereby extending its simulation optimisation capabilities, which in turn may be beneficial to industries using Tecnomatix as their decision support system.

A high-level overview of the proposed general-purpose optimisation framework is presented in Figure 1.2. Assume that the metamodel has been trained and that the initial solution(s) have been generated. The proposed simulation optimisation process *starts* by evaluating the initial solution(s), the hyperheuristic is *run* on the current solution, from which the hyperheuristic generates the neighbouring solution(s) $\boldsymbol{x}$. Thereafter, the metamodel predicts the objective function value(s), *i.e.* $F(\boldsymbol{x})$ of $\boldsymbol{x}$. If the predicted objective function value(s) is *good*, then the simulation model may *evaluate* $\boldsymbol{x}$, resulting in $f(\boldsymbol{x})$, otherwise $\boldsymbol{x}$ is discarded. This process repeats until the *stopping condition* is reached, after which the simulation optimisation process *terminates*.

Note that for single-solution based search framework the simulation model takes as input a single solution, *i.e.* one value for each decision variable, to evaluate which is then transformed by the hyperheuristic. For a population-based search framework the simulation model takes as input a population of solutions, *i.e.* each decision variable has several values to be evaluated (simultaneously) and then transformed by the hyperheuristic. In this study both search frameworks are considered, namely a single-solution based hyperheuristic and a population-based hyperheuristic framework.

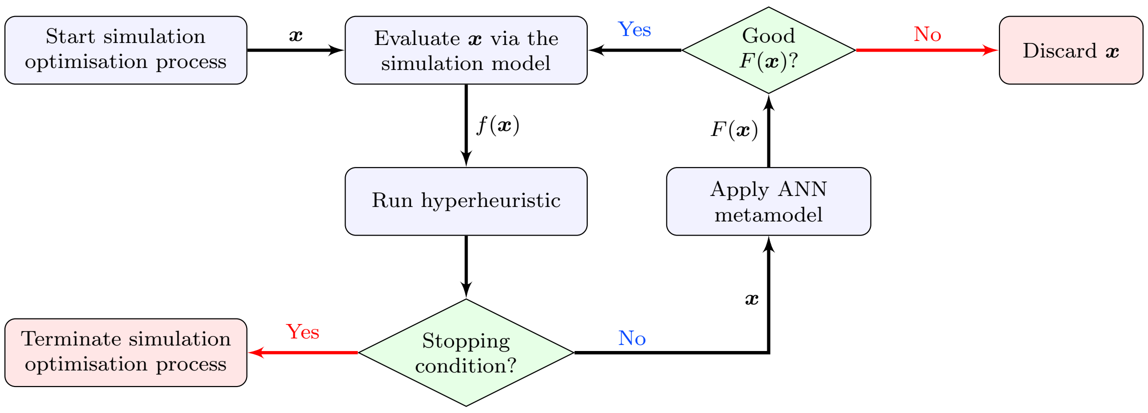


FIGURE 1.2: *An illustration of the proposed framework of a hyperheuristic optimiser with a metamodel filter [5].*

<u>To summarise</u>, the research assignment can be formulated as,

1. *Design* two hyperheuristics based on the two classifications mentioned previously, namely a single-solution based hyperheuristic and a population-based hyperheuristic to solve for *bi-objective simulation optimisation* of discrete-event *dynamic* and *stochastic* problems.

2. *Determine* whether an ANN metamodel is a feasible solution to enhance the simulation optimisation process and if so, *construct* the metamodel with the appropriate *architecture* that can be applied to a large subset of problems.

3. *Design*, *develop* and *implement* a computerised user-interface that will enable the simulation analyst using Tecnomatix to solve bi-objective simulation optimisation of discrete-event *dynamic*, *stochastic* problems with the hyperheuristics and possible ANN metamodel filter, thereby *extending* the bi-objective optimisation functionality of Tecnomatix.

In conclusion, the novelty of the research problem under investigation necessitates the design of a hyperheuristic optimiser with an ANN metamodel filter feature within Tecnomatix that may be used to solve bi-objective simulation optimisation problems. The research assignment warrants an in-depth exploration into the workings of metaheuristics and hyperheuristics as well as investigation into the different ANN architectures that can generalise well to unseen datapoints and consequently approximate various problems with sufficient accuracy.

1.3 Research scope

The following delimitations are adopted to narrow down the scope of the research problem considered in this thesis:

Simulation models used to evaluate the performance of the general-purpose optimisation tools are assumed to be built correctly, *i.e.* all models accurately reflect the real-world system that they were modelled for. A limited number of discrete-event, dynamic and stochastic simulation optimisation problems form the testbed so as to validate the optimisation tools. The problems include the (s, S) *inventory problem* (IP), three instances of the *buffer-allocation problem* (BAP), namely the BAP with *five machines* (BAP5), the BAP with *10 machines* (BAP10) and the *non-linear* BAP with *16 machines* (BAP16), and lastly the *Open mine problem* (OMP).

Bi-objective optimisation of *two* objective functions simultaneously. This may involve the following combinations of minimisation (min) or maximisation (max); *min–min*, *min–max* and *max–max*.

Hyperheuristic designs that result in more general-purpose, robust optimisation procedures that are novel in its contribution due to the given optimisation context, *i.e.* simulation optimisation. Hyperheuristics are considered more appropriate when solving different problems or even instances of the same problem and do so by managing a set of low-level (meta)heuristics (LLHs), *i.e.* LLHs are either complete metaheuristics or the heuristic components of metaheuristics.

Accordingly, the LLHs (heuristic components) considered for the inclusion in the single-solution based hyperheuristic (SBH) are *move operators* of the multi-objective variant of simulated annealing, namely *dominance-based multi-objective simulated annealing* (DBMOSA) [233]. The LLHs (metaheuristics) considered for the inclusion in the population-based hyperheuristic (PBH) are the multi-objective cross-entropy method (MOOCEM) [21] and the non-dominated sorting genetic algorithm (NSGA-II) [67].

Metaheuristics have a plethora of hyperparameters, each influencing the performance of the metaheuristic. For this reason, this study includes an empirical study conducted to determine common hyperparameters for the MOOCEM, the NSGA-II and DBMOSA, respectively.

Artificial neural networks are considered to form part of the feasibility analysis for metamodeling purposes. According to Hornik *et al.* [125] and supported by Faucett [84] an ANN with one hidden layer that feeds datapoints in a so-called forward fashion, referred to as a feedforward neural network (FNN), are considered universal function approximators. The fundamental premise of training ANNs are more or less the same however their applications are different. For that reason, ANNs such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are excluded from consideration in this study. Furthermore, hardware constraints provide further motivation for this delimitation. The interest is simulation metamodeling for regression problems.

Similar to simulation models, ANN as well as many machine learning models are essentially black box optimisers and as a result, building and gaining insight into how the ANN solves a problem is not straightforward. Interestingly, despite the possibilities of ANN, they are rarely part of introductory optimisation classes [237]. Consequently, hyperparameter optimisation of a FNN with one hidden layer is of great importance.

Supervised learning is the machine learning paradigm used to facilitate function approximation as the ANN learns the relationship between the input values(s) and the corresponding output values(s) to ultimately make predictions based on unseen inputs [81].

Regression models are used for comparison purposes to determine whether or not the FNN as simulation metamodel is feasible. This is done by comparing the performance of the

FNN to that of other regression models, namely linear regression [2], polynomial regression [113], support vector regression [234], decision tree regression [205] and random forest regression [168, 55].

Dataset is obtained, for the sake of relevancy, by running exhaustive enumeration on the (s, S) *inventory problem* (IP). This dataset is used to train, validate and test the network generalisation capabilities as well as to facilitate the hyperparameter optimisation.

Solution representation considered in this study includes; discrete and binary variables. Continuous variables, permutations and mixed solution representations are mentioned–however they are excluded from the functionality of the optimisation tool.

Development of the optimisation tools are built within Tecnomatix using the objects and methods to create an interactive user-interface facilitating the simulation optimisation process. The programming language used within Tecnomatix is *SimTalk*, and will be used to program the various algorithms.

Note that the DBMOSA, the MOOCEM and NSGA-II are built individually as well as part of their hyperheuristics to facilitate a comparison between the individual algorithms and its ensemble.

Algorithmic comparisons are limited to the BOCEGAH and BOSAH and their constituent LLHs and is facilitated using *non-parametric statistical testing.* The principal aim of this study as per the problem statement in §1.2, is twofold. The first aim is to build an optimisation tool (hyperheuristic) that performs better (measured accross five problems) than its constituent LLHs, and the second is to determine the feasibility of an ANN as metamodel to enhance the simulation optimisation process.

It is anticipated that further assumptions may be necessary as the project progresses, and these will be reported in detail at the appropriate time.

1.4 Research objectives

The following *eight* objectives are pursued in this thesis:

1. *Conduct* a review of the academic literature related to simulation optimisation, more precisely the application of hyperheuristics as well as literature related to applying FNNs as simulation metamodels.

2. *Perform* a preliminary study to determine the feasibility of a FNN with one-hidden layer as metamodel, to determine its capabilities as function approximator, before implementation.

3. *Determine* the specific parameters related to each simulation problem studied, parameters that are necessary to conduct a statistically sound simulation optimisation study, for example a sensible upper bound.

4. *Review* the literature covering the LLHs and *identify* based on the review, which hyperparameters to include in the hyperparameter tuning search space. Also, *build* a computerised user-interface within Tecnomatix to facilitate the simulation optimisation process for the MOOCEM, the NSGA-II and the DBMOSA algorithm.

5. *Design* the population-based and single-solution based hyperheuristic based using the chosen LLHs, and build a computerised user-interface within Tecnomatix to facilitate the simulation optimisation process.

6. *Perform* hyperparameter tuning for the LLHs, respectively, and *report* the results.

7. *Employ* the common best hyperparameters and *evaluate* the hyperheuristic optimisation tools within Tecnomatix on all five discrete-event dynamic stochastic simulation optimisation problems and *interpret* the results obtained from the various tools and report valuable insights realised.

8. *Recommend* possible follow-up work and further research avenues that may be pursued to continue the work done in this thesis.

1.5 Research methodology

The following methodological procedure is adopted to achieve the objectives set out for this thesis:

1. *Conduct* an extensive literature review on the following topics relating to this thesis and the completion thereof. The topics aim to assist the researcher in gaining the required knowledge and insight necessary to fulfil the research objectives 1–7.
   (a) Multi-objective simulation optimisation,
   (b) multi-objective optimisation preliminaries,
   (c) metaheuristics,
   (d) hyperheuristic design, and
   (e) simulation metamodeling by method of ANNs.

2. *Perform* a preliminary study on FNN metamodel feasibility, in fulfilment of Objective 2.
   (a) Perform hyperparameter optimisation on a FNN with one hidden layer,
   (b) evaluate the performance of the optimised FNN on the IP dataset,
   (c) evaluate the regression models on the IP dataset, and
   (d) compare their performances, *i.e.* FNN *versus* regressors.

3. *Define* the statistical prerequisites necessary to conduct a simulation study and *determine* the following specific to each simulation problem, in fulfilment of Objective 3.
   (a) The upper bounds,
   (b) the sufficient number of observations per solution.

4. Based on relevant literature, *report* the hyperparameters to be included in the hyperparameter tuning search space, in fulfilment of Objective 4.

5. *Design* the population-based hyperheuristic using the MOOCEM and the NSGA-II, called the **bi**-**o**bjective **c**ross-**e**ntropy and **g**enetic **a**lgorithm **h**yperheuristic (BOCEGAH). Also, *design* the single-solution based hyperheuristics, called the **bi**-**o**bjective **s**imulated **a**nnealing **h**yperheuristic (BOSAH) in fulfilment of Objective 5.

6. *Design* and *build* a computerised user-interface within Tecnomatix to facilitate the simulation optimisation process for the DBMOSA, MOOCEM, NSGA-II, BOCEGAH and BOSAH, also in fulfilment of Objectives 4 and 5.
   (a) Test and validate each metaheuristic and hyperheuristic in terms of logic and output using structured walkthroughs,
   (b) test the user-interface to ensure that it meets the conceptual and operational validity requirements and is credible,
   (c) the functionality must exhibit reasonableness in continuity, consistency, and produce errors based on absurd conditions,
   (d) reiterate the optimisation tools where necessary.
7. *Perform* hyperparameter tuning and extensive algorithmic parameter evaluations in a structured and statistically sound manner in order to determine the best hyperparameter combinations for BOCEGAH and BOSAH and its constituent LLHs, considering all five simulation problems. This is done in fulfilment of Objective 6.
8. *Evaluate* the optimisation tools within Tecnomatix for each simulation optimisation problem in a structured and statistically sound manner, in fulfilment of Objective 7.
   (a) Compare the performance of each hyperheuristic with it constituent LLHs as well as with each other,
   (b) evaluate the general application capabilities of the optimisation tools developed, and
   (c) report valuable insights realised.
9. *Discuss* the contribution of the addition of a general-purpose simulation optimisation tool within Tecnomatix and industries using Siemens and *recommend* future-work to further the work done in this thesis, based on the research findings, in fulfilment of Objective 8.

1.6 Thesis organisation

This thesis comprises a total of eight chapters, including this introductory chapter, a bibliography, and six appendices. Chapter 2 comprise a literature study of the work related to this thesis. Thereafter, the detailed documentation of the preliminary study conducted to determine the feasibility of an FNN as metamodel is delineated in Chapter 3. Chapter 4 reports the statistical prerequisites required to conduct a simulation study and introduced the simulation models studied. Chapter 5 comprises a review of the LLHs and their algorithm specific parameters used, as well as the population-based and single-solutions based search hyperheuristics proposed in this study. Chapter 6 reports the empirical study conducted to determine suitable hyperparameter combinations for each LLH, followed by the results chapter, Chapter 7, where both BOCEGAH and BOSAH are compared with their constituent algorithms and later compared with each other. Finally, Chapter 8 comprises a summary of the work conducted in this thesis and a documentation of the research contributions and concludes with recommended future work to be done.

This chapter contains a contextual description of time-consuming nature of simulation optimisation and the need for enhancements. This led to the formulation of a research assignment and research objectives constrained by the scope. In Chapter 2, a literature study is conducted on simulation optimisation and metamodeling. More specifically simulation optimisation approaches used in the literature are briefly discussed after which the multi-objective optimisation preliminaries are documented. Thereafter, literature pertaining to metaheuristics and hyperheuristics are discussed. Lastly, metamodels form the basis of discussion, specifically FNN.

Chapter 3 reviews the regression models that are used for comparison purposes. Next, the hyperparameter optimisation approach followed for the FNN is documented. Some performance metrics are introduced, some of which are used to compare the regressors. Lastly, the performances of the FNN is compared to that of the other regression models.

Chapter 4 briefly introduces some statistical prerequisites required to conduct a simulation study, thereafter the dynamic, stochastic simulation problems are discussed. For each problem a sensible upper bound and a sufficient number of observations per solution is determined. This chapter also documents the decision variables and objectives for each problem as well as the true Pareto fronts found by method of exhaustive enumeration and lastly, describes each simulation model in terms of its complexity. The simulation problems considered in this study include the (s, S) inventory problem, the buffer-allocation problem with five machines, the buffer-allocation problem with ten machines, the non-linear buffer-allocation problem with sixteen machines, and the Open mine problem. See Appendix D for the respective problem descriptions.

Chapter 5, presents the theoretical foundation of the LLHs, as well as its formulation and application to simulation optimisation specifically. Furthermore, the chapter discusses each LLHs algorithm specific parameters that will be included in the hyperparameter tuning search space. Thereafter, the theoretical foundation of the hyperheuristics proposed are presented, including its formulation and application to simulation optimisation context specifically.

Chapter 6 documents the empirical study conducted to determine common hyperparameter combinations for each LLH across all five simulation problems by method of statistical inferences. Penultimately, Chapter 7 documents the performances of the hyperheuristics as well as compares the hyperheuristics with its constituent algorithms and later with each other. The comparisons are performed using non-parametric statistical tests with relevant *post hoc* analysis, where applicable.

Finally, Chapter 8 summarises the work done, documents avenues for improvements and future work contributions as well as formally documents the research contributions made by the completion of this thesis.

CHAPTER 2

Literature Study

This chapter serves to fulfil Objective 1 as stated in Chapter 1. In this chapter, the reader is introduced to some of the fundamental concepts of multi-objective simulation optimisation (MOSO). First, a short overview is given of the simulation optimisation (SO) approaches in the literature. Next, an introduction to multi-objective optimisation (MOO) preliminaries pertaining to the notion of Pareto dominance and methods for determining the non-dominated solution sets, the quality of the Pareto sets are then discussed in terms of the convergence and diversity of the solution set. Thereafter an introduction to metaheuristic and hyperheuristic methods are given. Penultimately, an introduction to artificial neural networks (ANNs) as a metamodel is proposed and discussed. Finally, the chapter closes in with a summary of its contents.

2.1 Simulation optimisation

The following example demonstrates the need for simulation as a research field: A nuclear reactor operates within safe operating zones, according to which abnormal operating conditions could lead to events similar to that of the Chernobyl disaster in 1986. By method of simulation, however, abnormal operating conditions can be reproduced in theory and then avoided in practice. Simulation enables the *trail-and-error* process that real-world implementations do not, whereby reducing the cost, time and potential risk associated with the implementation of new designs [219].

Simulation can be characterised by the *time increment*, *variable types* and *time dependency* of the problem and whether or not the problem is *terminating*. In this study, the nature of the problems is terminating discrete-event simulation (DES) problems. To elaborate, problems are *discrete* (time increment), *stochastic* (variable type), *dynamic* (time dependency) and are evaluated for a specified amount of simulated time (terminating) [20].

In DES models, the state of the system changes at *discrete* and usually *random* points in time [224]. Take for example a single-server queueing system – customers arrive from an unknown population at an arrival rate lambda; the customer can go to the server if the server is idle, or join the queue and wait until the server becomes idle [20]. *To summarise*: the single-server queueing system is discrete (unique events can be identified), dynamic (the state of the system evolves over time), and stochastic (the arrival rate of customers is determined by the Poisson distribution) and starts when the store opens and terminates when the store closes.

Simulation optimisation provides a way to overcome the limitations of traditional optimisation, attributable to its ability to model uncertainty and complex interactions. The main drawback of simulation optimisation is the *computational cost* required for the trail-and-error (simulated) process [99]. Figure 2.1 shows the optimisation approaches used in literature in combination with simulation to enhance the simulation optimisation process and reduce the associated computational cost [3, 4, 18, 90, 258].

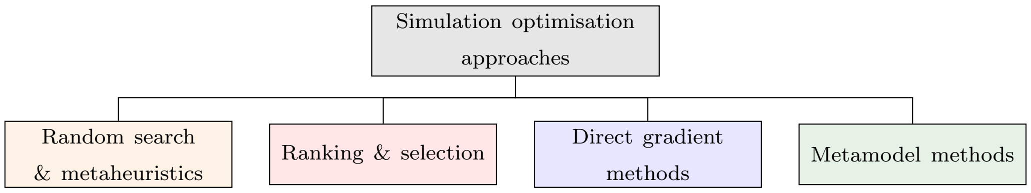

Simulation optimisation approaches
Random search & metaheuristics
Ranking & selection
Direct gradient methods
Metamodel methods

Figure 2.1: *A taxonomy of the simulation optimisation approaches in literature [18].*

Random Search & Metaheuristics are used in combination with simulation to search the decision space and find good solutions to large and typically complex problems intelligently. These methods provide the simulation model with inputs (or decision variable combinations) to evaluate and thereby reduce the search space by intelligently searching the decision space. The result is a smaller part of the total decision space that needs to be explored and typically less computational power is required to find good objective function values [20].

Ranking & Selection (R&S) procedures are statistical methods developed to compare a *finite* number of simulation alternatives and select the best system from a set of competing alternatives [105]. R&S determines the minimum number of observations per solution required, with a certain probability, to ensure that the optimal input parameter values over a finite set k, where k is 'small', *i.e.* 2–200, is found to be different at some level of statistical significance [3]. Statistical methods, specifically R&S, are recommended in case of a discrete search space with limited number of solutions, whereas random search and metaheuristics are preferred when there are a large number of alternatives, *i.e.* combinatorial nature [20]. For an overview of R&S methods, the reader is referred to [260].

Direct gradient methods are considered efficient estimators as they derive the gradient of the objective function(s) and determines the search direction used in simulation optimisation algorithms and include perturbation analysis [27, 94, 117], likelihood ratios or score functions [102, 215, 216]. *Metamodel* methods are used to find the functional relationship between inputs and their corresponding outputs. In the context of this study, metamodels are used to enhance the simulation optimisation process by filtering and screening out solutions that are predicted to be of *low-quality* when compared with the incumbent solution(s) [18], see §2.5 for more detail.

The use of these approaches is motivated by the sheer size of most real-world problems, specifically *combinatorial optimisation problems* (COPs). COPs are a class of optimisation problems comprising multiple discrete decision variables, as opposed to continuous decision variables. Even though COPs have a finite search space they can often be too large for exhaustive enumeration to be considered a feasible option [150].

In support of this statement, the so-called combinatorial relationship is explained. As the number of decision variables increase the number of combinations proverbially explode, which is often referred to as the *combinatorial explosion problem* [150]. Consider the buffer-allocation problem (BAP) with 16 machines as an example: there are 15 buffers and suppose that each buffer has 10 buffer spaces. This example results in a search space cardinality of 1×10^{15} solutions. Now imagine that each solution evaluation takes 0.5 seconds and because of the stochastic nature of simulation models, several observations per solution evaluation is required, say 100. This results in an exhaustive search that will take 1 585 489 599 years to complete. Also, to further this explanation, it is said that there are approximately 1×10^{25} grains of sand on earth.

Some examples of real-world COPs are vehicle routing and scheduling problems. Most COPs are considered NP-Hard, which requires exponential time to be solved to optimality [93]. For an exhaustive list of COPs being studied in literature, the reader is referred to [134].

There are two main approaches towards optimisation, namely *exact* and *approximate* (or heuristic) methods. Even though exact methods guarantee optimal solutions within a finite amount of time, they are not considered suitable for solving many real-world problems [198]. On the other hand, approximation methods, such as metaheuristics, do not guarantee optimality but are widely used for solving complex COPs [198]. Simulation used in combination with metaheuristics can guide the search, evaluated by the simulation model, in the most effective way, instead of blindly itemising scenarios or performing exhaustive enumeration [99].

The *simulation model* represents the objective function that needs to be optimised, representing the complex but unknown relationship between the decision variables, termed black-box optimisation [80]. The simulation optimisation process is illustrated in Figure 2.2. First, the number of observations per solution, the confidence interval and the simulation time specific to the simulation problem are to be defined as it influences the simulation output as well as the analysis of the output. This matter is discussed in detail in Chapter 4.

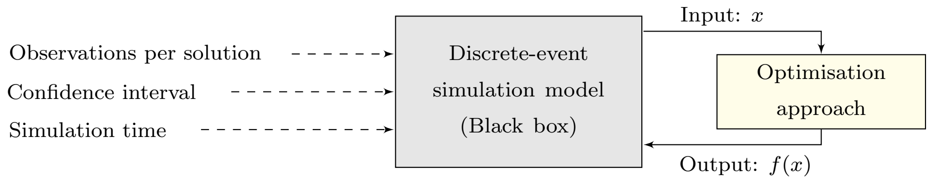


Figure 2.2: *Illustration of the black-box scenario used in simulation optimisation, adapted from [93].*

Next, values for the decision variables are used as input x which are then evaluated by the simulation model (or black-box or objective function) to generate the output $f(x)$. The output is then subjected to some optimisation approach (metaheuristic), which determines the next input value(s) of the decision variables to be evaluated by the simulation model. This procedure continues until the optimisation approach terminates due to a predefined stopping condition. Note that many decision variables may be evaluated, and as such $\boldsymbol{x}$ is a vector of n values, where n is the number of decision variables. Similarly many objective(s) may be of interest, and as such $f(\boldsymbol{x})$ corresponds to a vector of m values.

There is a trade-off relationship between *time*, *cost*, *accuracy* and *risk* for the real-world system, a simulation model and simulation metamodel. This relationship is depicted in Figure 2.3, where the inputs $\{x_1, \ldots, x_i, \ \ldots, \ x_n\}$ for the real-world system, the simulation model and metamodel are the same, however, the outputs are different and distinguishes each in terms of *time*, *cost*, *accuracy* and *risk* associated. In Figure 2.3 there are n inputs and m outputs. The inputs are natural numbers ($\mathcal{N} \in \{1, 2, \ldots, \infty\}$) since the inputs considered in this study are integers, while the outputs are real numbers ($\mathcal{R}$), *i.e.* integer or non-integer values.

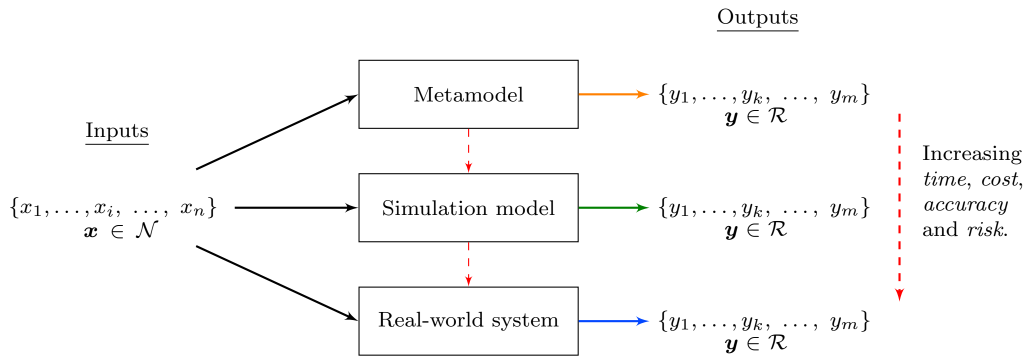

Outputs
Inputs
Metamodel
Simulation model
Real-world system
$\{x_1, \dots, x_i, \dots, x_n\}$ $\boldsymbol{x} \in \mathcal{N}$
$\{y_1, \dots, y_k, \dots, y_m\}$ $\boldsymbol{y} \in \mathcal{R}$
Increasing *time*, *cost*, *accuracy* and *risk*.

Figure 2.3: *Illustration of the relationships that exist among real-world systems, simulation models and simulation metamodels, adapted from [165].*

The real-world implementation is the most time-consuming, costly and embodies the most risk, but is the most accurate. Simulation metamodels require the least amount of time, are the least costly and embody the least risk, but is also the least accurate. Simulation represents a good trade-off between each of the aforementioned criteria and is therefore the preferred choice by many. Simulation model can be used in combination with a metamodel, to benefit from the accuracy of the simulation model and the time, cost and risk of metamodels [165].

Each of the simulation problems considered in this study are combinatorial in nature and therefore simulation in combination with some optimisation approach is the preferred choice. Furthermore, an optimisation problem can be classified as *single-objective* or *multi-objective*, in this study bi-objective simulation optimisation problems are considered. The next section addresses the preliminaries for solving multi-objective COPs.

2.2 Multi-objective optimisation preliminaries

Multi-objective optimisation problems (MOOPs) have multiple, usually *conflicting* and often *non-commensurate* objectives that need to be optimised simultaneously. For this reason, MOO searches for a set of *tradeoff* solutions, termed the *Pareto set* [186].

Formally, a MOOP has a set of m objective functions, n decision variables with J equality and K inequality constraints. As mentioned previously, in multi-objective simulation optimisation (MOSO) there is no formulation of an objective function, the simulation model is the objective function. However, for the sake of completeness, the mathematical formulation of a MOOP can be stated as follows, where the minimisation of all the objectives is assumed without loss of generality,

$$\begin{aligned}
&\text{minimise} && f(\boldsymbol{x}) = \{f_1(\boldsymbol{x}), \dots, f_m(\boldsymbol{x})\}, && && (2.1)\\
&\text{subject to} && g_k(\boldsymbol{x}) \geq 0, && k \in \{1, \dots, K\}, && (2.2)\\
& && h_j(\boldsymbol{x}) = 0, && j \in \{1, \dots, J\}, && (2.3)\\
& && x_i^{(L)} \leq x_i \leq x_i^{(U)}, && i \in \{1, \dots, n\}, && (2.4)\\
&\text{where} && \boldsymbol{x} = \{x_1, \dots, x_n\} \in X, && && (2.5)\\
& && \boldsymbol{f} = \{f_1, \dots, f_m\} \in F, && && (2.6)
\end{aligned}$$

where X denotes the decision space $x = (x_1,\ x_2)$, where x_1 and x_2 have discrete values, and F the objective space $y = (y_1,\ y_2)$. Decision variables are bounded by an upper and lower bound as stated in (2.4). If solution $\boldsymbol{x}$ satisfies the constraints (2.2)–(2.4), it is considered a feasible solution [46, 169, 266].

For any MOOP, there exists a *true* but *unknown* set of non-dominated (or Pareto optimal) solutions, when considering the entire feasible decision space. These solutions form the true Pareto set ($\mathcal{P}_T$) and visually they represent the boundary called the true Pareto front. Solutions are guided towards the Pareto front by method of *fitness assignment*. The fitness assignment strategy used in this paper is dominance-based [266]. Approximate methods search the decision space and identify a set of *high-quality* non-dominated solutions that form the approximate Pareto set, for a given MOOP, by means of the notion of *solution dominance* (or *Pareto dominance*) [186]. The notion of solution dominance is discussed next.

2.2.1 The notion of solution dominance

Solutions are considered non-dominated if no other solutions in the search space are superior to them, when *all* objectives are considered [266]. Formally, a solution x_1 dominates another solution x_2, if the following criteria are true:

1. Solution x_1 is no worse than solution x_2 in respect of *all* objectives, and
2. solution x_1 is better than solution x_2 in at least one objective, $f_1, \ldots, f_m$.

If criteria 1 and 2 are *true*, it can be said that x_1 *dominates* x_2, denoted by $x_1 \prec x_2$. This is elucidated in the *max–min* bi-objective optimisation problem (BOOP) given in Figure 2.4. The members $\{x_3,\ x_5\}$ form approximate Pareto Front 1, *i.e.* $\mathcal{F}_1 = \{3,\ 5\}$ and are considered non-dominated solutions [68, 266]. For the sake of brevity, when referring to true Pareto solutions, the solutions are explicitly referred to as *true*, otherwise it is regarded as approximate.

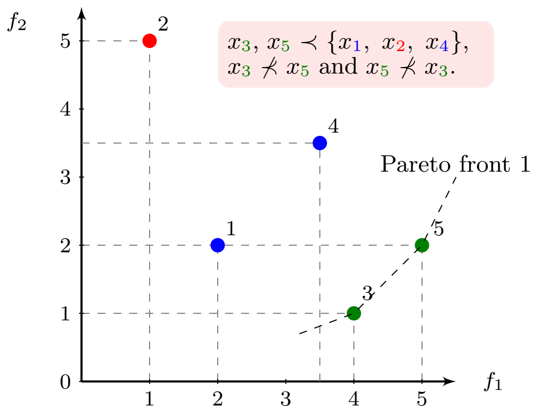


Figure 2.4: *An illustration of solution dominance using a population of five solutions, adapted from [60].*

Depending on the space considered, the number of Pareto solutions may differ. In Figure 2.5, a solution in the objective space F is considered as a single point $\{f_1,\ f_2\}$ (●), however it is represented by two different solutions x_1 and x_2 in the decision space (●). The convention used in this paper is that Pareto set refers to the objective vector, while Pareto optimal solutions refer to the corresponding decision variable combinations. If the system is operated using any of the Pareto optimal solutions (●), then the system is considered optimal. Mathematically, non-dominated solutions are regarded as *equally desirable*. However, in the real world, a final solution is selected for implementation and is based on some *preference information*.

This selection process is typically the responsibility of the decision maker [181, 266], but is not part of the scope of this study.

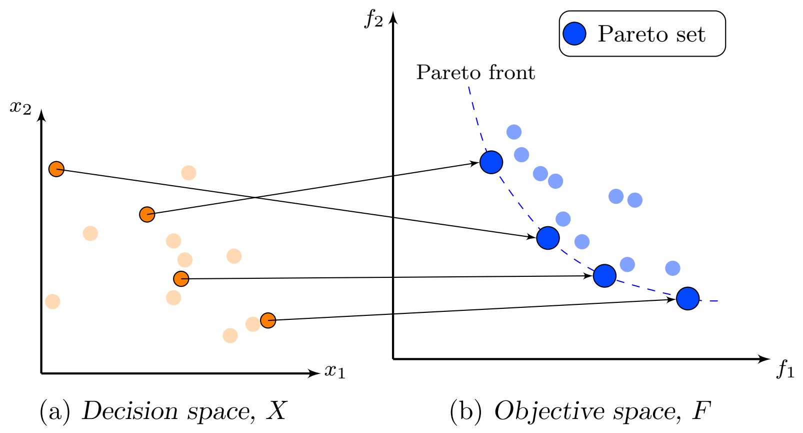


Figure 2.5: *Illustration of the concept of Pareto optimality for two decision variables* $\{x_1,\ x_2\} \in X$ *and two objectives* $\{f_1,\ f_2\} \in F$ *for a min–min and deterministic output [20].*

MOO of *stochastic output* is very complex and computationally intensive and is elucidated using Figure 2.6. For each decision variable combination $\{x_1,\ x_2\}$, the simulation model must execute a number of observations per solution to be confident, at a certain level of significance, that the point estimator value is a *good approximation* of the true but unknown values for $\{x_1,\ x_2\}$ [15]. This matter is discussed in detail in Chapter 4.

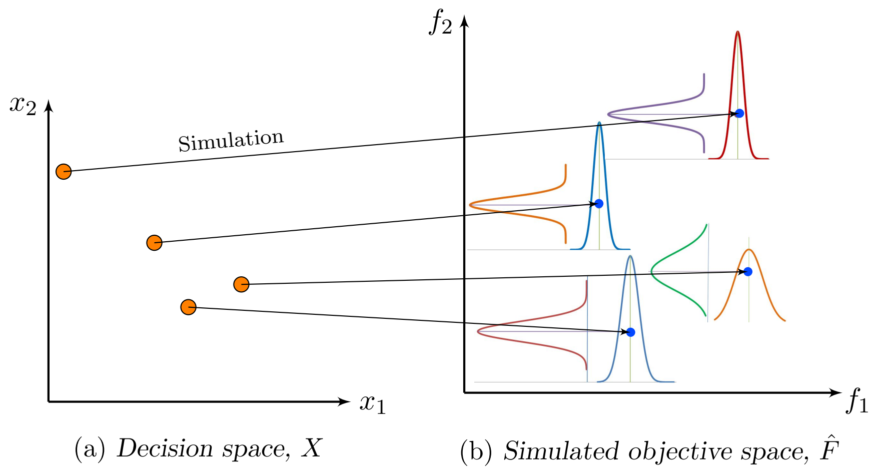


Figure 2.6: *Illustration of the computational burden of simulation optimisation for two decision variables* $\{x_1,\ x_2\} \in X$ *and two objectives* $\{f_1,\ f_2\}$ *in the simulated objective space* $(\hat{F})$ *for stochastic output, adapted from [20].*

Many solutions are evaluated during multi-objective simulation optimisation (MOSO), especially if metaheuristics are used to guide the search. This requires some form of ranking to determine which solutions form part of the approximate Pareto set. A popular ranking method is described in Algorithm 2.1, however, this method only applies to *deterministic results* [20, 104]. Recall that, the results generated by a simulation model are *stochastic*, therefore the more appropriate ranking method is multi-objective ranking and selection (MORS), as it ensures that the solutions that constitute the Pareto set are statistically significantly different [72, 160]. However, MORS falls outside of the scope of this project.

Consequently, some work has to be done to ensure that the solutions in the Pareto set are *statistically significantly different* to enable the use of Algorithm 2.1. MORS is replaced by determining a sufficient number of observations per solution that results in statistically significant solutions This is dicussed in more detail in Chapter 4.

Consider Algorithm 2.1 for a *min–min* BOOP. The algorithm takes as input the approximate Pareto set S and returns the non-dominated fronts and corresponding ranks. The algorithm starts by sorting the first objective (f_1) in descending order. Thereafter, the values of the second objective (f_2) are considered and determine its rank. Remember that both objectives are to be minimised, therefore if the value of f_2 in row i is larger than the value of f_2 in row j, the rank of row i is incremented since it is dominated. Initially each row has a rank of zero.

Algorithm 2.1: Pareto ranking algorithm, for a *min–max* bi-objective problem [104]

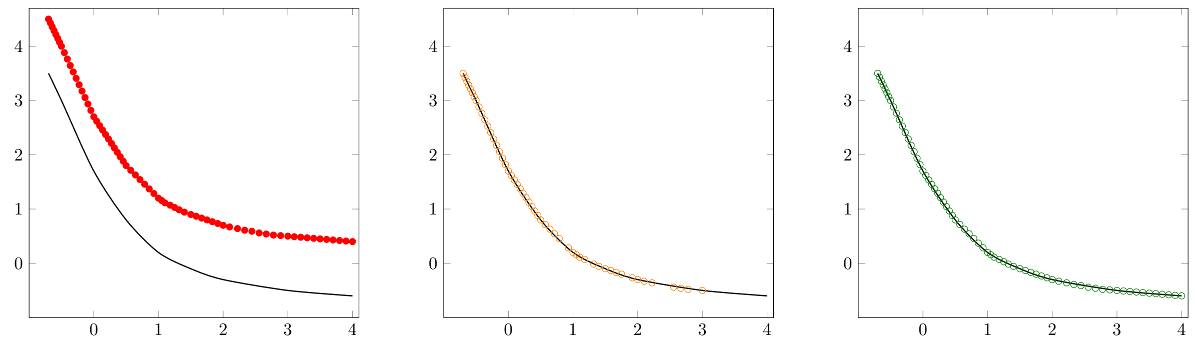
```
   Input  : A set of solutions $\mathcal{S}$ containing the values for two objective functions.
   Output : The approximate Pareto set $\mathcal{P}_S$ which contain all the solutions with $\rho = 1$
1  $\mathcal{S} \leftarrow \text{sort}(\mathcal{S}, 1)$;              // Sort the first objective in descending order
2  for $i \leftarrow 1$ to $|\mathcal{S}|$ do
3      $penalty \leftarrow 0$;                             // Initialise Pareto rank to zero
4      for $j \leftarrow (i+1)$ to $|\mathcal{S}|$ do
5          if ${f_j}^2 \prec {f_i}^2$ then // i.e. if ${f_j}^2 \geq {f_i}^2$
6              $penalty \leftarrow penalty + 1$;                    // Increment the penalty
7      $\rho_i \leftarrow penalty$;                                 // Rank of solution $i$
8  for $i \leftarrow 1$ to $|\mathcal{S}|$ do
9      if $\rho_i = 0$ then
10         $F_1 \leftarrow F_1 \cup \{i\}$
11         $\mathcal{P}_S \leftarrow \mathcal{P}_S \cup \{F_1\}$;            // Approximate Pareto set
```

Specifically, the rank of Row 1 is incremented for every other row it is dominated by. Similarly, the rank of Row 2 gets incremented for every other row it is dominated by, except itself and the rows before it. This is repeated until the last row, which is given a default rank of zero. All rows that have a rank of zero represent non-dominated solutions (or the Pareto set) and subsequently $\mathcal{F}_1$ and Pareto rank $\rho_1 = 1$. The rank value of zero indicates solution dominance, *i.e.* no solution pair $\{f_1,\ f_2\}$ dominates any other pair in the Pareto set ($\mathcal{P}_S$) [20].

In the next section the properties that determine the quality of an approximated Pareto front are discussed.

2.2.2 The quality of MOO algorithm

The quality of a MOO algorithm depends on two properties that describe the approximated Pareto front, namely *convergence* and *diversity*. The goal of a MOO algorithm is to converge toward the true Pareto front, *i.e.* generate solutions near the true Pareto solutions while maintaining a diverse set of solutions, *i.e.* generate solutions that represent the entire range of the Pareto front [61, 266]. Figure 2.7a shows an approximation front having a diverse set of solutions, but does not converge to the true Pareto front. Next, Figure 2.7b shows an approximation front that is close to the true Pareto front, *i.e.* converges, however some regions are unsearched resulting in poor diversity. Lastly, Figure 2.7c shows an approximation with both desirable properties, *i.e.* good convergence and diversity. It can be seen that too much diversity disrupts the search process and too little could lead to premature convergence [142, 266].



(a) *Bad convergence, good diversity* (b) *Good convergence, bad diversity* (c) *Good convergence and diversity*

Figure 2.7: *Examples of approximation sets illustrating (a) bad convergence and good diversity, (b) good convergence and bad diversity and (c) good convergence and diversity.*

An algorithm's ability to generate good quality approximations, as depicted in Figure 2.7c, is governed by its ability to efficiently *explore* the search space and effectively *exploit* prospective regions [77], *i.e.* to generate a diverse set of non-dominated solutions near the true Pareto front. Therefore, a balance between exploration and exploitation needs to be maintained [107]. Many MOO algorithms use an archive to prevent non-dominated solutions from being lost or to guide the search based on previously archived solutions [266]. Note that archived solutions are not necessarily only non-dominated solutions, for example simulated annealing (SA) [143] accepts *dominated* solutions into its archive with some probability. This method of accepting dominated solutions aids the search algorithm in escaping local optima. The notion of archiving is discussed next.

2.2.3 Archiving

For the sake of simplicity Figure 2.8 represents a *max–max* BOOP in which the contents of the archive consist only of non-dominated solutions, where the Pareto set will become clear by notion of archiving. The non-dominated solutions generated are represented by ● and form the archive $\mathcal{A}$.

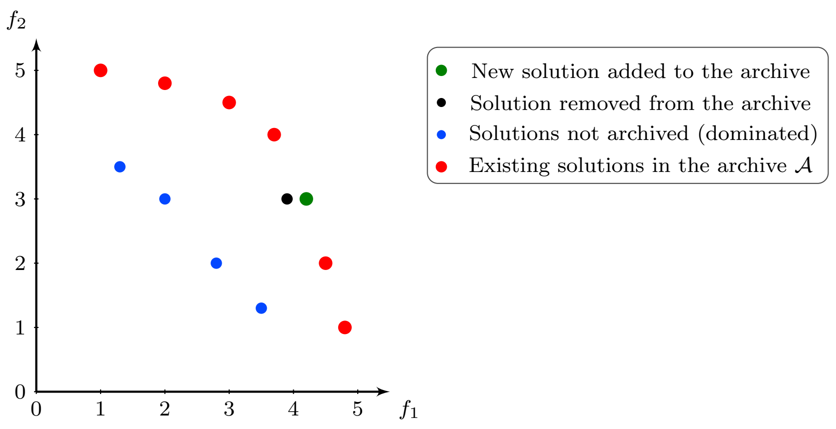


Figure 2.8: *An example illustrating the notion of archiving for a max–max bi-objective optimisation problem, adapted from [68].*

Solution ● was not dominated until solution ● was generated, replacing ● by ● in $\mathcal{A}$. Solution ● represent the dominated solutions that are not archived [68]. For comparison purposes, it is necessary to assess the performance of a search algorithm, a matter discussed in the next section.

2.2.4 Performance assessment of MOO algorithms

Attention is afforded to *quality performance indicators* because performance indicators refer to both *time* and *quality*. Simulation time is a function of the computer's processing power, *i.e.* the computer used, and the number of simulations optimised simultaneously. To elaborate, the amount of time it takes to run a single algorithm on a simulation problem differs from machine to machine, also, the simulation time increases when the number of simulation models being optimised simultaneously increases, however not linearly. For that reason time cannot be used to assess the performance of a MOSO algorithm. As a result, quality performance indicators are considered to assess the performance of a MOO algorithm in the objective space, *i.e.* in terms of the approximate Pareto set obtained [222].

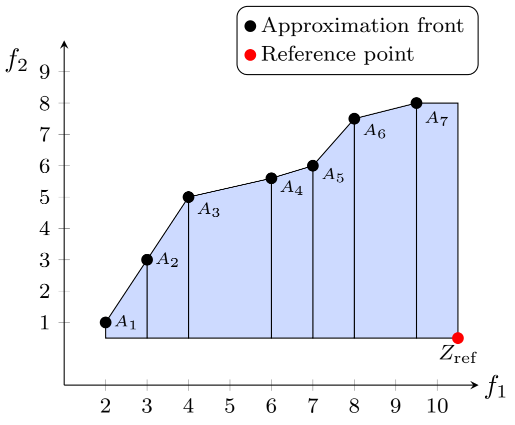
Indicator approaches measure quantify the performance of an approximation set as a numerical value. Quality performance indicators can be classified as *unary* or *binary*. Unary indicators take one approximation set as input and returns a numerical value (that describes its cardinality, convergence or diversity) as output. Whereas binary indicators, take as input two approximation sets and considers the relationship between them (in terms of dominance) and returns a numerical value (that dictates which set is better) as output [147]. In the absence of a true Pareto front, a unary performance assessment methodology is required to enable the comparative analysis between approximation fronts returned by each algorithm (for a specific problem), relative to one another.

An extensive review of performance indicators conducted by Riquelme *et al.* [210] concluded that the hypervolume indicator is the most widely used quality performance indicator, and is attributable to being *Pareto compliant*, evaluation properties and ease of implementation since it does not require a true Pareto set as reference [162]. The hypervolume[1] [267, 269] is a unary quality indicator that measures both the *convergence* and *diversity* of an approximation set [166], denoted by I_H. In this paper, I_H is referred to as a *hyperarea* since the simulation problems being optimised are bi-objective. The hyperarea indicator essentially measures the portion of the objective space that is dominated by the approximation front, relative to prespecified reference point. Note that the reference point has to be dominated by the entire approximation front, *i.e.* by both objectives. Formally, the hyperarea can be defined as

$$I_H(Z_{\text{ref}}, \boldsymbol{X}) = \Lambda\left(\bigcup_{X_n \in X} \left[f_1(X_n), f_1^{\text{ref}}\right] \times \left[f_2(X_n), f_2^{\text{ref}}\right]\right), \tag{2.7}$$

where $I_H(Z_{\text{ref}}, X)$ represents the area covered by the approximation set X, $Z_{\text{ref}} = \{f_1^{\text{ref}}, f_2^{\text{ref}}\}$ refers to the problem specific reference point and $\Lambda(\cdot)$ refers to the Lebesque measure. Note that when conducting multiple comparisons of optimisation algorithms, the reference point must be the same, otherwise the resulting I_H indicator values are not comparable [214, 268]. A simple example is shown in Figure 2.9 with the *min–max* bi-objective approximation set containing seven solutions, $\boldsymbol{X} = \{(2,1),\ (3,3),\ (4,5),\ (6,5.6),\ (7,6),\ (8,7.5),\ (9.5,8)\}$ and the reference point $Z_{\text{ref}} = (10.5, 0.5)$. The hyperarea is represented by the shaded (blue) area and corresponds to a hyperarea value $I_H = 44.53$, which is determined by summing the individual hyperareas as shown in Figure 2.9.

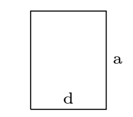
[1]Also termed the S-metric [269], hyperarea in the case of two objectives or Lebesgue measure [85]



$$I_H = A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7$$

$$A_1 = \tfrac{1}{2}((1 - 0.5) + (3 - 0.5)(3 - 2)) = 1.5$$
$$A_2 = \tfrac{1}{2}((3 - 0.5) + (5 - 0.5)(4 - 3)) = 3.5$$
$$A_3 = \tfrac{1}{2}((5 - 0.5) + (5.6 - 0.5)(6 - 4)) = 9.6$$
$$A_4 = \tfrac{1}{2}((6 - 0.5) + (5.6 - 0.5)(7 - 6)) = 5.3$$
$$A_5 = \tfrac{1}{2}((6 - 0.5) + (7.5 - 0.5)(8 - 7)) = 6.25$$
$$A_6 = \tfrac{1}{2}((7.5 - 0.5) + (8 - 0.5)(9.5 - 8)) = 10.86$$
$$A_7 = (10.5 - 9.5)(8 - 0.5) = 7.5$$

$$A = ad$$



$$A = \tfrac{1}{2}(a + b)h$$

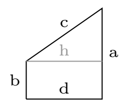


Figure 2.9: *An example illustrating the concept of hyperarea for a min–max bi-objective approximation front. The blue shaded area corresponds to the value of the hyperarea indicator for this approximation front, relative to the reference point.*

Notice that $A_1 - A_6$ are trapezoids and are calculated as such, whereas A_7 is simply a rectangle. A larger hyperarea value is preferred, and because the hyperarea indicator is Pareto compliant, it can be that that if the hyperarea of approximation set A is larger than approximation set B ($I_A > I_B$), then that approximation set A is preferable to B. However, the indicator value does not provide any information with regards to the extent according to which A outperforms B, and whether or not it is significant. This matter is discussed in detail in Chapter 6. For more on quality performance indicators refer to [267, 270].

This concludes the discussion of MOO preliminaries for solving COPs, hereafter metaheuristics for solving MOOPs are discussed.

2.3 Metaheuristics

It has been shown that metaheuristics provide acceptable solutions in a reasonable amount of computational time, and are therefore considered good substitutes for exact algorithms [93, 114, 198]. Metaheuristics are classified using many criteria, as seen in Table 2.1, namely *nature* vs. *non-nature* inspired, *memory* vs. *memoryless*, *deterministic* vs. *stochastic*, *population-based* vs. *single-solution based* search and *iterative* vs. *greedy* [93]. Metaheuristics that are considered memoryless, do not use the historical information during the search process, whereas metaheuristics that use memory, such as TS, use a specific structure to take historical information into account. Moreover, deterministic algorithms generate the same final solution(s) for the same initial solution(s), whereas stochastic algorithms, generate different final solutions from the same initial solution(s). Note that this characteristic influences how the performance of the metaheuristic is evaluated [93], refer to Chapter 6.

Most of the metaheuristics are iterative algorithms, in that they start with a complete solution and transform it using a set of search operators; this is performed at each iteration [93, 134]. Notice that the algorithms belonging to each family of metaheuristics, as discussed, share many search mechanisms. However, the most used classification of a metaheuristic takes into account the number of solutions transformed in each step of the iterative algorithm. For that reason, metaheuristics will be classified as either single-solution based which transform a single solution or population-based which transform a population of solutions [134]. Figure 2.10 demonstrates the complementary strengths and weaknesses of S-metaheuristics and P-metaheuristics.

Table 2.1: *Criteria used for the classification of metaheuristics.*

Single-solution based	Inspiration	Memory	Stochastic	Iterative
Simulated Annealing (SA) [143]	Physics		✓	✓
Tabu Search (TS) [100]	AI	✓		✓
Threshold Accepting (TA)	-			✓
Iterated Local Search (ILS) [170]	-		✓	✓
Breakout Local Search (BLS) [23]	-	✓	✓	
Descent-based Local Search (DLS) [265]	-		✓	
Guided Local Search (GLS) [253]	-	✓	✓	✓
Variable Neighborhood Search (VNS) [185]	-		✓	✓
Hill Climbing (HC) [131]	-			✓
Large Neighborhood Search (LNS) [228]	-	✓	✓	✓
Great Deluge (GD) [79]	-	✓	✓	✓
Greedy randomised adaptive search procedure (GRASP)	-		✓	
Population-based	**Inspiration**	**Memory**	**Stochastic**	**Iterative**
Genetic Algorithm (GA) [120]	Biology		✓	✓
Memetic Algorithm (MA) [187]	Biology	✓	✓	✓
Differential Evolution (DE) [240]	Biology	✓	✓	✓
Artificial Bee Colony (ABC) [133]	Bees	✓	✓	✓
Ant Colony Optimisation (ACO) [73]	Ants	✓	✓	✓
Particle Swarm Optimisation (PSO) [139]	Flocking birds	✓	✓	✓
Harmony Search (HS) [91]	Music	✓	✓	✓
Scatter Search (SS) [98, 101]	Evolution	✓	✓	✓
Water Wave Optimisation (WWO) [264]	Wave theory		✓	✓
Imperialist Competitive Algorithm (ICA) [7]	Society		✓	✓

P-metaheuristics are considered exploration (diversification) oriented as they are powerful in the approximation of the whole Pareto set while S-metaheuristics are efficient in exploiting (intensify) the search around the obtained approximations [196, 197, 204]. Hybrid metaheuristics combine S- and P-metaheuristics to work together [134].

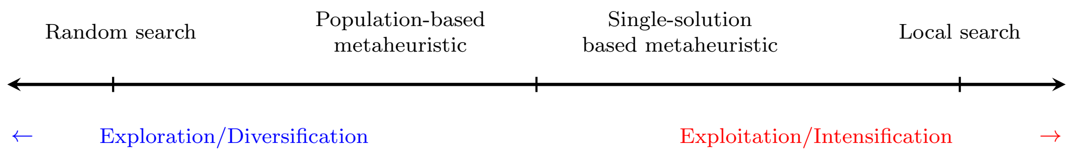

Random search
Population-based metaheuristic
Single-solution based metaheuristic
Local search
← Exploration/Diversification
Exploitation/Intensification →

Figure 2.10: *The two conflicting criteria in the design of a metaheuristic, namely exploitation and exploration, adapted from [93].*

Random search only *explores* the search space which means that non-improving solutions are also accepted and because promising regions are not exploited it could result in suboptimal solutions. Local search on the other hand, only *exploits* the search space which means that only improving solutions are accepted and because of the lack of exploration, promising regions may not be found and could result in premature convergence and consquently suboptimal solutions. There is a balance that needs to be maintained, called the *exploration–exploitation* tradeoff. Exploration of the search space and exploitation of promising regions which are determined by *good* solutions [93]. Some of the main classes of metaheuristics are listed below, with examples, some of which are regularly used or have been adapted to solve large COPs.

Local Search Methods start with an initial solution and seeks to improve the solution by searching in the neighbourhood (or locally). When an improvement is found, the process seeks to improve the solutions, *etc.* Examples are *variable neighbourhood search* [111, 185] and *guided local search* [246, 253].

Simulated Annealing resembles local search, but accepts non-improving solutions with some probability, whereby escaping the local optimas [143].

Tabu Search resembles local search, but uses some memory structure to prohibit recent solutions to be revisited, TS also escapes from local optima by allowing non-improving moves [97, 100, 115, 167].

Evolutionary Algorithms apply the theory of evolution to a population of solutions. Examples include *genetic algorithms* [120], *evolutionary strategies* [93], *scatter search* [101], genetic programming [17].

Swarm intelligence inspired by the collective behaviour of swarms and social insect colonies [80, 77]. Examples include artificial bee colony [133], ant-colony optimisation [73] and particle swarm optimisation [139].

Mememic Algorithms are population-based search techniques in combination with local search techniques, where information is transferred by imitation, rather than genetically [187].

Despite the individual success of metaheuristics and their widespread adoption, there is some reluctance to use them. This is attributable to each having several parameter or operator choices which directly influence their performance, but to which choice of parameters of operators to use are unclear to non-expert users [213].

Consequently, the application of *hyperheuristics* to the field of optimisation, spesifically simulation optimisation, is motivated by the above-mentioned drawbacks of metaheuristic search methodologies. Hyperheuristics mitigate, to some extent, the difficulty of characterising and designing problem-specific algorithms, by operating in the search space of metaheuristics (or heuristic components) before operating on the search space of solutions, whereas metaheuristics operate directly on the search space of solutions.

2.4 Hyperheuristics

The main motivations for research on hyperheuristics are to attempt to respond to the legitimate limitations of existing search methods, some of these limitations include [213]:

1. Metaheuristics cannot be applied to new optimisation problems from diffrent domains or even instances of the same problem without some changes to the algorithm, and

2. usually there is a large number of parameters or algorithm choices that must be defined, without guidelines for how to select them [36, 39].

Consequently, hyperheuristics have been developed to represent a class of high-level adaptive search techniques to raise the level of generality of search algorithms [36, 39]. Accordingly, the application of hyperheuristics is more appropriate to solve different problems or even instances of the same problem and does so by managing a set of low-level (meta)heuristics (LLHs).

By combining LLHs, ensemble algorithms are constructed that enable each heuristic to compensate, to some degree, for the weaknesses of others by method of *algorithmic cooperation* [142]. The result is a more effective search method that leads to a better overall performance when compared with the performance of the individual heuristics, *i.e.* hyperheuristics are adept at solving a larger range of problems [80, 36, 137]. Hyperheuristics can be classified in terms of (i) the different sources of feedback information used and (ii) the nature of the heuristic search space, as shown in Figure 2.11. Firstly, a hyperheuristic can be considered a *learning algorithm* when it uses some form of feedback *during* the search process. The use of information *during* the search process is termed *online learning*, used *after* the search process is termed *offline learning* [38]. Some studies incorporate both, called *mixed learning*, see [6]. Whenever *no learning* takes place, a random or exhaustive process dictates the selection strategy [36].

Secondly, the nature of the heuristic search space, classified as either *heuristic selection* where existing LLHs are selected or *heuristic generation* where new heuristics are generated from the components of existing LLHs [38]. The second level corresponds to the *nature* of the search structure, delineated into two types of LLHs, either *constructive* or *perturbative* [38]. Constructive heuristics incrementally build a complete solution from nothing by applying a selected LLH to partial solutions at each step. Whereas, perturbation heuristics operate on complete solutions, iteratively performing local search operations until some stopping (or termination) criterion is reached [76, 123].

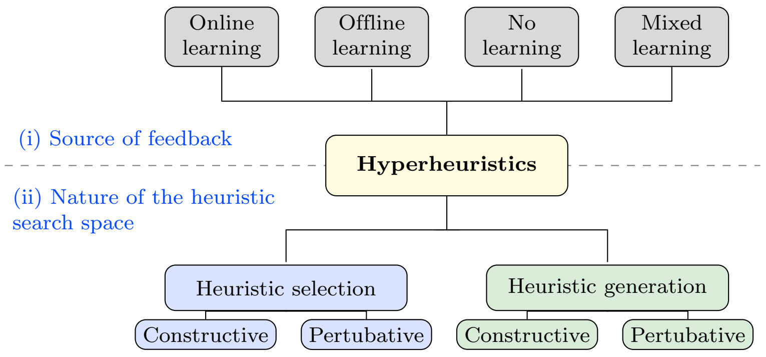


Figure 2.11: *Classification of hyperheuristic approaches according to two dimensions: (i) the source of feedback during learning, and (ii) the nature of a heuristic search space, adapted from [38].*

MO selection hyperheuristics can either manage multiple MO metaheuristics [163, 164, 173] within a single execution, for example MOEAs, or manage the components of a single MO metaheuristic such as the *crossover* or *mutation operators* [108]. To conclude, a set of LLHs can either be a set of metaheuristics or a set of metaheuristic components. A selection hyperheuristic consists of two components: *heuristic selection* and *move acceptance* [29, 199]. In heuristic selection, given a set of perturbative LLHs, a strategy is used to *select* the most appropriate LLH to run at a certain point in the search. Thereafter, the perturbed solution(s) is either accepted or rejected based on the move acceptance strategy.

The nature of the move acceptance strategy for selection hyperheuristics is classified as *stochastic* if the accept (or reject) decision is determined with some probability, for example SA, or *deterministic*, otherwise [38]. It is important to note that improving moves are always accepted, the move acceptance strategy determines whether or not *worsening* moves are accepted [142]. Tables 2.3 and 2.2 shows some of the different strategies for move acceptance and heuristic selection, respectively, note that this is not a taxonomy of all the strategies. When applying selection hyperheuristics to population-based methods, it is worth noting that the move acceptance strategy often becomes a replacement strategy [37, 38].

Note that it is possible to apply more than one heuristic selection and or move acceptance strategy, however only one Heuristic Selection–Move Acceptance pair is considered in this paper. Selection hyperheuristics are described in terms of these two components, allowing for a high level of modularity, *i.e.* when either component is replaced by another, a new hyperheuristic is created. An instance of this will be denoted as Heuristic Selection–Move Acceptance from this point forward. A plethora of different *Heuristic Selection–Move Acceptance* combinations have been explored within the context of perturbative selection hyperheuristics, refer to [36].

It is advised to introduce some learning into the heuristic selection process, to select the next LLH intelligently and thereby aid the decision-making process [36]. Note that two approaches are available when employing perturbative selection hyperheuristics, namely single-solution based search or population-based search. The majority of perturbative selection hyperheuristic research covers single-solution based search methods [51, 108, 201]. However, research is growing for hyperheuristics using population-based search methods [76] and mixed approaches, *i.e.* using single-solution and population-based search methods [126, 161].

Table 2.2: *Move acceptance strategies in perturbation selection hyperheuristics.*

Deterministic methods	Description	References
All moves	Accepts all moves.	[50, 51]
Only improving	Only accepts improving moves.	[50, 51]
Improving or equal	Accepts improving or equal moves.	[50, 51]
Threshold acceptance	Accepts non-improving moves that are less than a prefixed threshold, in terms of solution quality.	[29, 138, 182]
Late acceptance	The new solution is compared to the L^{th} solution step and the decision to accept is made accordingly.	[35, 200]
Stochastic methods	**Description**	**References**
Monte Carlo	Accepts non-improving moves with a probability that decreases in relation to the decrease in objective function value.	[9]
Great Deluge	Accepts a move as long as it is no worse than an expected objective value that changes linearly at each step.	[29, 138]
Tabu search	Accepts the move if it is not on the tabu list.	[42]
Simulated annealing	Accepts non-improving moves with a probability, according to the Metropolis rule.	[12, 13, 29, 74]

The majority selection hyperheuristics generate an online score based on each LLHs performance, and based on the selection strategy chosen, the scores are processed and/or combined in a systematic manner to select the heuristic to be applied to the candidate solution or population of solutions at each step. A *score-based* hyperheuristic framework comprises of the following components: (i) an initial score, (ii) memory length adjustment, (iii) heuristic selection strategy based on the scores, (iv) score update rules in case of improvement of solution quality and (v) score update rules in case of degradation of solution quality, respectively [37].

The first component, the *initial score* is assigned to all the LLHs, the initial score is typically zero. The *memory length adjustment* controls how the previous LLHs performance influences the selection strategy. Note that the choice of heuristic selection strategy may directly affect the performance of the hyperheuristic [37]. For example, the *max* heuristic selection strategy selects the heuristic with the maximum score, the *Roulette wheel* selects an LLH based on its probability which corresponds to its performance [36]. The *choice function* (CF) is an example of the max strategy and maintains a utility score for each LLH, the heuristic with the maximum score is selected at each iteration. Another example and commonly used method is *reinforcement learning* (RL), for more detail, refer to [132, 242].

Table 2.3: *Heuristic selection strategies in perturbation selection hyperheuristics.*

Max strategy	Description	References
Greedy	Exhaustively applies all LLHs and selects the LLH that produced the best solution.	[50, 51]
Choice function	Adaptively ranks the LLHs with respect to a combined score on its individual performance and selects the LLH with the best score.	[50, 51]
Roulette-wheel strategy	**Description**	**References**
Simple random	Randomly selects a LLH at each step.	[50, 51]
Random descent	A LLH is selected at random and applied until no improvement is made.	[50, 51]
Random permutation	Generates a random order in which a LLH is selected at each step.	[50, 51]
Random permutation descent	Generates a random order in which a LLH is selected but does so when no improvement is made.	[50, 51]
Reinforcement learning	Tries to learn which LLH to select at which step and is based on scores or cumulative reward.	[40, 74, 183]

RL heuristic selection selects an LLH by method of trial-and-error, the system attempts to learn which actions result in reward and which actions result in penalty. To elaborate, if an LLH improves a non-dominated solution, then it is rewarded and its score increases, otherwise it is punished and its score decreases [36]. Empirical evidence shows that different combinations of Heuristic Selection–Move Acceptance components yield different performances [29, 199]. The results show that All Moves (AM) perform the worst, regardless of the selection strategy. Experiments have shown that the move acceptance strategy significantly affects the performance of the hyperheuristic as compared with heuristic selection. The most used and successfully implemented heuristic selection methods include the choice function and the reinforcement learning variants. For move acceptance methods, the best is SA and great deluge variants. Experiments have shown that the choice of move acceptance methods significantly affects the performance of the hyperheuristics, more so than the choice of heuristic selection methods [142].

Different frameworks for population-based searches have also been proposed, for example *a multi-algorithm genetically adaptive multi-objective* (AMALGAM) approach [254]. AMALGAM applies a set of LLHs simultaneously, where the number of solutions that each LLH may produce is proportional to its individual past performance, *i.e.* the percentage of previously created solutions that are non-dominated [38, 76]. An enhanced AMALGAM method was proposed for population evolution during the optimisation process [140]. Later AMALGAM was applied to single-solution based searches, proposed in [128]. So far, this chapter has discussed simulation and its optimisation approaches, MOO preliminaries, metaheuristics and hyperheuristics. The next section discusses artificial neural networks for the purpose of simulation metamodeling.

2.5 Simulation metamodeling

The DES models studied in this paper are complex, *i.e.* they have multiple sources of *randomness* that require substantial computational effort to simulate. The computational inefficiency of simulation motivated the development of *metamodels* in order to enhance the overall effectiveness of the simulation optimisation process [5]. Using metamodels in combination with simulation reduces the computationally expensive (time-intensive) nature of simulation by searching the solution space more *intelligently* and less *exhaustively* [5]. The computational time (or simulation time) required by a simulation model depends greatly on the complexity of the model and may range from minutes to hours and even weeks.

The more complex the system, the harder it is to model, resulting in lengthy computational requirements [248], which is particularly true for complex COPs [18, 248, 258]. Despite the potential of SO, the excessive computational time required is a limiting factor, and reducing the model complexity is not an option. Because even if an optimal solution is found, it is for a model that does do not adequately represent the real system [156]. To this end, metamodels are proposed to aid in the simulation optimisation procedure and reduce the computational time required, by acting as a filter for potentially *low-quality* solutions, thereby mitigating computationally expensive evaluations of the simulation model [5, 156].

As discussed previously, and as seen in Figure 2.3, metamodels require less computational time than simulations because they are deterministic rather than stochastic and can be used in combination with simulation so as to leverage the trade-off between the time, cost, risk and quality [109], thereby simplifying the simulation optimisation process. Simulation metamodels offer a trade-off between *accuracy* and *efficiency*, by trying to predict the *input-output* relationships inherent to the simulation model, thereby offering significant advantages regarding computational efficiency [87]. For an example, see the development of the OptQuest tool [99].

Metamodels can provide a *fast* decision support that aids the overall effectiveness of the simulation optimisation process [109, 219]. Note that the inputs (or decision variables) of the simulation model are also the inputs of the metamodel, and the outputs (or objective function values) of the simulation model are used to train the metamodel [109]. There are several techniques for simulation metamodelling, each having advantages, disadvantages and applicable domains [18, 248]. The most frequently used techniques in recent literatures include *artificial neural networks* (ANNs) [86, 87], *genetic programming* (GP), *multivariate adaptive regression splines* (MARS) [88], *spatial correlation* or *Kriging* (KG) [144, 145, 146, 220], *radial basis functions* (RBF) [178, 190, 230] and *support vector regression* (SVR) [44].

Publications combining ANNs and simulation are on the rise, the reader is referred to [41, 86, 155, 191, 248], either for simulation metamodeling purposes [248] or as part of the simulation optimisation process which is part of simulation software such as *OptQuest*, *Optimiz* and *Simul8* [3, 5], however not yet in *Tecnomatix* [16]. The next section discusses the pertinent literature related to the fundamentals of ANNs, mainly to introduce concepts and the notations used throughout this paper.

2.5.1 The fundamentals of artificial neural networks

ANNs are mathematical models inspired by the generally accepted notion of how the brain *learns from experience* by *processing information* enabling machines to learn in an automated fashion and is the fundamental underpinning of the domain of *machine learning* (ML) [106]. To elaborate, ANNs learn from training data which consist of a set of input values and, in the case of supervised learning, its corresponding output values. The network is able to map the input-output relationship intrinsic to the data and does so in an iterated manner. Mitchell *et al.* [184] provides a succinct definition:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

Learning from experience is defined as the method of attaining the ability to perform a task, and can be accomplished in a *supervised*, *semi-supervised*, *unsupervised* or *reinforced* manner, depending on the experience E.

Unsupervised learning requires a model to learn from inputs what the underlying function of the dataset is without having the corresponding outputs to compare to (or learn from). Semi-supervised learning combines supervised and unsupervised learning, *i.e.* some of the inputs include a corresponding output to learn from while others do not. Reinforcement learning (RL) is different to the other machine learning paradigms as it follows a trail-and-error search approach and delays reward. RL attempts to learn what the best action is that will maximise the cumulative reward, the reader is referred to [242].

The two main supervised learning tasks that ANNs can perform are *classification* and *regression* and some applications include *clustering*, *function approximation* and *optimisation* [179]. Given some input, a classification task would predict the category to which some input belongs, whereas a regression task would predict some numerical value [106]. To evaluate the performance of an ML algorithm, a quantitative measure of its performance P is required and is dependent on the task T being performed and is discussed later. This study focusses on function approximation for regression tasks. The focus of this study is on *regression tasks* T and *supervised learning experiences* E using ANNs.

ANNs are generally classified into three main categories, exhibiting noteworthy differences, *recurrent neural networks* (RNNs), *convolutional neural networks* (CNNs) and *Feedforward neural networks* (FNNs). RNNs use memory in the form of feedback connections, where information is transmitted to and from preceding layers, that enables the network to process and analyse sequential data and perform tasks such as speech recognition with exceptional performance [262]. CNNs perform tasks such as pattern-and-image classification by taking as input an image and encoding specific features into the network architecture that allows the network to learn from the pixels and make predictions [81, 106]. For a systematic introduction see [211]. FNNs (or multi-layer perceptrons (MLPs)) allow data to be transmited thought the network, strictly in a forward direction, *i.e.* no feedback connections exist [106].

The three basic elements of an ANN are illustrated by the perceptron in Figure 2.12 [153]:

1. A set of *weighted* connections, also called *synapses*. Specifically, when referring to Figure 2.12, input signal x_i is connected (or multiplied) by weighted connection w_i.

2. The weighted sum z, sums the input signals multiplied by their respective weighted connections and adding a bias, denoted by

$$z = \sum_{i=1}^{n} x_i w_i + b, \tag{2.8}$$

   where b is the bias associated with each neuron and n is the number of inputs. The purpose of incorporating this bias value is to shift the activation function either to the left or right, adjusting the neuron's inherent threshold.

3. An activation function, denoted by $\sigma(z)$, mathematically models the firing process of a biological neuron, *i.e.* a node only relays information if the weighted sum of that node exceeds some inherent threshold, called the activation of the node.

A biological brain comprises of a network of *biological neurons*, therefore neurons are the core building blocks of ANNs as they try to mimic how the brain functions by controlling a network of artificial neurons, referred to as a node from this point on [153]. As shown in Figure 2.12, a neuron receives inputs $\{x_1, \ldots, x_i, \ldots, x_n\}$, weighted with $\{w_1, \ldots, w_i, \ldots, w_n\}$ and a bias b, respectively. The sum of the weighted connections $z = \sum_{i=1}^{n} x_i w_i + b$ then produces the predicted output $\hat{y} = \sigma(z)$ of the neuron, according to the activation function σ.

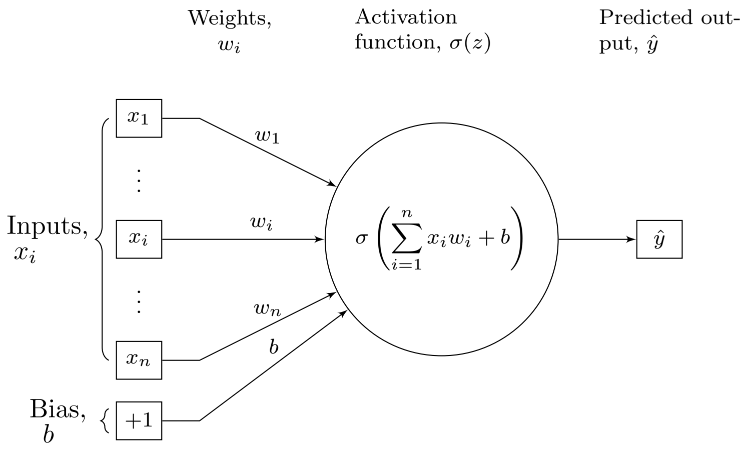

Weights, w_i
Activation function, $\sigma(z)$
Predicted output, $\hat{y}$
Inputs, x_i
x_1
x_i
x_n
Bias, b
$+1$
w_1
w_i
w_n
b
$\sigma\left(\sum_{i=1}^{n} x_i w_i + b\right)$
$\hat{y}$

Figure 2.12: *Mathematical model of an artificial neuron and its processing capabilities.*

The output represents the transmitted signal strength of the neuron which is influenced by the weighted connections and bias. Unfortunately, a perceptron is unable to learn from data that are not linearly separable, which is the case for the logical operation exclusive or (XOR). However, two perceptrons combined could. ANNs may be defined by the topology in which the neurons are connected, termed the *network architecture*, the training algorithm used and activation function applied [84]. These components and other relevant information are discussed in the following sections.

2.5.2 The network architecture

In terms of network architecture, an ANNs can be classified as a *single-layer* or *multi-layer network*. A neural network may be described in terms of layers, where the layers consist of neurons, *i.e.* neurons are partitioned into different layers as depicted in Figure 2.13, the first layer is called the *input layer* and the neurons in that layer input neurons, the last layer is called the *output layer* and the neurons output neurons. The layers in between are called *hidden layers* and the neurons hidden neurons. Typically, the same activation function is applied to neurons in the same layer.

Single-layer networks, similar to the network presented in Figure 2.12, do not have hidden layers. Multi-layer networks consist of an input layer, at least one hidden layer and an output layer and is capable of learning in respect of linearly separable data [84]. An ANN with more than one hidden layer is referred to as a *deep neural network* (DNN) [192]. Note that there are other conventions being followed, as seen in [192].

In FNNs, signals (or information) are propagated from one neuron to the next in a forward direction through the network layers, starting at the input layer moving through the hidden layers towards the output layer [84]. Consequently, the architecture of a *fully-connected* (or interconnected) ANN is explained using an FNN with one hidden layer, as depicted in Figure 2.13. The network has n input neurons each representing an input, h hidden neurons with a bias term in each layer and m output neurons each representing an output. The bracketed superscripts denote the relevant layers, adopting the convention that the input layer is layer one, hidden layer is layer two and output layer is layer 3. It should be noted that the neurons from one layer are connected to the neurons in the subsequent layer, however, neurons in the same layer are not connected to one another.

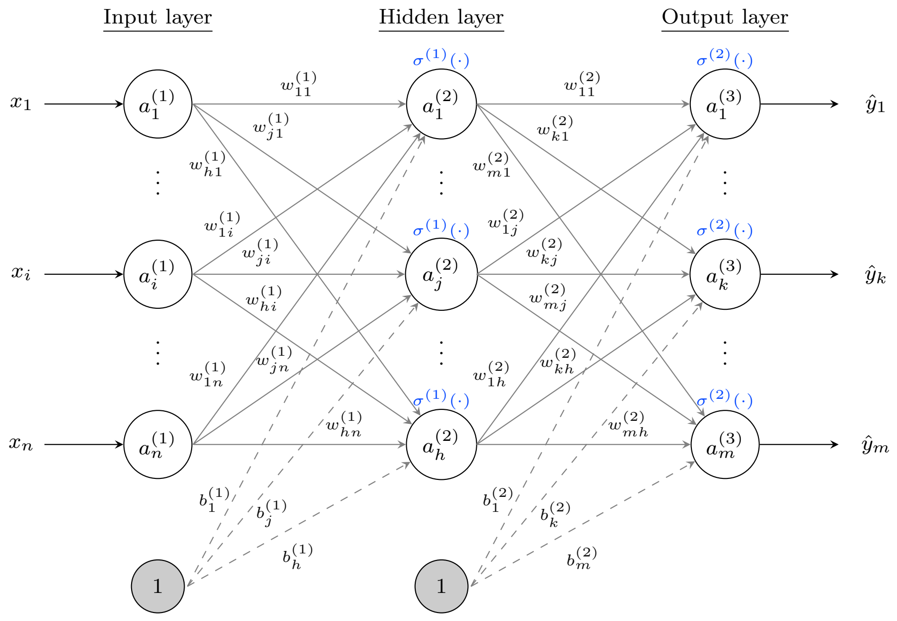


Figure 2.13: *Illustration of the notation used for a FNN with one hidden layer.*

Furthermore, Figure 2.13 demonstrates the notation adopted throughout this study. The weights are numbered sequentially, starting at the first neuron in the input layer to the first hidden neuron in the hidden layer. Specifically, let $w_{ji}^{(1)}$ denote the weighted connection associated with the input neuron i and the hidden neuron j, and let $b_j^{(1)}$ denote the bias for hidden neuron j. Similarly, let $w_{kj}^{(2)}$ denote the weighted connection associated with the hidden neuron j and output neuron k, and let $b_k^{(2)}$ denote the bias for output neuron k. The biases are represented as adjustable weights from additional inputs with a value of one [30].

The main components of the FNN with one hidden layer presented in Figure 2.13 can be summarised as:

- Input layer activations, denoted by $\boldsymbol{a}^{(1)} = [a_1^{(1)}, \ldots, a_i^{(1)}, \ldots, a_n^{(1)}]^T$,
- hidden layer activations, denoted by $\boldsymbol{a}^{(2)} = [a_1^{(2)}, \ldots, a_j^{(2)}, \ldots, a_h^{(2)}]^T$,
- output layer activations, denoted by $\boldsymbol{a}^{(3)} = [a_1^{(3)}, \ldots, a_k^{(3)}, \ldots, a_m^{(3)}]^T$,
- weights corresponding to the connection between input neuron $i \in \{1, \ldots, n\}$ and hidden neuron $j \in \{1, \ldots, h\}$, denoted by $w_{ji}^{(1)}$ and contained within a weight-matrix $\boldsymbol{W}^{(1)}$ and bias denoted by $b_j^{(1)}$,
- weights corresponding to the connection between hidden neuron $j \in \{1, \ldots, h\}$ and output neuron $k \in \{1, \ldots, m\}$, denoted by $w_{kj}^{(2)}$ and contained within a weight-matrix $\boldsymbol{W}^{(2)}$ and bias denoted by $b_k^{(2)}$,
- input layer activation functions are excluded as the identity function is used, the hidden layer activation functions are denoted by $\sigma^{(1)}(\cdot)$ and the output layer activation functions are denoted by $\sigma^{(2)}(\cdot)$.

The process followed to determine the predicted output $\hat{y}_k$ is termed *forward propagation.* Given the aforementioned information, the FNN with one hidden layer can be expressed analytically in terms of a mathematical function. Recall that the activation function used in the input layer is the identity function, therefore $\sigma(x_i) = a_i^{(1)} = 1(x_i)$, *i.e.* the input signal is simply transmitted onwards, as is. The input layer activations are subsequently used to calculate the net input (or weighted sum) to hidden neuron j, denoted by $z(j)^{(2)}$ for $j \in \{1, \dots, h\}$.

The net input is given by

$$z_j^{(2)} = \sum_{i=1}^{n} w_{ji}^{(1)} a_i^{(1)} + bj^{(1)}. \tag{2.9}$$

The matrix expression that defines the net input for all the hidden nodes may be written as

$$\begin{bmatrix} w_{11}^{(1)} & w_{1i}^{(1)} & \dots & w_{1n}^{(1)} \\ w_{j1}^{(1)} & w_{ji}^{(1)} & \dots & w_{hi}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{h1}^{(1)} & w_{hi}^{(1)} & \dots & w_{hn}^{(1)} \end{bmatrix} \odot \begin{bmatrix} a_1^{(1)} \\ a_i^{(1)} \\ \vdots \\ a_n^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_j^{(1)} \\ \vdots \\ b_h^{(1)} \end{bmatrix} = \begin{bmatrix} z_1^{(2)} \\ z_j^{(2)} \\ \vdots \\ z_h^{(2)} \end{bmatrix},$$

where $\odot$ is the *Hadamard product* (or element-wise multiplication). Next, the activation of hidden neuron j can be calculated by applying the activation function $\sigma^{(1)}(\cdot)$ to the weighted sum calculated in (2.9), such that

$$a_j^{(2)} = a\left(z_j^{(2)}\right) = \sigma^{(1)}\left(z_j^{(2)}\right). \tag{2.10}$$

The matrix expression that defines the activation for all the hidden nodes may be written as

$$\sigma^{(1)}\begin{pmatrix} z_1^{(2)} \\ z_j^{(2)} \\ \vdots \\ z_h^{(2)} \end{pmatrix} = \begin{bmatrix} a_1^{(2)} \\ a_j^{(2)} \\ \vdots \\ a_h^{(2)} \end{bmatrix}.$$

The calculations for calculating the weighted sum and activation of output node k is similar to that of hidden neuron j, and is given as

$$z_k^{(3)} = \sum_{j=1}^{h} w_{kj}^{(2)} a_j^{(2)} + b_k^{(2)}, \tag{2.11}$$

where h is the number of hidden nodes. The matrix expression that defines the weighted sum for all the output nodes may be written as

$$\begin{bmatrix} w_{11}^{(2)} & w_{1j}^{(2)} & \dots & w_{1n}^{(2)} \\ w_{k1}^{(2)} & w_{kj}^{(2)} & \dots & w_{kn}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}^{(2)} & w_{mj}^{(2)} & \dots & w_{mn}^{(2)} \end{bmatrix} \odot \begin{bmatrix} a_1^{(2)} \\ a_j^{(2)} \\ \vdots \\ a_h^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_k^{(2)} \\ \vdots \\ b_m^{(2)} \end{bmatrix} = \begin{bmatrix} z_1^{(3)} \\ z_k^{(3)} \\ \vdots \\ z_m^{(3)} \end{bmatrix}.$$

Next, the activation of output neuron k can be calculated by applying the activation function $\sigma^{(2)}(\cdot)$ to the weighted sum calculated in (2.11), such that

$$a_k^{(3)} = a\left(z_k^{(3)}\right) = \sigma^{(2)}\left(z_k^{(3)}\right), \tag{2.12}$$

where $a_k^{(3)}$ represents the output $\hat{y}_k$. The matrix expression that defines the activation for all the output nodes may be written as

$$\sigma^{(2)}\begin{pmatrix} z_1^{(3)} \\ z_k^{(3)} \\ \vdots \\ z_m^{(3)} \end{pmatrix} = \begin{bmatrix} a_1^{(3)} \\ a_k^{(3)} \\ \vdots \\ a_m^{(3)} \end{bmatrix}.$$

The FNN is simply a non-linear function which is controlled by adjusting the network weights and biases [30]. The weights and biases that represent the underlying input-output relationship of the data are determined by a *training* (or *learning algorithm*) according to which network weights and biases are updated in an iterative and algorithmic fashion.

2.5.3 The network training algorithms

Recall that supervised learning is adopted in this study, *i.e.* the ANN is presented with input–output pairs from which the network adjusts its weights and biases so as to best approximate the functional mapping from inputs to outputs. There are three approaches that may be adopted to train an ANN, namely *first-order optimisation algorithms*, *second-order optimisation algorithms* and *metaheuristics* [106]. This section focuses on first-order algorithmic approaches as second-order approaches are rendered intractable as the problem complexity and cardinality increases [106]. More on this in Chapter 3. The backpropagation training algorithm introduced a computationally efficient method for minimising the the error function that measures the prediction error of the network. The derivatives of the error function with respect to the network weights and biases are used to determine what changes to the weights and biases would result in the largest decrease in network error [30, 83, 194]. There are four steps involved during backpropagation training of neural networks [83, 194]:

1. Forward propagate an input training vector (or training example) through the network, as shown in the previous section using (2.9)–(2.12).

2. Calculate the network errors using

$$E = \frac{1}{2}\sum_{k=1}^{m}(a_k - y_k)^2, \tag{2.13}$$

where m is the number of output nodes, and the network error is the sum of the individual output errors denoted by $E = \frac{1}{2}\sum_{k=1}^{m} E_k$. Recall that $a_k = \hat{y}_k$, *i.e.* the predicted output.

3. Determine the derivatives of the error function with respect to the network weights and biases.

4. Adjust the network weights and biases according to the chosen *learning rate*[2].

[2]A hyperparameter which controls the *step size* used to minimise the error function, *i.e.* speed of learning.

During forward propagation, a training example is transmitted through the network. The activation of the output layer is the predicted output for the given training example. Next, the network error may be calculated by comparing the predicted output (or response) a_k with the actual output (or target value) y_k using (2.13). Now the derivatives of the error function with respect to the network weights and biases can be calculated and the necessary adjustments to the weights and biases can be made. The derivation for the backpropagation algorithm is presented within the context of the FNN with one hidden layer, as depicted in Figure 2.13 in Appendix C.

The adjustments to the weights and biases can be performed by stochastic or batch update. Stochastic (or online) updates occur for each training example, whereas when batch updates are performed, the error accumulates and one adjustment is made after n training examples, where n represents the batch size. Stochastic updates have the advantage of escaping local minima, however batch updates enable the network to learn faster because the adjustments can typically be more considerable. The weights regulate how much an input influences the predicted output and the biases can have an influence on feature importance. The goal of training the network is to find the weights and biases that result in the predicted output matching the actual output [134, 188].

When training a neural network, only a percentage of the entire dataset is used, typically 80% of the dataset which is commonly referred to as the *training set*. After training the network, the remaining 20% of the dataset is used to measure the network performance, which is called the *test set*. When the network exhibits good performance when measured on the training set, but poor performance when measured on the testing set, the network is deemed to overfit the data, *i.e.* it memorises well but generalises poorly [30]. Consequently, performance with respect to the test set is of paramount importance, *i.e.* how well the network was able to generalise to new (or unseen) data. Note that the performance measure used depends on the task, in this case regression. This matter is discussed in detail in Chapter 3.

It is also considered good practice to have a *validation set* which is used for regularisation purposes and to adjust certain parameters of the network, called the hyperparameters (*e.g.* learning rate). Accordingly, the dataset is partitioned into the training set, validation set and test set, typically with a proportion of 60%:20%:20%, as seen in Figure 2.14. Note that it is also common practice to partition the dataset into only a training and test set and then partition the training dataset into a training and validation set, *i.e.* a 80%:20% partition and then the 80% is split again to form 60%:20%, *i.e.* a quarter of the training dataset becomes the validation set [34, 154, 212, 218, 227].

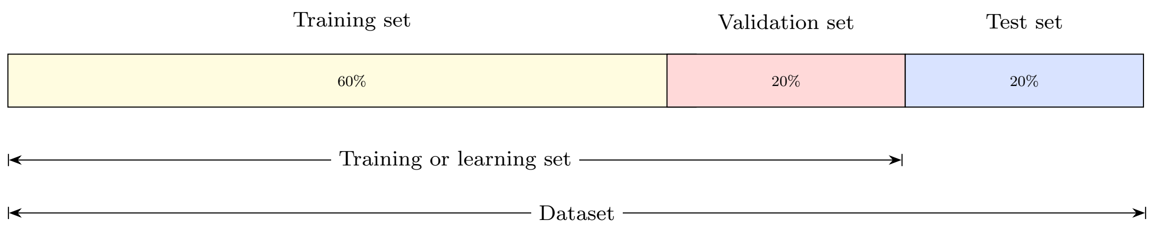


Figure 2.14: *Illustration of a dataset partitioned into a training, validation and test set respectively, adapted from [227].*

It should be noted that during training, only the parameters, *i.e.* the weights and biases, are typically optimised, not the hyperparameters. Hyperparameters may be adjusted and has been reported to considerably influence the performance of the network [30].

The hyperparameters typically include the number of hidden neurons per hidden layer, the number of hidden layers, the activation function used in the hidden layers and output layer, the training algorithm and its learning rate, see Chapter 3. According to Fausett [83], when using a gradient-based training algorithm, the activation function should be *continuous* and *differentiable*, this is because the derivative of the activation function is required to determine the partial derivatives of the activation of a neuron with respect to the weighted sum of the neuron which in turn is required to determine the derivative of the error function with respect to the weights and biases. Some of the activation functions used in literature are discussed next.

2.5.4 Activation functions

The choice of activation function (or *transfer function*) is a critical part of the network architecture design, this is because the activation function can greatly affect the performance of the network. Different activation functions can be used in different layers. A caveat when choosing an activation function for both the hidden and output layers. When choosing an activation function for the hidden layer, the nature of the input variables has to be considered, since it controls the networks ability to learn from the inputs. The choice of activation function for the output layer depends on the nature of the output variables, since it determines what predictions (of the outputs) the network can make [83].

Activation functions essentially model the firing process of biological neurons mathematically, which can be a *linear* or *non-linear* function. However, non-linear functions exhibit the same level of functionality in a network with a single hidden layer as linear activation functions do in a network with multiple hidden layers [192]. Moreover, the non-linear nature of the activation functions is what empowers neural networks with respect to their nonlinear capabilities [159]. As such, non-linear activation functions are discussed in this section and will include the form, mathematical expression, the advantages and disadvantages, as well as suitable applications of each. The two simplest activation functions are the *identity* and the *heavside* (or *binary step*) function, as shown in Figure 2.15a and 2.15b, respectively. The mathematical expression for the identity function is given as $\sigma(z) = 1(z)$, for all weighted sums z. The heaveside function is given by

$$\sigma(z) = \begin{cases} 0, & z < 0, \\ 1, & z \geq 0. \end{cases} \tag{2.14}$$

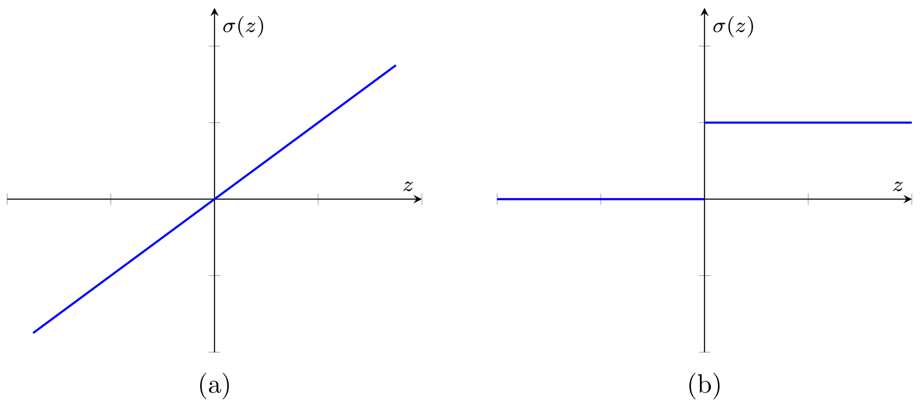


Figure 2.15: *Illustration of the two simplest activation functions (a) the identity function and (b) the heavside function, as given in* (2.14).

Recall, that the activation used in the output layer is dependent on the data type of the output variable that the model must predict. Similar to the activation function chosen for the output layer being dependent on the data type of the output variable, so is the activation function chosen for the hidden layers [195]. The nature of the inputs and outputs must be considered. For example, for binary inputs, an appropriate activation function could the *logistic sigmoid* or the *hyperbolic tangent* function [83].

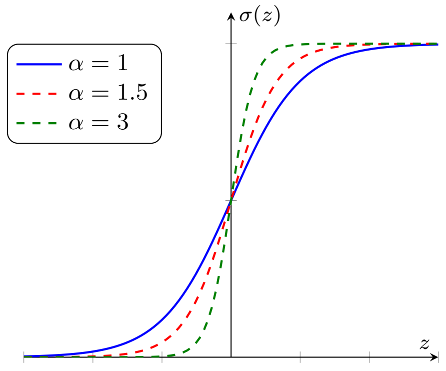
The sigmoid function can be seen in Figure 2.16a, where the logistic sigmoid function has a slope of $\alpha = 1$. These functions take any real number and map it to a probability between 0 and 1 depending on the threshold chosen. Suppose the threshold is set to 0.5, then a probability smaller than 0.5 would be mapped to the outcome 0 (*i.e.* false), otherwise the outcome would be 1 (*i.e.* true). The sigmoid function is one of the most common non-linear activation functions [81, 106], given by (2.15). The slope parameter is a *hyperparameter*, and may be adjusted to influence the performance of the network, or may be learnt in a trail-and-error manner. Figure 2.16 illustrates the sigmoid function for $\alpha = \{1,\ 1.5,\ 3\}$,

$$\sigma(z) \ = \ \frac{1}{1+e^{-\alpha z}}, \tag{2.15}$$

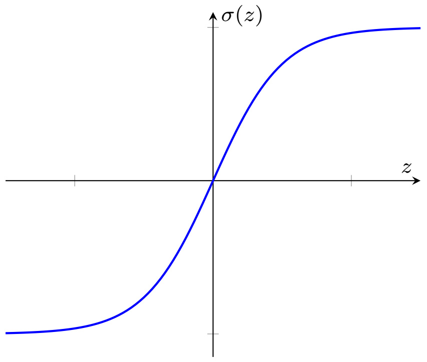
and its derivative is given by

$$\sigma'(z) \ = \ \alpha\left(\frac{1}{1+e^{-\alpha z}}\right)\left(1 - \frac{1}{1+e^{-\alpha z}}\right), \tag{2.16}$$

which can be written as $\sigma'(z) = \ \alpha\ (\sigma(z)(1-\sigma(z)))$ for the sake of brevity. It is clear by the 'S' shape of the *hyperbolic tangent function* that it is a member of the class of sigmoid functions, illustrated in Figure 2.16b. Unfortunately, both functions in Figure 2.16 are insensitive to small changes in the inputs and saturate when the network weights are either too large or too small. The result is slow gradient descent, *i.e.* slow learning [116].



(a) *The sigmoid function*



(b) *The hyperbolic tangent function*

Figure 2.16: *The class of sigmoid activation functions.*

The mathematical expression of the hyperbolic tangent function is given by

$$\sigma(z) \ = \ \tanh(z), \tag{2.17}$$

and its derivative is given by

$$\sigma'(z) \ = \ (1+\tanh(z))(1-\tanh(z)), \tag{2.18}$$

which can be written as $\sigma'(z) = (1+\sigma(z))(1-\sigma(z))$, for the sake of brevity.

It has been reported that the performance of FNNs can be improved by employing piecewise linear activation functions, such as rectified linear units (ReLUs), in the hidden layers rather than sigmoid or hyperbolic tangent functions [106]. ReLU activation function was first proposed for *restricted Boltzmann machines* (RBMs) [81, 189] and then successfully applied to neural networks in [96]. It can be seen that the ReLU activation function is a combination of Figure 2.15b (for negative values) and Figure 2.15a (for positive values) and is reported to perform better than the sigmoid and hyperbolic tangent functions [151].

The application of the ReLUs in DNNs have been reported to deliver promising results [96], however because all negative values are zero, neurons may be made inactive prematurely. This phenomenon is referred to as the '*dying ReLU*' problem. The parametric ReLU (PReLU), *leaky* ReLU (LReLU) and the *exponential linear unit* (ELU) or Scaled ELU (SELU) are alternatives that permit small negative values by having some slope in the negative region, thereby preventing the issue [172]. Figure 2.17 is the graphical representation of the different activation functions, and are discussed accordingly.

PReLU has a hyperparameter *alpha* (α) that controls the shape of the negative region of the function and can be trained and improve the networks performance. PReLU function achieves a notable performance advantage in terms of speed of convergence of the gradient-descent training algorithm when compared with the sigmoid, hyperbolic tangent functions[135], and with negligible additional computational cost [112]. Note that ReLU is the PReLU function, where $\alpha = 0$. The mathematical expression of PReLU is given by

$$\sigma(z) = \begin{cases} \alpha z, & z < 0, \\ z, & z \geq 0. \end{cases} \tag{2.19}$$

LReLU is the PReLU function, where $\alpha = 0.01$, *i.e.* LReLU has a predetermined slope whereas PReLU learns the slope. LReLU replaces the negative region of the ReLU function with a linear function and has exhibited superior performance with respect to to ReLU [172]. ELU was first proposed by Clevert *et al.* [45], designed to combine the advantages of ReLU and LReLU — to avoid the so-called dying ReLU problem. ELU is similar to the LReLU, having a small slope for negative values, however an exponential function controls the negative region as opposed to a straight line. The mathematical expression of the ELU (or SELU) function is given by

$$\sigma(z) = \begin{cases} \alpha(e^z - 1), & z < 0, \\ z, & z \geq 0. \end{cases}$$

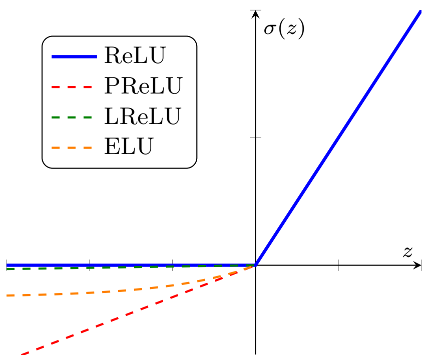


Figure 2.17: *The different ReLU activation functions, for* $\alpha \in \{0,\ 0.01,\ 0.25\}$, *and ELU.*

Given a sufficient number of hidden neurons in the hidden layer(s), MFNNs are considered universal function approximators, capable of approximating function arbitrarily [125].

Note that more hidden layers and/or hidden neurons do not directly translate to a better function fit. It does, however, result in a more complex network, a matter discussed in greater detail later. Supporting this, Fausett [84] suggests that one hidden layer is sufficient for most function approximations. Moreover, the universal approximation theorem [56, 124] support the use of FNNs as simulation metamodels, as it states that an FNN with a single hidden layer containing a finite number of neurons is sufficient to approximate any measurable function. However, the theorem does not give any information about the network parameterisation and requires a preliminary study to determine its feasibility. Intuitively, more training examples coincide with improved network performance, here overfitting is not a consideration since the motivation behind a metamodel is to reduce the computational time and therefore the goal is to build a metamodel with limited training and be able to approximate the simulation models with a certain level of accuracy [248].

2.6 Summary

This chapter contained a review of the most relevant and pertinent literature pertaining to simulation optimisation methods for MOO as well as ANNs as metamodels. The reader was presented with the necessary information to facilitate an understanding of the remainder of the research reported in this study.

First, the need for simulation optimisation was discussed as well as methods for doing so. Next, the MOO preliminaries required for solving MOOPs were documented. Metaheuristics and hyperheuristics were discussed next. Lastly, ANNs as metamodels were discussed as they are considered good candidates for metamodelling purposes, specifically FNN with one hidden layer. The next chapter reports the preliminary study conducted to determine the feasibility of an FNN with one hidden layer as a metamodel.

CHAPTER 3

ANN as Metamodel Pilot Study

The previous chapter introduced the fundamental concepts that will be used and build-upon throughout this study.

The main objective of this chapter is to determine whether or not an artificial neural network (ANN) as metamodel is a feasible solution to enhance the simulation optimisation process, in fulfilment of Objective 2 as stated in Chapter 1. The first section introduces the supervised learning regression models that are used as a benchmark for comparison. Next, the performance metrics used to train the regression models as well as to quantify their performances are discussed. Thereafter, the hyperparameters considered for the hyperparameter search space are discussed. The performance of the respective regression models are presented, followed by the performance results of the best hyperparameters for the ANN. Penultimately, the regression models (including the ANN) are compared and a conclusion is made whether or not an ANN as metamodel is feasible.

3.1 Machine learning models: Regression

It is necessary to compare the ANN with other regression models, specifically other supervised learning regression models, which include *linear regression* (LR) [2], *polynomial linear regression* (PLR) [113], *support vector regression* (SVR) [234], *decision tree regression* (DTR) [205], and *random forest regression* (RFR) [55, 168]. The following sections provide a high-level overview of the theory of each regressor.

3.1.1 Linear regression

Linear regression approximates the linear relationship between the independent variable (or decision variable) and the dependent variable (or objective variable). However, more than one independent variable is of interest, *i.e.* multiple linear regression (MLR) and is formulated as [2]

$$\hat{y} = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n, \tag{3.1}$$

where $\hat{y}$ is the dependent variable (to be predicted), β_0 is the y-intercept and $x_1, \ldots, x_n$ are the independent variables with their corresponding coefficients $\beta_1, \ldots, \beta_n$ that the regressor is solving for. Moreover, the coefficients (β_n) is the effect that x_n has on the predicted value y. For the sake of completeness, in this study, $\hat{y}$ is a vector of two elements, *i.e.* two objective variables each having their own linear relationship with the decision variables.

The goal of MLR, as visually presented in Figure 3.1, is to find the line that minimises the distance between the actual value y and the predicted value $\hat{y}$ [81], given by $\sum(y - \hat{y})^2$.

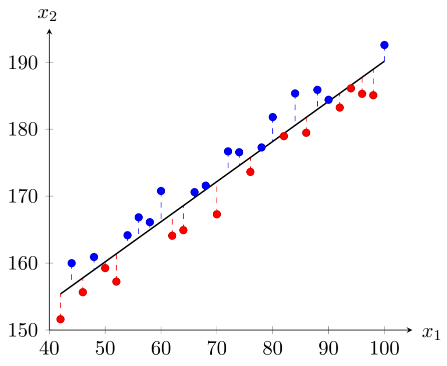


Figure 3.1: *An illustration of multiple linear regression.*

A caveat, however, is that the assumptions of linear regression which include linearity, homoscedasticity, multivariate normality, independence of errors and lack of multi-collinearity [2] are not guaranteed, the model will simply perform poorly if the relationships embedded within the data are non-linear [81]. Polynomial regression is discussed next.

3.1.2 Polynomial regression

Polynomial regression, a special case of linear regression, however, can approximate non-linear relationships by method of an n^{th} degree polynomial. The function $\hat{y}$ can be expressed as a linear combination of the coefficients. The n^{th} degree polynomial for one decision variable is given by [113]

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n, \tag{3.2}$$

where $\beta_0, \ldots, \beta_n$ represents the coefficients of x, not the x variable, that the regressor is solving for. Figure 3.2 illustrates how polynomial regression works by fitting a 1^{st}, 2^{nd} and 3^{rd} degree polynomial to the data.

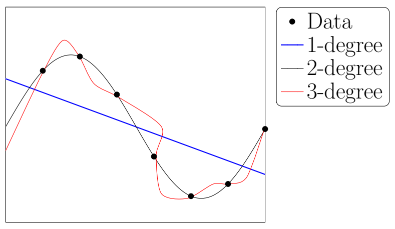


Figure 3.2: *Illustration of polynomial regression fitting data.*

3.1.3 Support vector regression

Support vector regression is based on statistical learning theory [251] used to find the decision boundary between groups of data. Future data is predicted based on where they are in terms of the boundary [234]. For a more in-depth overview of SVR, the reader is referred to [176, 223, 8]. The formulation for SVR is given by [49]

$$\frac{1}{2}||w||^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \rightarrow \min, \tag{3.3}$$

where $||w||^2$ is the $l2$-norm of the coefficient vector and C is a hyperparameter that determines the trade-off between the flatness of the function f and the tolerance for solutions outside ε that are allowed. The slack variables (ξ) contribute to the cost, if the solution is above the ε-insensitive tube, the solution is ξ otherwise, it is ξ^*.

Figure 3.3 depicts this graphically, only the points outside the tube, *i.e.* support vectors ξ_1^*, ξ_2, ξ_3^*, ξ_4^*, ξ_5, ξ_6^*, ξ_7, ξ_8^*, ξ_9, contribute to the cost and therefore determine how the tube is created and consequently the flexibility of the model.

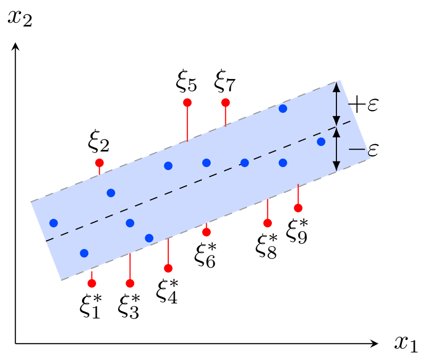


Figure 3.3: *An illustration of support vector regression, adapted from [223].*

For the sake of completeness, note that the boundary can only be found for linearly separable data, otherwise SVR has to be extended by either mapping the data to a higher dimension, thereby rendering the data linearly separable, or by applying a kernel to the data [81, 234].

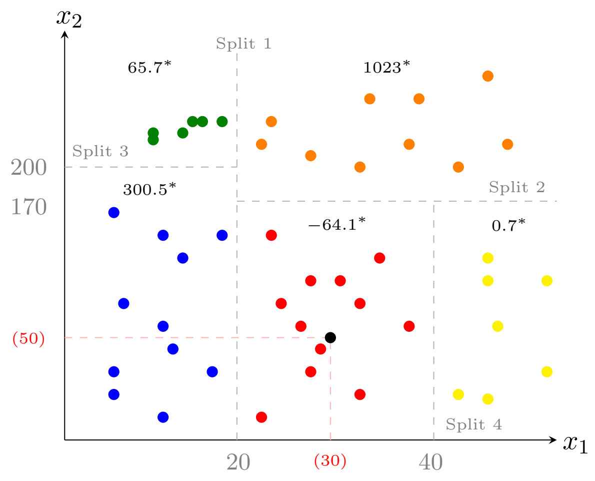
3.1.4 Decision tree regression

Decision tree regression performs predictions by learning decision rules inferred from the data. For example, in Figure 3.4, a dataset is divided into smaller subsets, which create the associated decision tree. The subsets are determined by splitting the data. The splits are determined by the information entropy, *i.e.* a split is made if it results in an increase in the amount of information about the dataset [205].

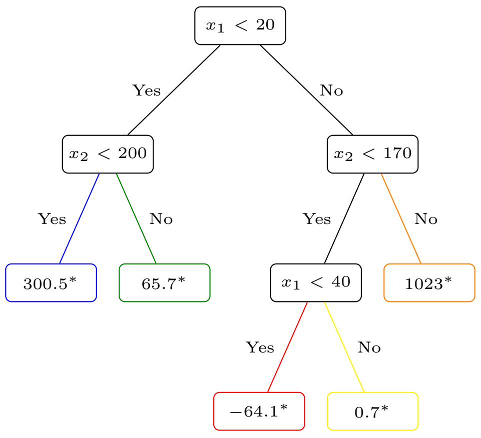
Figure 3.4a is used to aid in illustrating the working of decision tree regression. How and where the splits are made is determined by the algorithm, but for demonstration purposes assume the algorithm determined the splits as shown in Figure 3.4a.

Split 1 is x_1 at 20, splitting the data for the first time. Therefore, the first decision is $x_1 < 20$ as seen in Figure 3.4b. Next, the algorithm makes a second split, Split 2 x_2 at 170, creating the second decision, where $x_2 < 170$, and so on, until Split 4. In this example, Split 4 was the last split which means no more information could be added (or gained) by splitting a leaf again.

The final split resulted in five terminal leaves. The associated decision tree based on the splits is given in Figure 3.4b.



(a) *The terminal leaves or splits.*



(b) *The resulting decision tree.*

Figure 3.4: *An illustration to aid in explaining how decision tree regression works, adapted from [81].*

For example, the predicted value of y where $x_1 = 30$ and $x_2 = 50$ is determined as follows. It can be seen that the point $\{30, 50\}$ falls within the red terminal leaf depicted by the black data point. The predicted value for y is the average of the red terminal leaf, *i.e.* $y = -64.1$.

3.1.5 Random forest regression

Random forest regression applies ensemble learning[1] to regression trees, *i.e.* it creates a number of regression trees (N), and averages the prediction of all the trees. The y prediction is based on N trees (or a whole forest of trees), whereby improving the accuracy of the predictions because an average is taken of N trees (or predictions) [81]. The random forest regression algorithm is given in Algorithm 3.1.

Algorithm 3.1: Random Forest Regression [81]

```
   Input  : Training set and testing set, the number of trees you want to build, denoted by $N_{tree}$.
   Output : Predicts the output $y$ based on the input $x$
1  repeat
2      Pick at random $K$ data points from the training set;
3      Build the Decision Tree associated to these $K$ data points;
4  until Number of trees built = $N_{tree}$ ;
5  for New data point $x$ do
6      foreach Tree $N$ do
7          Predict $y_N$ for tree $N$
                   $x \leftarrow \frac{\sum_{i=1}^{N} y_i}{N}.$
```

This concludes the high-level overview of the regression models used in this study. It is sufficient to know how to use the algorithms at a high-level of abstraction, since each regressor was implemented in Python using existing libraries [81].

[1]When multiple algorithms or a single algorithm is used multiple times, to create a more powerful algorithm.

Additionally, because of its popularity, extreme gradient boosting (XGBoost) is also used for comparison purposes. XGBoost is an open-source implementation of the gradient boosted trees algorithm. For more detail the reader is referred to [43]. The next section discusses the performance measures that are used to train the regression models as well as quantify their performances.

3.2 Performance measures for assessing regression models

There are many performance metrics (within the context of regression problems) that describe a model's capability to learn from data (or so-called goodness of fit), some of which are discussed here [239].

Mean Squared Error (MSE) takes the square difference between the actual value and the predicted value. Therefore, when the difference is great, the squared difference is even greater, emphasising the error and can be expressed as [188]

$$\text{MSE} = \frac{1}{Q}\sum_{k=1}^{m}(y_k - \hat{y}_k)^2, \tag{3.4}$$

where Q is the number of training examples, y_k is the target output value, $\hat{y}_k$ the predicted value and m the number of outputs. MSE has been extended to *root Mean Square Error* (RMSE), which measures the standard deviation of the error between the predicted and the actual value, indicating the overall approximation capability of the regressor and can be expressed as [188]

$$\text{RMSE} = \sqrt{\frac{1}{Q}\sum_{k=1}^{m}(y_k - \hat{y}_k)^2}. \tag{3.5}$$

The smaller the RMSE, the better the approximation.

Mean Absolute Error (MAE) represents the average of the absolute differences between the predicted and actual value and can be expressed as [188]

$$\text{MAE} = \frac{1}{Q}\sum_{k=1}^{m}|y_k - \hat{y}_k|. \tag{3.6}$$

Coefficient of Determination (R^2) is a statistical measure used to explain the correlation between variables and can be expressed as

$$R^2 = 1 - \frac{\sum_{k=1}^{m}(\hat{y}_k - y_k)^2}{\sum_{k=1}^{m}(\bar{y}_k - y_k)^2}, \tag{3.7}$$

where $\bar{y}_k$ is the average. Note that R^2 will continue to increase as the number of parameters increase, therefore an adjusted R^2 is proposed which adds a penalty term to penalise complex models, because greater complexity does not necessarily translate to improved performance [188]. Adjusted R^2 is given by

$$R^2_{\text{adj}} = 1 - (1 - R^2)\frac{Q-1}{Q-n-1}, \tag{3.8}$$

where n is the number of decision variables. If $R^2 = 1$, the variables are perfectly correlated.

Each measure provides some indication of the prediction error made by the model and may be interpreted as the cost of predicting. If a batch learning approach is adopted, then the error is calculated across the Q input-output training examples that constitute the batch. If an online learning approach is followed, on the other hand, then the summation is omitted [188].

The next section discusses the hyperparameter optimisation performed to determine the best hyperparameter combination for the FNN with one hidden layer.

3.3 Hyperparameter optimisation

In the context of ANNs, hyperparameters are parameters of the neural network that are set prior to the training process, that is, prior to determining the weights and biases [33]. Determining the best hyperparameter combination, however, is not a trivial task and requires some form of optimisation. There are several hyperparameter optimisation techniques, Table 3.1 shows the high-level overview of the different techniques for sequential and parallel computing. Parallel algorithms are listed for the sake of completeness, however only sequential algorithms are considered. For in-depth discussions the reader is referred to [25, 26, 235]. Grid search exhaustively

Table 3.1: *A summary of hyperparameter optimisation algorithms.*

	Random	Adaptive	Evolutionary
Sequential	Grid search or random search	Bayesian optimisation	Genetic algorithm
Parallel	Asynchronous successive halving algorithm (ASHA)	Bayesian optimisation with hyperband (BOHB)	Population-based training

evaluates a model for each hyperparameter combination specified, whereas random search samples each hyperparameter combination from a distribution, over all possible parameter values, and is likely to discover better combinations (faster) than grid search due to its exploration of the parameter space [25]. Bayesian optimisation samples regions that result in a better network performance with a higher likelihood. Initially, Bayesian optimisation resembles random search, but as the search progresses the algorithm uses information inferred from past trials to guide the search. It has been shown that Bayesian optimisation is able to find better hyperparameter combinations than random search [26].

Consequently, Bayesian optimisation is used to determine the best hyperparameter combination to use for the FNN with one hidden layer. Figure 3.5a shows one function evaluation of what the actual function g is at that point, next a Gaussian process is fit to this one training example and the grey shaded area represents the approximate function of g for one observations, it should be clear that the more function evaluations are performed the more the approximated functions mirrors g. Next, an acquisition function is used to determine the next point to evaluate g, with a high probability of getting a lower function value (for a minimisation problem). It can be seen in Figure 3.5h that after only eight function evaluations, the Gaussian process is a relatively good approximation of g.

Due to the time-consuming process of selecting and evaluating hyperparameter combinations, with the goal of finding the best combination [33], the next section discusses the most appropriate design choices that are considered in the hyperparameter optimisation search space.

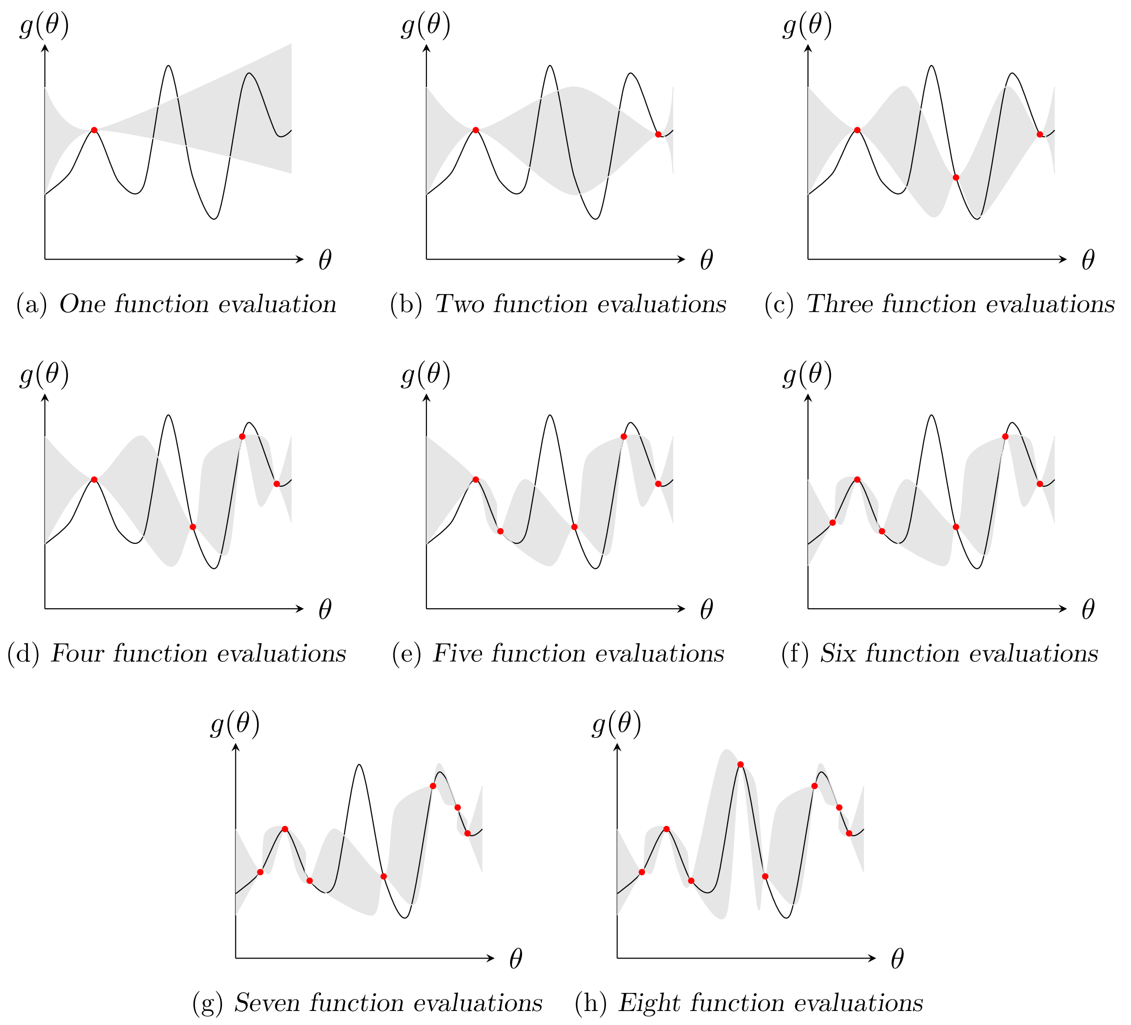


Figure 3.5: *An example illustrating how Bayesian optimisation works, by fitting a Gaussian process for each training example.*

3.3.1 ANN hyperparameters

The Nobel prize winning physicist *Enrico Fermi* said:

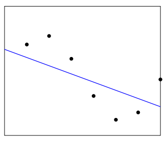
> "I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk."

The point is that models with many parameters might be able to fit to the training data, but overfitting does not make it a good model. For example, a 3^{rd} degree polynomial overfits to the data presented in Figure 3.2. Formally, the capability of the network to fit the training data is known as *memorisation*, which means that the model will be able to predict values for the existing data, but will fail to generalise to unseen data, known as *generalisation* [195].

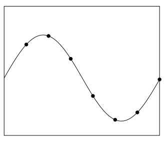
Therefore, the main goal of an ANN is to be able to generalise to unseen data, *i.e.* make predictions based on data it has not been trained on. Consequently, a trade-off exists between the competing abilities of memorisation and generalisation [153].

Network architecture and training epochs

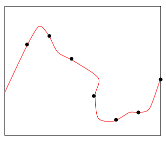
In Figure 3.6d and 3.6e, it can be seen that the network's *generalisation error* is the gap between the training error and validation error. The generalisation error is characterised into two terms: *bias* and *variance*. Bias measures the expected deviation between the predicted and the actual value, while variance measures how much the network predicted output varies between datasets. To elaborate, when a model performs poorly on the training set but well on the validation set (or test set), the model is biased. Alternatively, when a model fits the training set well, but not the validation set (or test set), the model is said to have high variance. Both cases are undesirable, because the result is the same: poor predictions in respect of out-of-sample data [30].



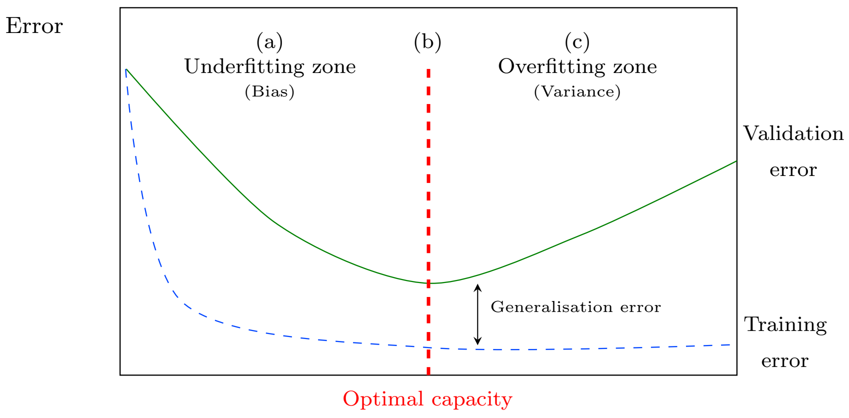
(a) *Underfitting*



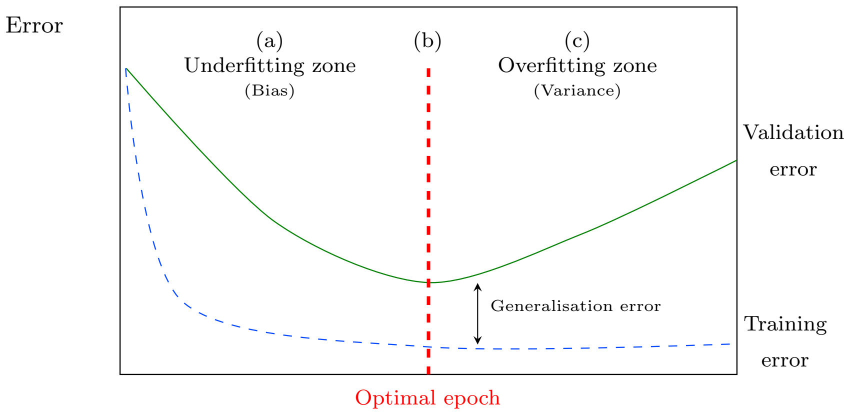
(b) *Good fit*



(c) *Overfitting*



(d) *An illustration of how the model's complexity influence the models generalisation capabilities*



(e) *An illustration of how the number of epochs influence the models generalisation capabilities*

Figure 3.6: *An illustration of the curve fitting phenomena that occurs when the ANN is too complex, or that is trained for too long. These may result in a learning model that either underfits or overfits the training set, resulting in large prediction errors, i.e. lack of generalisation in respect of the validation set, adapted from [222].*

The bias-variance tradeoff is strongly associated with the concepts of *capacity* (or complexity), *underfitting* and *overfitting*. Informally, models with low capacity (for example an ANN with a small number of hidden neurons) may struggle to fit to the training set, while models with high capacity (for example an ANN with a large number of hidden neurons) can overfit by memorising properties of the training set. Generally, increasing capacity and number of epochs that the network is trained for tends to increase variance and decrease bias [30]. It can be shown that the minimum network error is when the sum of the bias and variance are minimal, illustrated by the red dashed lines in Figure 3.6d and 3.6e for the network capacity and number of epochs trained respectively.

To elucidate this further, refer to Figure 3.6d. On the left side of the *optimal capacity* (red dashed line), the training and validation errors are high and the model is underfit (or biased). This is represented by the underfitting zone (a) in Figure 3.6d and the result is a poor function fit as seen in Figure 3.6a. However, as the capacity increases, the training error decreases, but the validation error increases and as a result, the generalisation error increases. On the right of the optimal capacity, the model is considered overfit (or has high variance). This is represented by the overfitting zone (c) in Figure 3.6d, also resulting in a poor function fit as seen in Figure 3.6c. The optimal capacity is where the smallest generalisation error exists, *i.e.* the point where the validation error starts increasing while the training error gradually decreases [30, 179].

Early on in the training process, the networks predictions are far from the actual output and therefore the bias is large, this can be seen in zone (a) of Figure 3.6e, the underfitting zone. As training progresses, the bias decreases as the network learns the underlying function, however, when the network is trained for too many epochs, the network may learn the noise present in the dataset it was trained on causing the variance to increase, represented by zone (c) of Figure 3.6e the overfitting zone [151]. From this, two important questions arise: During the construction of an FNN (with one hidden layer), how many neurons should be used and for how many epochs should the model be trained for? Where the goal is to create a model that can generalise well to out-of-sample data. It is suggested that the required number of neurons depends on many factors, namely, training set size, problem complexity and the generalisation techniques used to name but a few [244].

Deciding on the *right* number of hidden neurons is clearly problem dependent and requires expert knowledge to infer. Essentially, it must be determined empirically, serving as motivation for its inclusion in the ANNs hyperparameter optimisation. Next, the learning in literature and suggested rates are discussed.

Learning algorithms and learning rates

The ideal learning algorithm has low bias and can accurately model the underlying input-to-output relationship found by finding the trade-off between underfitting and overfitting. Gradient descent is the standard technique used for training ANNs. There are three variants of gradient descent, which are defined based on the amount of data used to compute the gradient of the error function and accordingly the frequency with which the weights and biases are updated. Again, a trade-off exists, between the accuracy of the weight and bias updates and the computational time required to perform the updates [194].

Figure 3.7 is used to elucidate this behaviour. Assume that there are 10 training examples in the training set. *Batch learning* is where the weights and biases are only updated after *all* or a *subset* of the training examples in the training set have been propagated through to the network [194]. When the whole training set is used, it is termed *batch gradient descent*, whereas, when a subset of training set is used, it is termed *mini-batch gradient descent*.

In Figure 3.7, the mini-batch size is two. During *online learning* the weights and biases are updated after each training example, termed *stochastic gradient descent* (SGD). It is termed stochastic since the examples are selected at random, rather than using the whole dataset [194].

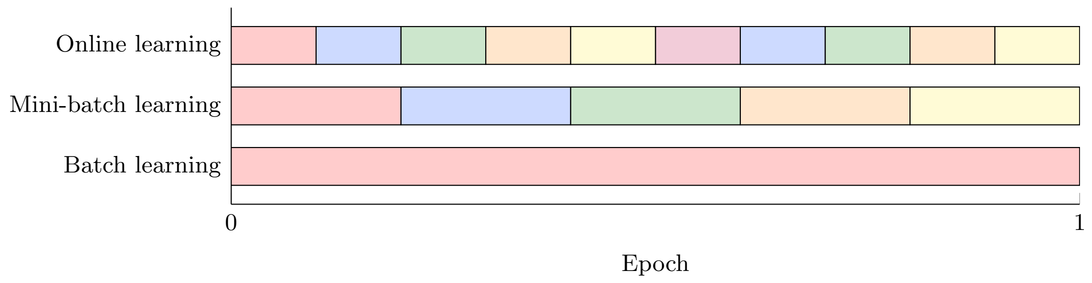


Figure 3.7: *An illustration of online, mini-batch and batch learning.*

Whenever the weights and biases are updated, an iteration is incremented, whether it was updated after a single training example (online learning) or a batch of training examples (batch learning). Moreover, when the entire training set has been evaluated by the training algorithm, an *epoch* is incremented. For example, to elucidate these two definitions further, take the previous example of 10 training examples. If online learning is employed, then when 10 iterations pass an epoch is completed. On the other hand, when batch learning is employed with a batch size of 2, then an epoch will be incremented after 5 iterations have been completed. When batch learning is employed, where the batch size is equal to the number of training examples, then one iteration is equal to one epoch.

Stochastic gradient descent maintains a single learning rate for all weight updates and the learning rate does not change during training, *i.e.* adaptively. For this reason, some gradient descent optimisation algorithms have been proposed, where the learning rates are adapted during the search. The algorithms include:

1. Nesterov momentum [208, 241], was developed to help accelerate SGD in the relevant direction.

2. Nesterov accelerated gradient (NAG) [193], was developed to improve upon the momentum term, instead of blindly following the slope, Nesterov evaluates gradients using the previous velocity.

3. Adaptive gradient algorithm (AdaGrad) [78] improves upon NAG and is an algorithm that adapts the learning rate to the parameters.

4. Adadelta [263] is an extension of Adagrad that seeks to reduce its radically decreasing learning rates, by adapting the updates for each individual parameter.

5. Root mean square propagation (RMSprop), proposed by Geoffrey Hinton [116, 245], similar to Adadelta was developed to resolve Adagrad's radically diminishing learning rates.

6. Adaptive moment estimation (Adam) [141] is a combination of RMSprop and momentum, and compares favourably to other adaptive learning-method algorithms.

7. AdaMax [141] adapts the update rule in Adam.

8. Nesterov-accelerated Adaptive Moment Estimation (Nadam) [75], is a combination of Adam and NAG, Adam is essentially RMSprop with momentum, Nadam is Adam with Nesterov momentum.

SGD is included in the hyperparameter search space of the ANN, and because of their simplicity, computational efficiency, relatively low memory requirements and combined advantages [106], Adam and Nadam are also included in the ANN's hyperparameter search space. The learning rate determines how fast the model learns during training or the amount that the weights are updated during training, also referred as the *step size*. Therefore, the learning rate affects both the learning speed as well as the convergence of the model. In Figure 3.8, an illustration is presented of the effect of the learning rate on the networks capability to learn. Note that smaller learning rates require more training epochs as the 'steps' are smaller, whereas larger learning rates require fewer training epochs as a result of the large 'steps' taken to update the weights [106].

Figure 3.8a illustrates the effect of employing a small learning rate, training takes long and it is possible that the algorithm can get stuck at a local optimum because the 'step' is too small. Figure 3.8c illustrates the effect of employing a high learning rate, which can cause premature convergence to suboptimal values for the weights. The learning rate that results in the fastest convergence is η_{opt}, as seen in Figure 3.8b. The largest learning rate that can be used without causing divergence is $\eta_{max} = 2\eta_{opt}$ [159], if $\eta > 2\eta_{opt}$ the algorithm will diverge and high-quality weights will not be found, as illustrated in Figure 3.8d.

According to Goodfellow *et al.* [106], the learning rate might be the most important hyperparameter, serving as motivation for its inclusion in the hyperparameter search space. Note that a single learning rate is considered for the weights.

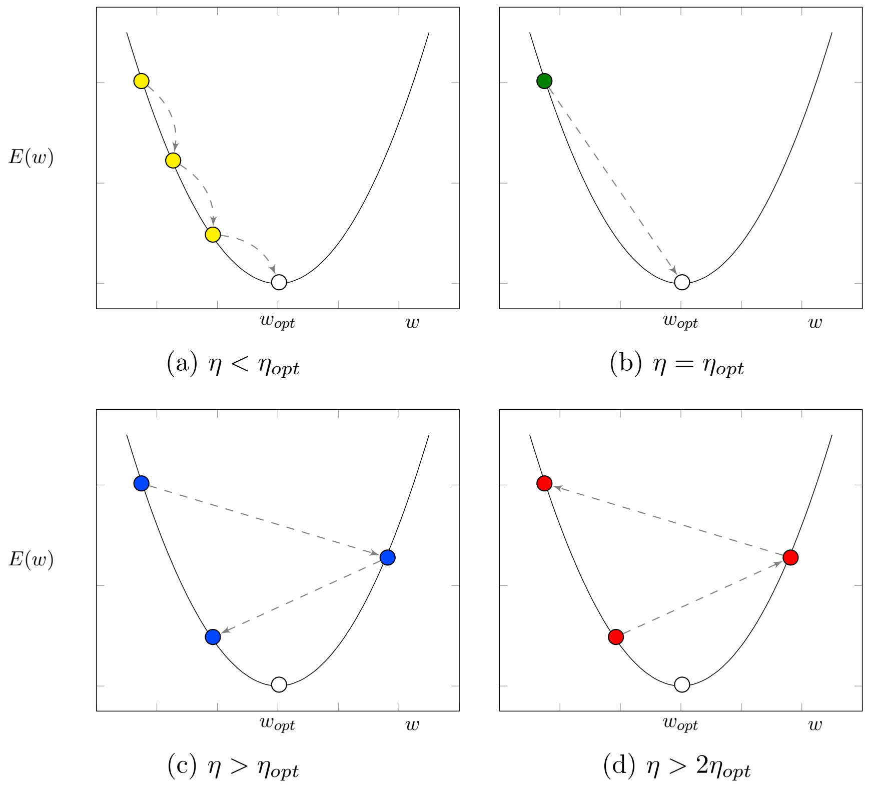


Figure 3.8: *Gradient descent for different learning rates, adapted from [159].*

Next, methods for network weight initialisation are discussed.

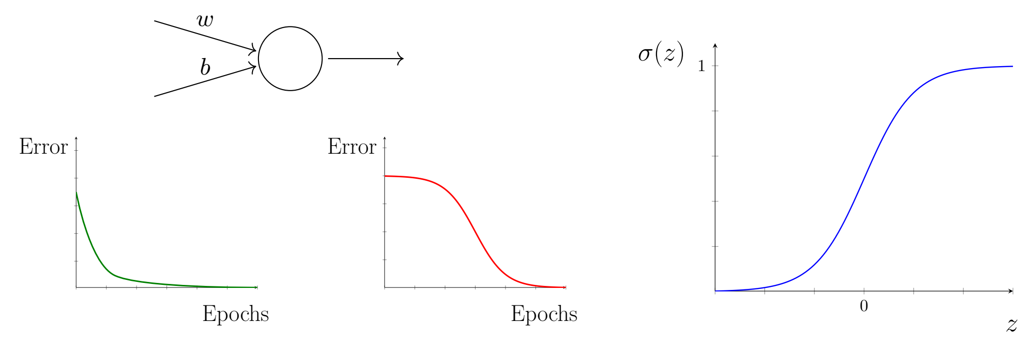
Network weight initialisation

Humans tend to learn faster when they make mistakes, however, the same does not hold true for neural networks. For example: Take a network with one input to a node. Suppose the goal was to train the single input neuron to take the input 1 and change it to the output 0. However, trivial this task may seem, it will help to explain the *learning slowdown phenomenon* that occurs when the initial weight value is too large, and is therefore worth computing the weight w and bias b using gradient descent.

Experiment 1 The first experiment uses the parameters and hyperparameter (the learning rate) as specified in Figure 3.9a, which leads to an initial output of 0.82 which is far from the desired 0.0. Figure 3.9a shows how the neuron rapidly learns the weight and bias that minimises the error and produces an output of 0.09 which is much closer to 0.0.

Experiment 2 The second experiment uses the parameters and hyperparameter as specified in Figure 3.9b, which leads to an initial output of 0.98 which is even further from the desired 0.0, although they use the same learning rates. Figure 3.9b shows how the neuron learns markedly slower, in fact, at the beginning the weight and bias barely change. Then, as learning kicks in the neuron's output rapidly moves closer to 0.0. The learned weight and bias minimises the error and gives an output of 0.20.

To summarise, Experiment 1 performs better and yields a better prediction for the output than Experiment 2 and is attributable to the chosen initial weight and bias. Based on the experiments and the results it is clear that the artificial neuron encounters considerable difficulty learning when the initial weight and bias is large, *i.e.* either too large or too small. Therefore, it can be concluded that smaller initialised values for w and b are preferred. This learning slowdown phenomenon occurs in larger networks too [195]. Consequently, weight and bias initialisation is included in the hyperparameter search space.



(a) *Experiment 1:* $w = 0.6,\ b = 0.9,\ \eta = 0.15$ (b) *Experiment 2:* $w = 2,\ b = 2,\ \eta = 0.15$ (c) *The Sigmoid function*

Figure 3.9: *The learning slowdown phenomenon, illustrating the effect of weight initialisation and the chosen activation function.*

The training algorithm is essentially an optimisation algorithm which requires a starting point in the space of possible network weight values from which the optimisation (learning or training) of the neural network can begin. Consequently, before training (or learning) can transpire, the network weights must be initialised. This is an important design choice, since the initial weight values have a substantial effect on the *convergence speed* and *quality of solutions* obtained during the training process [159, 106], as discussed previously.

A *too-large* initialisation leads to exploding gradients, resulting in the error to oscillate around the minimum value (or even diverge). Conversely, *too-small* initialisations lead to vanishing gradients, resulting in convergence of the error before it has reached the minimum value [106]. The basic convention is to initialise all the weights to values randomly drawn from a *uniform* or *Gaussian probability distribution*. Weight initialisation methods use information about the network, such as the activation function (and its associated derivative) and the number of weighted connections. The heuristic initialisation methods provide the prospects of a more effective optimisation process and became the *de facto* standard [106].

The following weight initialisation procedures are suggested based on their prolific use in literature.

1. The Sigmoid activation function requires a modification, since the function saturates for markedly small/large values of s, as shown previously in Figure 2.16(a), given by (2.15). The modification enables the linear region to dictate the range of values from which the weights can be drawn.

   - LeCun *et al.* [159] suggested that the weights be drawn from some probability distribution, with a mean of zero and a standard deviation of $\sqrt{\frac{1}{m}}$, where m is the number of weighted connections entering the neuron (or fan-in number).
   - Xavier Glorot and Yoshua Bengio [95] proposed the normalised '*xavier*' or '*glorot*' initialisation method, where the weights are drawn from the *uniform probability distribution* referred to as GlorotUniform, given by

   $$w = \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n+m}}, \frac{\sqrt{6}}{\sqrt{n+m}}\right], \tag{3.9}$$

     where n is the number of outgoing weighted connections (or fan-out number). Another initialisation method was proposed using the normal distribution, referred to as GlorotNormal.

2. When employing the PReLU activation function, as previously shown in Figure 2.17, He *et al.* [112] recommended the *'he'* initialisation method. Where the weight w is calculated as a random number from the *Gaussian probability distribution*, referred to as HeNormal, where the weights have a mean of zero and a standard deviation of $\sqrt{\frac{2}{m}}$. Another initialisation method was proposed using the uniform distribution, referred to as HeUniform.

Other than the number of hidden neurons and the number of training epochs, there are some other techniques that can be used to improve the generalisation capabilities of the regression model, some of which are discussed next.

3.3.2 An introduction to generalisation techniques

Network generalisation is achieved by controlling the capacity of the network. Goodfellow *et al.* [106] defines regularisation as any improvement to the learning algorithm to 'reduce its generalisation error but not its training error.' The most relevant generalisation strategies found in literature are *parameter norm penalisation*, known as *L1*, *L2* and *elastic-net regularisation*, *batch normalisation*, *early stopping*, *dropout* and *cross-validation* [30]. Essentially, these techniques control the complexity of the network in one way or another to avoid the behaviour observed in the overfitting zone (c), depicted in Figures 3.6d and 3.6e [30].

L1, and L2 attempt to control model complexity by the addition of a *penalty term* to the error function. By adding a small amount of bias, a smoother input-to-output mapping is encouraged [30, 106, 152, 243]. Elastic-net combines L1 and L2 regularisers. According to Goodfellow *et al.* [106], early stopping is commonly used attributable to its effectiveness and simplicity. The premise of this strategy is as follows: During training, the validation error is monitored which can provide an indication for when the network is overfitting, as seen in Figure 3.6d and 3.6e. Initially the training and validation error decreases, but at some point, at the optimal number of epochs, the validation error starts to increase. The standard procedure is to stop the training process (with some patience) if the validation error does not improve. Patience is the number of epochs that the validation error is permitted to increase before training is stopped and is set to 15 in this study [106].

The dropout regularisation technique involves randomly shutting down (or dropping) hidden neurons in each iteration, however, is not considered in this study. Batch normalisation re-centres and re-scales the activations of each input variable, for each mini-batch, by normalising them to have a mean of zero and a standard deviation of one [31]. Batch normalisation is said to stabilise the learning process and dramatically reduce the amount of training required [106]. It is important to note that not all the hyperparameters previously discussed form part of the hyperparameter search space. This is because hyperparameter optimisation is a time-consuming process due to the combinatorial relationship that exists.

The next section presents the results obtained during model training and testing for the respective regression models as well as delineates the methodology followed to determine the best hyperparameters for the FNN with one hidden layer.

3.4 Training, validating and testing the regression models

If a model is trained and evaluated on the same dataset then the evaluation metric produced is known as *training loss* (or error). Unfortunately, this leads to a model that has overfit to the data and is unlikely to generalise to out-of-sample data. Alternatively, the train-test split can be used, where the dataset is split into two sets, the training and test set. The model is trained on the training set and evaluated on the test set and results in a performance metric known as the *testing loss*. However, the testing loss is a high variance estimate of the out-of-sample data and is markedly sensitive to how the data is split, *i.e.* if you split the data with a different random number seed you are likely to obtain a different testing loss than previously obtained.

As mentioned, cross-validation can be used to train a machine learning model and then estimate how well a model is likely to perform on out-of-sample data. There are various ways to perform cross-validation, however, the method employed in this study is k-fold cross-validation, where k refers to the number of groups that the data set is split into. The training set is partitioned into k groups, as illustrated in Figure 3.10, for $k = 10$, *i.e. 10-fold CV*. The test set is held separate for the final evaluation of the ML model, while the remaining data forms the training set and is split into k groups (or folds). At each iteration, one of the k folds are used as the validation set, while the other $k - 1$ remaining folds are used for training.

By training and testing the ML model k times on different subsets of the training data and averaging the scores of each iteration allows for a more accurate representation of the model's generalisation capability, compared to one training and one validation set, a sort of ensemble method. The regression models were built following the proposed machine learning pipeline presented in Figure 3.11.

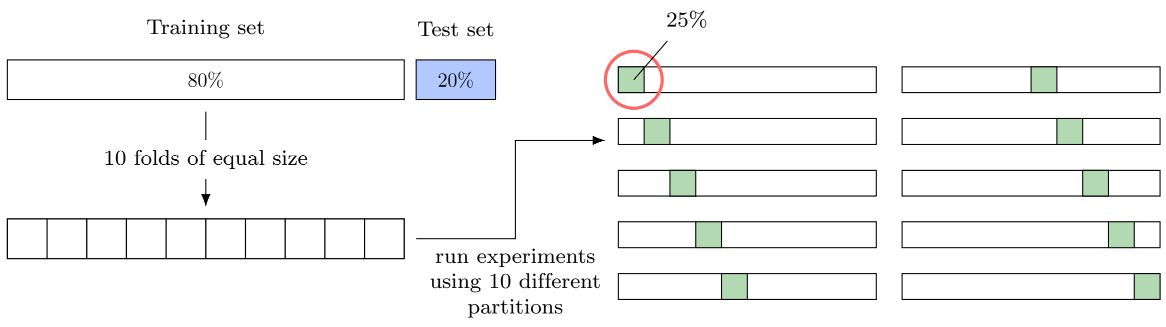


Figure 3.10: *An illustration of how the dataset is split for k-fold cross validation.*

The data collection phase consisted of performing exhaustive enumeration for the (s, S) inventory problem (IP) and store the data as inputs and corresponding outputs as illustrated in Table 3.2, which provides a snapshot of the first five examples from 250 000 training examples. The reorder point (ROP) and reorder quantities (ROQ) represent the decision variable values (or inputs) and the total inventory cost (TIC) and service level (SL) the objectives (or outputs).

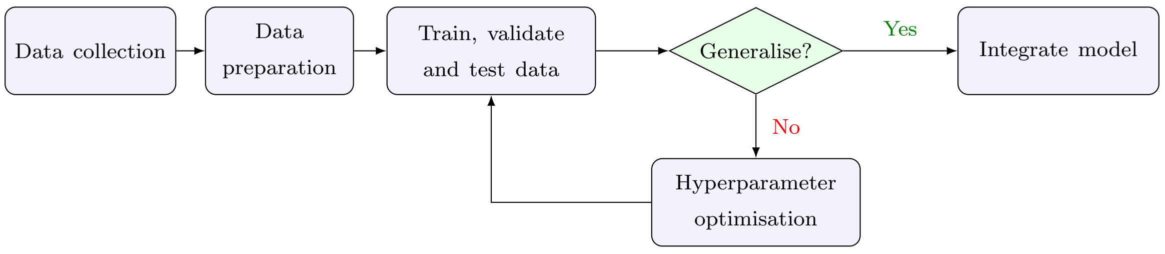


Figure 3.11: *The proposed machine learning pipeline.*

Table 3.2: *An example of the data in the IP dataset.*

	ROP	ROQ	TIC	SL
1	20	20	7686.53	16.80
2	20	30	7394.59	18.79
3	20	40	7261.08	20.80
4	20	50	7154.16	22.66
5	20	60	7074.46	24.55

Assume the necessary libraries have been imported. In the data preparation phase, the dataset is imported, then partitioned into the training and test set. Accordingly, the training set consists of 200 000 training examples and the test set of 50 000. The validation set is then 25% of the training set, *i.e.* 12 500 training examples. After the data is split, feature scaling is applied, *i.e.* normalisation of the data to ensure all values are on the same scale. This prevents one feature (or variable) from dominating another [81]. Next, build the regressors (as described in the Python script presented in Listing A.1), then train the regressor on the training set using 10-fold CV, measuring the MSE, MAE, as discussed previously, on the training, validation and test set. The results for the 30 trails are presented in Tables A.1–A.5 for each regression model.

The stochastic nature of the regression models necessitates a sample of algorithmic performances, therefore the performance measure scores presented in Table 3.3 were calculated by performing 30 trials (using 30 different random number seeds to produce 30 different datatset splits) of the learning model and then averaging the individual scores for each performance measure.

The Python script demonstrating this for MLR is given in Listing A.2. The best performing algorithm is RFR, which achieved the lowest training, validation and test scores in terms of MSE and MAE, and the worst performing algorithm is MLR.

Table 3.3: *The performance of the respective regressors in terms of MSE and MAE for the IP dataset.*

	Training		Validation		Test	
	MSE	*MAE*	MSE	MAE	MSE	MAE
MLR	0.148	0.311	0.148	0.311	0.148	0.311
SVM	0.003	0.044	0.003	0.044	0.003	0.044
DTR	0	0	0	0.003	0	0.002
RFR	0	0.001	0	0.002	0	0.002
XGBoost	0	0.007	0	0.007	0	0.007

As discussed previously, ANNs have many hyperparameters that can be adjusted, however, it requires expert knowledge because the hyperparameter combination is dataset specific (problem specific). The hyperparameter search space considered for optimisation is presented in Table 3.4, resulting in 12 960 possible combinations, further motivating the use of Bayesian optimisation. Recall that only one hidden layer is considered, as supported by the literature presented in Chapter 2.

Table 3.4: *The proposed hyperparameters search space for the FNN with one hidden layer.*

(a) *Network hyperparameters*

Neural network search space	
Number of neurons	2, 4, . . . , 20
Activation function	Sigmoid, ReLU, ELU
Weight initialisation	HeNormal, HeUniform, GlorotNormal, GlorotUniform
Batch normalisation	Yes, No
Training algorithm	SGD, Adam, Nadam
Kernal regulariser	$L1$, $L2$, $L1L2$

(b) *Algorithm hyperparameters*

Algorithm search space	
Learning rate	0.01, 0.001, 0.0001
Batch size	512, 1028

The results obtained by employing Bayesian optimisation (by running the Python code in Listing A.3) is presented in Table 3.5, for a batch size of 512 and 1028 and with or without batch normalisation. For example, for a batch size of 512 training examples and employing batch normalisation, the optimal hyperparameters, for the FNN with one hidden layer, are Nadam with a learning rate of 0.01, L_2 regularisation, the ELU activation function, HeUniform weight initialisation and 12 hidden neurons, which result in a MSE of 0.052.

The network has to be trained again, this time with the hyperparameter combinations found during Bayesian optimisation, as listed in Table 3.5. Once the model training is completed the generalisation capabilities of the network can be evaluated on the test set. Again, the stochastic nature of the regression model necessitates a sample of algorithmic performances, therefore the FNN is run for 30 independent trials, presented in Tables A.6–A.9 for the hyperparameter combinations A–D, as specified in Table 3.5. Note that each trial is run with a different random number seed in order to generate 30 different dataset splits. The performance of the model is then represented by the average of the 30 trials, as described by the Python code in Listing A.4. The performance measure scores are tabulated in Table 3.6, where the metrics represent the average for both objectives for all 30 trials.

Table 3.5: *The best hyperparameters obtained for the FNN, with one hidden layer, during Bayesian optimisation.*

Convention	*A*	*B*	*C*	*D*
Batch size	512	512	1028	1028
Batch normalisation	✓	✗	✓	✗
Hyperparameters				
Training algorithm	Nadam	Adam	Nadam	Nadam
Learning rate	0.01	0.001	0.001	0.0001
Regulariser	L2	L2	L2	L2
Activation	ELU	ELU	ELU	ELU
Initialiser	HeUniform	HeNormal	HeNormal	HeUniform
Units	12	14	14	20
Score:	0.052	0.052	0.052	0.055

According to Table 3.6, the best performance (in terms of the smallest MSE and MAE for the training, validation and test set) corresponds to hyperparameter combination B, *i.e.* a batch size of 512 without employing batch normalisation, employing the Adam optimiser with a learning rate of 0.001, L_2 regularisation, ELU as activation function, HeNormal as weight initialisation method and with 14 hidden neurons.

Table 3.6: *The average performance of the respective hyperparameter combinations, as shown in Table 3.5, in terms of MSE and MAE for the IP dataset.*

	Batch size	Batch normalisation	Training set		Validation set		Test set	
			MSE	MAE	MSE	MAE	MSE	MAE
A	512	✓	0.0532	0.1466	0.0532	0.1466	0.0669	0.1474
B	512	✗	0.0492	0.1319	0.0492	0.1319	0.0494	0.132
C	1028	✓	0.0511	0.1391	0.0511	0.1391	0.0508	0.1376
D	1028	✗	0.0599	0.1506	0.0599	0.0151	0.0601	0.1508

3.5 Conclusion: Chapter 3

The main objective of this chapter was to demonstrate whether or not an ANN as metamodel would be a feasible solution to enhance the simulation optimisation process. Table 3.7 summarises the performances of each regression model in terms of MSE and MAE. Overall, RFR performed the best and MLR the worst. The FNN with one hidden layer (with a batch size of 512 without employing batch normalisation) performed better than MLR. This chapter provided an in-depth inverstigation into the workings of an ANN, considering the network structures, network initialisation methods, learning algorithms and generalisation techniques, providing valuable insight into the complex interactions within an ANN. Therefore, this pilot study is considered *novel* and can be used as a starting point for further experimentation. This pilot study proposed to demonstrate the feasibility of an ANN metamodel, however, from the results in Table 3.7 it is not clear whether or not an ANN metamodel adds sufficient value.

Further experimentation is required regarding the following:

1. At what stage during training can the network start predicting whether or not solutions should be evaluated?

Table 3.7: *Comparison of the performances of the respective regression models, in terms of MSE and MAE for the test set for the IP dataset.*

	MSE	MAE
MLR	0.1471	0.3101
SVM	0.0026	0.0438
DTR	0	0.002
RFR	0	0.0015
XGBoost	0.0001	0.0071
ANN	0.0494	0.132

2. What is the optimal batch size? Consider for example a mini-batch size of 16, 32 and 64.
3. Consider the regularisation parameter λ for L1, L2 and L1L2 regularisation which controls the trade-off between minimising the error function and keeping the network parameters (*i.e.* weights and biases) small [195].
4. Consider dropout regularlisation and the dropout rate.

However, this is out of the scope of this study but should be considered for future work. The next chapter presents the simulation models studied as well as some statsitical prerequisites required for simulation output analysis.

CHAPTER 4

Simulation Models and Statistical Prerequisites

Previously, Chapter 3 reported on the preliminary analysis conducted to determine whether or not the application of an ANN as a metamodel is a feasible solution to enhance the simulation optimisation process.

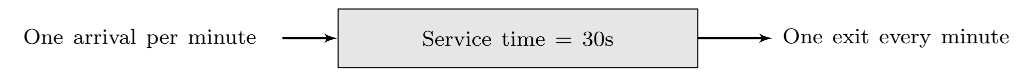
This chapter serves to fulfil Objective 3, as stated in Chapter 1. This chapter introduces some of the statistical aspects with regards to simulation output and each reports the problem-specific simulation parameters studied. Firstly, the statistical prerequisites necessary for simulation output analysis are discussed. Thereafter, the respective simulation models are discussed in terms of their decision variables (inputs) and objectives (or outputs) as well as their boundaries. Next, the sufficient number of observations per solution is determined scientifically and because R&S is out of the scope of this study, analysis-of-variance (ANOVA) is used to determine whether 100 observation per solutions are sufficient for each problem. Penultimately, a description of the complexity of each simulation model is presented as well as its true Pareto front. Finally, the chapter closes with a summary of its contents.

4.1 Statistical analysis preliminaries

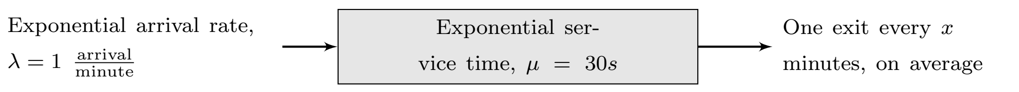
The simulation of stochastic systems is simply *statistical sampling* and therefore relies on statistical inferences to be able to analyse and draw conclusions from the results. Stochastic systems or systems containing stochastic elements (or randomness) respond stochastically, *i.e.* the output(s) obtained for the same set of input(s) will differ for different runs, the reader is referred to [136] for further reading.

Consequently, several observations per solution is required to describe the output variable in terms of its point and interval estimators to then analyse and draw conclusions from. The analysis of the output depends on the type of system (terminating) analysed. For terminating systems, every termination (or simulation time) results in one observation for the output variable(s) being studied. Each observation is assumed to be statistically independent from other observations during the same simulation run [20, 157].

For example, a simple deterministic system is given in Figure 4.1a. A customer arrives and leaves every minute, with a constant service time of 30s. If the system is changed to stochastic, as shown in Figure 4.1b, it is no longer possible to predict exit time. Assume the simulation time is from 07:00–17:00, where the mean of all the exit times observed, from 07:00–17:00, represent one observation. Suppose 100 days are observed, then the mean of the mean observed exit times are taken and represent the point estimator (x) for the output, *i.e.* 100 observations per solution was observed. A sufficient number of arrivals are required, to draw a conclusion from the unknown exit parameter x.



(a) *Deterministic system*

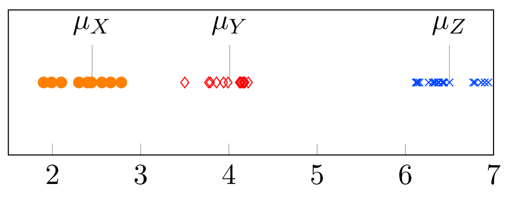

Exponential arrival rate, $\lambda = 1 \frac{\text{arrival}}{\text{minute}}$
Exponential service time, $\mu = 30s$
One exit every x minutes, on average

(b) *Stochastic system*

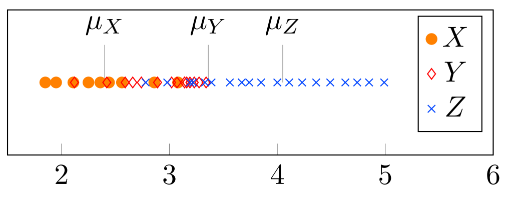
Figure 4.1: *Illustration of (a) a simple deterministic and (b) a simple stochastic system, adapted from [20].*

Formally, it is necessary to determine the number of observations per solution required to be 95% confident that the mean value x, is equal to the true but unknown population mean, where the confidence interval (CI) specifies the range in which it is to be expected. The output (or estimated value) generated by a stochastic system may be observed as numerically different, however they may not be *statistically significantly different* [20].

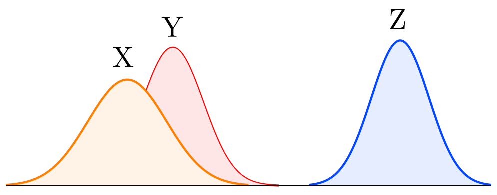
For example, $\bar{X}_1 = 22.4$ and $\bar{X}_2 = 23.5$ differ numerically, however, it is necessary to determine statistically whether they differ statistically significantly or not. To elaborate, Figures 4.2a and 4.2b are used. Figure 4.2a shows samples without a common mean and cross-sample variation, *i.e.* X, Y and Z are different and Figure 4.2b shows samples with a common mean and in-sample variation, *i.e.* X, Y and Z are considered similar.



(a) *Samples without a common mean and high cross-sample variability*



(b) *Samples with a common mean and high in-sample variability*



(c) *Samples with different distribution characteristics*

Figure 4.2: *Illustration of the difference between (a) cross-sample, (b) in-sample variation and (c) distributions having these characteristics, adapted from [20].*

Figures 4.2a and 4.2b can also be explained considering Figure 4.2c, where distribution X and Y are from the same population that exhibit in-sample variability whereas distribution Z is from a different population that exhibits cross-sample variation between X, Y *versus* Z. As stated in the project scope, it is assumed that all models have been built correctly for the purpose of validating the proposed solutions. Also, unless stated otherwise the simulation time for each model is *five simulation days*, denoted by 05:00:00:00. The simulation problems studied as well as other relevant information are discussed next. For a formal description of each, refer to Appendix D.

4.2 The simulation model problems

To reiterate, due to the stochastic nature inherent to simulation models, small-sample theory must be applied to deal with the observations generated by the simulation model. Typically, the expected values for the objectives are described by *point* and *interval estimators* which require, as mentioned, several observations per solution [157]. The more observations per solution, the smaller the CI becomes, *i.e.* less variance of the output distribution, which leads to a *good* approximation (or point estimator) of the output parameter(s) being studied [20].

However, this requires more simulation time to complete, *i.e.* the computational time may become considerable [28, 82]. For example, assume one observation per solution takes 0.1 seconds to evaluate and 100 observations per solution are required, then 10 seconds are required to evaluate one decision variable combination. Recall that earch algorithms evaluate many solutions during the search process, assuming 1 000 decision variable combinations are evaluated then the simulation optimisation process would take 2.77 hours.

Upon considering the above-mentioned, it can be derived that a good search algorithm should find good quality approximation sets and do so by the least number of function evaluations, *i.e.* find good solutions fast. The amount of time it takes to evaluate a single solution varies from simulation model to simulation model and is attributable to the complexity of the model. There are four elements that influence the complexity of a simulation model, namely the number of objectives, number of decision variables, number of stochastic elements, and size of the search space (or combinatorial nature).

Accordingly, all the problems studied differ in terms of their complexities and are reported in the following sections, respectively. In this study, the complexity of the simulation problems differ by the number of decision variables, stochastic elements and the search space cardinality, not influenced by the number of objectives since all the problems in this study are bi-objective simulation optimisation problems.

After careful consideration of the problem context, a sensible upper bound for each problem was chosen. The upper bounds were introduced, because when going beyond a certain value, a point of diminishing return is reached, also it is used as a method to contain the problem. The BAP with five machines (BAP5) is used to illustrate this. Table D.4 lists the experiments used for the hypothesis test to determine the upper bound for BAP5, and is summarised in Table 4.1.

Table 4.1: *Summary of the experiments (in the columns) used to determine a sensible upper bound as given in Table D.4.*

Groups	Count	Sum	Average	Variance
Work-in-progress	9	21.72	2.41	0.02
Throughput	9	830.09	92.23	0.03

The possible outcomes of the hypothesis test is, either the data suppports the research question stated as the *null hypothesis* H_0 (which is assumed to be true) or disproves the research prediction stated as the *alternate hypothesis* H_1 [71]. Specifically, the null hypothesis states that the means do not differ and consequently there is no statistically significant difference between the work-in-progress (WIP) or throughput objective values for different buffer sizes, formally stated as (4.1). Alternatively, (4.2) the means differ significantly and therefore the buffer sizes influence the objectives significantly.

Note that one-way ANOVA is used, *i.e.* the test is conducted for each objective respectively. If H_0 is rejected for one objective then that objective becomes the deciding factor, be it for the upper bound chosen or the number of observations per solution. The following hypothesis is used for both the work-in-progress and throughput objectives for determining a sensible upper bound for BAP5.

$$H_0 \quad : \quad \mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5 = \mu_6 = \mu_7 = \mu_8 = \mu_9 \tag{4.1}$$

$$H_1 \quad : \quad \mu_i \neq \mu_j, \quad i \neq j, \quad i, \ j \in \{1, \dots, 9\}. \tag{4.2}$$

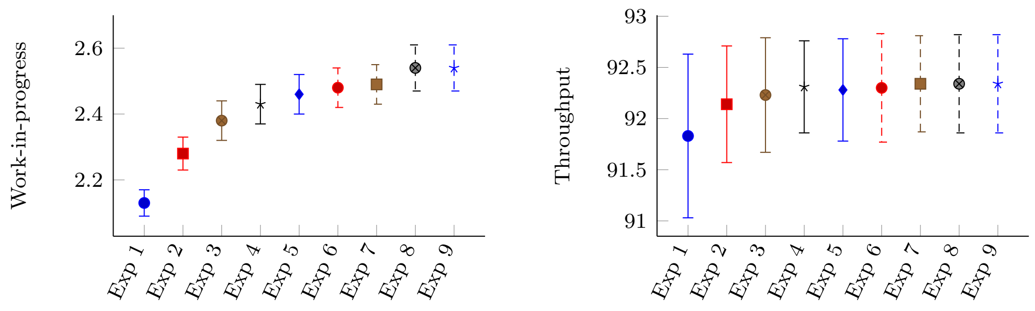
The values for each objective in Table D.4 may differ numerically, however ANOVA is conducted to determine whether they differ statistically. The ANOVA test results are summarised in Table 4.2 and describe the sum of squares (SS), degrees of freedom (df), mean squared (MS), the F-statistic (F) the corresponding critical value (Fcrit) and p-value. Since $F\text{crit} < F$ and $p < 0.05$ the outcome of the ANOVA test is to reject H_0, which indicates that the mean of at least one pair is statistically different and therefore a *post hoc* test is required to determine which experiments differ significantly. To identify the significant pair(s) *post hoc* analysis is required,. The multiple comparisons t-test is used for *post hoc* analysis.

Table 4.2: *The results of the ANOVA test to determine the upper bound for BAP5.*

Source of variation	SS	df	MS	F	Fcrit	p-value	*Outcome*
Between groups	36 303.27	1	36 303.27	1 599 757.20	4.49	0	Reject H_0
Within groups	0.36	16	0.02				
Total	36 303.63	17					

Figures 4.3a and 4.3b illustrate the CI plot per experiment for objectives work-in-progress and throughput and also presents the pair-wise comparisons. Note that the dot represents the expected mean and the line indicates the range of 95% of the observations. From the visual inspection of Figure 4.3b, it can be seen that the throughput objective values do not differ statistically and consequently has no influence on the decision regarding the problem upper bound. Considering the work-in-progress objective a sensible upper bound is chosen. The larger the upper bound, the larger the combinatorial nature of the problem, therefore a lower upper bound is preferred.

From the p-values in Table 4.3 obtained from the *post hoc* t-test, regarding the work-in-progress objective, it can be said that *Exp 5* is not *statistically significantly different* to Exp 6–9. Consequently, implementing more than 17 buffers per machine does not yield a work-in-progress value that is significantly better, *i.e.* the threshold is reached and the work-in-progress becomes dependent on the arrival rate of the work. This means that even if 100 buffer spaces are allocated, the point of diminishing return is met at 17 buffers. Logically, further analysis should be limited to 17 buffer spaces per machine. A similar process was followed to determine sensible upper bounds for the *Open mine problem* (OMP), the (s, S) *inventory problem* (IP), the *BAP* with 10 machines (BAP10) and the *non-linear BAP* with 16 machines (BAP16). The resulting upper bounds are reported in the following respective sections.



(a) *CI plot for the work-in-progress values* (b) *CI plot for the throughput values*

Figure 4.3: *Illustration of the meticulous consideration taken with the choice of upper bound for BAP5.*

Table 4.3: *Table of the p-values of the t-test for the work-in-progress objective for BAP5.*

	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6	Exp 7	Exp 8	Exp 9
Exp 1	0	0	0	0	0	0	0	0
Exp 2		0.01	0	0	0	0	0	0
Exp 3			0.16	0.04	0.01	0.01	0	0
Exp 4				0.54	0.29	0.17	0.02	0.02
Exp 5					0.66	0.44	0.08	0.08
Exp 6						0.74	0.18	0.18
Exp 7							0.31	0.31
Exp 8								1

The next sections set out to describe each simulation problem in terms of their inputs and outputs, upper bounds and solution cardinality. Next, hypothesis testing is conducted to determine whether or not 100 observations per solution are sufficient and lastly, a description of the complexity of each problem is given as well as its true Pareto front.

4.2.1 The Open mine problem

There are two conflicting objectives that need to be optimised, namely minimise the *total cost* while maximising the *number of trains served*. The decision variables are the number of shovels, crushers and trucks. The sensible upper bound was found at 15 for all decision variables, resulting in a discrete search space cardinality of 3 375 solutions.

The MOO question can be formulated as: what values of *shovels*, *crushers* and *trucks* that result in a minimum total cost and maximum number of trains served? In the context of simulation optimisation, a search algorithm samples values for shovels, crushers and trucks within the specified range, the values are then evaluated by the simulation model and point estimators for the objectives are returned. The evaluation of a decision variable combination $\{shovels, crushers, trucks\}$ results in a corresponding value for total cost and number of trains served.

A wide spectrum of problems sizes are studied, from small (the OMP) to very large (BAP16). The reason for including a smaller problem such as the OMP is to validate that the algorithms can solve the problems studied. The reasoning is that if the algorithms are able to find the true Pareto set (or markedly good approximations of the true Pareto set) for a reasonably small problem then they should be able to solve the larger problems.

Observations per solution

The proposed number of observations per solution is 100 due to the limited timeframe in which the study must be completed. To motivate this decision further, the respective problems are tested for 10, 100 and 1 000 observations per solution for 10 different experiments (or decision variable combinations), respectively, to test whether there is a statistically significant difference in the number of observations chosen and whether or not 100 observations are sufficient.

The following procedure was followed for all the problems. Note that the decision variable combinations are chosen arbitrarily and has no effect on the outcome of the hypothesis test conducted. The null hypothesis states that the means do not differ and consequently there is no statistically significant difference between the number of observations used, formally stated as (4.3). Alternatively, (4.4) the means differ significantly and therefore the number of observations used matter.

$$H_0 \quad : \quad \mu_1 = \mu_2 = \mu_3 \tag{4.3}$$
$$H_1 \quad : \quad \mu_i \neq \mu_j, \quad i \neq j, \quad i,\ j \in \{1, \ldots, 3\} \tag{4.4}$$

The 10 experiments that are used to determine whether 100 observations per solution are sufficient and are given in Table D.14, a summary of which is presented in Table D.15. The ANOVA test results are summarised in Table 4.4. Note that the simulation time for the OMP is *one simulation day*, denoted by 01:00:00:00.

Table 4.4: *The results of the ANOVA test for the OMP.*

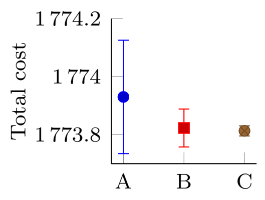
	Total cost						Served trains					
Source of Variation	SS	df	MS	F	Fcrit	p-value	SS	df	MS	F	Fcrit	p-value
Between Groups	48 677 990.34	9	5 408 665.59	613 461 121.17	2.39	0	16.13	9	1.79	190.39	2.39	0
Within Groups	0.18	20	0.01				0.19	20	0.01			
Total	48 677 990.51	29					16.32	29				

Table 4.5: *The experiments that are statistically significantly different for the OMP.*

	$p < 0.05$	
	Total cost	Served trains
Exp 1	✓	✓
Exp 3		✓
Exp 4	✓	✓
Exp 5		✓
Exp 6	✓	✓
Exp 7		✓
Exp 8	✓	✓
Exp 9		✓
Exp 10	✓	✓

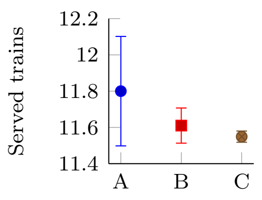
Since Fcrit $< F$ and $p < 0.05$, the outcome is to reject H_0 which indicates that the mean of at least one pair is significantly different and therefore a *post hoc* test is required to determine which. As mentioned, the multiple comparisons are used to determine the significant pair(s). The t-test concluded that there is a significant difference for *Exp 1* and *Exp3–10*, as seen in Table 4.5. *Exp 1* is used to visually present the differences in terms of CIs for the total cost

(Figure 4.4a) and served trains (Figure 4.4b) objectives, for 10, 100 and 1 000 observations as well as the corresponding p-values. From Figures 4.4a and 4.4b it can be seen that for *Exp 1*, 10 observations per solution (A) are *statistically significantly different* to 100 and 1 000 observations per solution (B and C, respectively), *i.e.* $p < 0.05$.



	B	C
A	0.01	0
B		0.81

(a) *CI plot and p-values for Exp 1*



	B	C
A	0.01	0
B		0.38

(b) *CI plot and p-values for Exp 1*

Figure 4.4: *The respective CI plots and p-value tables obtained for the work-in-progress and throughput objectives, respectively, for the OMP.*

Therefore, 10 observations per solution would not be enough to say with 95% confidence that the mean generated by 10 observations is the same as the mean generated by 100 or 1 000 observations. However, B and C are not statistically significantly different and therefore, 100 observations per solution is sufficient for OMP.

Problem complexity

To summarise, the OMP has two objectives and *three* decision variables, with a search space cardinality of 3 375 solutions. The problem has *five* sources of randomness, namely shovel processing time, crusher processing time, the processing time for loader 1 and loader 2, and an inspection processing time. It is also important to note that each search technique only evaluates 1 000 solution evaluations, *i.e.* only 29.63% of the total solution space is searched.

Exhaustive enumeration

To obtain the true Pareto set, exhaustive enumeration was performed with 100 simulation observations per {Shovels, Crushers, Trucks} combination. All decision variables were adjusted in steps of 1 unit from 1 to 15, thus exploring a solution space of 3 375 possibilities. The resulting front (containing 36 points) is considered the true Pareto front and is shown in Figure 4.5. The true hyperarea for the OMP is 14 485.04, determined with reference point $\{2800,\ 3.8\}$.

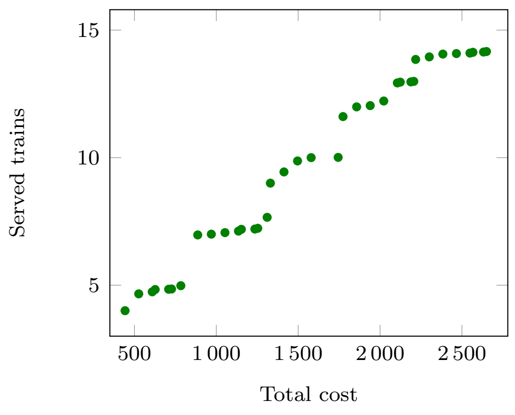


Figure 4.5: *The true Pareto front of the OMP obtained by method of exhaustive enumeration.*

4.2.2 The (s, S) inventory problem

There are two conflicting objectives that need to be optimised, namely minimise *total inventory cost* while maximising *service level*. The decision variables are the *reorder point* (s) and *reorder quantity* (S). The sensible upper bound was found as $s, S \in \{1, \dots, 500\}$, resulting in a discrete search space cardinality of 250 000 solutions.

The MOO question can be formulated as: what values of s and S result in a minimum total inventory cost and maximum service level? Again, in the context of simulation optimisation, a search algorithm samples values for s and S within the specified range, the values are then evaluated by the simulation model and point estimators for the objectives are returned. The evaluation of a decision variable combination s, S results in a corresponding value for total inventory cost and service level.

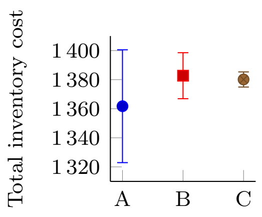
Observations per solution

A similar process is followed as documented for the OMP, the null (4.3) and alternative (4.4) hypothesis stated remain the same, but for the IP. The 10 experiments used to determine if 100 observations per solution are sufficient are given in Table D.1, where *group A* represents 10 observations, *B*–100 observations and *C*–1 000 observations, a summary of which is presented in Table D.2. The ANOVA test results are summarised in Table 4.6.

Table 4.6: *The results of the ANOVA test for the IP.*

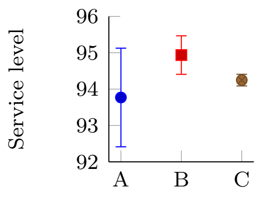
	Total inventory cost						Service level					
Source of variation	SS	df	MS	F	Fcrit	p-value	SS	df	MS	F	Fcrit	p-value
Between groups	8 998 297	9	999 810.82	18 942.63	2.39	0	15 051.09	9	1 672.34	9 350.76	2.39	0
Within groups	1 055.62	20	52.78				3.58	20	0.18			
Total	8 999 353	29					15 054.66	29				

Since Fcrit $< F$ and $p < 0.05$, the outcome is to reject H_0 which indicates that the mean of at least one pair is significantly different and therefore a *post hoc* test is required to determine which pair. Again, multiple comparisons t-test is used to determine the significant pair(s). The t-test concluded that there is a significant difference for *Exp 6* for the total inventory cost objective and *Exp 10* for the service level objective. *Exp* 6 and *Exp 10* are used to visually present the differences in terms of CIs for the total inventory cost (Figure 4.6a) and service level (Figure 4.6b) objectives, for 10, 100 and 1 000 observations as well as the corresponding p-values. From Figures 4.6a and 4.6b it can be seen that for *Exp 6* and *Exp 10*, 10 observations per solution (A) are *statistically significantly different* to 100 observations per solution (B), *i.e.* $p < 0.05$.



	B	C
A	0.03	0.07
B		0.82

(a) *CI plot and p-values for Exp 6*



	B	C
A	0	0.14
B		0.07

(b) *CI plot and p-values for Exp 10*

Figure 4.6: *The respective CI plots and p-value tables obtained for the total inventory cost and service level objectives, respectively, for the IP.*

Therefore, 10 observations per solution would not be enough to say with 95% confidence that the mean generated by 10 observations is the same as the mean generated by 100 observations. However, B and C are not statistically significantly different and therefore 100 observations per solution are sufficient for the IP.

Problem complexity

To summarise, the IP has two objectives and two decision variables, with a search space cardinality of 250 000 solutions. The problem has *three* sources of randomness (or stochastic elements), namely the arrival rate, customer demand and the inventory replenishment lead times. It is also important to note that the search techniques only evaluate a small percentage of the solution space. Specifically, only 1 000 evaluations were allowed, *i.e.* only 0.4% of the total solution space is searched.

Exhaustive enumeration

To obtain the true Pareto set, exhaustive enumeration was performed with 100 simulation observations per (s, S) combination. Both s and S were adjusted in steps of 1 unit from 1 to 500, thus exploring a solution space of 250 000 possibilities. The resulting front (containing 1 393 points) is considered the true Pareto front $\mathcal{P}_T$ for this problem and is shown in Figure 4.7. The true hyperarea for the IP is 114 443.61, determined with reference point $\{2500,\ 20\}$.

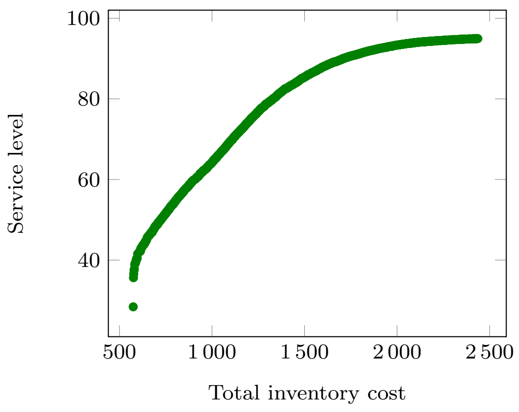


Figure 4.7: *The true Pareto front of the IP obtained by method of exhaustive enumeration.*

4.2.3 The buffer-allocation problem: five machines

For each BAP instance, there are two conflicting objectives that need to be optimised, namely minimise *work-in-progress* while maximising *throughput*. The decision variables are the buffer spaces (or niches) buffer 1–4 with a sensible upper bound that was found at 17, as described previously, resulting in a discrete search space cardinality of 83 521 solutions.

The MOO question for the BAP can be formulated as: what values of the buffer spaces result in the minimum work-in-progress and maximum throughput? This is the same for all the BAP instances, however the buffer spaces differ for each instance. Again, in the context of simulation optimisation, a search algorithm samples values for buffer 1–4 within the specified range.

The values are then evaluated by the simulation model and point estimators for the objectives are returned. The evaluation of a decision variable combination {buffer 1, buffer 2, buffer 3, buffer 4}, for the sake of brevity {B1, B2, B3, B4}, results in a corresponding value for the work-in-progress and throughput.

Observations per solution

Again, a similar process is followed as documented for the OMP, the null (4.3) and alternative (4.4) hypotheses stated remains the same, but for BAP5. The 10 experiments that are used to determine whether 100 observations per solution are sufficient are given in Table D.5, a summary of which is presented in Table D.6. The ANOVA test results are summarised in Table 4.7.

Table 4.7: *The results of the ANOVA test for BAP5.*

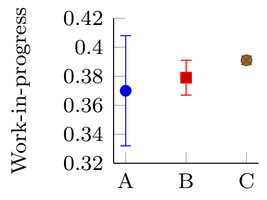
	Work-in-progress						Throughput					
Source of Variation	SS	df	MS	F	Fcrit	p-value	SS	df	MS	F	Fcrit	p-value
Between Groups	9.46	9	1.05	468.35	2.39	0	1 252.24	9	139.14	145.24	2.39	0
Within Groups	0.04	20	0				19.16	20	0.96			
Total	9.51	29					1 271.39	29				

Since Fcrit $< F$ and $p < 0.05$, the outcome is to reject H_0 which indicates that the mean of at least one pair is significantly different and therefore a *post hoc* test is required to determine which. Again, multiple comparisons are used to determine the significant pair(s). The t-test concluded that there is a significant difference for *Exp* 1, *Exp* 3–4 and *Exp* 7–10, as seen in Table 4.8. *Exp1* is used to visually present the differences in terms of CIs for the work-in-progress (Figure 4.8a) and throughput (Figure 4.8b) objectives, for 10, 100 and 1 000 observations as well as the corresponding p-values.

Table 4.8: *The experiments that are statistically significantly different for BAP5.*

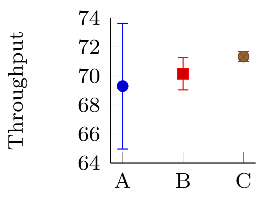
	$p < 0.05$	
	Work-in-progress	Throughput
Exp 1	✓	✓
Exp 3		✓
Exp 4		✓
Exp 7	✓	
Exp 8	✓	
Exp 9	✓	
Exp 10		✓

From Figures 4.8a and 4.8b it can be seen that for *Exp 1*, 10 observations per solution (A) are *statistically significantly different* to 1 000 observations per solution (C), *i.e.* $p < 0.05$. Therefore, 10 observations per solution would not be enough to say with 95% confidence that the mean generated by 10 observations is the same as the mean generated by 1 000 observations. However, B and C are not statistically significantly different and therefore, 100 observations per solution is sufficient for BAP5.



	B	C
A	0.25	0.01
B		0.14

(a) *CI plot and p-values for Exp 1*



	B	C
A	0.30	0.02
B		0.14

(b) *CI plot and p-values for Exp 1*

Figure 4.8: *The respective CI plots and p-value tables obtained for the work-in-progress and throughput objectives, respectively, for the BAP5.*

Problem complexity

To summarise, BAP5 has two objectives and four decision variables, with a search space cardinality of 83 521 solutions. The problem has *fifteen* sources of randomness, namely five machine processing times, five failure distributions and five mean time to repair (MTTR) distributions. Only a small percentage of the total search space is explored, specifically only 1 000 evaluations were allowed, *i.e.* only 1.2% of the total solution space is searched.

Exhaustive enumeration

To obtain the true Pareto set, exhaustive enumeration was performed with 100 simulation observations per {B1, B2, B3, B4} combination. All buffers were adjusted in steps of 1 unit from 1 to 17, thus exploring a solution space of 83 521 possibilities. The resulting front (containing 306 points) is considered the true Pareto front $\mathcal{P}_T$ for this problem and is shown in Figure 4.9. The true hyperarea for the BAP is 33.84, determined with reference point {2.5, 70}.

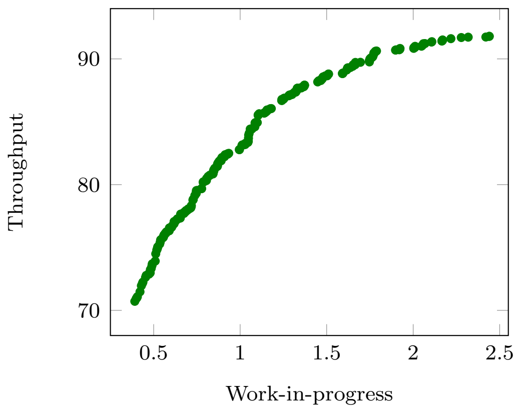


Figure 4.9: *The true Pareto front of BAP5 obtained by method of exhaustive enumeration.*

4.2.4 The buffer-allocation problem: 10 machines

The decision variables are the buffer spaces buffer 1–9 (B1–B9). The sensible upper bound was found at 10, resulting in a discrete search space cardinality of 1 000 000 000 solutions. Again, in the context of simulation optimisation, a search algorithm samples values for B1–9 within the specified range. The values are then evaluated by the simulation model and point estimators for the objectives are returned. The evaluation of a decision variable combination {B1, B2, B3, B4, B5, B6, B7, B8, B9}, results in a corresponding value for the work-in-progress and throughput.

Observations per solution

Again, a similar process is followed as documented for the OMP, the null (4.3) and alternative (4.4) hypothesis stated remain the same, but for BAP10. The 10 experiments that are used to determine whether 100 observations per solution are sufficient are given in Table D.8, a summary of which is presented in Table D.9. The ANOVA test results are summarised in Table 4.9.

Table 4.9: *The results of the ANOVA test for BAP10.*

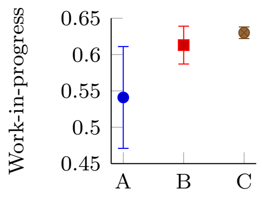
	Work-in-progress						Throughput					
Source of Variation	SS	df	MS	F	Fcrit	p-value	SS	df	MS	F	Fcrit	p-value
Between groups	4.47	9	0.50	703.67	2.39	0	730.72	9	81.19	17.32	2.39	0
Within groups	0.01	20	0				93.77	20	4.69			
Total	4.48	29					824.49	29				

Since $F\text{crit} < F$ and $p < 0.05$, the outcome is to reject H_0 which indicates that the mean of at least one pair is significantly different and therefore a *post hoc* test is required to determine which. Again, multiple comparisons are used to determine the significant pair(s). The t-test concluded that there is a significant difference for *Exp 1–8* and *10*, as seen in Table 4.10.

Table 4.10: *The experiments that are statistically significantly different for BAP10.*

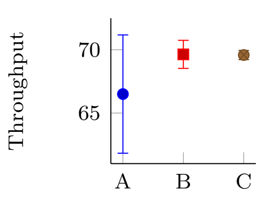
	$p < 0.05$	
	Work-in-progress	Throughput
Exp 1	✓	
Exp 2	✓	✓
Exp 3		✓
Exp 4		✓
Exp 5		✓
Exp 6		✓
Exp 7		✓
Exp 8		✓
Exp 10		✓

Exp 2 is used to visually present the differences in terms of CIs for the work-in-progress (Figure 4.10a) and throughput (Figure 4.10b) objectives, for 10, 100 and 1 000 observations as well as the corresponding p-values.



	B	C
A	0	0
B		0.35

(a) *CI plot and p-values for Exp 2*



	B	C
A	0	0
B		0.95

(b) *CI plot and p-values for Exp 2*

Figure 4.10: *The respective CI plots and p-value tables obtained for the work-in-progress and throughput objectives, respectively, for BAP10.*

From Figures 4.10a and 4.10b it can be seen that for *Exp 2*, 10 observations per solution (A) are *statistically significantly different* to 100 and 1 000 observations per solution (B and C, respectively), *i.e.* $p < 0.05$.

Therefore, 10 observations per solution would not be enough to say with 95% confidence that the mean generated by 10 observations is the same as the mean generated by 100 or 1 000 observations. However, B and C are not statistically significantly different and therefore, 100 observations per solution is sufficient for BAP10.

Problem complexity

To summarise, BAP10 has two objectives and nine decision variables, with a search space cardinality of 1×10^9 solutions. The problem has *thirty* sources of randomness, namely ten machine processing times, ten failure distributions and ten mean time to repair (MTTR) distributions. It is also important to note that the search techniques only evaluate a small percentage of the solution space. Specifically, only 1 000 evaluations were allowed, *i.e.* only 0.0001% of the total solution space is searched.

Exhaustive enumeration

An exhaustive search approach involves searching through the entire solution space (*i.e.* 100 simulation observations per {B1, B2, B3, B4, B5, B6, B7, B8, B9} combination and all buffers adjusted in steps of 1 unit from 1 to 10) to find the true Pareto set. This approach is not feasible for large decision spaces, such as 1×10^9 possibilities, as it becomes computationally time consuming and practically impossible. Therefore, design of experiments was used to obtain a good approximate of the true Pareto set which for the purposes of this study will represent the true Pareto set. The resulting front (containing 120 points) is considered the true Pareto front and is presented in Figure 4.11. The true hyperarea for BAP10 is 22.66, determined with reference point {1.4, 60}.

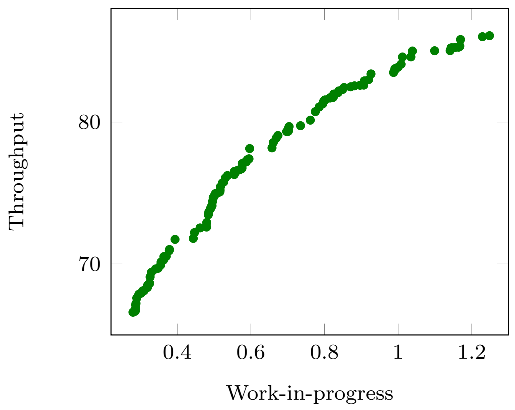


Figure 4.11: *The true Pareto front of BAP10 obtained by design of experiments.*

4.2.5 The non-linear buffer-allocation problem: 16 machines

The decision variables are the buffer spaces buffer 1–15 (B1–B15). The sensible upper bound was found at 10, resulting in a discrete search space cardinality of 1×10^{15} solutions. Again, in the context of simulation optimisation, a search algorithm samples values for B1–15 within the specified range. The values are then evaluated by the simulation model and point estimators for the objectives are returned. The evaluation of a decision variable combination {B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, B11, B12, B13, B14, B15}, results in a corresponding value for the work-in-progress and throughput.

Obervations per solution

Again, a similar process is followed as documented for the OMP, the null (4.3) and alternative (4.4) hypothesis stated remain the same, but for BAP16. The 10 experiments that are used to determine whether 100 observations per solution are sufficient are given in Table D.11, a summary of which is presented in Table D.12. The ANOVA test results are summarised in Table 4.11.

Table 4.11: *The results of the ANOVA test for BAP16.*

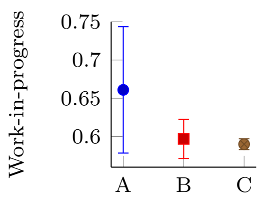
	Work-in-progress						Throughput					
Source of Variation	SS	df	MS	F	Fcrit	p-value	SS	df	MS	F	Fcrit	p-value
Between Groups	2.67	9	0.30	193.14	2.39	0	34 889.22	9	3 876.58	101.25	2.39	0
Within Groups	0.03	20	0				765.77	20	38.29			
Total	2.71	29					35 654.99	29				

Since Fcrit $< F$ and $p < 0.05$, the outcome is to reject H_0 which indicates that the mean of at least one pair is significantly different and therefore a *post hoc* test is required to determine which. Again, multiple comparisons are used to determine the significant pair(s). The t-test concluded that there is a significant difference for *Exp 2–4* and *8–10*, as seen in Table 4.12.

Table 4.12: *The experiments that are statistically significantly different for BAP16.*

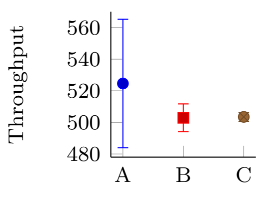
	$p < 0.05$	
	Work-in-progress	Throughput
Exp 2		✓
Exp 3	✓	✓
Exp 4	✓	
Exp 8	✓	
Exp 10	✓	✓

Exp 3 is used to visually present the differences in terms of CIs for the work-in-progress (Figure 4.12a) and throughput (Figure 4.12b) objectives, for 10, 100 and 1 000 observations as well as the corresponding p-values. From Figures 4.12a and 4.12b it can be seen that for *Exp 10*, 10 observations per solution (A) are *statistically significantly different* to 100 and 1 000 observations per solution (B and C, respectively), *i.e.* $p < 0.05$.



	B	C
A	0	0.01
B		0.93

(a) *CI plot and p-values for Exp 3*



	B	C
A	0	0
B		0.95

(b) *CI plot and p-values for Exp 3*

Figure 4.12: *The respective CI plots and p-value tables obtained for the work-in-progress and throughput objectives, respectively, for the BAP16.*

Therefore, 10 observations per solution would not be enough to say with 95% confidence that the mean generated by 10 observations is the same as the mean generated by 100 or 1 000 observations. However, B and C are not statistically significantly different and therefore 100 observations per solution is sufficient for BAP16.

Problem complexity

To summarise, BAP16 has two objectives and *fifteen* decision variables, with a search space cardinality of 1×10^{15} solutions. The problem has *fifty-one* sources of randomness, namely sixteen machine processing times, sixteen failure distributions and sixteen mean time to repair (MTTR) distributions. Lastly, the routing from B1, B3 and B7 are determined stochastically, *i.e.* there is a 50:50 chance for the part to go from B1 to M1 or M2, B3 to M4 or M5 and B7 to M8 or M9, as seen in Figure D.8. Again, note that the search techniques only evaluate a small percentage of the solution space. Specifically, only 1 000 evaluations were allowed, *i.e.* only 1×10^{-10}% of the total solution space is searched.

Exhaustive enumeration

An exhaustive search approach involves searching through the entire solution space (*i.e.* 100 simulation observations per {B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, B11, B12, B13, B14, B15} combination and all buffers adjusted in steps of 1 unit from 1 to 10) to find the true Pareto set. However, as mentioned, this approach is not feasible for large decision spaces, such as 1×10^{15} possibilities. Therefore, design of experiments was used to obtain a good approximate of the true Pareto set which for the purposes of this study will represent the true Pareto set. The resulting front (containing 132 points) is considered the true Pareto front and is presented in Figure 4.13. The true hyperarea for BAP16 is 214.33, determined with reference point $\{1.4,\ 350\}$.

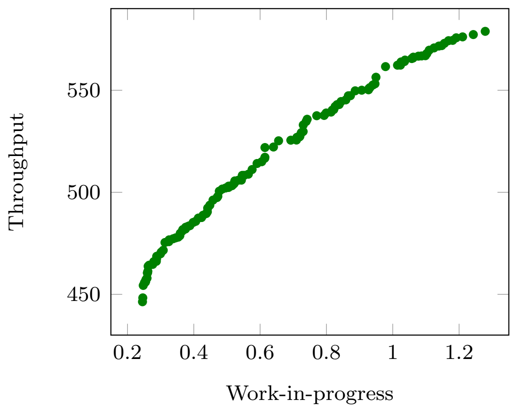


Figure 4.13: *The true Pareto front of BAP16 obtained by design of experiments.*

4.3 Conclusion: Chapter 4

This chapter discussed some of the statistical prerequisites required to understand the analysis of simulation outputs. The respective simulations models were also described in terms of their inputs, outputs, upper bounds, search space cardinality, complexity and the true Pareto front was given. This chapter also set out to determine whether 100 observation per solution are sufficient.

Table 4.13 summarises the results of the statistical tests conducted in this chapter. *Overall*, 100 observations per solution are sufficient for all five problems, whereas 10 observations were not and 1 000 are not computationally feasible. Also, with 100 observations per solution the central limit theorem can be applied and the observed solutions are approximately normally distributed.

Table 4.13: *Summary of the sufficient number of observations per solution for the respective problems.*

	Observations		
	10	100	1 000
IP	×	✓	✓
BAP5	×	✓	✓
BAP10	×	✓	✓
BAP16	×	✓	✓
OMP	×	✓	✓

To revise, the more observations per solution, the smaller the CI becomes, however the effort required is greater, *i.e.* the values for 100 and 1 000 observations per solution may differ numerically, but statistically they are considered the same. However, there is a significant difference in the computational effort required to run 1 000 observations per solution when compared with 100. Also, it is more important to obtain an approximate Pareto set showing a trend, rather than having more accurate estimations of the objectives, *i.e.* it is more important to find approximations that are considered accurate enough in the minimum amount of simulation runtime possible.

Next, Chapter 5 documents the metaheuristics chosen as LLHs as well as the proposed hyperheuristics.

CHAPTER 5

Metaheuristics and Hyperheuristics

The previous chapter introduced the simulation optimisation problems studied as well as statistical prerequisites required to conduct a simulation study.

This chapter serves to fulfil Objective 4 and 5 as stated in Chapter 1. Metaheuristics are constantly being improved upon, either by designing new approaches or by extending current approaches, to better and more efficiently solve COPs. Consequently, deciding which metaheuristic (or heuristic operators) to use is not a trivial task. This chapter presents three metaheuristics from different classes in an attempt to represent the diversity of algorithms available in the literature, namely the multi-objective cross-entropy method (MOOCEM), the non-dominated sorting genetic algorithm II (NSGA-II) and the dominance-based multi-objective simulated annealing (DBMOSA) algorithm. First, some main concepts of S- and P-metaheuristics are discussed with specific focus on their implementation in this study. Next, each metaheuristic is discussed, in great detail, in the context of simulation optimisation and its integration in Tecnomatix.

Next, the two hyperheuristics proposed in this study are presented, first for population-based search and then for single-solution based search. The MOOCEM and the NSGA-II are employed for the population-based search hyperheuristic and DBMOSA and three move operators are employed for the single-solution based search hyperheuristic. Each hyperheuristic is also discussed in the context of simulation optimisation and its integration in Tecnomatix. The chapter closes with a summary of its contents.

5.1 The main concepts for metaheuristics

In this section a discussion follows on the basic concepts for metaheuristics, namely initial solution(s), solution representations and stopping conditions. These concepts are considered when designing *any* metaheuristic [93]. The concept of solution dominance and archiving, as discussed in §2.2.1 and §2.2.3, forms part of all the LLHs discussed in this chapter.

5.1.1 Initial solution

Generating initial solutions is considered a crucial step when designing a metaheuristic, as the initial solution(s) have a great impact on the exploration-exploitation capabilities of the metaheuristic and can therefore affect its performance [93]. In this regard, it is important to consider the strategy employed to ensure diverse initial solutions are generated to prevent premature convergence [134].

The two main strategies are *random* and *greedy*. The random strategy randomly generates initial solutions, whereas the greedy strategy finds determines an initial solution that is considered to be of *good quality* by method of applying a greedy algorithm to determine the initial solution. Again, a trade-off exists between the amount of exploration and exploitation. Hybrid strategies combine both random and greedy strategies, *i.e.* a pool of initial solutions are generated using both random and greedy algorithms [93, 134].

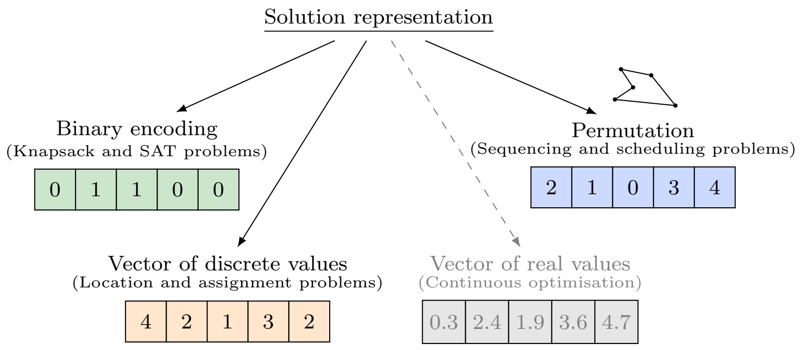
The strategy followed for generating initial populations for the P-metaheuristics is the random strategy. The initial populations are generated using the *normal* and *Poisson distributions*. It is important to note that the random seed values are different for every run to ensure that the P-metaheuristics start with a different initial population for each run. This is done for both MOOCEM and NSGA-II. The initial solution for the S-metaheuristic is randomly selected in the range of possible decision variable values. If a solution has been used as an initial solution, it is removed from the set of possible initial solutions, ensuring that DBMOSA starts at different initial solutions for every run.

5.1.2 Solution representation

Deciding on a *solution representation* (or encoding) is a fundamental design question to consider during the development of a metaheuristic. The representation influences the efficiency and effectiveness of a metaheuristic and constitutes careful consideration. The representation must be suitable and relevant to the optimisation problem, *i.e.* COPs. The efficiency is also related to the search operators applied to the representation, for example move operators or crossover operators. In conclusion, the choice of solution representation requires consideration of how the solution is evaluated and how the search operators manipulate the representation. Figure 5.1 highlights the main solution representations in the literature, namely binary encodings, vector of discrete values, permutations and vector of real values [93, 250].

For the sake of completeness, Figure 5.1 includes the representation for a vector of real values, however, it is not applicable for COPs and is excluded from further discussion.

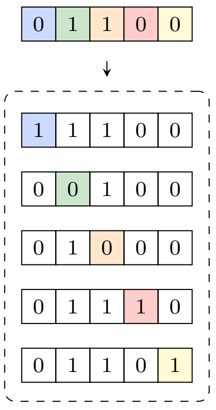
In Figures 5.2a–5.2c, an illustration is presented of the neighbourhoods that correspond to the solution representations applicable for discrete variable optimisation. A binary representation, represents a solution in a vector of bits, where a neighbour is created by flipping a single bit, as illustrated in Figure 5.2a. The neighbourhood is typically based on a Hamming distance of 1, where the size of the neighbourhood is the size of the vector.



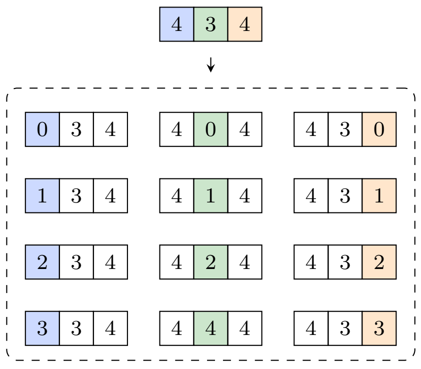
Solution representation
Binary encoding
(Knapsack and SAT problems)
0 1 1 0 0
Permutation
(Sequencing and scheduling problems)
2 1 0 3 4
Vector of discrete values
(Location and assignment problems)
4 2 1 3 2
Vector of real values
(Continuous optimisation)
0.3 2.4 1.9 3.6 4.7

Figure 5.1: *Some classical solution representations for optimisation problems: vector of binary values, vector of discrete values, vector of real values and permutation, adapted from [250].*

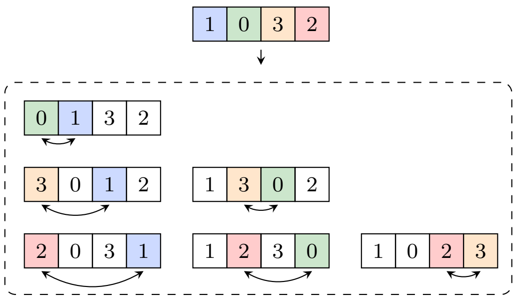
In Figure 5.2b, an illustration is presented of the neighbourhood for the solution $\{4, 3, 4\}$ with a lower bound of 0 and an upper bound of 4, for all decision variables. The size of the neighbourhood is $n(k-1) = 12$, where $n = 3$ represents the number of decision variables and the cardinality of the decision variables, is denoted by $k = 5$. The neighbourhood in Figure 5.2c is created by a pair-wise exchange, where the order is of significance. The neighbourhood size depends on the permutation length n and can be calculated as $\frac{n(n-1)}{2}$ [250].



(a) *The neighbourhood for the binary representation*



(b) *The neighbourhood for the discrete vector representation*



(c) *The neighbourhood for the permutation representation*

Figure 5.2: *Illustration of the neighbourhood for (a) binary encoding, (b) discrete vectors and (c) permutation representations, adapted from [250].*

A majority of the COPs studied either have permutation representations (*e.g.* travelling salesman problem (TSP)) or discrete vector representations (*e.g.* quadratic assignment problem (QAP)), which is attributable to the number of metaheuristics that are available and the simplicity of manipulating these types of representations [1, 148]. The simulation optimisation problems studied are considered COPs. Solutions are evaluated by the simulation model which takes as input a vector of discrete values and because the search operators can be modified to manipulate discrete vectors, discrete vector representation is chosen.

5.1.3 Stopping condition

The search terminates if the given stopping condition is met. A stopping condition may be a fixed number of iterations, or a convergence measure [93]. The stopping conditions of the LLHs have been modified to control explicitly the duration of each algorithm and to facilitate fair comparisons between the algorithms. A fixed number of iterations is adopted, namely 1 000 solution evaluations.

The following sections document each LLH with specific focus on the integration of simulation and optimisation, starting with MOOCEM, then NSGA-II and finally DBMOSA.

5.2 Multi-objective cross-entropy method

The single-objective cross-entropy method (CEM) [217] has been adapted to solve MOOPs. The multi-objective extension, called MOOCEM, was purposefully designed by Bekker and Aldrich [22] to solve (computationally expensive time-dependent) stochastic multi-objective simulation optimisation problems — naturally, motivating its inclusion in this study. The CEM is based on *Importance Sampling* and the *Kullback–Leibler* (or cross-entropy) distance [58]. For a theoretical discussion, the reader is referred to [217]. The following section introduces some preliminaries specific to the MOOCEM algorithm.

5.2.1 Preliminaries for the MOOCEM

The literature does not prescribe a specific probability distribution for the CEM [21, 22]. In this study the normal and Poisson distributions are used for both population initialisation and to sample decision variable values from. To enable sampling of feasible decision variable values, it is necessary to truncate the probability distribution so as to sample in the range $[l_i, u_i]$ for decision variable i.

The truncated normal (or Poisson) distribution on the given range $[0,\ u_i]$ is obtained by normalising the fundamental normal (or Poisson) distribution. This is described next, followed by how to sample from these distributions. The probability density function (pdf) for the normal distribution is given as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}, \quad -\infty < x < \infty, \tag{5.1}$$

where σ denotes the standard deviation and μ the mean. The pdf for the truncated normal distribution with μ_i and σ_i, on the range $[a, b]$, is given as

$$f_{t,i}(x,\ \mu_i,\ \sigma_i,\ a,\ b) = \frac{f(x)}{\int_a^b f(x)dx}, \quad a \leq x \leq b, \tag{5.2}$$

where $f(\cdot)$ is the pdf, as defined in (5.1).

The probability mass function (pmf) for the Poisson distribution can be expressed as

$$f(x) = \frac{\lambda^x \exp^{-\lambda}}{x!}, \quad x = 0, 1, 2, \ldots \tag{5.3}$$

where λ is the mean. Note that μ and σ (or λ) are arbitrarily initialised. The pmf for the truncated Poisson distribution with rate λ_i, on the range $[a, b]$, is given as

$$f_{t,i}(x,\ \lambda_i,\ a,\ b) = \frac{e^{-\lambda_i}\lambda_i^x}{\sum_{x=a}^{b} \frac{e^{-\lambda_i}\lambda_i^x}{x!} x!}, \quad x = a,\ \ldots,\ b. \tag{5.4}$$

The pseudo-code description of truncating a probability distribution is presented in Algorithm 5.1. The truncation process is similar for the normal and Poisson distributions, therefore one algorithm is sufficient to describe the process for both probability distributions. In Algorithm 5.1 the truncation process is first described for the normal distribution and then for the Poisson distribution in brackets.

First, the pdf (or pmf) values $f_i(x)$ for normal (or Poisson) distribution are calculated, using (5.1) (or (5.3)), for decision variable i in the range $[0,\ u_i]$. Next, each $f_i(x)$ value is normalised, by dividing it by the sum of the $f_i(x)$ values in the range $[l_i,\ u_i]$. This results in the truncated pdf (or pmf) values $f_{t,i}(x)$. Now, the truncated cdf values $F_{t,i}(x)$ can be calculated as the sum of $f_{t,i}(x)$, where x is in the range $[l_i,\ u_i]$ and $F_{t,i}(l_i) = f_{t,i}(l_i)$, $F_{t,i}(l_i + 1) = f_{t,i}(l_i) + f_{t,i}(l_i + 1)$ and so on. To validate that the distributions are truncated correctly, the cdf value for the upper bound should be equal to 1, *i.e.* $F_{t,i}(u_i) = 1$.

Algorithm 5.1: Truncate a probability distribution

Input : For the normal distribution, specify the μ_i and σ_i and (for the Poisson distribution, specify λ_i) also specify the upper and lower bound $[l_i,\ u_i]$ for decision variable i.
Output : The truncated normal (or Poisson) distribution.

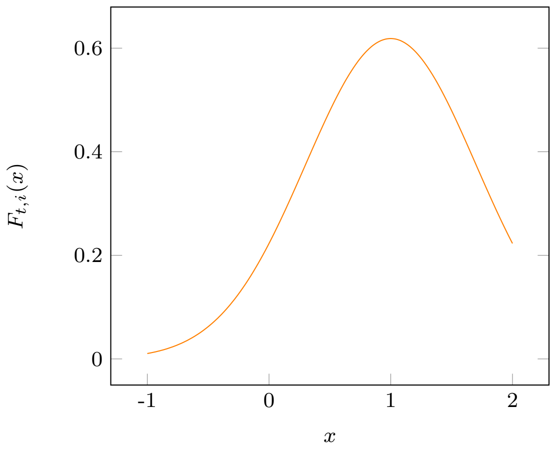
```
1  $sum \leftarrow 0$;                                              // Initialise $sum$
2   for $x := 0$ to $u_i$ do
3     Calculate the pdf (or pmf) value $f_i(x)$ for $x$, using (5.1) (or (5.3));   // $x \in [0,\ u_i]$
4  for $x := l_i$ to $u_i$ do
5     $sum \leftarrow sum + f_i(x)$;                         // Determine the sum, $x \in [l_i,\ u_i]$
6  for $x := l_i$ to $u_i$ do
7     Calculate the truncated pdf (or pmf) value, $f_{t,i}(x) \leftarrow f_i(x)/sum$;
8  Calculate the first truncated cdf value, $F_{t,i}(l_i) \leftarrow f_t(l_i)$;
9  for $x := l_i + 1$ to $u_i$ do
10    Calculate the truncated cdf value, $F_{t,i}(x) \leftarrow f_{t,i}(x) + f_{t,i}(x-1)$;   // $x \in [l_i + 1,\ u_i]$
```

The next step is to sample from the truncated normal (or Poisson) distribution which is described in Algorithm 5.2. To sample from the truncated normal (or Poisson) distribution, a random number $r_u \sim \mathcal{U}(0, 1)$ is generated and $x \leftarrow l_i$. If $r_u < F_{t,i}(x)$, where x is in the range $[l_i,\ u_i]$ for decision variable i, then x is incremented. The algorithm terminates when $r_u \geq F_{t,i}(x)$ and x is returned as the random sample in the range $[l_i,\ u_i]$.

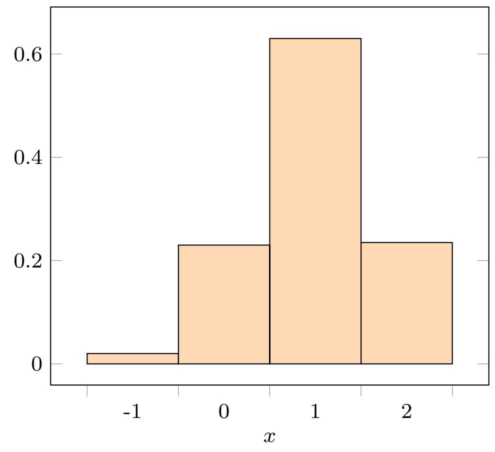
Sampling from these distributions is simple and the sample values, for decision variable i on the defined range, are easily obtained. Figures 5.3a and 5.3b illustrate the truncated normal distribution with μ_i and σ_i and the Poisson distribution with λ_i, for decision variable i, where x is defined in the range $[-1,\ 2]$.

Algorithm 5.2: Sample from a truncated probability distribution

```
Input  : The truncated normal (or Poisson) distribution, $F_{t,i}$.
Output : Random sample $x$, where $x$ is in the range $[l_i,\ u_i]$.
1 Generate a random number $r_u \sim \mathcal{U}(0, 1)$;
2 Set $x \leftarrow l_i$;
3 while $r_u < F_{t,i}(x)$ do
4     $x \leftarrow x + 1$;                          // Increment $x$
5 Return $x$;            // Random sample $x$, where $x$ is in the range $[l_i,\ u_i]$
```



(a) *Truncated normal distribution*



(b) *Truncated Poisson distribution*

Figure 5.3: *An example illustrating the (a) truncated normal distribution with μ_i and σ_i and (b) truncated Poisson distribution with λ_i, for decision variable i, where x is defined in the range $[-1,\ 2]$, adapted from [22].*

The next section provides a detailed description of the MOOCEM algorithm, supported by the user-interface of the MOOCEM tool used in Tecnomatix, to illustrate the integration of simulation and optimisation.

5.2.2 The MOOCEM algorithm

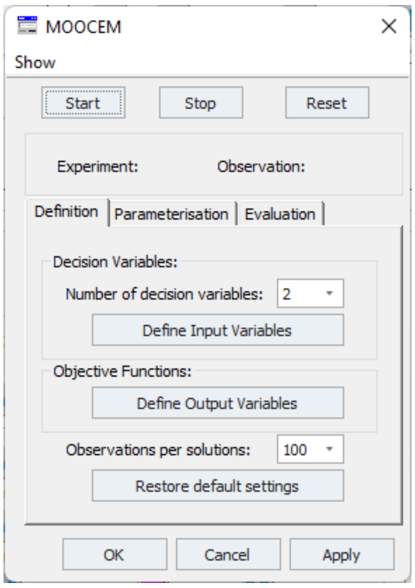
Figures 5.4a and 5.4b aids with elucidating the simulation optimisation process *via* the MOOCEM tool as it is used in Tecnomatix. The steps for using the MOOCEM tool are as follows: starting with the *Definition tab* as shown in Figure 5.4a.

1. Define the number of decision variables (n), from the drop-down list, then click the *Define Input Variables* button and define each decision variable in terms of its path and lower and upper bounds, as seen in Figure 5.5a.

2. Click the *Define Output Variables* button and define each output variable in terms of its path and select the optimisation direction from the drop-down list, *i.e. Min* or *Max* as seen in Figure 5.5b.

The next step is to navigate to the *Parameterisation tab*, as shown in Figure 5.4b:

3. Input the *Population size* (N), *Number of loops*, *Smoothing parameter*, *Probability of inversion* and *Epsilon*.


MOOCEM

Show

Start Stop Reset

Experiment: Observation:

Definition | Parameterisation | Evaluation

Decision Variables:

Number of decision variables: 2

Define Input Variables

Objective Functions:

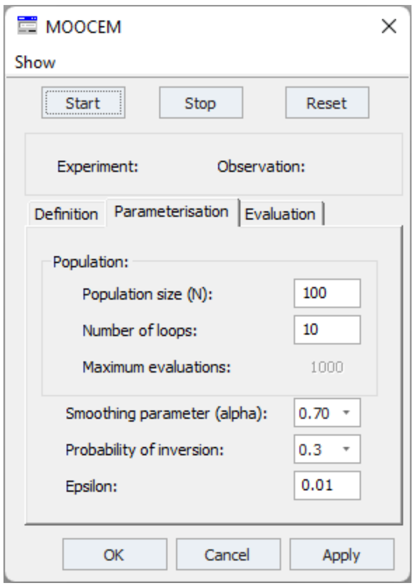
Define Output Variables

Observations per solutions: 100

Restore default settings

OK Cancel Apply

(a) *Definition tab*


MOOCEM

Show

Start Stop Reset

Experiment: Observation:

Definition | Parameterisation | Evaluation

Population:

Population size (N): 100

Number of loops: 10

Maximum evaluations: 1000

Smoothing parameter (alpha): 0.70

Probability of inversion: 0.3

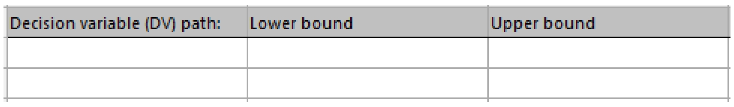
Epsilon: 0.01

OK Cancel Apply

(b) *Parameterisation tab*

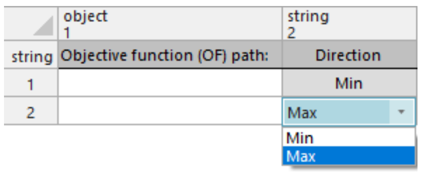
Figure 5.4: *The MOOCEM tool user-interface.*

The above-mentioned parameters are used as input to Algorithm 5.4. To execute the MOOCEM algorithm and optimise the simulation problem at hand click the *Start* button. The *Evaluation tab* is where the simulation time for the specific simulation problem is specified as discussed in Chapter 4.



Decision variable (DV) path:	Lower bound	Upper bound

(a) *The decision variable table*



	object 1	string 2
string	Objective function (OF) path:	Direction
1		Min
2		Max

Min
Max

(b) *The output variable table*

FIGURE 5.5: *The user-interface for the input and output tables used in Tecnomatix.*

As mentioned, the initial population is generated randomly. The Poisson distribution is a discrete probability distribution, denoted by $X \sim \text{Pois}(\lambda)$, where λ is the mean. The Poisson distribution is unable to generate sample values larger than 130, therefore if the upper bound of a decision variable exceeds 130, as in the case of the IP, the normal distribution is used. The normal distribution is a continuous probability distribution, denoted by $X \sim \mathcal{N}(\mu,\ \sigma^2)$, where μ is the mean and σ the standard deviation.

When sampling from the normal distribution, discrete intervals are used to enable the generation of discrete sample values and because decision variable i has a lower and upper bound, it is necessary to truncate the distributions as to contain the search to only generating sample values within the specified bounds, as discussed in §5.2.1.

The algorithm starts by initialising the iteration counter $t \leftarrow 1$, the current loop $k \leftarrow 1$, the current number of evaluations $NumEvaluations \leftarrow 0$ and a boolean variable $NotTerminate \leftarrow$ true. Next, the parameters specific to the probability distribution are initialised [21, 22]. If the normal distribution is used, then the mean μ_i and standard deviation σ_i is initialised, for decision variable i. If the Poisson distribution is used, then the mean λ_i is initialised for decision variable i. The following discussion assumes the normal distribution is used and therefore a mean and standard deviation are required.

1. Initialise μ_i and σ_i.

Initialise $\mu_i \leftarrow l_i + (u_i - l_i)r_u$, where l_i is the lower bound and u_i the upper bound for decision variable i, for $i \in \{1, \ldots, n\}$ and r_u is a random number, between 0 and 1, generated from the uniform distribution, denoted by $r_u \sim \mathcal{U}(0, 1)$. Next, initialise an arbitrarily large value for σ_i, for example $\sigma_i \leftarrow 10(u_i - l_i)$ [21, 22]. (In the case of the Poisson distribution, λ_i is arbitrarily initialised as $\lambda_i \leftarrow u_i r_u$).

2. Construct the truncated probability distribution for each decision variable i for μ_i and σ_i, using Algorithm 5.1. Refer to §5.2.1.
3. Generate an initial population, of size N, by sampling from the truncated probability distribution, using Algorithm 5.2 for each decision variable i. Again, refer to §5.2.1.
4. Evaluate the initial population, *i.e.* the simulation model evaluates N decision variable combinations.

Now, the loop initiates, while the current loop k is less than the user-defined *Number of loops*, denoted by k_{max}, do the following:

5. Rank the solutions obtained by the simulation model, using Algorithm 5.3 for specified ranking threshold ρ_E.
6. Append the solutions with ranking values not exceeding a specified threshold value ρ_E to a so-called elite vector, titled *Elite*.

Algorithm 2.1 is adapted to consider the ranking threshold ρ_E, as given in Algorithm 5.3.

Consequently, *Elite* represents the current *weakly* non-dominated set. Solution x_1 is said to weakly dominate x_2 (denoted by $x_1 \preceq x_2$) if $f_k(x_1) \leq f_k(x_2)$ for $k \in \{1, \ldots, m\}$ [162]. To ensure exploration and exploitation during the search, the initial ranking threshold is selected as $\rho_E = 2$, *i.e. Elite* contains solutions with rank $\rho \in \{0, 1, 2\}$. The ranking value of the solution indicates the number of solutions that dominate it [21, 22]. For example, a solution with $\rho = 2$ is dominated by two solutions in the population.

7. If the elite vector is not an empty set, denoted as $Elite \neq \emptyset$, then update μ_i and σ_i according to (5.5).
8. Construct the truncated probability distribution for each decision variable i for μ_i and σ_i, using Algorithm 5.1.
9. Generate the new population of N possible solutions by sampling from the truncated probability distribution, using Algorithm 5.2 for each decision variable i and continue from step 19 onwards.

Algorithm 5.3: Ranking threshold adaptation of the Pareto ranking algorithm, for a *min-max* bi-objective optimisation problem

```
   Input  : A population of solutions $\mathcal{P}$ and ranking threshold $\rho_E$.
   Output : The elite vector containing solutions with rank $0 - \rho_E$.
1  $\mathcal{P} \leftarrow \mathsf{sort}(\mathcal{P}, 1)$;          // Sort the first objective in descending order
2  for $i \leftarrow 1$ to $|\mathcal{P}|$ do
3      $penalty \leftarrow 0$;                                        // Initialise Pareto rank to zero
4      for $j \leftarrow (i+1)$ to $|\mathcal{P}|$ do
5          if ${f_j}^2 \prec {f_i}^2$ then // i.e. if ${f_j}^2 \geq {f_i}^2$
6              $penalty \leftarrow penalty + 1$;                      // Increment the penalty
7      $\rho_i \leftarrow penalty$;                                   // Rank of solution $i$
8  for $i \leftarrow 1$ to $|\mathcal{P}|$ do
9      if $\rho_i = \rho_E$ then
10         $Elite \leftarrow Elite \cup \{i\}$;                       // Solutions with rank $0 - \rho_E$
```

10. Check for duplicate solutions and replace them with non-duplicate solutions.

The parameter vectors $(\mu_i,\ \sigma_i)$ are smoothed by means of the smoothing function and the decision variable values in the elite vector, given as

$$\hat{\boldsymbol{v}}_t = \alpha\hat{\boldsymbol{v}}_t + (1-\alpha)\hat{\boldsymbol{v}}_{t-1}, \tag{5.5}$$

where t is the current iteration and α is the smoothing constant, typically in the range $[0.6,\ 0.9]$. For example, the value of $\sigma_{i,t}$, for the first decision variable, is updated as follows

$$\hat{\sigma}_{1,t} = \alpha\tilde{\sigma}_{1,t} + (1-\alpha)\hat{\sigma}_{1,t-1}, \tag{5.6}$$

after iteration t [21, 22]. The idea behind smoothing μ_i and σ_i is to account for some information from the previous iteration.

11. If $Elite \neq \varnothing$ and $k > 1$, then construct a histogram for each decision variable i using the values in *Elite*.

The histograms are used to guide the search and inform the MOOCEM on what regions it needs to explore next. The histogram concept is implemented as follows and explained using Figure 5.6 for a decision variable i defined in the range $[l_i,\ u_i]$. In this example, there are $r = 5$ equally sized classes that are formed between the two boundary classes (*i.e.* the first and last class), resulting in a total of $r + 2$ classes [21, 22].

The first step is to determine the boundaries of the classes. For the first class, the lower bound is set equal to l_i (the lower bound of decision variable i) and the upper bound is set equal to the minimum value found in *Elite*, *i.e.* $\min(Elite(\cdot, i))$. Similarly, for the last class, the upper bound is set equal to u_i (the upper bound of decision variable i) and the lower bound is set equal to the maximum value found in *Elite*, *i.e.* $\max(Elite(\cdot, i))$ [21, 22].

Next, the equally sized class intervals can be calculated as

$$\frac{\max(Elite(\cdot, i)) - \min(Elite(\cdot, i))}{r}.$$

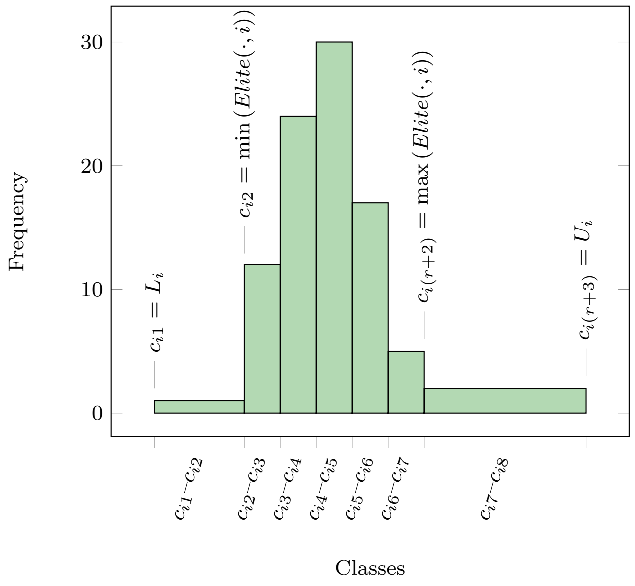


Figure 5.6: *An example illustrating the histogram concept for decision variable i and $r = 5$ equally sized classes, with a total of $r + 2$ classes, adapted from [22].*

In summary, consider again the example in Figure 5.6, suppose the class boundaries are recorded in a vector $\boldsymbol{C}_i = \{c_{i1},\ c_{i2}, \ldots, c_{i(r+2)},\ c_{i((r+2)+1)}\}$, where c_{i1} corresponds to the lower boundary of the first class ($i.e.\ c_{i1} = l_i$) and $c_{i((r+2)+1)}$ corresponds to the upper boundary of the last class ($i.e.\ c_{i((r+2)+1)} = u_i$). Note that $\boldsymbol{C}_i$ contains $r + 3$ elements because there are $r + 2$ classes in the histogram, also, the width of the first class ($[c_{i1},\ c_{i2}]$) and last class ($[c_{i(r+2)},\ c_{i((r+2)+1)}]$) may differ from each other, whereas the r classes inbetween have equal widths [21, 22].

12. Determine the frequency value $\tau_{i\kappa}$ for each class, for decision variable i, that is present in *Elite*.

Informally, the frequency of class $c_{i1} - c_{i2}$ is determined by counting how many solutions in *Elite* are smaller than or equal to c_{i2}. Next, the frequency of class $c_{i2} - c_{i3}$ is determined by counting how many solutions in *Elite* are greater than c_{i2} and smaller than or equal to c_{i3} and so on, until the frequency of class $c_{i7} - c_{i8}$ is determined.

Formally: X_{ij} belongs to the class $[c_{i\kappa},\ c_{i(\kappa+1)})$ if $c_{i\kappa} < X_{ij} \leq c_{i(\kappa+1)}$, where $1 \leq \kappa \leq r+2$ and j is the number of solutions in *Elite*. To conclude, the histogram frequency value τ_{i1} represents the frequency count of decision variable i in the range $[c_{i1},\ c_{i2})$, τ_{i2} represents the count in the range $[c_{i2},\ c_{i3})$, and so on. The new population is formed proportionally, according to the class frequencies for each decision variable [21, 22].

13. Calculate the proportions that determine the number of solutions to generate in each class, respectively, for decision variable i.

Suppose the elite vector contains E_r solutions and $\tau_{i\kappa}$ occurrences in class $[c_{i\kappa},\ c_{i(\kappa+1)})$ for a given decision variable. If E_r is not equal to N, then $\lfloor N_{\tau_{ik}}/E_r \rfloor$ values are created from this range. Note that the proportional numbers may not add up to N because of the rounding down calculation, however the difference is simply added to the last class [21, 22].

14. Calculate the temporary $\mu'_{i\kappa}$ and $\sigma'_{i\kappa}$ values associated with the specific histogram class ranges, *i.e.* for the class $[c_{i\kappa},\ c_{i(\kappa+1)})$ corresponding to the decision variable i. The parameter estimators are given as

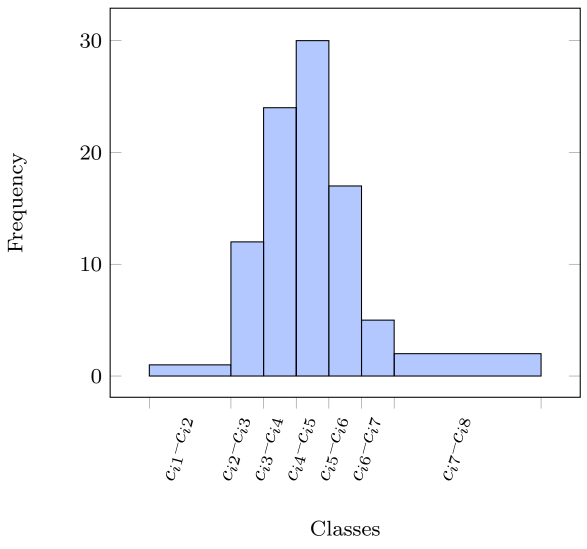
$$\mu'_{i\kappa} = c_{i\kappa} + r_u(c_{i(\kappa+1)} - c_{i\kappa}), \text{ and} \tag{5.7}$$
$$\sigma'_{i\kappa} = c_{i(\kappa+1)} - c_{i\kappa}, \tag{5.8}$$

where $1 \leq \kappa \leq r + 2$ and $r_u \sim \mathcal{U}(0, 1)$ [21, 22].

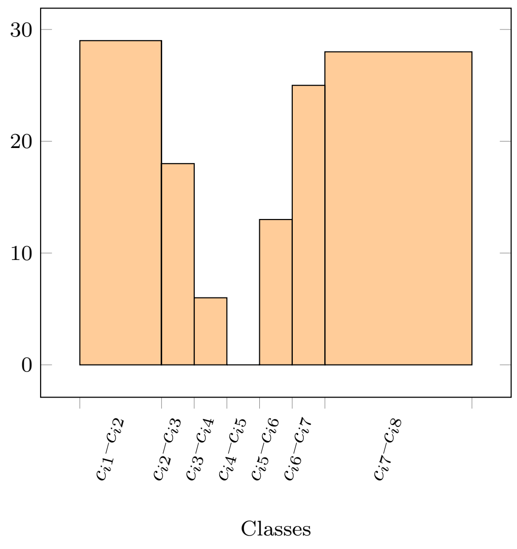
Since *Elite* is used to determine the frequencies, according to the rankings returned by the candidates, classes that do not contribute to the elite vector are effectively eliminated from the search (as the search progresses). For this reason, the histogram frequencies are purposefully inverted to prevent premature convergence albeit with low probability. This is done for each iteration t. Note that the probability of inverting the histogram frequencies is denoted by p_h, where the range is typically $[0.1,\ 0.3]$ [21, 22].

15. Generate a random number $r_u \sim \mathcal{U}(0, 1)$. If $r_u < p_h$ the histogram frequencies are inverted, otherwise the frequencies remain the same and the new population may be generated accordingly.

For decision variable i, the maximum frequency over all class frequencies is determined, denoted as τ_{max}. This corresponds to the frequency of class $c_{i4}-c_{i5}$, as seen in Figure 5.7a. The frequency, for each class ($\tau_{i\kappa}$), is then subtracted from τ_{max}, resulting in an inverted histogram, as shown in Figure 5.7b. After the frequency inversion, search ranges with small proportions of population candidate allocations receive higher proportions of allocations, while search ranges with high proportions of population allocations receive fewer allocations, *i.e.* forcing the algorithm to explore the regions possibly eliminated during the search [21, 22].



(a) *Histogram after iteration t*



(b) *Inverted histogram*

Figure 5.7: *An example illustrating the inversion of the histogram frequencies at iteration t for decision variable i, adapted from [22].*

The number of classes increases as the search progresses which renders it possible to maintain good combinations of decision variable values because the class ranges become smaller and consequently more fine-tuned.

The number of classes should not become too large because the algorithm will become inefficient and may not be larger than the decision variable bound range. For example, the number of classes for the BAP5 with $l = 1$ and $u = 17$, may not exceed $u - l = 16$ because then the boundaries of the classes become real values. Finally, the new population can be generated by sampling from the relevant truncated probability distribution and class frequencies [21, 22].

16. Construct the truncated probability distribution for each decision variable i for each class, using $\mu'_{i\kappa}$ and $\sigma'_{i\kappa}$ and Algorithm 5.1.

17. Generate the new population of possible solutions by sampling from the truncated probability distribution for each class, using Algorithm 5.2.

18. Check for duplicate solutions and then replace them with non-duplicate solutions.

19. Evaluate the new population, *i.e.* the simulation model evaluates N new decision variable combinations.

This concludes the loop which is repeated several times. If all $\sigma_{i,t} < \epsilon_c$, then the ranking threshold is set to $\rho_E \leftarrow 1$, *i.e.* *Elite* contains solutions with $\rho \in \{0,\ 1\}$, and the number of loops k is incremented, consequently the number of classes of the histograms is incremented, $r \leftarrow k + 5$. MOOCEM usually terminates when the value for σ_i of each decision variable has decreased below a user-specified threshold ϵ_c.

As mentioned, the stopping condition employed in this study is fixed at 1 000 solution evaluations (or when $k = k_{\max}$). At termination, the ranking threshold is set to $\rho_E \leftarrow 0$ and *Elite* is ranked one final time, using Algorithm 5.3, and returned as an approximate of the true Pareto set of solutions to the bi-objective simulation optimisation problem under consideration [21, 22]. The pseudo-code description of the entire process described above is provided in Algorithm 5.4, with specific reference to the above-mentioned steps.

Although it is not explicitly reported, when a new generation is generated, the algorithm has to check if duplicate solutions are present, and if so, replace them with non-duplicate solutions. This is a critical step to ensure that the computationally expensive function evaluator (*i.e.* the simulation model) does not evaluate the same solution. Even though checking for duplicate solutions may add to the computational time, it is less computationally expensive than unnecessarily evaluating a solution, this is done for each algorithm.

In conclusion, the properties of the MOOCEM algorithm are stated. The MOOCEM algorithm

1. is a population-based search technique,

2. has a statistical basis,

3. uses an archive (*Elite*) to preserve weakly-dominated solutions during the search and finally non-dominated solutions after the search terminates,

4. employs a Pareto ranking scheme to find non-dominated solutions,

5. achieves proximity to the true Pareto front *via* the convergence mechanism of the CEM,

6. and ensures diversity of the search *via* the inversion of the histogram (with probability p_h) which determines the search regions.

Next, the NSGA-II algorithm is discussed.

Algorithm 5.4: The MOOCEM algorithm, adapted from [21, 22], for the simulation optimisation process

```
   Input  : Previously described using Figures 5.4a and 5.4b.
   Output : The approximate Pareto set ($\mathcal{P}_S$) for the given simulation problem.
1  Initialise the iteration counter $t \leftarrow 1$;
2  Initialise the current loop $k \leftarrow 1$;
3  Initialise $NumEvaluations \leftarrow 0$;              // The current number of evaluations
4  Initialise $NotTerminate \leftarrow$ true;              // Boolean variable
5  Initialise $\mu_i$ and $\sigma_i$;                     // Refer to step 1
6  $Elite \leftarrow \emptyset$;                          // Initialise Elite vector
7  Construct the truncated distribution, using Algorithm 5.1;   // Refer to step 2
8  Generate an initial population, using Algorithm 5.2;        // Refer to step 3
9  Evaluate the initial population;                            // Refer to step 4
10 while $k \leq k_{max}$ and $NumEvaluations < MaxEvaluations$ do
11     $t \leftarrow t + 1$;                                   // Increment $t$
12     Rank the solutions, for $\rho_E = 2$, using Algorithm 5.3;   // Refer to step 5-6
13     if $Elite \neq \emptyset$ then
14         Update $\mu_i$ and $\sigma_i$, using (5.5);         // Refer to step 7
15         if $All\ \sigma_{i,t} < \epsilon_c$ then
16             $NotTerminate \leftarrow$ false;
17     $NotTerminate \leftarrow (NotTerminate$ and $(Nt \leq MaxEvaluations/2))$;
18     $NumEvaluations \leftarrow NumEvaluations + N$;
19     if $NotTerminate = false$ then
20         Rank the solutions, for $\rho_E = 1$, using Algorithm 5.3;
21         $k \leftarrow k + 1$;
22         Update $\mu_i$ and $\sigma_i$, as explained in Step 1;
23         $t \leftarrow 0$;
24         $NotTerminate \leftarrow$ true;
25     else if $NotTerminate = true$ then
26         if $Elite \neq \emptyset$ and $k > 1$ then
27             Follow the steps described in 11–19;            // Refer to step 11-19
28         else
29             Construct the truncated distribution, using Algorithm 5.1;   // Refer to step 8
30             Generate a new population, using Algorithm 5.2;              // Refer to step 9
31             Evaluate the new population;                                 // Similar to step 19
32 Rank the solutions in $Elite$, for $\rho_E \leftarrow 0$, using Algorithm 5.3;
33 $\mathcal{P}_S \leftarrow Elite$;                                       // Approximate Pareto set
```

5.3 Non-dominated sorting genetic algorithm II

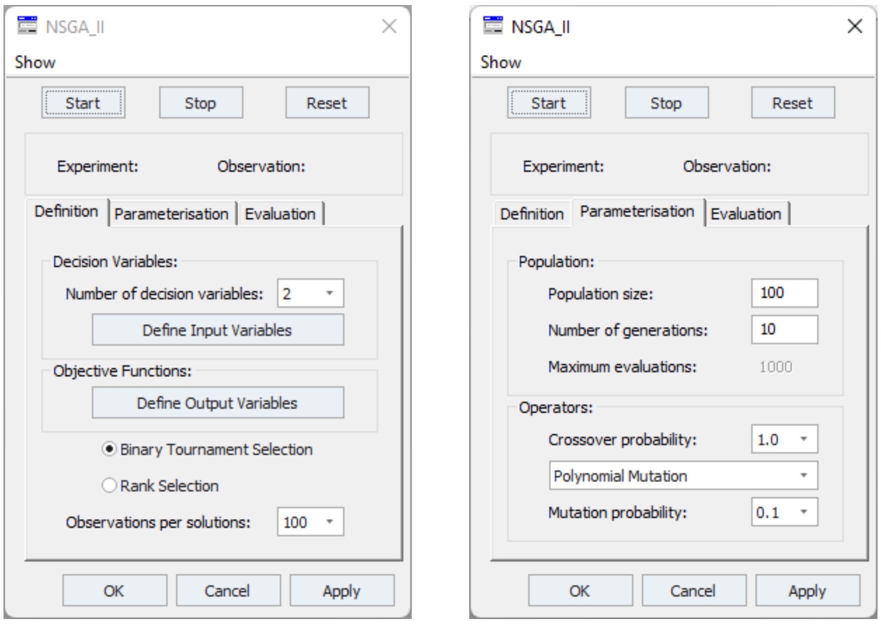
Many multi-objective extensions of the single-objective genetic algorithm have been proposed to solve MOOPs and some have even been improved upon. Deb *et al.* [67] developed the NSGA-II to improve upon NSGA (its predecessor) [238] and is classified under evolutionary algorithms (EAs). NSGA-II was one of the first algorithms to purposefully maintain the diversity of the solution set with the focus of converging towards the true Pareto front [238]. EAs are inspired by the principles of natural selection and evolution [10]. For an introductory survey on EAs the reader is referred to [11], whereas for an introduction to multi-objective EAs (MOEAs), the reader is referred to [47].

The following section starts by presenting a detailed description of the NSGA-II algorithm, supported by the user-interface of the NSGA-II tool used in Tecnomatix, to illustrate the integration of simulation and optimisation. Thereafter, the selection, crossover and mutation operators are discussed. The crossover and mutation operators considered in this study, are real-coded operators applicable to the discrete search space.

5.3.1 The NSGA-II

Figures 5.8a and 5.8b aids with elucidating the simulation optimisation process *via* the NSGA-II tool as it is used in Tecnomatix. The first two steps for using the NSGA-II tool are the same as for the MOOCEM tool as described in §5.2.2, refer to steps 1–2. Accordingly, starting with the *Definition tab* as shown in Figure 5.8a, follow the first two steps of the MOOCEM tool, *i.e.* steps 1 and 2. After steps 1 and 2 have been completed continue with step 3 as described here:

3. Select the selection operator to apply, *i.e.* choose between *Binary tournament selection* and *Rank selection*.



Decision Variables:
Number of decision variables: 2
Objective Functions:
(●) Binary Tournament Selection
( ) Rank Selection
Observations per solutions: 100

Population:
Population size: 100
Number of generations: 10
Maximum evaluations: 1000
Operators:
Crossover probability: 1.0
Polynomial Mutation
Mutation probability: 0.1

(a) *Definition tab* (b) *Parameterisation tab*

Figure 5.8: *The NSGA-II tool user-interface.*

The next step is to navigate to the *Parameterisation tab*, as shown in Figure 5.8b:

4. Input the *Population size* (N), *Number of generations*, *Crossover probability*, *Mutation operator* and *Mutation probability*.

The mutation operator is selected from a drop-down list as shown in Figure 5.9, where the choice is *Polynomial mutation* or *Dynamic mutation*. The above-mentioned parameters are used as input to Algorithm 5.7. To execute the NSGA-II algorithm and optimise the simulation problem at hand, click the *Start* button.

The algorithm starts by initialising the generation counter, $t \leftarrow 0$. Recall that the initial population is generated randomly, using the truncated normal or Poisson distribution as described in §5.2.1, where μ_i and σ_i or λ_i are arbitrarily initialised, as suggested in step 1.

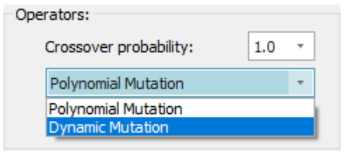


Figure 5.9: *The mutation operators.*

5. Generate the initial parent population $\mathcal{P}_0$, of size N.
6. Evaluate the initial population, *i.e.* the simulation model evaluates N decision variable combinations.

If $t = 0$, then $\mathcal{P}_0$ is the mating pool, *i.e.* no selection operator is applied and the offspring population $\mathcal{Q}_0$ is simply generated from $\mathcal{P}_0$.

8. Generate the offspring population $\mathcal{Q}_0$, of size N, by applying the *reproduction operators*, namely the *crossover operator* with probability p_c and *mutation operator* with probability p_m to the mating pool ($\mathcal{P}_0$).
9. Check for duplicate solutions and then replace them with non-duplicate solutions.

A random number $r_u \sim \mathcal{U}(0, 1)$ is generated. If $r_u < p_c$, crossover is performed, otherwise the solution remains the same, *i.e.* no modification is made. Similarly, if $r_u < p_m$, mutation is performed. Note that different random numbers r_u are generated for each decision step.

9. Evaluate the offspring population ($\mathcal{Q}_0$), *i.e.* the simulation model evaluates N decision variable combinations.

Now, the loop initiates, while the current generation $t < t_{\max}$, where $t_{\max}$ is the fixed number of iterations permitted, divided by the population size N, *i.e.* $t_{\max} = 10$. Do the following:

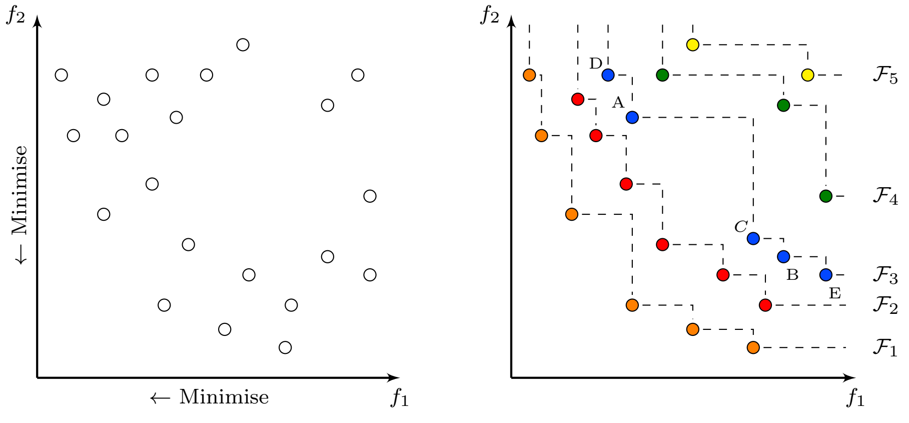
10. Combine the parent and offspring populations, *i.e.* $\mathcal{R}_t \leftarrow \mathcal{P}_t \cup \mathcal{Q}_t$, of size $2N$.
11. Assign fitness values to the combined population $\mathcal{R}_t$.

The fitness values are used to determine which solutions should constitute the mating pool. NSGA-II generally follows the same structure as MOEAs, however, its *fitness assignment* and *selection procedures* distinguish it from other MOEAs. Each solution in the population is assigned two attributes that constitute its fitness, namely a Pareto rank and crowding distance. The procedure is as follows: The first step is to apply the fast non-dominated sorting algorithm (FNSA), described in Algorithm 5.5, in order to rank the solutions in $\mathcal{R}_t$ according to their Pareto rank and partition them into their respective fronts. Thereafter, the solutions within each rank (or front) are ranked again according to their crowding distance by applying the crowding distance sorting and assignment algorithm, as described in Algorithm 5.6 [67]. Algorithm 2.1 is adapted to return the non-dominated fronts $\mathcal{F}_1, \ldots, \mathcal{F}_k$, as seen in Algorithm 5.5. The nature of Algorithm 5.5 is described using Figure 5.10.

The population of solutions in Figure 5.10a is ranked and partitioned into their respective sets and fronts (as depicted in Figure 5.10b). Front 1 (denoted by $\mathcal{F}_1$), contains all the solutions with a rank value of zero and Pareto rank, *i.e.* $\rho_1 = 0$ (denoted by ●), forming the Pareto front.

Algorithm 5.5: Fast non-dominated sorting algorithm, for a *min-max* bi-objective optimisation problem [67]

```
Input  : A population of solutions $\mathcal{P}$.
Output : The population of $\mathcal{P}$ partitioned into $k$ successive non-dominated fronts $\mathcal{F}_1, \ldots, \mathcal{F}_k$
         with a corresponding Pareto rank $\rho_1, \ldots, \rho_k$ and each solution $x$ is also assigned a
         Pareto rank $\rho_x$, where $\mathcal{P}_S \leftarrow \mathcal{F}_1$, containing all the solutions with $\rho = 1$.
1  $\mathcal{P} \leftarrow$ sort$(\mathcal{P}, 1)$;                 // Sort the first objective in descending order
2  for $i \leftarrow 1$ to $|\mathcal{P}|$ do
3      $penalty \leftarrow 0$;                                   // Initialise Pareto rank to zero
4      for $j \leftarrow (i+1)$ to $|\mathcal{P}|$ do
5          if ${f_j}^2 \prec {f_i}^2$ then // i.e. if ${f_j}^2 \geq {f_i}^2$
6              $penalty \leftarrow penalty + 1$;                 // Increment the penalty
7      $\rho_i \leftarrow penalty$;                              // Rank of solution $i$
8  $j \leftarrow 1$;
9  while $\mathcal{F}_j \neq \emptyset$ do ;                     // Populating $\mathcal{F}_1, \ldots, \mathcal{F}_k$
10
11     for $i \leftarrow 1$ to $|\mathcal{P}|$ do
12         if $\rho_i$ = j-1 then
13             $\mathcal{F}_j \leftarrow \mathcal{F}_j \cup \{i\}$;
14     $j \leftarrow j + 1$;
```



(a) *Candidate solutions* (b) *Pareto fronts, $\mathcal{F}_1, \ldots, \mathcal{F}_5$*

Figure 5.10: *An illustration of candidate solutions partitioned into their respective sets and fronts and ranked accordingly, using Algorithm 5.5.*

Next, $\mathcal{F}_2$ comprises all ● solutions with $\rho_2 = 1$, $\mathcal{F}_3$ comprises all ● solutions with $\rho_3 = 2$, and so forth until Front 5 (denoted by ●) is formed containing all the solutions with $\rho_5 = 4$.

As mentioned, Algorithm 5.6 is applied to each non-dominated front obtained by the FNSA. The algorithm takes as input the non-dominated front $\mathcal{F}$ and returns the crowding distances $d_1, \ldots, d_\ell$ as output, where ℓ denotes the number of solutions in $\mathcal{F}$. The first step is to initialise all the crowding distances to zero. Thereafter, the solutions in each objective are sorted in ascending order irrespective of the optimisation context. Next, the distances of the boundary solutions are set to infinity. For the intermediary solutions (*i.e.* 2 to $\ell - 1$), the crowding distance is assigned by calculating the normalised distance between the solutions above and below its rank, using (5.9). The boundary solutions are always added first, with equal probability.

Algorithm 5.6: Crowding distance sorting and assignment algorithm [67]

```
    Input  : A non-dominated front $\mathcal{F}$ of cardinality $\ell$.
    Output : The crowding distances $d_1, \ldots, d_\ell$ for each solution in front $\mathcal{F}$.
1   $\ell \leftarrow |\mathcal{F}|$;                          // Number of solutions in front $\mathcal{F}$
2   for $i \leftarrow 1$ to $\ell$ do
3       $d_i \leftarrow 0$;                       // Initialise crowding distances to zero
4   for $f_k$ to $|f_m|$ do
5       $I^k \leftarrow \mathsf{sort}(\mathcal{F}, k)$;
6       $d_1 = d_\ell = \infty$;
7       for $i \leftarrow 2$ to $(\ell - 1)$ do // Apply to all solutions between the endpoints
8           $d_i \leftarrow d_i + \frac{(f_k^{I[i+1]} - f_k^{I[i-1]})}{(f_k^{\mathsf{max}} - f_k^{\mathsf{min}})}$    (5.9)
```

The fitness values of two solutions may be compared according to the *crowded comparison operator*, given that each solution i is assigned a Pareto rank ρ_i and crowding distance d_i. Accordingly, solution a is said to dominate solution b, denoted as $a \prec_{cc} b$ if one of two scenarios, as stated in (5.10), are true. The first: if the solutions have different Pareto ranks (*i.e.* $\rho_a < \rho_b$) then $a \prec_{cc} b$, the solution with the lowest Pareto rank is preferred, *i.e.* solution a. The second: if two solutions a and b have the same Pareto ranks (*i.e.* $\rho_a = \rho_b$), then the solution with the greater crowding distance is preferred as they promote more diversity, *i.e.* if $d_a > d_b$ then $a \prec_{cc} b$ and solution a is preferred [67]. The crowded comparison operator is formally given as

$$a \prec_{cc} b \quad \text{if} \quad \begin{cases} \rho_a < \rho_b, \text{ or} \\ \rho_a = \rho_b \text{ and } d_a > d_b. \end{cases} \tag{5.10}$$

During the entire search process, an archive is maintained and captured in the elite vector *Elitist* and includes all the non-dominated solutions found at each generation t, *i.e.* the solutions in $\mathcal{F}_1$ with $\rho = 1$.

12. Maintain the elite vector by appending $\mathcal{F}_1$ to *Elitist*, denoted as *Elitist* $\leftarrow$ *Elitist* $\cup \mathcal{F}_1$.

Next, the mating pool $\mathcal{P}_{t+1}$ can be formed based on the crowded comparison operator $\prec_{cc}$ fitness measure, as previously described.

13. Create the new mating pool $\mathcal{P}_{t+1}$ by applying the selection operator.

The selection process is depicted in Figure 5.11. A selection operator is applied to the combined population $\mathcal{R}_t$, consisting of $2N$ solutions. The selection operator selects N solutions to form the mating pool, thereafter two offspring solutions are generated from two parent solutions, *i.e.* N parents are used to generate N offsprings. The selection operator selects solutions based on the crowded comparison operator.

It can be seen that the new population consists of the solutions in $\mathcal{F}_1$ and $\mathcal{F}_2$ because the solutions with the lower Pareto ranks are included first. A subset of $\mathcal{F}_3$ also forms part of the new population, *i.e.* $\mathcal{P}_{t+1} = \{\mathcal{F}_1, \mathcal{F}_2, D\}$, where $D \subset \mathcal{F}_3$.

Next, the offspring population can be generated from $\mathcal{P}_{t+1}$.

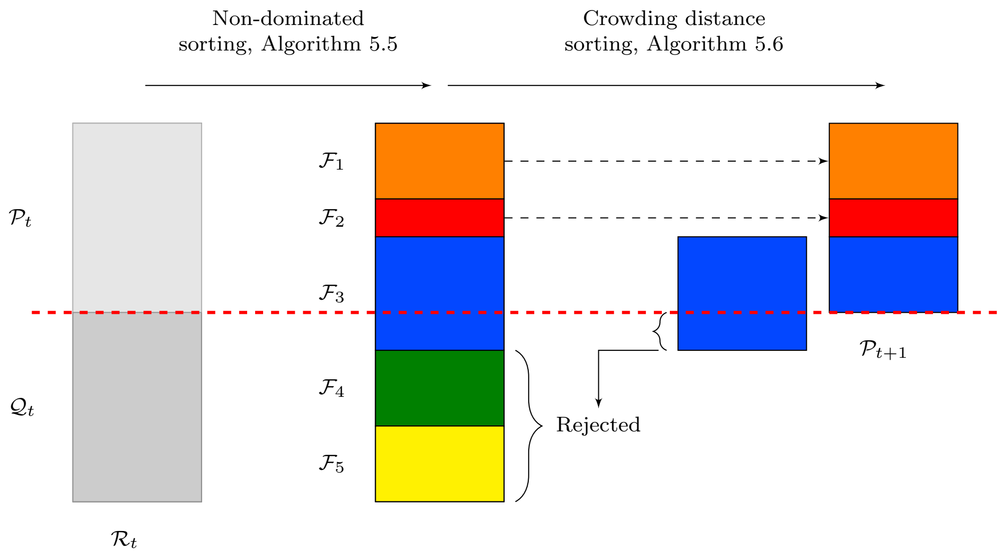


Figure 5.11: *The fitness assignment procedure followed by NSGA-II to generate population $\mathcal{P}_{t+1}$, indicating the partitioning of the parent and offspring populations $\mathcal{P}_t$ and $\mathcal{Q}_t$, adapted from [67].*

14. Generate the offspring population $\mathcal{Q}_t$ by applying the crossover operator with probability p_c and mutation operator with probability p_m to the solutions in the mating pool.

15. Check for duplicate solutions and then replace them with non-duplicate solutions.

Again, a random number $r_u \sim \mathcal{U}(0,1)$ is generated. If $r_u < p_c$ crossover is performed on decision variable i, otherwise, no modification is made. Similarly, a random number $r_u \sim \mathcal{U}(0,1)$ is generated, *i.e.* different random numbers r_u are generated for each decision step. If $r_u < p_m$ then mutation is performed on decision variable i, otherwise, no modification is made.

15. Evaluate the offspring population $\mathcal{Q}_t$, *i.e.* the simulation model evaluates N new decision variable combinations.

16. Increment the generation counter, $t \leftarrow t+1$.

This concludes the loop which is repeated until the stopping condition $t \geq t_{\max}$ is met. At termination, the archive *Elitist* is ranked, according to Algorithm 2.1 and returned as an approximate of the true Pareto set of solutions to the bi-objective simulation optimisation problem under consideration, *i.e.* $\mathcal{P}_S \leftarrow Elitist$. Recall that (although it is not explicitly reported) when a new generation is generated, the algorithm has to check if duplicate solutions are present, and if so, replace them with non-duplicate solutions to ensure that the expensive function evaluator does not unnecessarily evaluate a solution and is done for each algorithm. A pseudo-code description of the entire process described above is provided in Algorithm 5.7, with specific reference to the above-mentioned steps.

During each generation, solutions are selected based on fitness values (and not objective function values) to form the parent population [149]. The crowded comparison operator fitness measure ($\prec_{cc}$) enables the selection process used in NSGA-II to favour the exploration of diverse solutions [67], refer to the discussion around step 11. Next, an offspring population is created by applying reproduction operators (crossover and mutation) to the solutions in the parent population.

Algorithm 5.7: NSGA-II, adapted from [67, 222], for the simulation optimisation process

```
Input  : Previously described using Figures 5.8a and 5.8b.
Output : The approximate Pareto set ($\mathcal{P}_S$) for the given simulation problem.
1  $t \leftarrow 0$;                                                  // Initialise the generation counter
2  $Elitist \leftarrow \emptyset$;                                    // Initialise Elitist
3  Generate an initial population $\mathcal{P}_0$;                    // Refer to step 5
4  Evaluate $\mathcal{P}_0$;                                          // Refer to step 6
5  Generate the offspring population $\mathcal{Q}_0$;                 // Refer to step 8
6  Evaluate $\mathcal{Q}_0$;                                          // Refer to step 9
7  while $t < t_{max}$ do
8      $\mathcal{R}_t \leftarrow \mathcal{P}_t \cup \mathcal{Q}_t$;   // Refer to step 10
9      Assign fitness values, using Algorithm 5.5 and 5.6 respectively; // Refer to step 11
10     $Elitist \leftarrow Elitist \cup \mathcal{F}_1$;               // Refer to step 12
11     Create the new mating pool of solutions ($\mathcal{P}_{t+1}$); // Refer to step 13
12     Generate offspring population $\mathcal{Q}_{t+1}$;             // Refer to step 14
13     Evaluate $\mathcal{Q}_{t+1}$;                                  // Refer to step 15
14     $t \leftarrow t + 1$;                                          // Refer to step 16
15 Rank the solutions in Elitist, using Algorithm 2.1;                // Get non-dominated solutions
16 $\mathcal{P}_S \leftarrow Elitist$;                                // Approximate Pareto set
```

Finally, a replacement scheme determines which solutions of the population will survive from the offspring and parent populations. This process, as depicted in Figure 5.12, is iterated until a stopping condition is met [93].

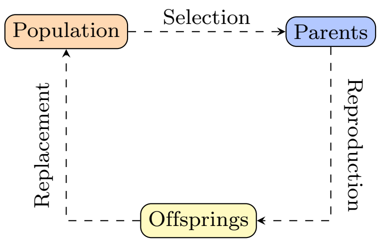


Figure 5.12: *An illustration of a generation in EAs, adapted from [93].*

Selection pressure is the paradigm in which solutions that have superior fitness values are selected with a higher probability, without it — the search process is arbitrary. Selection pressure and diversity preservation are complimentary [255]. The selection pressure guides the population to better solutions resulting in a loss of population diversity. Conversely, preserving population diversity means selecting worse solutions thereby negating the effect of selection pressure. Consequently, a trade-off exits and should be balanced to obtain a diverse population in close proximity to the true Pareto front, *i.e.* aiding the exploration and exploitation of the search and not premature convergence [171, 175]. The selection operators considered for its inclusion in this study are discussed next.

5.3.2 Selection operators

As mentioned, the selection operator determines which solutions in the parent population are used to reproduce (by method of reproduction operators) to create the offspring population. This strategy is based on the principle of *survival of the fittest*, where superior parents typically result in better and fitter offsprings. This selection process is critical as it guides the search toward the true Pareto front [93].

There are many selection strategies, namely, tournament selection, rank selection and roulette-wheel selection, refer to [11, 14, 32, 103]. *Roulette-wheel selection* is dependent on the optimisation direction, whereas tournament and rank selection are not. For more information regarding roulette-wheel selection, the reader is referred to [209]. During tournament selection, k solutions are selected at random from the combined population R_t, which compete against each other. The solution with the best fitness value, as discussed in step 11, wins and is included in the mating pool. In this study, the *tournament size* (or number of solutions competing) is $k = 2$, referred to as *binary tournament selection* (BTS) [93, 209].

Rank selection (RS) simply selects the solutions with the best fitness values, as discussed in step 11 in §5.3.1, *i.e.* the mating pool consists of $\mathcal{F}_1, \mathcal{F}_2, \ldots$ until the mating pool consists of N solutions. Note that binary tournament selection enables solutions with lower fitness values to enter the mating pool. Rank selection also enables solutions with lower fitness values to enter the mating pool, however, solutions with the greatest fitness values are selected first and consequently, if the number of solutions in Front 1 is equal to the population size (*i.e.* $|\mathcal{F}_1| = N$) then only non-dominated solutions are selected to enter the mating pool.

The decision to incorporate discrete vector representations is further motivated by the disadvantages of binary representations. The process in Figure 5.13 is as follows: crossover and mutation are performed in respect of the binary string (the genotype or encoded solution), which is then decoded (as the phenotype) and evaluated by the simulation model evaluator, *i.e.* the reproduction operators act on the genotype level while the simulation model evaluator uses the phenotype of the associated genotype. At this point the selection operator is applied and a mating pool created.

The selected parents are then encoded and crossover as well as mutation are performed on the encoded solutions, and so on. Real-coded reproduction operators have been suggested in the literature [64, 66, 69, 70] and adapted for discrete vector representations due to the unnecessary computational inefficiency of encoding and decoding solutions.

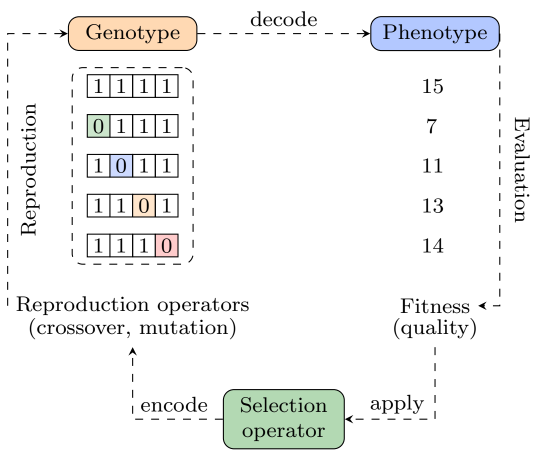


Figure 5.13: *Illustration of the effect that changing a single bit has on the value the string represents, adapted from [93].*

Furthermore, in order to move to a neighbouring solution, more than one bit might need to be changed which introduces incumbrance, as changing one bit does not translate into a neighbour. An example is presented in Figure 5.13, with specific focus on the binary string [1 1 1 1] = 15 — by converting a single bit the string can take the following values {7, 11, 13, 14}, two of which are neighbours and two that are not. There is, however, the option of converting the binary code to so-called Gray code, where changing one bit would result in a neighbouring solution.

However, the challenge remains of keeping the genotypes within the bounds of the decision variables and requires considerable effort. Consequently, a so-called direct encoding is used, where the genotype is similar to the phenotype. The type of operations that can be applied to a solution depends on its representation, *i.e.* if the solution representation is discrete then the application of the reproduction operators should modify the discrete value and return a discrete value for the solution and is dependent on the problem under consideration. Many optimisation problems have discrete search spaces [221, 252], as do the simulation problems in this study.

For this reason, techniques exist to adapt continuous heuristics, for example the recombination operators, for the discrete search space [52]. In this study, the technique of rounding off is used, *i.e.* the value generated by the continuous heuristic is simply rounded off to the nearest integer value. This approach was first used by Yoshida *et al.* [261] and is used in this study due to its simplicity and low computational cost.

The real-coded crossover operators that have been adapted for discrete search spaces and that are considered for its inclusion in this study, are discussed next.

5.3.3 Crossover operators

The next step in creating the offspring population is *crossover*. A crossover probability p_c is specified by the user and represents the proportion of crossover operations, *i.e.* p_c determines whether or not the crossover operator is applied. Crossover operators are n-ary operators [93]. In this study, the crossover operators are 2-ary (or binary), *i.e.* the crossover operator modifies two solutions in the parent population, simultaneously.

The general procedure is as follows: generate a random number $r_u \sim \mathcal{U}(0, 1)$, if $r_u < p_c$ the crossover is applied to two parent solutions. Crossover creates two new offspring solutions by using parts of both parent solutions with the purpose of inheriting some characteristics of both parents. The offsprings may potentially be generated far from the region being sampled, depending on how far the parent solutions are from each other and the crossover operator used, consequently introducing diversity into the population [104]. The primary task of the crossover operator is therefore to search the neighbourhood of the parent solutions without any bias to specific neighbourhoods. It is the task of the selection operator to guide the search towards better, fitter neighbourhoods [64]. As mentioned, if $r_u \geq p_c$, then the solution remains unchanged, *i.e.* the parents become the offsprings.

There are several real-coded crossover operators, namely *simplex crossover* [247], *simulated binary crossover* (SBX) [62, 66], *Laplace crossover* (LX) [69] and *blend crossover*, to name a few. SBX and LX are considered *parent-centric*, *i.e.* each parent has an equal probability of creating an offspring in its neighbourhood [93].

Simulated binary crossover

SBX uses a probability distribution function $P(\beta)$ to simulate the single-point crossover operator, as used in binary-coded operators [62, 66], to generate two offspring solutions.

The probability distribution function is given as

$$P(\beta) = \begin{cases} 0.5(\eta + 1)\beta^{\eta}, & \text{if } \beta \leq 1 \\ 0.5(\eta + 1)\beta^{\frac{1}{\eta+2}}, & \text{otherwise.} \end{cases} \tag{5.11}$$

The distribution index η controls the distance between the offsprings and the parents, where a large η results in a higher probability of generating offsprings that are near their parents.

For MOO the interval value for η is suggested as $[5, 10]$ [93]. The procedure of computing the offspring solutions x_1^{t+1} and x_2^{t+1} from parent solutions x_1^t and x_2^t is described in Algorithm 5.8, assuming $r_u < p_c$. First, a random number $r_u \sim \mathcal{U}(0, 1)$ is generated, thereafter the ordinate β can be determined by solving for (5.12) using r_u and specified η. Finally, the offsprings can be computed. For more detail, the reader is referred to [62, 66, 93].

Algorithm 5.8: Simulated binary crossover, adapted for the discrete search space [62, 66]

```
  Input  : Two parent solutions $x_1^t$ and $x_2^t$ and a specified value for the distribution index $\eta$.
  Output : Two discrete valued offspring solutions $x_1^{t+1}$ and $x_2^{t+1}$.
1 Generate a random number, $r_u \sim \mathcal{U}(0, 1)$;
2 Calculate $\beta$ as
```

$$\beta = \begin{cases} 2r_u^{\frac{1}{\eta+1}}, & \text{if } r_u \leq 0.5 \\ \left[\frac{1}{2(1-r_u)}\right]^{\frac{1}{\eta+1}}, & \text{otherwise.} \end{cases} \tag{5.12}$$

```
3 Compute offsprings as
```

$$\begin{aligned} x_1^{t+1} &= 0.5\lfloor(1+\beta)x_1^t + (1-\beta)x_2^t\rfloor \\ x_2^{t+1} &= 0.5\lfloor(1-\beta)x_1^t + (1+\beta)x_2^t\rfloor \end{aligned}$$

Laplace crossover

The LX operator uses the Laplace distribution to create offsprings. The procedure of computing the offspring solutions x_1^{t+1} and x_2^{t+1} from parent solutions x_1^t and x_2^t is described in Algorithm 5.9, assuming $r_u < p_c$.

Algorithm 5.9: Laplace crossover, adapted for the discrete search space [69]

```
  Input  : Two parent solutions $x_1^t$ and $x_2^t$ and specified values for the location parameter $a$ and
           the scale parameter $b$.
  Output : Two discrete valued offspring solutions $x_1^{t+1}$ and $x_2^{t+1}$.
1 Generate a random number, $r_u \sim \mathcal{U}(0, 1)$;
2 Calculate $\beta$ as
```

$$\beta = \begin{cases} a - b\log_e(r_u), & \text{if } r_u \leq 0.5 \\ a + b\log_e(r_u), & \text{otherwise.} \end{cases} \tag{5.13}$$

```
3 Compute offsprings as
```

$$\begin{aligned} x_1^{t+1} &= x_1^t + \lfloor\beta \ \mathsf{abs}(x_1^t - x_2^t)\rfloor \\ x_2^{t+1} &= x_2^t + \lfloor\beta \ \mathsf{abs}(x_1^t - x_2^t)\rfloor \end{aligned}$$

Similar to SBX, the value of b (the scale parameter) influences the distance between the offsprings and the parents. However, for smaller values of b, offsprings are likely to be near the parents. For fixed values of a and b, the offspring are created proportional to the parents, *i.e.* the offspring are expected to be near each other if the parents are near each other, and *vice versa* [69]. First, a random number $r_u \sim \mathcal{U}(0, 1)$ is generated, thereafter β can be determined by solving for (5.13) using r_u and specified values for a and b, then the offsprings can be computed.

Blend crossover

The BX procedure of computing offsprings x_1^{t+1} and x_2^{t+1} from x_1^t and x_2^t is described in Algorithm 5.10, assuming $r_u < p_c$. First, a random number $r_u \sim \mathcal{U}(0, 1)$ is generated, thereafter the offsprings are created randomly in the range $[x_1^t - r_u(x_2^t - x_1^t),\ x_2^t + r_u(x_2^t - x_1^t)]$ [203].

Algorithm 5.10: **Blend crossover, adapted for the discrete search space [203]**

```
   Input  : Two parent solutions $x_1^t$ and $x_2^t$, where $x_1^t < x_2^t$.
   Output : Two discrete valued offspring solutions $x_1^{t+1}$ and $x_2^{t+1}$.
1  Generate a random number, $r_u \sim \mathcal{U}(0, 1)$;
2  Compute offsprings as
       $x_1^{t+1} = \lfloor r_u x_1^t + (1 - r_u) x_2^t \rceil$
       $x_2^{t+1} = \lfloor (1 - r_u) x_1^t + r_u x_2^t \rceil$
```

Upon initial experimentation, the LX and BX operators proved to be troublesome, generating infeasible offsprings and proving difficult to keep values within bound irrespective of the user-specified parameter values (*e.g.* the location or scale parameters). Consequently, LX and BX are excluded from further consideration.

Next, the real-coded mutation operators adapted for discrete search spaces and considered for its inclusion in this study, are discussed.

5.3.4 Mutation operators

After crossover, the next step in creating the offspring population, is *mutation*. A mutation probability p_m is specified by the user and is typically markedly lower than the crossover probability. In contrast to crossover operators, mutation operators are *unary operators*, modifying a single solution of the parent population independent of the remainder of the population [65]. The general procedure is as follows: a uniform random number $r_u \sim \mathcal{U}(0, 1)$ is generated, if $r_u < p_m$ the solution is mutated, *i.e.* the current variable value (which is the parent if crossover was not performed or offspring if crossover was performed) is perturbed to a neighbouring value (the offspring). This is performed for each respective decision variable and each solution.

Mutation introduces and maintains diversity in the population [104, 119]. At the start of the search, it may be desirable that the mutation operator favours values markedly different from the current solution, but as the search progresses it should favour values more similar to the parent solutions [203]. As mentioned, if $r_u \geq p_m$, then the solution remains unchanged. There are several real-coded mutation operators, namely *random mutation* [57], *Gaussian mutation* [225], *polynomial mutation* (PM) [59, 63], *power mutation* [70], *dynamic mutation* (DM) [203], *non-uniform mutation*, to name a few. PM and DM operators create solutions within the user-specified range $[l, u]$, naturally motivating their inclusion in this study.

Polynomial mutation operator

PM uses the polynomial probability distribution, with a user-defined distribution index ($\eta_m \approx 20$), to perturb a solution near the parent (or crossover solution) [93]. The mutation operator creates a solution x' in the range $[l, u]$, by mutating x as described in Algorithm 5.11, assuming

$r_u < p_m$. First, a random number $r_u \sim \mathcal{U}(0,1)$ is generated, thereafter δ can be determined by solving for (5.14) using r_u and specified η_m. Finally, the mutated offspring can be computed.

```
Algorithm 5.11: Polynomial mutation, adapted for the discrete search space [65, 93]
    Input  : Solution $x$ and distribution index $\eta_m$. If crossover was not applied, then $x$ is the parent
             solution, otherwise $x$ is the offspring solution created by from the crossover operation.
    Output : Mutated offspring $x'$.
1   Generate a random number, $r_u \sim \mathcal{U}(0,1)$;
2   Calculate $\delta$ as
        $\delta = \begin{cases} 2r_u^{\frac{1}{\eta_m+1}} - 1, & \text{if } r_u < 0.5 \\ 1 - 2(1-r_u)^{\frac{1}{\eta_m+1}}, & \text{otherwise.} \end{cases}$   (5.14)
3   Apply mutation operator
        $x' = \begin{cases} x + \lfloor (x-l)\delta \rfloor, & \text{if } r_u \leq 0.5, \\ x + \lfloor (u-x)\delta \rfloor, & \text{otherwise.} \end{cases}$
```

Dynamic mutation

As mentioned, it may be desirable to start out with a mutated solution where all possible values are equally probable (called uniform mutation) and as the search progresses to focus the search near the parent solutions [203]. The mutated solution x' is created from solution x, for $r_u \sim \mathcal{U}(0,1)$, as described in Algorithm 5.12, assuming $r_u < p_m$. First, a random number $r_u \sim \mathcal{U}(0,1)$ is generated, thereafter α can be determined by solving for (5.15) using r_u and specified γ. Finally, the mutated offspring can be computed.

```
Algorithm 5.12: Dynamic mutation, adapted for the discrete search space [203]
    Input  : Solution $x$ and mutation parameter $\gamma$. If crossover was not applied, then $x$ is the parent
             solution, otherwise $x$ is the offspring solution created by from the crossover operation.
    Output : Mutated offspring $x'$.
1   Generate a random number, $r_u \sim \mathcal{U}(0,1)$;
2   Calculate $\xi$ as
        $\xi = \left(1 - \frac{t-1}{t_{\max}}\right)^{\gamma},$   (5.15)
      where $t$ is the current generation count, $t_{\max}$ the total number of generations and $\beta$ the user
      specified mutation parameter. For the purposes of this study $\beta = 5$.
      Apply mutation operator
        $x' = \begin{cases} l + \lfloor (r_u - l)^{\xi}(x-l)^{1-\xi} \rfloor, & \text{if } r_u \leq x, \\ u - \lfloor (u-r_u)^{\xi}(u-x)^{1-\xi} \rfloor, & \text{otherwise.} \end{cases}$
```

In conclusion, the properties of the NSGA-II algorithm are stated. The NSGA-II algorithm

1. is a population-based search technique,
2. has a biological basis,

3. uses an archive to preserve non-dominated solutions (*Elitist*) during the search,

4. employs a Pareto ranking scheme to find non-dominated solutions,

5. achieves proximity to the true Pareto front *via* the selection operator,

6. and ensures diversity of the search *via* the crowded comparison operator, crossover and mutation operators.

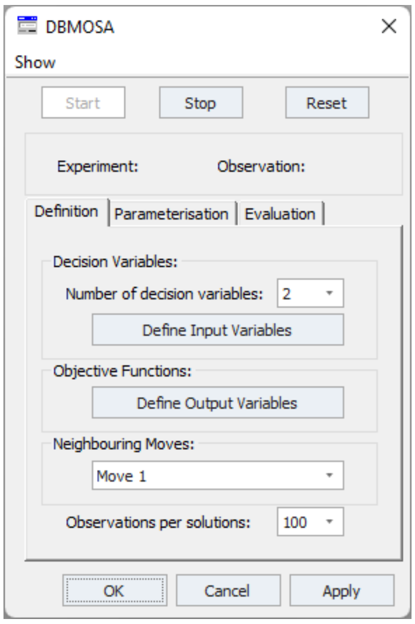
Finally the DBMOSA algorithm is discussed.

5.4 Dominance-based bi-objective simulated annealing

The single-objective SA algorithm has been adapted to solve MOOPs, DBMOSA being one such manifestation. SA was first proposed by Kirkpatrick *et al.* [143] and later Smith *et al.* developed the multi-objective adaptation DBMOSA [233]. For a summative description of the different multi-objective extensions of SA, the reader is referred to [236]. The following section provides a detailed description of the DBMOSA algorithm, supported by the user-interface of the DBMOSA tool used in Tecnomatix, to illustrate the integration of simulation and optimisation.

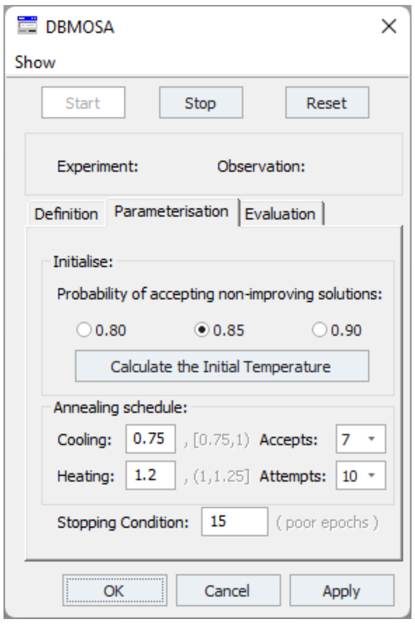
5.4.1 The DBMOSA algorithm

Figures 5.14a and 5.14b are purposefully included to elucidate the simulation optimisation process *via* the DBMOSA tool as it is used in Tecnomatix. The steps for using the DBMOSA tool are as follows: starting with the *Definition tab* as shown in Figure 5.14a.



Experiment:
Observation:
Decision Variables:
Number of decision variables: 2
Objective Functions:
Neighbouring Moves: Move 1
Observations per solutions: 100

(a) *Definition tab*



Experiment:
Observation:
Initialise:
Probability of accepting non-improving solutions: ( ) 0.80 (● 0.85) ( ) 0.90
Annealing schedule:
Cooling: 0.75 , [0.75,1) Accepts: 7
Heating: 1.2 , (1,1.25] Attempts: 10
Stopping Condition: 15 (poor epochs)

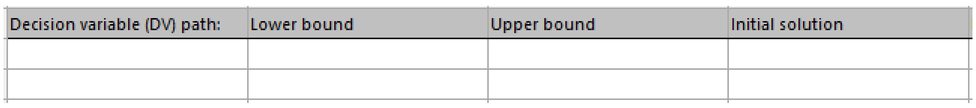
(b) *Parameterisation tab*

Figure 5.14: *The DBMOSA tool user-interface.*

1. Define the number of decision variables, then click the *Define Input Variables* button and define the variables in terms of their paths, lower and upper bounds and initial solutions, as seen in Figure 5.15.

2. Define the output variables, click the *Define Output Variables* button, in terms of their paths and select the optimisation direction, *i.e.* *Min* or *Max* as seen in Figure 5.5b.

3. Choose the neighbouring move operator that should be applied. Refer to §5.4.4 for the detailed documentation regarding the respective move operators.



Decision variable (DV) path:	Lower bound	Upper bound	Initial solution

Figure 5.15: *The user-interface for the decision variable table used for DBMOSA in Tecnomatix.*

The next step is to navigate to the *Parameterisation tab*, as shown in Figure 5.14b:

4. Input the *Cooling* and *Heating* schedule parameters, select the number of *Accepts* before the temperature should decrease and the number of *Attempts* before it should increase.

5. Input the *Stopping Condition*, *i.e.* the number of poor epochs, denoted by $maxpoorEpochs$.

6. Select the probability of accepting non-improving moves and then click the *Calculate the Initial Temperature* button, refer to §5.4.3.

After the initial temperature has been calculated the *Start* button appears. The above-mentioned parameters are used as input to Algorithm 5.13, click the *Start* button to execute the DBMOSA algorithm and optimise the simulation problem at hand. As mentioned, the initial solution is generated randomly, represented by $\boldsymbol{x}$. The algorithm starts by initialising the iteration counter, $t \leftarrow 1$, the counters $accepts \leftarrow 1$ (since the initial solution is accepted), $attempts \leftarrow 0$, poor epochs, $poorEpochs \leftarrow 0$ and epochs, $epochs \leftarrow\ 0$.

1. Evaluate the initial solution $(\boldsymbol{x})$, *i.e.* the simulation model evaluates $\boldsymbol{x}$.

An archive $\mathcal{A}$ is maintained throughout the search process and contains all the non-dominated solutions found. Because $\boldsymbol{x}$ is the first solution evaluated it is considered non-dominated and enters the archive. However, as the search progresses, $\boldsymbol{x}$ might become dominated after which it is removed from the archive.

2. Initialise the archive to contain the initial solution, *i.e.* $\mathcal{A} \leftarrow \boldsymbol{x}$.

Now the loop initiates, while the current iteration is less than or equal to the fixed number of iterations permitted (*i.e.* $t \leq 1\,000$), do the following:

3. Increment the iteration counter, $t \leftarrow t + 1$.

4. Generate a neighbouring solution $\boldsymbol{x'}$ from the previously accepted solution $\boldsymbol{x}$ according to the selected move operator (Move operator 1–4), denoted by $\boldsymbol{x'} \leftarrow \textsf{move operator}(\boldsymbol{x})$. Refer to §5.4.4.

5. Check if a duplicate solution are present and then replace it with a non-duplicate solution.

6. Evaluate $\boldsymbol{x'}$, *i.e.* the simulation model evaluates $\boldsymbol{x'}$.

Smith *et al.* [233] proposed the use of an energy function which is defined in terms of the proposed archive ($\tilde{\mathcal{A}}$) containing all the non-dominated solutions found as well as the current and neighbouring solutions, *i.e.* $\tilde{\mathcal{A}} = \mathcal{A} \cup \{\boldsymbol{x}\} \cup \{\boldsymbol{x'}\}$. Consequently, the change in energy for the current solution $\boldsymbol{x}$ and neighbouring solution $\boldsymbol{x'}$ may be expressed as

$$\Delta_E(\boldsymbol{x'}, \boldsymbol{x}) = \frac{|\tilde{\mathcal{A}}_{\boldsymbol{x'}}| - |\tilde{\mathcal{A}}_{\boldsymbol{x}}|}{|\tilde{\mathcal{A}}|}, \tag{5.16}$$

where the difference is normalised by dividing by $|\tilde{\mathcal{A}}|$ (the number of solutions in the archive). It has been reported that normalising the energy promotes both coverage of and convergence towards the true Pareto set. The archive $\tilde{\mathcal{A}}$ includes the current solution to ensure that $\Delta_E(\boldsymbol{x'}, \boldsymbol{x}) < 0$ when the neighbouring solution dominates the current solution (*i.e.* $\boldsymbol{x'} \prec \boldsymbol{x}$) [233].

In Figure 5.16 an illustration is presented of the nature of the energy measure for the current solution $\boldsymbol{x}$ and neighbouring solution $\boldsymbol{x'}$, where the number of solutions in $\tilde{\mathcal{A}}$ is 11 (denoted as $|\tilde{\mathcal{A}}| = 11$) and $|\mathcal{A}| = 9$. It can be seen that $\boldsymbol{x}$ is dominated by 3 solutions in the archive (denoted as $|\tilde{\mathcal{A}}_{\boldsymbol{x}}| = 3$), and $\boldsymbol{x'}$ is dominated by 1 solution (denoted as $|\tilde{\mathcal{A}}_{\boldsymbol{x'}}| = 1$). The resulting change in energy of $\Delta_E(\boldsymbol{x'}, \boldsymbol{x}) = \frac{1-3}{11} = -\frac{2}{11}$. Therefore, accepting $\boldsymbol{x'}$ as the new current solution for the next iteration would encourage exploration by guiding the search towards less explored neighbourhoods of the non-dominated front (this is because $\boldsymbol{x'}$ is dominated by fewer solutions than $\boldsymbol{x}$) [128].

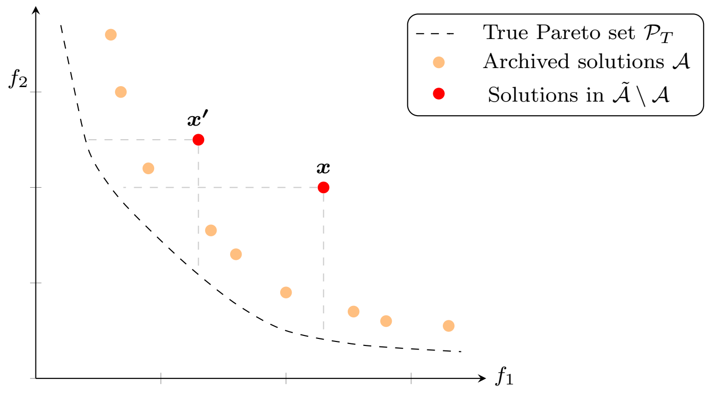


Figure 5.16: *An example illustrating the energy measure of current solution $\boldsymbol{x}$ and neighbouring solution $\boldsymbol{x'}$ for a min–min bi-objective problem with objective functions f_1 and f_2, adapted from [128].*

5. Define the proposed archive $\tilde{\mathcal{A}}$ that includes the current and the neighbouring solution, *i.e.* $\tilde{\mathcal{A}} \leftarrow \{\boldsymbol{x},\ \boldsymbol{x'}\}$. The number of solutions in the archive is (currently) two.

6. Define the subset of solutions in $\tilde{\mathcal{A}}$ that dominates $\boldsymbol{x}$ (in terms of objective function values), *i.e.* $\tilde{\mathcal{A}}_{\boldsymbol{x}} = \{\mathbf{y} \in \mathrm{a}\mathcal{P}_S \mid \mathbf{y} \prec \boldsymbol{x}\}$.

7. Define the subset of solutions in $\tilde{\mathcal{A}}$ that dominates $\boldsymbol{x'}$ (in terms of objective function values), *i.e.* $\tilde{\mathcal{A}}_{\boldsymbol{x'}} = \{\mathbf{y} \in \mathrm{a}\mathcal{P}_S \mid \mathbf{y} \prec \boldsymbol{x'}\}$.

8. Calculate the change in energy (Δ_E) for $\boldsymbol{x}$ and $\boldsymbol{x'}$ using (5.16).

Every solution that is *accepted* during the search, is a candidate for inclusion in the archive ($\mathcal{A}$). If the accepted solution is not dominated by any solution in the archive, it is included. If the accepted solution dominates solutions currently in the archive, it is included, and the dominated solutions are removed [233]. If, however the accepted solution is dominated by any other solution in the archive, it is not included, as discussed in §2.2.3.

10. The objective values of $\boldsymbol{x}$ and $\boldsymbol{x'}$ are compared and if $\boldsymbol{x'}$ is not dominated by any other solution in the archive, *i.e.* $|\mathcal{A}_{\boldsymbol{x'}}| = 0$, it is accepted as the new current solution for the next iteration (with a probability of 1). Otherwise, $\boldsymbol{x'}$ is accepted stochastically, according to the *Metropolis acceptance rule* [180], where the probability of accepting a non-improving neighbouring solution is given in (5.17).

To elucidate this further, take for example a bi-objective *max-max* problem with two decision variables $\{x_1,\ x_2\}$, if $f(\boldsymbol{x'_1}, \boldsymbol{x'_2}) \geq f(\boldsymbol{x_1}, \boldsymbol{x_2})$, then $\boldsymbol{x'}$ is accepted with a probability 1, otherwise $\boldsymbol{x'}$ is accepted according to the Metropolis acceptance rule, given as

$$P_r(\boldsymbol{x'}) = \min\left\{1,\ \exp^{\left(-\frac{\Delta_E(\boldsymbol{x'},\boldsymbol{x})}{q_t}\right)}\right\}, \tag{5.17}$$

where q_t is the temperature at iteration t [233].

11. Generate a random number $r_u \sim \mathcal{U}(0, 1)$. If $r_u < P_r(\boldsymbol{x'})$ then the neighbouring solution enters the archive ($\tilde{\mathcal{A}}$) and becomes the new current solution, *i.e.* $\boldsymbol{x} \leftarrow \boldsymbol{x'}$.

DBMOSA is executed iteratively, each iteration is known as an epoch, where the temperature q_t remains constant during an epoch and only increases or decreases when an epoch is incremented. An epoch is incremented if either *maxAccepts* or *maxAttempts* are reached. Suppose the maximum number of attempts (or accepts) are defined as 5, denoted as *maxAttempts* $\leftarrow$ 5 (or *maxAccepts* $\leftarrow$ 5), then *epochs* is incremented if *attempts* = 5 (or if *accepts* = 5), irrespective if the attempts or accepts occured successively.

12. If the solution is accepted then *accepts* is incremented, *i.e. accepts* $\leftarrow$ *accepts* + 1.
13. If *accepts* $\geq$ *maxAccepts* then *epochs* $\leftarrow$ *epochs* + 1 and the temperature q_t is cooled. Refer to §5.4.2.
14. If the solution is attempted then *attempts* is incremented, *i.e. attempts* $\leftarrow$ *attempts* + 1.
15. If *attempts* $\geq$ *maxAttempts* then *epochs* $\leftarrow$ *epochs* + 1 and the temperature is heated. Refer to §5.4.2.

Again, suppose *maxAttempts* $\leftarrow$ 5, then *poorEpochs* is incremented when *maxAttempts* number of *successive epochs* have elapsed without accepting a solution. This is illustrated in the example given in Table 5.1. It can be seen that after 5 solution evaluations are attempted, *epochs* is incremented and if 5 epochs are incremented consecutively, without accepting a solution then *poorEpochs* is incremented, *i.e.* if 25 attempts are made then *poorEpochs* $\leftarrow$ *poorEpochs*.

16. If a solution is attempted (*i.e.* not accepted) *maxAttempts* consecutive number of times then *poorEpochs* is incemented, *i.e. poorEpochs* $\leftarrow$ *poorEpochs* + 1.

Table 5.1: *An example illustrating the incrementing process followed for an epoch and a poorEpoch.*

	attemps		*epochs*		*poorEpochs*
1	✓		0		0
2	✓		0		0
3	✓		0		0
4	✓		0		0
5	✓	+1	1		0
⋮	⋮		⋮		⋮
21	✓		4		0
22	✓		4		0
23	✓		4		0
24	✓		4		0
25	✓	+1	5	+1	1

This concludes the loop. The algorithm terminates when the current iteration is greater than the maximum number of iterations permitted, *i.e.* $t > t_{\max}$, where $t_{\max} = 1\,000$ solution evaluations. At termination, the archive $\mathcal{A}$ is returned as an approximate of the true Pareto set of solutions to the bi-objective simulation optimisation problem under consideration, *i.e.* $\mathcal{P}_S \leftarrow \mathcal{A}$.

It is worth mentioning that there are other stopping conditions that may be implemented, namely terminating when the number of $poorEpochs \geq maxPoorEpochs$ (as depicted in the user-interface) or when $q_t \geq q_f$, where q_f is a user-specified final temperature. Specific reference is made to this in Algorithm 5.13.

Although it is not explicitly reported, when a neighbouring solution is generated, the algorithm has to check if it is a duplicate solution, and if so, replace it with a non-duplicate solution. Again, this is done to ensure that the expensive function evaluator does not unnecessarily evaluate a solution and is done for each algorithm.

A pseudo-code description of the entire process described above is provided in Algorithm 5.13, with specific reference to the above-mentioned steps.

The temperature q_t controls the probability of accepting non-improving solutions. If the temperature is high, the probability of accepting a non-improving solution is greater and *vice versa*. The temperature is controlled by the chosen *annealing schedule*, this process and the annealing schedule adopted in this study is discussed next.

5.4.2 The annealing schedule

The *annealing schedule* determines by how much the temperature is decreased (using the cooling schedule) or increased (using the reheating schedule). Recall that the temperature is only decreased or increased if an epoch is incremented, which is based on what happens during the search, *i.e.* how many solutions are accepted or not. If an excessive number of solutions are accepted, the cooling schedule decreases the temperature at some rate and, conversely, if too few solutions are accepted, the reheating schedule increases the temperature at some rate (depending on the annealing schedule).

Intuitively, the cooling schedule encourages exploitation, since the probability of accepting a non-improving solution becomes less significant. On the other hand, the reheating schedule encourages exploration, since the probability of accepting a non-improving solution becomes more significant [169, 233], as depicted in Figure 5.17.

Algorithm 5.13: The DBMOSA algorithm adapted from [233], for the simulation optimisation process

```
   Input  : Previously described using Figures 5.14a and 5.14b.
   Output : The approximate Pareto set ($\mathcal{P}_S$) for the given simulation problem.
 1 Initialise the iteration counter, $t \leftarrow 1$;
 2 Initialise counters, $accepts \leftarrow 1$, $attempts \leftarrow 0$ and $poorEpochs \leftarrow 0$;
 3 Initialise the number of epochs, $epochs \leftarrow 0$;
 4 Evaluate the initial solution $\boldsymbol{x}$;                          // Refer to step 1
 5 Initialise the archive, $\mathcal{A} \leftarrow \{\boldsymbol{x}\}$;      // Refer to step 2
 6 while $t \leq t_{max}$ do // Other stopping condition(s) (e.g. $poorEpochs < maxPoorEpochs$)
 7     $t \leftarrow t + 1$;                                                  // Refer to step 3
 8     Generate a neighbouring solution $\boldsymbol{x'}$;                    // Refer to step 4
 9     Evaluate $\boldsymbol{x'}$;                                            // Refer to step 6
10     Calculate the change in energy $\Delta_E(\boldsymbol{x'}, \boldsymbol{x})$, according to (5.16);   // Refer to step 5-8
11     if $|\mathcal{A}_{\boldsymbol{x'}}| = 0$ then
12         Add $\boldsymbol{x'}$ to the archive, $\mathcal{A} \leftarrow \mathcal{A} \cup \{\boldsymbol{x'}\}$;   // Refer to step 10
13         $\boldsymbol{x} \leftarrow \boldsymbol{x'}$;
14         for $y \in \mathcal{A}$ do
15             if $x \prec y$ then
16                 $\mathcal{A} \leftarrow \mathcal{A} \setminus \{y\}$;          // Remove $y$ from archive
17                 $accepts \leftarrow accepts + 1$;                              // Refer to step 12
18             else
19                 $A \leftarrow |\tilde{\mathcal{A}}| + 1$;     // Increment the number of solutions in the archive
20     else
21         Generate a random number, $r_u \sim \mathcal{U}(0, 1)$;             // Refer to step 11
22         if $r_u < P_r(\boldsymbol{x'})$ then
23             Add $\boldsymbol{x'}$ to the archive, $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\boldsymbol{x'}\}$
24             $\boldsymbol{x} \leftarrow \boldsymbol{x'}$;
25             $A \leftarrow |\tilde{\mathcal{A}}| + 1$;         // Increment the number of solutions in the archive
26             $accepts \leftarrow accepts + 1$;                                  // Refer to step 12
27         else
28             $attempts \leftarrow attempts + 1$;                                // Refer to step 14
29 if $accepts \geq maxAccepts$ then
30     $epochs \leftarrow epochs + 1$;                                            // Refer to step 13
31     Decrease the temperature;          // According to cooling schedule, refer to §5.4.2
32     $accepts \leftarrow 0$;                                                    // Reset counter
33 else if $attempts \geq maxAttempts$ then
34     $epochs \leftarrow epochs + 1$;                                            // Refer to step 15
35     Increase the temperature;          // According to reheating schedule, refer to §5.4.2
36     $attempts \leftarrow 0$;                                                   // Reset counter
37 if $successiveAttempts = maxAttempts$ then
38     $poorEpochs \leftarrow poorEpochs + 1$;                                    // Refer to step 16
39 $\mathcal{P}_S \leftarrow \mathcal{A}$;                                        // Approximate Pareto set
```

The choice between different *annealing schedules* involves a compromise between the *quality* of the solutions found and the *speed* of cooling [93]. SA is computationally analogue to the process of physically annealing a metal, *i.e.* to slowly cool a metal to induce an equilibrium state of low-energy. Similarly, if the temperature is decreased sufficiently slow, then better-quality solutions are obtained, however requires significantly more computational time. Some annealing schedules include *linear* schedules, *exponential* schedules [143], *geometric*, *Geman* *and Geman* or *logarithmic* annealing schedules [92], *nonmonotonic* schedules [110] and *adaptive* schedules [93]. The geometric schedule is considered the most popular of the annealing schedules and is used for both cooling and reheating.

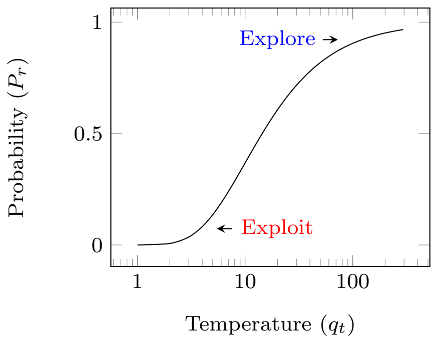


Figure 5.17: *An illustration of how the probability of accepting non-improving moves changes as the temperature changes, resulting in either exploration or exploitation of the search.*

The geometric cooling schedule is given by,

$$q_{t+1} \leftarrow \alpha q_t, \ \ \alpha \in \left[\frac{3}{4}, 1\right), \tag{5.18}$$

where α is the cooling parameter. The geometric reheating schedule is given by,

$$q_{t+1} \leftarrow \beta q_t, \ \ \beta \in \left(1, \frac{5}{4}\right], \tag{5.19}$$

where β is the reheating parameter [233].

It has been reported that the overall *cooling rate* is more important than the specific cooling function. The initial temperature q_0 should also be considered. If the starting temperature is notably high, the search resembles random search, otherwise, the search resembles local search. Consequently, the initial temperature should be high enough to allow for sufficient exploration without being completely random [93].

5.4.3 The initial temperature

As mentioned previously, the DBMOSA tool determines an initial temperature by pressing the *Calculate Initial Temperature* button. The methodology followed is briefly described in this section. Many have proposed methods for determining a suitable initial temperature that is specific to the problem being optimised, refer to [127, 143, 249].

Typically, an initial temperature is found where the probability of accepting non-improving solutions is close to 1. The algorithm randomly generates trial solutions until the neighbouring solution is dominated by the solutions currently in the archive, thereafter the initial temperature q_0 can be determined, based on the probability chosen, by solving for

$$q_0 = -\frac{\Delta_E}{\ln P_r(\boldsymbol{x'})}. \tag{5.20}$$

In this study the initial temperature is chosen arbitrarily as $q_0 = 10$, which is relatively high so as to encourage exploration at the start of the search.

The motivation is to facilitate the hyperparameter tuning with specific focus on the *move operator* (or neighbouring function) used and no other parameters that may influence the performance of the algorithm, *i.e.* observe the effect on the performance of DBMOSA based on the move operator used. It may be that the chosen initial temperature is better for a specific problem, however, this does not influence the analysis of which move operators are best, since single-problem comparisons are done.

5.4.4 Neighbourhood move operators

In order to apply the DBMOSA algorithm, a perturbation method (or neighbouring move operator) has to be defined [222]. The move operators reported for SA typically perform a *random perturbation* of the current solution $\boldsymbol{x}$, according to a set of feasible moves while considering the specific combinatorial context of the problem. The move operators should keep some part of the solution and strongly perturb another part to escape local optima and possibly move to more attractive neighbourhoods, as depicted in Figure 5.18.

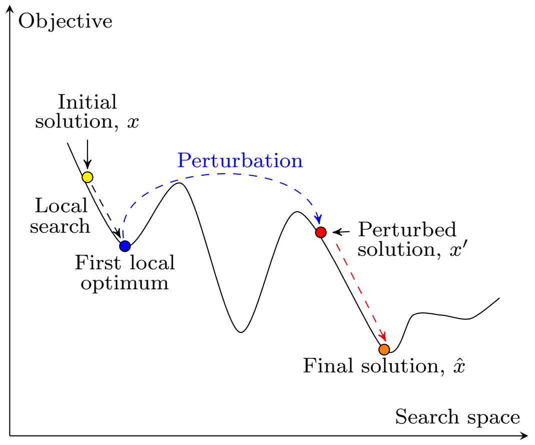


Figure 5.18: *The perturbation of solution x for a minimisation problem, adapted from [93].*

Consequently, a neighbour is generated by the application of a *move operator* that performs a small perturbation of the solution, where perturbation is a random move considering the set of feasible moves [93]. The motivation behind the proposed move operators is the lack of move operators applicable for discrete search spaces.

Move operator 2 simply uses a linear regression model to guide the search. *Move operators 3* and *4* borrow mutation operators from the evolutionary computation literature and adapt them for discrete search spaces. These constitute the set of neighbourhood move operators proposed for the DBMOSA algorithm and are discussed in the following sections, respectively. Note that all move operators consider the solution representation and generate feasible solutions.

Move operator 1: Temperature dependent move

Move operator 1 was designed to change as the temperature parameter q_t changes and was designed during a quantitative pilot study. Consequently, when the temperature is high, the corresponding move should be large as to encourage exploration, and conversely, when the temperature is small the corresponding move should be small as to encourage exploitation. The proposed move operator is described in Algorithm 5.14 and is applied to the solution previously accepted into the archive.

Algorithm 5.14: Neighbouring move 1

```
Input  : Previously accepted solution $x$.
Output: Neighbouring solution $x'$.
1 Generate a random number, $r_u \sim \mathcal{U}(0,1)$;          // Random number between 0 and 1
2 if $r_u < 0.5$ then
3     $x \leftarrow x + \min(u - x,\ x - l,\ stepsize)$     (5.21)
4 else
5     $x \leftarrow x - \min(u - x,\ x - l,\ stepsize)$     (5.22)
```

Table 5.2: *The temperature ranges used to specify the stepsize of a move.*

q_t	*stepsize*
$q_t \geq 0.75q_0$	$(0.3 - 0.39)u$
$0.5q_0 \leq q_t < 0.75q_0$	$(0.20 - 0.29)u$
$0.25q_0 \leq q_t < 0.5q_0$	$(0.10 - 0.19)u$
$q_t < 0.25q_0$	$(0.01 - 0.09)u$

Suppose that the initial temperature for a given problem is $q_0 = 2$ and the current temperature for iteration t is $q_t = 1.6$. Then the relationship $\frac{q_t}{q_0} = 0.8$, *i.e.* the temperature range is $q_t \geq 0.75q_0$ as seen in Table 5.2, and the corresponding step size is between 30%–39% of the upper bound for decision variable i. Also, suppose that the lower and upper bounds for decision variable i are $l = 0$ and $u = 100$ and currently $x = 50$. The step size can be is determined by first generating a random number between 0.3 and 0.39. Suppose 0.32 was generated and that $r_u > 0.5$ then the neighbouring solution $\boldsymbol{x'}$ is created by adding the minimum of expression in (5.21) as

$$\begin{aligned} x &= 50 + \min(100 - 50,\ 50 - 0,\ (0.32)100) \\ &= 82, \end{aligned}$$

otherwise the neighbouring solution $\boldsymbol{x'}$ is created by subtracting the minimum of expression (5.22) as

$$\begin{aligned} x &= 50 - \min(100 - 50,\ 50 - 0,\ (0.32)100) \\ &= 5. \end{aligned}$$

Move operator 2: Linear regression

Move operator 2 applies a linear regression model to the solutions that are evaluated and then uses the coefficients for the respective decision variables to guide the search. The regression model is updated after every 50 solution evaluations. The motivation behind move operator 2 is its ease of implementation when compared with other regression models. Linear regression is discussed in Chapter 3. The idea is to use the coefficients in respect of the decision variable's effect on the two objectives to determine how far the decision variables should move from its current position.

Move operator 3: Dynamic and polynomial mutation

As mentioned, *move operator 3* and *4* are borrowed from evolutionary computation literature and adapted to form move operators that mutate discrete vector representations. Polynomial and dynamic mutation are combined to form move operator 3 discussed in §5.3.4 and §5.3.4. The mutation operators are applied as described in Algorithm 5.15, to the solution previously accepted into the archive. For every decision variable, a random number $r_u \sim \mathcal{U}(0,1)$ is generated and if $r_u < 0.5$ polynomial mutation is applied, otherwise, dynamic mutation is applied, however (5.15) is adapted as $\xi = 1 - |(\mathsf{counter}\ \mathsf{mod}\ 100)/(10)|^5$, where counter is the current solution. By combining polynomial mutation and dynamic mutation, the hope is that a balance between exploitation and exploration is established.

Algorithm 5.15: Neighbouring move 3

```
  Input : Previously accepted solution $x$.
  Output: Neighbouring solution $x'$.
1 Generate a random number, $r_u \sim \mathcal{U}(0,1)$;        // Random number between 0 and 1
2 if $r_u < 0.5$ then
3     Polynomial mutation;                                      // See Section 5.3.4
4 else
5     Dynamic mutation;                                         // See Section 5.3.4
```

Move operator 4: Temperature dependent dynamic mutation

Move operator 4 adapts the dynamic mutation operator (discussed in §5.3.4) to determine a move based on the relationship between the initial temperature and the current temperature. Consequently, (5.15) is adapted to include the temperature parameters and is given as

$$\xi = 1 - \left(\frac{q_0 - q_t}{q_0}\right), \tag{5.23}$$

where q_0 is the initial temperature and q_t is the temperature at iteration t. The smaller the value of ξ the closer the neighbouring solution x' is created to the previously accepted solution x.

In conclusion, the properties of the DBMOSA algorithm are stated. The DBMOSA algorithm

1. is a single-solution based search technique,
2. has a physics basis,
3. uses an archive to preserve non-dominated solutions ($\mathcal{A}$) during the search,
4. uses an archive ($\tilde{\mathcal{A}}$) to control the probability of acceptancing non-improving solutions,
5. employs a Pareto ranking scheme to find non-dominated solutions,
6. achieves proximity to the true Pareto front *via* the cooling schedule,
7. and ensures diversity of the search *via* the energy measure and the Metropolis acceptance rule which determine whether or not a non-improving solution is accepted into the archive.

Recall that, based on the NFL theorem, no single algorithm outperforms all other algorithms when considering a diverse set of optimisation problems, irrespective of the performance metric used [257]. Consequently, the implementation of hyperheuristics proposed a way to try to mitigate, to some extent, the effect of the NFL theorem by combining more than one algorithm. AMALGAM was used as inspiration for both the population-based and single-solution based search hyperheuristics, as proposed in this study. As mentioned, AMALGAM is a population-based search hyperheuristic proposed by Vrugt and Robinson [254], which incorporates multiple metaheuristics (or LLHs) simultaneously and has also been adapted and applied to single-solution based searches, as proposed in [128].

The choice of hyperheuristic framework (or method for managing the LLHs) is crucial as it is often difficult to determine which LLH to apply to the current search space and serves as the motivation for using AMALGAM as inspiration. AMALGAM applies the LLHs in parallel (or proportionally) rather than successively. The heuristic selection procedure proposed in AMALGAM determines how many solutions each LLH may generate during each generation (or loop), *i.e.* given a population size N, then each LLH is allocated a proportion of N (based on its individual past performance) to contribute to and form the offspring population [38, 76]. The move acceptance procedure implemented in AMALGAM is that of NSGA-II, namely Pareto rank and crowding distance (as discussed in §5.3.1).

The next sections document each hyperheuristic, first, a detailed description of the population-based search hyperheuristic inspired by AMALGAM is given, called the **bi**-**o**bjective **c**ross-**e**ntropy and **g**enetic **a**lgorithm **h**yperheuristic (BOCEGAH). Next, a detailed description of the single-solution based search hyperheuristic, also inspired by AMALGAM is given, called the **bi**-**o**bjective **s**imulated **a**nnealing **h**yperheuristic (BOSAH). Each description is supported by the user-interface of the *Hyperheuristic tool* used in Tecnomatix, to illustrate the integration of simulation and optimisation.

5.5 The BOCEGAH algorithm

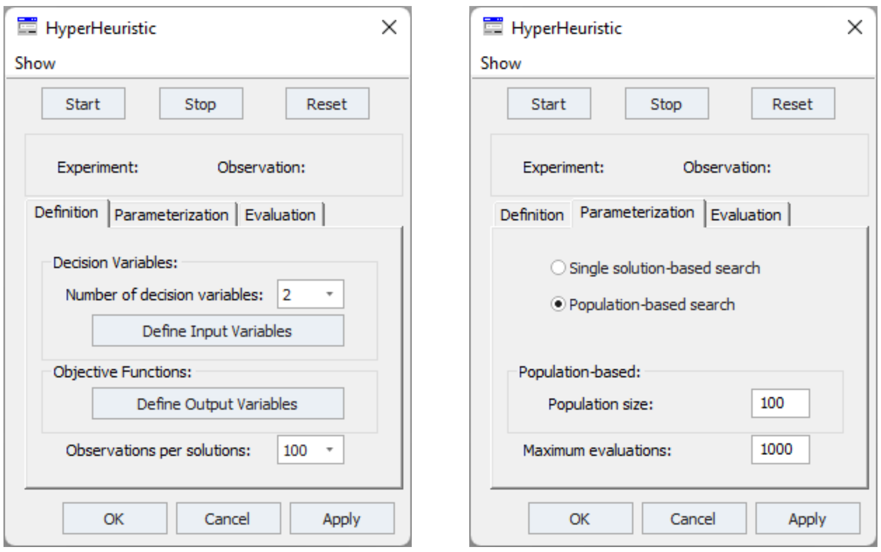
The typical procedure for solving MOOPs (approximately) by means of a population-based search approach is to implement a single algorithm that evolves a population of solutions (iteratively). AMALGAM follows a similar approach, however, multiple algorithms are implemented in parallel and each evolve a proportion of the population [254]. The MOOCEM and the NSGA-II are also applied in parallel (or simultaneously), evolving their own (partial) populations, enabling them to (ultimately) exploit their shared advantages while compensating for their respective weaknesses [192]. To conclude, by applying the LLHs in parallel, the hyperheuristic is able to determine the extent to which each LLH may contribute to the search, based on its past performance and effectively allocate solutions to each LLH for the current search space.

The BOCEGAH implements AMALGAMs heuristic selection procedure, however, each LLH (*i.e.* the MOOCEM and the NSGA-II) maintain their own move acceptance strategies as designed (specifically) for them. The reasoning pertains to the trade-off between exploration and exploitation of the search space and algorithms have to be designed to balance this trade-off. The simulation optimisation process (for population-based search) is illustrated using the Hyperheuristic tool in Tecnomatix, as shown in Figures 5.19a and 5.19b. Note that the same tool user-interface is used for both hyperheuristics. The steps for using the Hyperheuristic tool are as follows: starting with the *Definition tab* as shown in Figure 5.19a.

1. Define the number of decision variables (n), from the drop-down list, then click the *Define*

Input Variables button and define each decision variable in terms of its path and lower and upper bounds (as previously shown in Figure 5.5a).

2. Click the *Define Output Variables* button and define each output variable in terms of its path and select the optimisation direction from the drop-down list, *i.e. Min* or *Max* (as previously shown in Figure 5.5b).



Decision Variables:
Number of decision variables: 2
Objective Functions:
Observations per solutions: 100

○ Single solution-based search
● Population-based search
Population-based:
Population size: 100
Maximum evaluations: 1000

(a) *Definition tab* (b) *Parameterisation tab*

Figure 5.19: *The Hyperheuristic tool user-interface for population-based search.*

The next step is to navigate to the *Parameterisation tab*, as shown in Figure 5.19b:

3. Select *Population-based search*, enter the *Population size* (N) and *Maximum evaluations* permitted.

Recall that the population size $N = 100$ and a fixed number of 1 000 solution evaluations are permitted in this study. The above-mentioned parameters are used as input to Algorithm 5.16 as well as the best hyperparameter combination specific to MOOCEM and NSGA-II, as reported in Chapter 6. To execute the Hyperheuristic tool for population-based search and solve for the simulation problem at hand, click the *Start* button. A similar process is followed to execute and evaluate the Hyperheuristic tool as for the individual LLHs, where the initial population is also generated randomly.

Start with the initialisation of the parameters for the MOOCEM algorithm (as described in §5.2.2):

1. Initialise the elite vector, $Elite \leftarrow \emptyset$.
2. Initialise μ_i and σ_i (refer to §5.2.2 step 1).

Next, initialise the parameters for the NSGA-II algorithm (as described in §5.3.1):

3. Initialise the elite vector ($Elitist \leftarrow \emptyset$).

Next, initialise the parameters specific to the BOCEGAH algorithm. First, initialise the loop counter, $l \leftarrow 0$ and the number of LLHs (or heuristic components) considered, $nLLH \leftarrow 2$. Also, initialise the elite vector $Elites \leftarrow \emptyset$, used to combine the elite vectors of MOOCEM and NSGA-II at termination. Now, the general procedure starts.

4. Generate an initial population (of size N) by sampling from a truncated probability distribution, using Algorithm 5.1 and 5.2 as discussed in §5.2.1.
5. Evaluate the initial population, *i.e.* the simulation model evaluates N decision variable combinations.
6. Increment the loop counter, $l \leftarrow l + 1$.

At the start of the search procedure each LLH (*i.e.* MOOCEM and NSGA-II) is allocated the same number of solutions, *i.e.* for $l = 1$ and $N = 100$, each LLH is allocated $N/2 = 50$ (offspring) solutions, denoted by $N_{h,l}$ for $h \in \{1,\ 2\}$, where $h = 1$ is the MOOCEM and $h = 2$ is the NSGA-II. The allocations are updated for every loop l.

To summarise, for loop $l = 1$, MOOCEM generates 50 offspring solutions (referred to as offspring population $\mathcal{Q}_{1,1}$) from the 50 solutions in the random initial population, and similarly, NSGA-II generates 50 offspring solutions ($\mathcal{Q}_{2,1}$) from the remaining 50 solutions in the random initial population.

7. Initialise the allocations $N_{h,l}$ for $h \in \{1,\ 2\}$ and loop $l = 1$, *i.e.* $N_{h,1} \leftarrow N/2$.

Now, the loop initiates, while the current loop $l < 10$ (resulting in 1 000 solution evaluations), do the following. Generate the offspring populations $\mathcal{Q}_{h,l}$ for $h \in \{1,\ 2\}$. The process of generating the offspring populations is first described for the MOOCEM and then for the NSGA-II. The MOOCEM generates (partial) offspring population $\mathcal{Q}_{1,l}$ of size $N_{1,l}$ by following steps 11–30 in Algorithm 5.4. Also, refer to the supporting documentation given in §5.2.2.

8. Generate offspring population $\mathcal{Q}_{1,l}$ of size $N_{1,l}$ for loop l by following steps 11–30 in Algorithm 5.4.

Next, the NSGA-II generates the other partial offspring population $\mathcal{Q}_{2,l}$ of size $N_{2,l}$ by following steps 8–12 and 14 in Algorithm 5.7 for loop l. Also, refer to the supporting documentation given in §5.3.1.

9. Generate offspring population $\mathcal{Q}_{2,l}$ of size $N_{2,l}$ for loop l by following steps 8–12 and 14 in Algorithm 5.7.

After the offspring populations $\mathcal{Q}_{1,l}$ and $\mathcal{Q}_{2,l}$ are generated, it is necessary to check if duplicate solutions are present and then replace them with non-duplicate solutions by generating new (offspring) solutions, as described in steps 8 and 9. This is a critical step that ensures that the expensive function evaluator does not unnecessarily evaluate solutions.

10. Check for duplicate solutions and then replace them with non-duplicate solutions.

Next, combine the individual offspring populations $\mathcal{Q}_{h,l}$ for $h \in \{1,\ 2\}$ and then evaluate the combined population, denoted by $\mathcal{P}_{C,l}$.

11. Combine the offspring populations generated in step 8 and 9 for loop l, *i.e.* $\mathcal{P}_{C,l} \leftarrow \mathcal{Q}_{1,l} \cup \mathcal{Q}_{2,l}$.

12. Evaluate the combined population $\mathcal{P}_{C,l}$ of size N, for loop l, *i.e.* the simulation model evaluates N decision variable combinations.

Next, determine the new allocations for loop $l+1$, following steps 13–15.

13. Rank the combined population $\mathcal{P}_{C,l}$ for loop l, using Algorithm 5.5.

Next, determine how many solutions from $\mathcal{Q}_{1,l}$ and $\mathcal{Q}_{2,l}$ contributed to the new population, based on the non-dominated solutions found in step 13. The contributions are referred to as $C_{h,l}$ for $h \in \{1,\ 2\}$.

14. Determine the contributions $C_{h,l}$ for $h \in \{1,\ 2\}$, for loop l.

Now, the new allocations can be determined. This is where the heuristic selection procedure proposed in AMALGAM [254] is implemented. Each LLH is allocated a number of solutions based on their contributions ($C_{h,l}$). Suppose LLH $h \in \{1, \ldots, nLLH\}$ contributes $C_{h,l}$ solutions during loop l, then let the number of solutions that LLH $h \in \{1,\ 2\}$ may generate during loop $l+1$ be given by

$$N_{h,l+1} = \frac{\frac{C_{h,l}}{N_{h,l}}}{\sum_{h=1}^{nLLH} \frac{C_{h,l}}{N_{h,l}}} N. \tag{5.24}$$

When $l = 1$, then $N_{1,1}, N_{2,1} = 50$ because (as mentioned) each LLH $h \in \{1,\ 2\}$ is given equal opportunity at the start. Now, suppose $h = 1$ (MOOCEM) contributed 20 solutions (*i.e.* $C_{1,1} = 20$) and $h = 2$ (NSGA-II) contributed 15 solutions (*i.e.* $C_{2,1} = 15$). Then, the new allocation for $l = 2$ and $h = 1$ (*i.e.* $N_{1,2}$) can be calculated using (5.24), as depicted in Figure 5.20.

l	h	$N_{h,l}$	$C_{h,l}$
1	1	50	20
	2	50	15
2	1	$\boxed{57}$	-
	2	43	-

$$N_{1,2} = \frac{\frac{C_{1,1}}{N_{1,1}}}{\frac{C_{1,1}}{N_{1,1}} + \frac{C_{2,1}}{N_{2,1}}} N \quad (5.24)$$

$$= \frac{\frac{20}{50}}{\frac{20}{50} + \frac{15}{50}} 100$$

Figure 5.20: *An example illustrating the heuristic selection procedure implemented for the BOCEGAH.*

Accordingly, $h = 2$ (NSGA-II) is allocated $N_{2,2} = 43$ solutions. Note that provision is made to ensure that the number of solutions allocated to each LLH h for loop l adds up to the population size, *i.e.* $N_{1,l} + N_{2,l} = N$. Also, to ensure that a LLH is not eliminated from the search, a minimum allocation is set at 5, *i.e.* the minimum number of offspring solutions that LLH h may generate is $N_{h,l} = 5$. If the allocations for either MOOCEM or NSGA-II is less than 5, the allocation is updated to equal 5. For example, if $\{N_{1,2},\ N_{2,2}\} = \{2,\ 98\}$ then it is updated so that $\{N_{1,2},\ N_{2,2}\} = \{5,\ 95\}$.

15. Update the allocations $N_{h,l}$ for $h \in \{1,\ 2\}$ for loop l, using (5.24).

16. Increment the loop counter, $l \leftarrow l + 1$.

This concludes the loop. At termination, the elite vectors for MOOCEM (*Elite*) and NSGA-II (*Elitist*) are joined in elite vector *Elites*, *i.e.* $Elites \leftarrow Elite \cup Elitist$. Finally, the solutions in *Elites* are ranked using Algorithm 2.1 and returned as an approximate of the true Pareto set of solutions to the bi-objective simulation optimisation problem under consideration, *i.e.* $\mathcal{P}_S \leftarrow Elites$.

A pseudo-code description of the entire process described above is provided in Algorithm 5.16, with specific reference to the above-mentioned steps.

Algorithm 5.16: The BOCEGAH algorithm for population-based search

```
   Input  : Previously described using Figures 5.19a and 5.19b.
   Output : The approximate Pareto set ($\mathcal{P}_S$) for the given simulation problem.
   /* MOOCEM initialisation                                                        */
1  Refer to step 1-2;
   /* NSGA-II initialisation                                                       */
2  Refer to step 3;
   /* Hyperheuristic initialisation                                                */
3  $l \leftarrow 0$;                                      // Initialise the loop counter
4  $nLLH \leftarrow 2$;                                   // Initialise the number of LLHs
5  $Elites \leftarrow \emptyset$;                         // Initialise the elite vector
   /* General procedure starts                                                     */
6  Generate the initial population (of size $N$) using Algorithm 5.1;      // Refer to step 4
7  Evaluate initial population;                                            // Refer to step 5
8  $l \leftarrow l + 1$;                       // Increment the loop counter, refer to step 6
9  Initialise the allocations, $N_{h,l}$ for $h \in \{1,\ 2\}$;            // Refer to step 7
10 while $l < 10$ do
11     Follow steps 11–30 as described in Algorithm 5.4;                   // Refer to step 8
12     Follow steps 8–12 and 14 as described in Algorithm 5.7;             // Refer to step 9
13     Combine offspring populations, $\mathcal{P}_{C,l} \leftarrow \mathcal{Q}_{1,l} \cup \mathcal{Q}_{2,l}$;   // Refer to step 11
14     Evaluate the combined offspring population, $\mathcal{P}_{C,l}$ of size $N$;   // Refer to step 12
15     Rank the solutions in $\mathcal{P}_{C,0}$, using Algorithm 5.5;     // Refer to step 13
16     Determine the contributions $C_{h,l}$ for $h \in \{1,\ 2\}$;         // Refer to step 14
17     Update allocations $N_{h,l}$ for $h \in \{1,\ 2\}$, using (5.24);   // Refer to step 15
18     $l \leftarrow l + 1$;                                               // Refer to step 16
19 $Elites \leftarrow Elite \cup Elitist$;                                 // Combine elite vectors
20 Rank the solutions in $Elites$;                                         // Using Algorithm 2.1
21 $\mathcal{P}_S \leftarrow Elites$;                                      // Approximate Pareto set
```

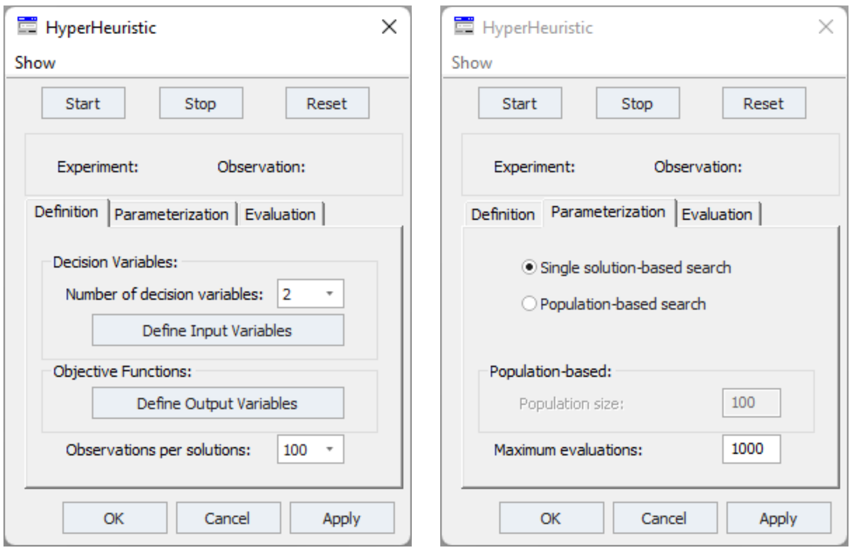
The next section documents BOSAH with specific focus on the integration of simulation and optimisation.

5.6 The BOSAH algorithm

The typical procedure for solving MOOPs (approximately) by means of a single-solution based search approach is to implement a single algorithm that perturbs a single solution (iteratively). For that reason, the heuristic selection procedure proposed for AMALGAM is adapted for successive heuristic assignment.

Each LLH (or move operator) is assigned successively instead of simultaneously to facilitate single-solution based search. The heuristic selection implemented in AMALGAM is employed to determine how many consecutive solutions a LLH may be assigned for. Three LLHs are assigned sequentially, *i.e.* move operator 1–2–3–1$\cdots$, each generating their assigned number of solutions. For this reason, another step is proposed as part of the heuristic selection procedure to ensure that if a LLH is performing poorly then instead of completing its assigned number of solutions, the next move operator in the sequence is assigned the rest of its allocations. The purpose of this step is to ensure that the search does not stagnate (due to the NFL theorem), because the LLHs are assigned sequentially, it is possible that the best move operator for the current point in the search is not assigned and should be addressed accordingly.

The simulation optimisation process (for single-solution based search) is illustrated using the Hyperheuristic tool in Tecnomatix as shown in Figures 5.21a and 5.21b. The steps for using the Hyperheuristic tool are as follows: starting with the *Definition tab*, as shown in Figure 5.21a.



Experiment: Observation:

Decision Variables:
Number of decision variables: 2
Objective Functions:
Observations per solutions: 100

Experiment: Observation:

(●) Single solution-based search
( ) Population-based search
Population-based:
Population size: 100
Maximum evaluations: 1000

(a) *Definition tab* (b) *Parameterisation tab*

Figure 5.21: *The Hyperheuristic tool user-interface for single-solution based search.*

1. Define the number of decision variables (n), from the drop-down list, then click the *Define Input Variables* button and define each decision variable in terms of its path, lower and upper bounds and initial solutions (as previously shown in Figure 5.15).

2. Click the *Define Output Variables* button and define each output variable in terms of its path and select the optimisation direction from the drop-down list, *i.e. Min* or *Max* (as previously shown in Figure 5.5b).

The next step is to navigate to the *Parameterisation tab*, as shown in Figure 5.19b:

3. Select *Single-solution based search* and enter the *Maximum evaluations* permitted.

Recall that a fixed number of 1 000 evaluations are permitted in this study. The above-mentioned parameters are used as input to Algorithm 5.17 as well as the three best move operators for DBMOSA as found in Chapter 6. To execute the Hyperheuristic tool for single-solution based search and optimise the simulation problem at hand click the *Start* button. A similar process is followed to execute and evaluate the Hyperheuristic tool as for the DBMOSA and its move operators (LLHs), where the initial solution is also generated randomly, represented by $\boldsymbol{x}$.

Start with the initialisation of the parameters for the DBMOSA algorithm (as described in §5.4):

1. Initialise the counters, accepts ($accepts \leftarrow 1$), attempts ($attempts \leftarrow 0$), poor epochs ($poorEpochs \leftarrow 0$) and epochs ($epochs \leftarrow 0$).

Next, initialise the parameters specific to the BOSAH algorithm. First initialise the iteration counter, $t \leftarrow 1$, and the number of LLHs (or heuristic components) considered, $nLLH \leftarrow 3$. Next, initialise the move sequence counter (denoted by i) and the move operator (denoted by m) assigned at the start of the search. Note that move operator $m \in \{1,\ 2,\ 3\}$, where $m = i$ implies move operator i.

2. Initialise the move sequence counter, $i \leftarrow 1$.
3. Initialise the move operator assigned, $m \leftarrow 1$.

The search starts with move operator 1, denoted as $m = 1$. Initially, each LLH (*i.e.* move operator 1–3 or $m \in \{1,\ 2,\ 3\}$) is allocated the same number of solutions that it is permitted to generate. The allocations are denoted by $N_{m,i}$ and are initialised to 20, as illustrated in Table 5.3, *i.e.* $N_{m,1} = 20$ for $m \in \{1,\ 2,\ 3\}$. Note that the initial allocations were chosen arbitrarily.

Table 5.3: *An example illustrating the initial allocations for the BOSAH.*

i	m	$N_{m,i}$	$T_{m,i}$
	1	20	20
1	2	20	40
	3	20	60

4. Initialise the allocations $N_{m,i}$ for $m \in \{1,\ 2,\ 3\}$, for move sequence $i = 1$.

The allocations are updated after each move sequence i, where a move sequence consists of move operator $m = 1$ generating $N_{1,i}$ solutions, then $m = 2$ generating $N_{2,i}$ solutions and lastly $m = 3$ generating $N_{3,i}$ solutions. This process continues until 1 000 solutions have been generated and evaluated. Now, the general procedure starts. Recall, that an archive $\mathcal{A}$ is maintained throughout the search and contains all the non-dominated solutions generated by the individual move operators (or LLHs). The random initial solution $\boldsymbol{x}$ is evaluated and inserted into the archive, $\mathcal{A} \leftarrow \{\boldsymbol{x}\}$.

5. Evaluate the initial solution $\boldsymbol{x}$, *i.e.* the simulation model evaluates $\boldsymbol{x}$.
6. Initialise the archive to contain the initial solution, $\mathcal{A} \leftarrow \{\boldsymbol{x}\}$.

As part of the heuristic selection strategy, it is necessary to count the number of non-dominated solutions found per move operator assignment, *i.e.* the number of non-dominated solutions found by move operator m for move sequence i and stored in variable NDS. Initially $NDS \leftarrow 1$ since one solutions has been evaluated. Note that NDS is initialised to zero every time a new move operator m is assigned.

7. Initialise the non-dominated solutions counter, $NDS \leftarrow 1$.

As part of the proposed heuristic selection procedure, a rate (denoted by $rate_t$) of non-dominated solutions is updated at each iteration t, *i.e.* $rate_t$ is the number of non-dominated solutions found divided by t (the number of solutions evaluated so far). Initially, $rate_t \leftarrow 1$ as the initial solution is considered non-dominated.

8. Initialise the rate of non-dominated solutions found, $rate_1 \leftarrow 1$.

Now, the loop initiates, while $t \leq 1\,000$ do the following. Increment the iteration counter, $t \leftarrow t + 1$. The next step, as described in §5.4, is to generate the neighbouring solution $\boldsymbol{x'}$ using a move operator. At each iteration t, the algorithm checks if move operator m should be employed to generate neighbouring solution $\boldsymbol{x'}$.

To facilitate the heuristic selection procedure (of the move operators $m \in \{1,\ 2,\ 3\}$) three steps are required. The first step ensures that each move operator (m) generates its allocated number of solutions ($N_{m,i}$). The second step is proposed in this study to ensure that the search does not stagnate, making it a novel contribution. The third step determines the new allocations for the next move sequence i and is similar to the strategy implemented in BOCEGAH and proposed for AMALGAM [254]. The heuristic selection procedure is described in steps 9–20.

The *first* step

Suppose $t = 15$ (*i.e.* 15 solutions have been evaluated) then from Table 5.3 it can be seen that it is currently the first move sequence (*i.e.* $i = 1$) and move operator $m = 1$ is assigned. Also, from Table 5.3, it can be seen that move operator $m = 1$ is allocated 20 solutions (*i.e.* $N_{1,1} = 20$). Therefore, once the current iteration t reaches the cumulative total number of solutions that move operator m for move sequence i may generate (*i.e.* $T_{m,i}$), then the next move operator in the move sequence is assigned (*i.e.* $m \leftarrow m + 1$).

If $t = N_{m,i} + 1$, then execute steps 9–11:

9. Assign the number of non-dominated solutions (NDS) contributed by move operator m for move sequence i, denoted by $C_{m,i} \leftarrow NDS$.
10. Re-initialise the NDS counter to zero, $NDS \leftarrow 0$.
11. Increment the move operator assigned, $m \leftarrow m + 1$.

The *second* step

The motivation behind the second heuristic selection step is to negate the effect of the NFL theorem by playing to the strengths of the individual move operators, *i.e.* if move operator $m = 1$ is not generating (enough) solutions that are evaluated as non-dominated then the rest of its allocation should be transferred to the next move operator in the move sequence (*i.e.* $m = 2$) and so forth. Ultimately, compensating for their respective weaknesses and exploiting their individual strengths.

Specifically, when the current move operator m has generated half of its allocated solutions (*i.e.* $t = N_{m,i}/2 + T_{m-1,i}$), then test if the move operator's rate of generating non-dominated solutions (*i.e.* $\frac{NDS}{N_{m,i}/2}$) is smaller than the overall rate ($rate_t$). If this is true, then move operator

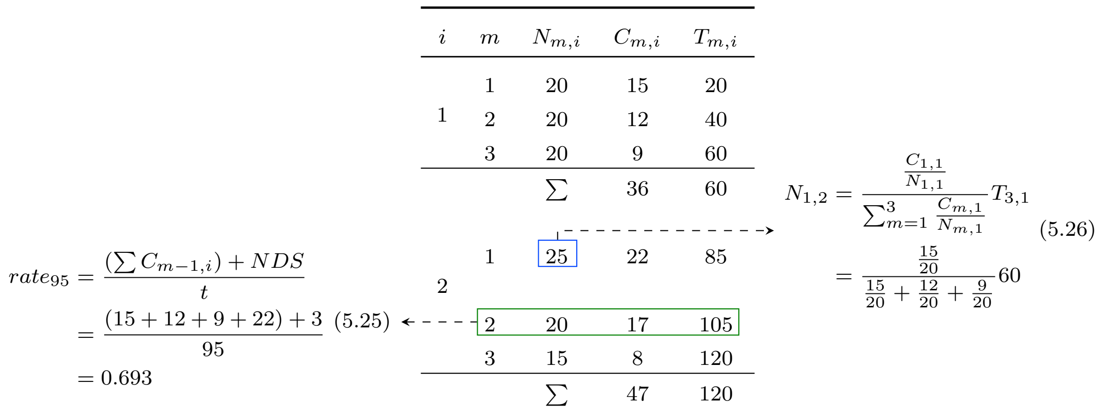
m is performing below the search standard and should not continue to generate the rest of its allocated solutions. Rather, the rest of the solutions are transferred to the next move operator (*i.e.* $m+1$) in the same move sequence i. However, if $m = nLLH$, then the rest is not transferred over to the next move sequence.

Formally, the overall rate of non-dominated solutions found, $rate_t$ can be determined as

$$rate_t = \frac{\sum C_{m-1,i} + NDS}{t}, \tag{5.25}$$

where $m \in \{1,\ 2,\ 3\}$, $\sum C_{m-1,i}$ is the sum of the individual contributions up until $m-1$ for move sequence $i = 1$ to the current move sequence i, $NDS \leftarrow 3$ (currently) and t is the current iteration. If $m = 1$ and $i = 2$, then the contribution $C_{m-1,i}$ is represented by $C_{3,1}$, *i.e.* the previous move sequence.

Suppose that the number of non-dominated solutions found by move operator $m = 2$ is 3 (denoted by $NDS = 3$) out of the $N_{2,2}/2 = 10$ solutions generated (for move sequence $i = 2$), then the rate at $t = 95$ can be calculated using (5.25) as depicted in Figure 5.22. Consequently, $\frac{NDS}{N_{2,2}/2} = 0.3$ and $0.3 < 0.693$.



i	m	$N_{m,i}$	$C_{m,i}$	$T_{m,i}$
1	1	20	15	20
	2	20	12	40
	3	20	9	60
		$\sum$	36	60
2	1	$\boxed{25}$	22	85
	2	20	17	105
	3	15	8	120
		$\sum$	47	120

$$N_{1,2} = \frac{\frac{C_{1,1}}{N_{1,1}}}{\sum_{m=1}^{3} \frac{C_{m,1}}{N_{m,1}}} T_{3,1} = \frac{\frac{15}{20}}{\frac{15}{20}+\frac{12}{20}+\frac{9}{20}} 60 \tag{5.26}$$

$$\begin{aligned} rate_{95} &= \frac{(\sum C_{m-1,i}) + NDS}{t} \\ &= \frac{(15+12+9+22)+3}{95} \quad (5.25) \\ &= 0.693 \end{aligned}$$

Figure 5.22: *An example illustrating the heuristic selection procedure proposed for the BOSAH.*

If $i > 1$ and $t = N_{m,i}/2 + T_{m-1,i}$ and $\frac{NDS}{N_{2,2}/2} < rate_t$ and $N_{m,i} > 5$, then execute steps 12–16:

12. Update the number of allocations assigned to move operator m, *i.e.* $N_{m,i} \leftarrow N_{m,i}/2$.
13. Assign the non-dominated solutions (NDS) contributed by move operator m for move sequence i, *i.e.* $C_{m,i} \leftarrow NDS$.
14. Re-initialise the NDS counter to zero, $NDS \leftarrow 0$.

Consequently, move operator $m = 2$ is assigned fewer solutions, *i.e.* $N_{2,2} \leftarrow 10$ instead of 20. Also, the contribution of move operator $m = 2$ becomes $C_{2,2} \leftarrow 3$. Next, the next move operator $m = 3$ in the move sequence is assigned and its allocation updated as $N_{3,2} \leftarrow 15 + 20/2$.

15. Increment the move operator, $m \leftarrow m + 1$.
16. Update the number of allocations assigned to move operator m for move sequence i, *i.e.* $N_{m,i} \leftarrow N_{m,i} + N_{m-1,i}/2$.

The *third* step

Recall that there is a sequence of move operators being assigned, called a move sequence. The sequence is as follows, move operator $m = 1$, then $m = 2$ and lastly $m = 3$, until termination is reached. Therefore, if the increment of move operator m results in $m = nLLH + 1$, then a move sequence is complete and move operator $m = 1$ is assigned again and the move sequence counter is incremented, $i \leftarrow i + 1$. Next, determine the new allocations $N_{m,i}$ for move sequence i and move operators $m \in \{1,\ 2,\ 3\}$, using (5.26).

$$N_{m,i} = \frac{\frac{C_{m,i-1}}{N_{m,i-1}}}{\sum_{m=1}^{3} \frac{C_{m,i-1}}{N_{m,i-1}}} T_{3,1}. \tag{5.26}$$

Figure 5.22 aids with elucidating the process followed to determine the new allocation for move sequence $i = 2$, move operator $m = 1$, *i.e.* $N_{1,2}$. In the example, the contributions are $C_{m,1} = \{15,\ 12,\ 9\}$, *i.e.* move operator $m = 1$ contributed $C_{1,1} = 15$ (non-dominated) solutions from $N_{1,1} = 20$. From this $N_{1,2} = 25$ is calculated using (5.26) as shown in Figure 5.22.

Following a similar process for $N_{2,2}$ and $N_{3,2}$ results in allocations $N_{2,2} = 20$ and $N_{3,2} = 15$. Note that the total number of allocations should be equal to the sum of the individual allocations for the first move sequence, namely 60 solutions, which is arbitrarily chosen.

If $m = nLLH + 1$, then execute steps 17–19:

17. Restart the move operator sequence, $m \leftarrow 1$.
18. Increment the move sequence, $i \leftarrow i + 1$.
19. Update the allocations $N_{m,i}$ for move sequence i and $m \in \{1,\ 2,\ 3\}$, using (5.26).

A minimum allocation threshold n_a is set at $n_a = 5$ and is required so that none of the move operators recieve an allocation of 0, and thereby be eliminated from the search. Consequently, if step 19 results in an allocation of less than five solutions, then it is simply set to 5 and the other allocations are updated accordingly.

20. If $N_{m,i} < n_a$, then $N_{m,i} \leftarrow n_a$ for $m \in \{1,\ 2,\ 3\}$.

Finally, the neighbouring solution $\boldsymbol{x}'$ can be generated. Recall that the neighbouring solution is generated from the previously accepted solution (as discussed in §5.4).

21. Generate the neighbouring solution $\boldsymbol{x}'$ according to move operator m, denoted by move operator$_m(\boldsymbol{x})$.
22. Check if a duplicate solution are present and then replace it with a non-duplicate solution.

After the neighbouring solution $\boldsymbol{x}'$ is generated using move operator m, it is necessary to check if a duplicate solution is present. If so, then replace it with a non-duplicate solution by generating a new neighbouring solution, (again) using move operator m. This step ensures that the expensive function evaluator does not unnecessarily evaluate a solution. Now, evaluate the non-duplicate neighbouring solution $\boldsymbol{x}'$.

Algorithm 5.17: The BOSAH method for single-solution based search

```
   Input  : Previously described using Figures 5.19a and 5.21b.
   Output : The approximate Pareto set ($\mathcal{P}_S$) for the given simulation problem.
   /* DBMOSA initialisation                                                     */
 1 Refer to step 1;
   /* Hyperheuristic initialisation                                             */
 2 $t \leftarrow 1$;                                      // Initialise iteration counter
 3 $nLLH \leftarrow 3$;                                   // Initialise the number of LLHs
 4 Initialise the move sequence counter, $i \leftarrow 1$;        // Refer to step 2
 5 Initialise move operator assigned, $m \leftarrow 1$;            // Refer to step 3
 6 Initialise allocations;                                         // Refer to step 4
   /* General procedure starts                                                  */
 7 Evaluate the initial solution $\boldsymbol{x}$;                 // Refer to step 5
 8 Initialise the archive, $\mathcal{A} \leftarrow \{\boldsymbol{x}\}$;   // Refer to step 6
 9 $NDS \leftarrow 1$;                                             // Refer to step 7
10 Initialise the $rate_1 \leftarrow 1$;                           // Refer to step 8
11 while $t \leq 1\,000$ do
12     $t \leftarrow t + 1$;                              // Increment the iteration counter
13     if $t = T_{m,i} + 1$ then
14         Assign the contribution of $m$, $C_{m,i} \leftarrow NDS$;   // Refer to step 9
15         $NDS \leftarrow 0$;                                     // Refer to step 10
16         $m \leftarrow m + 1$;                                   // Refer to step 11
17     if $i > 1$ and $t = N_{m,i}/2 + T_{m-1,i}$ and $\frac{NDS}{N_{m,i}/2} < rate_t$ and $N_{m,i} > 5$ then
18         Update allocations;                                     // Refer to step 12
19         Assign the contribution of $m$, $C_{m,i} \leftarrow NDS$;   // Refer to step 13
20         $NDS \leftarrow 0$;                                     // Refer to step 14
21         $m \leftarrow m + 1$;                                   // Refer to step 15
22         Update allocations;                                     // Refer to step 16
23     if $m = nLLH + 1$ then
24         $m \leftarrow 1$;                                       // Refer to step 17
25         $i \leftarrow i + 1$;                                   // Refer to step 18
26         Update allocations, $N_{m,i}$ for $m \in \{1,\ 2,\ 3\}$;    // Refer to step 19
27         if $N_{m,i} < 5$ then
28             $N_{m,i} \leftarrow 5$ and update allocations accordingly;   // Refer to step 20
29     Generate neighbouring solution $\boldsymbol{x}'$ according to move operator $m$;   // Refer to step 21
30     Evaluate neighbouring solution $\boldsymbol{x}'$;               // Refer to step 23
31     Follow steps 8–36 as described in Algorithm 5.13;           // Refer to step 24
32     Update $rate_t$, using (5.25);                              // Refer to step 25
33 $\mathcal{P}_s \leftarrow \mathcal{A}$;                          // Approximate Pareto set
```

23. Evaluate the neighbouring solution ($\boldsymbol{x}'$), *i.e.* the simulation model evaluates $\boldsymbol{x}$.

Next, execute the move acceptance mechanism of the DBMOSA algorithm, by following steps 8–36 in Algorithm 5.13, as discussed in §5.4. Naturally, the move acceptance strategy was chosen as simulated annealing.

24. Apply the move acceptance mechanism, by following steps 8–36 in Algorithm 5.13, refer to §5.4.

25. Update $rate_t$, using (5.25).

This concludes the loop. At termination, the archive $\mathcal{A}$ is returned as an approximate of the true

Pareto set of solutions to the bi-objective simulation optimisation problem under consideration, *i.e.* $\mathcal{P}_S \leftarrow \mathcal{A}$. A pseudo-code description of the entire process described above is provided in Algorithm 5.17, with specific reference to the above-mentioned steps and concludes the documentation of both hyperheuristics proposed in this study.

5.7 Summary

This chapter discussed some of the main concepts of S- and P-metaheuristics, with specific focus on their implementations for each respective metaheuristic considered in this study. The MOOCEM, the NSGA-II and the DBMOSA algorithms were discussed in the context of simulation optimisation and their integration in Tecnomatix. Next, the two hyperheuristics were presented, first for population-based search employing MOOCEM and NSGA-II as LLHs and then for single-solution based search employing three move operators.

The next chapter determines which hyperparameter combination, for each metaheuristic respectively, should be employed for the respective hyperheuristics.

CHAPTER 6

Algorithmic Parameter Evaluation

The previous chapter discussed the hyperheuristics BOCEGAH and BOSAH and their respective LLHs with specific reference to the simulation optimisation process as it occurs in Tecnomatix.

This chapter documents the empirical (and statistically sound) hyperparameter study conducted which is deemed necessary due to the high number of hyperparameter combinations (or configurations) that algorithms have and a lack of insight on how to choose them, attributable to their complex interactions. Moreover, when considering more than one problem it becomes markedly challenging to suggest a common hyperparameter combination suitable for the subset of problems considered. Consequently, deciding which metaheuristic (or heuristic operators) to use is not a trivial task.

The first section introduces the non-parametric hypothesis test and corresponding *post hoc* test followed to determine the best hyperparameter combinations. Next, the hypothesis tests and *post hoc* tests (where necessary) are documented for the MOOCEM, then the NSGA-II and lastly for the DBMOSA, for all five discrete-event dynamic stochastic simulation optimisation problems, as discussed in Chapter 4. After the individual tests have been conducted (for each algorithm), the results are consolidated in order to make an informed decision regarding the hyperparameter combination when considering all five simulation problems. The chapter closes with a conclusion of its contents.

6.1 Introduction to multi-objective statistical analysis

As mentioned in §2 the term *quality performance* is used, instead of the term *performance* which pertains to both time and quality. In this study *only* quality is assessed based on an algorithm's performance in the objective (or solution) space for a fixed number of evaluations (chosen as

1 000). As mentioned (in Chapter 5) the solutions obtained by the MOOCEM, the NSGA-II and the DBMOSA are referred to as approximation sets which aim to approximate the true Pareto set. In this chapter, the approximation sets are compared with each other based on the obtained hyperareas and number of non-dominated solutions in the approximation sets.

Recall that each algorithm contains stochastic elements and operate on stochastic problems (in this study), therefore, the resulting approximation sets vary from experimental run to experimental run. In order to derive significant statistical results, many experimental runs must be carried out, at least 10 or more than 100 if possible [93]. However, due to the limited timeframe of this study, 100 runs are carried out (unless stated otherwise). Consequently, any statements about the quality of an experimental run is probabilistic in nature, *i.e.* the estimated parameter values may be numerically different, however, statistical tests are required to determine whether they are statistically different. For the sake of brevity, experimental runs are hereafter simply referred to as runs.

There are two main types of statistical tests in the literature: *parametric* and *nonparametric tests*. The decision between the two tests depends on the assumptions made about the properties of the underlying population (or distribution) [121]. To elaborate, parametric tests assume that the data fulfills three conditions, namely *normality*, *independency* and *homoscedasticity* [229]. However, these conditions cannot be assumed for output data generated by stochastic optimisation algorithms and therefore nonparametric tests are considered [48].

Recall (from Chapter 4) that the possible outcomes of a hypothesis test is that either the data supports the research question stated as the *null hypothesis* H_0 (which is assumed to be true) or disproves the research prediction stated as the *alternate hypothesis* H_1 [71]. The Friedman test is regarded an omnibus test, *i.e.* more than three groups of data is to be analysed and is considered the nonparametric counterpart of the parametric two-way (repeated measures) analysis of variance (ANOVA), performed on ordinal (or ranked) data [89].

The test statistic suggested by Friedman is given as [231]

$$\chi_F^2 = \frac{12}{nk(k+1)} \sum_{k=1}^{k} R_k^2 - 3n(k+1), \tag{6.1}$$

where $R_k = \sum_{i=1}^{n} R_{ik}$ is the sum of the ranks for sample k over the n observations and χ_F^2 is approximately χ^2-distributed. At a significance level of $\alpha = 0.05$, the null hypothesis is rejected if $\chi_F^2 \geq \chi_{k-1,1-\alpha}^2$, where $\chi_{k-1,1-\alpha}^2$ is the $(1-\alpha)$ quantile of the Chi-square distribution with $k-1$ degrees of freedom [206].

Iman and Davenport [129] noted that the χ^2 approximation was occasionally poor or too conservative and is therefore also included. The test statistic suggested by Iman and Davenport is referred to as *Iman-Davenport* is given by [129]

$$F_{ID} = \frac{(n-1)\chi_F^2}{n(k-1) - \chi_F^2}. \tag{6.2}$$

Similarly, at $\alpha = 0.05$, the null hypothesis is rejected if $F_{ID} \geq F_{k-1,(k-1)(n-1),1-\alpha}$, where $F_{k-1,(k-1)(n-1),1-\alpha}$ is the $(1-\alpha)$ quantile of the F-distribution with $(k-1)$ and $(k-1)(n-1)$ degrees of freedom.

Both statistical tests yield a p-value, and if $p < \alpha$, then H_0 is rejected. Indicating that at least one of the summed ranks are statistically significantly different, however, it does not indicate which groups. Therefore, a *post hoc* test is required to determine where the differences are.

There are several *multiple comparison post hoc* tests that can be applied, some of which are listed in Figure 6.1 [89].

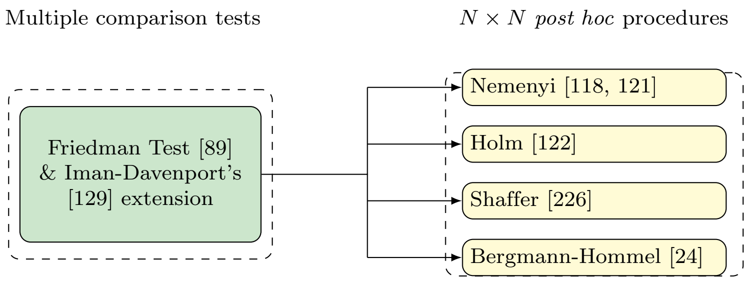

Multiple comparison tests
$N \times N$ *post hoc* procedures
Friedman Test [89] & Iman-Davenport's [129] extension
Nemenyi [118, 121]
Holm [122]
Shaffer [226]
Bergmann-Hommel [24]

Figure 6.1: *Non-parametric tests and post hoc procedures for $N \times N$ comparisons.*

The Nemenyi, formally called the Wilcoxon-Nemenyi-McDonald-Thompson *post hoc* test is categorised as more powerful than that of Holm and Hommel *post hoc* tests [121]. The procedure of determining the individual differences involves performing two-tailed pairwise significance tests between all pairs of samples, using the Friedman ranks. The test is performed using rank sums, and H_0 is rejected if

$$|R_{.k} - R_{.j}| \geq q_{k,\ n-k,\ 1-\alpha}\sqrt{\frac{nk(k+1)}{12}}, \tag{6.3}$$

where $q_{k,\ n-k,\ 1-\alpha}$ is the $(1-\alpha)$ of the studentised range distribution with k and $(n-k)$ degrees of freedom and requires equal sample sizes, *i.e.* $n_1 = n_2 = \ldots = n_k$ for each group k.

The Nemenyi *post hoc* test controls the familywise Type I error rate, correcting for the multiple inferences made by dividing the calculated p-values by the number of combinations determined as $\frac{k(k-1)}{2}$, thereby ensuring that the experiment-wide significance level $\alpha = 0.05$ is not exceeded.

The hypothesis testing procedure followed in this study is listed below:

1. State the null H_0 and alternate hypotheses H_1. Formally stated as,

   H_0: There is **no** significant difference between the ranked sums of the groups being compared.

   H_1: There **is** a significant difference between the ranked sums of the groups being compared and a post-hoc test is required to determine which groups.

2. State the level of significance, alpha: $\alpha = 0.05$.

3. Calculate the degrees of freedom:

   Friedman $df = k - 1$

   Iman-Davenport $df = \{k-1,\ (k-1)(n-1)\}$

   where k is the number of groups being compared and n is the number of observations. For the purpose of this paper $n = 100$ (unless stated otherwise), and k varies from algorithm to algorithm.

4. Get the corresponding critical values for both distributions, *i.e.* the χ^2-distribution and F-distribution, as χ^2_{k-1} and $F_{k-1,(k-1)(n-1)}$, respectively.

5. Calculate the test statistics for Friedman using (6.1) and Iman-Davenport using (6.2).

6. Determine the p-values.

7. Reject H_0 or fail to reject H_0.

   Friedman The null hypothesis is rejected if $\chi_F^2 \geq \chi_{k-1,\alpha}^2$ or if $p < \alpha$.

   Iman-Davenport The null hypothesis is rejected if $F_{ID} \geq F_{k-1,(k-1)(n-1),\alpha}$ or if $p < \alpha$.

6.2 Determining the MOOCEM algorithm hyperparameters

The MOOCEM contain many interacting hyperparameters to be adjusted and consequently, the purpose of this section is to determine the single best hyperparameter combination (for the MOOCEM) that results in the best algorithmic performance, across all five simulation problems. Algorithmic performance is compared based on the hyperareas obtained by the approximation fronts, as well as the number of non-dominated solutions in the fronts. The Friedman test and Iman-Davenport extension is used to compare these performances with regards to both measures. If the Friedman test or Iman-Davenport extension indicates significance, then the Nemenyi post-hoc test is conducted. Ultimately, the results are combined and used to decide which hyperparameter combination to implement in the BOCEGAH.

The population size N is typically chosen in the range [30, 100], in this study $N = 100$. Bekker [21] found that $\epsilon_c \in [0.1,\ 1]$ was sufficiently small, accordingly $\epsilon_c = 0.1$. The smoothing parameter α is typically in the range [0.6, 0.9] and the probability of inverting the decision variable histograms p_h is typically in the range [0.1, 0.3] [21].

It is important to note that a different *random number stream* (RNS) is used for each run, and the same RNS is used for the respective hyperparameter combinations, *i.e.* RNS = 1 is used for run 1, RNS = 2 for run 2 and so on, until RNS = 100 for run 100. The same setting applies for all the hyperparameter combinations as well as all the problems. The RNS is responsible for generating the initial population and therefore needs to be consistent for the same runs across the hyperparameter combinations. This applies for both the MOOCEM and the NSGA-II.

The MOOCEM parameters that are considered for the hyperparameter study are summarised in Table 6.1, where the naming convention is adopted to help navigate this section, *i.e.* A1.1.1 refers to the hyperparameter combination $\alpha = 0.65$ and $p_h = 0.2$ for the OMP specifically. Nine hyperparameter combinations are explored per simulation problem, *i.e.* the number of groups being compared is $k = 9$. Moreover, each hyperparameter combination (or group) is run for 100 individual (simulation) runs (unless stated otherwise), *i.e.* $n = 100$ observations per group resulting in 100 individual hyperareas and number of non-dominated solutions. Also, each run consists of 1 000 solution evaluations (as mentioned previously) to facilitate fair comparison.

The hypothesis tests conducted are first discussed for the OMP, then the IP, BAP5, BAP10 and finally for BAP16. Thereafter, the results of the respective tests are combined and a single best hyperparameter combination is chosen.

6.2.1 Open mine problem

This study includes a wide range of problem sizes, from the smallest problem the OMP with a decision space of 3 375 solutions to the largest problem, BAP16 with a decision space of 1×10^{15} solutions. The OMP, which has a relatively small decision space, is included as a method to validate the algorithms, because if an algorithm is able to approximately solve the OMP, then it can be assumed that the algorithm will be able to solve the larger problems too.

Table 6.1: *The hyperparameter search space and corresponding naming convention adopted refer to the specific hyperparameter combinations for the specific simulation problem, resulting in the entire hyperparameter search space considered for the MOOCEM.*

Smoothing parameter	Probability of inversion	OMP	IP	BAP5	BAP10	BAP16
	$p_h = 0.2$	A1.1.1	A1.2.1	A1.3.1	A1.4.1	A1.5.1
$\alpha = 0.65$	$p_h = 0.3$	A1.1.2	A1.2.2	A1.3.2	A1.4.2	A1.5.2
	$p_h = 0.4$	A1.1.3	A1.2.3	A1.3.3	A1.4.3	A1.5.3
	$p_h = 0.2$	A1.1.4	A1.2.4	A1.3.4	A1.4.4	A1.5.4
$\alpha = 0.70$	$p_h = 0.3$	A1.1.5	A1.2.5	A1.3.5	A1.4.5	A1.5.5
	$p_h = 0.4$	A1.1.6	A1.2.6	A1.3.6	A1.4.6	A1.5.6
	$p_h = 0.2$	A1.1.7	A1.2.7	A1.3.7	A1.4.7	A1.5.7
$\alpha = 0.75$	$p_h = 0.3$	A1.1.8	A1.2.8	A1.3.8	A1.4.8	A1.5.8
	$p_h = 0.4$	A1.1.9	A1.2.9	A1.3.9	A1.4.9	A1.5.9

Note that for the OMP, only 40 simulation runs per hyperparameter combination were executed, *i.e.* $n = 40$ observations. The reason for not executing 100 simulation runs as with the other problems is due to the time constraint of the project and the purpose of its inclusion in this study. The critical values for $k = 9$ and $n = 40$ for the Chi-square and F-distributions are $\chi^2_8 = 15.51$ and $F_{8,312} = 2.68$, respectively.

When calculating the hyperareas for the various approximation fronts, larger hyperarea values (than that of the true hyperarea) were obtained, this is demonstrated in Figure 6.2. Upon investigation of the approximation fronts, the discrepancy seems reasonable, and is attributable to the discontinuity of the approximation fronts and the method used to calculate the hyperarea of the front. From literature, it is clear that the better approximation front should have the largest hyperarea and consequently the hyperarea of the true approximation front should be the largest. For this reason, a linear adjustment is proposed to account for the discontinuity present in the OMPs approximation fronts, given as

$$I_{H\text{adj}} = I_H \left(\frac{NDS}{NDS_{\text{true}}} \right), \text{ for } NDS \leq NDS_{\text{true}}, \tag{6.4}$$

where I_H is the hyperarea calculated as described in §2.2.4, NDS is the number of non-dominated solutions in the approximation front and NDS_{true} is the number of non-dominated solutions in the true Pareto front, refer to Chapter 4.

Note that the approximation front with the largest hyperarea does not necessarily correspond to the approximation front with the most non-dominated solutions found, *i.e.* it is possible for an approximation front to have more non-dominated solutions than that of the true non-dominated front, because the non-dominated solutions obtained by the approximation fronts are not necessarily the true non-dominated solutions of the true Pareto front.

For example, consider the two Pareto fronts in Figure 6.2, the true Pareto front contains 36 non-dominated solutions with a true hyperarea value of 14 485.04 presented by the blue dot area, and the other (larger) approximation front has 34 non-dominated solutions with a hyperarea value of 14 617.45 with the adittional red area. Due to the discontinuity, the approximation front is larger denoted by the red area. To account for this, the adjusted hyperarea is calculated by applying (6.4), such that

$$
\begin{aligned}
I_{H\text{adj}} &= 14617.45\left(\frac{34}{36}\right)\\
&= 13805.37.
\end{aligned}
$$

The correction holds true for the case where the true Pareto set is found, since the obtained hyperarea would simply be multiplied by 1, as both sets contain 36 non-dominated solutions.

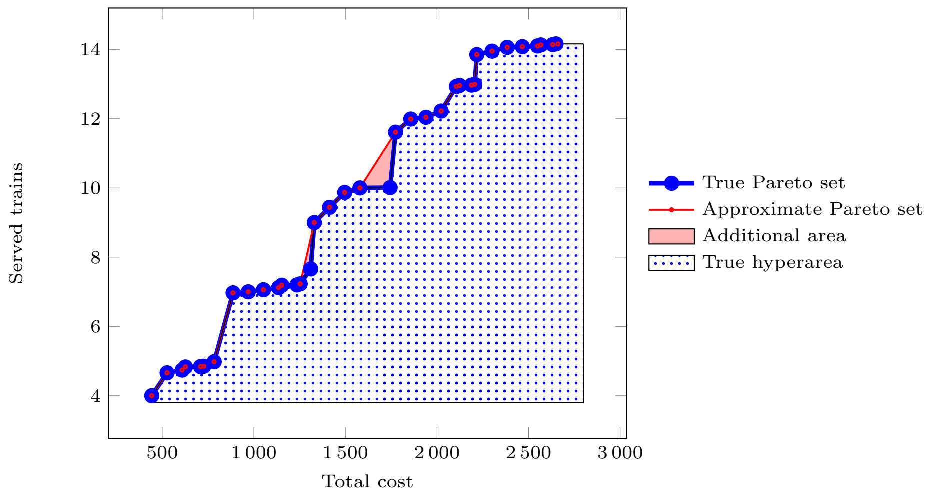


Figure 6.2: *An example illustrating that a larger hyperarea value than the true hyperarea value can exist (for a worst approximation front), when the front is discontinuous.*

Tables B.1 and B.2 present the adjusted hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 39–40 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. For example, the values in the columns of Table B.1 represent a snapshot of the adjusted hyperareas and their ranks obtained for $\alpha = 0.65$ and $p_h = 0.2$ (or A1.1.1) for *Run 1–10 and 39–40*. The boxplots in Figure 6.3 provide a graphical summary of the results for hyperparameter combinations A1.1.1–A1.1.9, as presented in Tables B.1 and B.2.

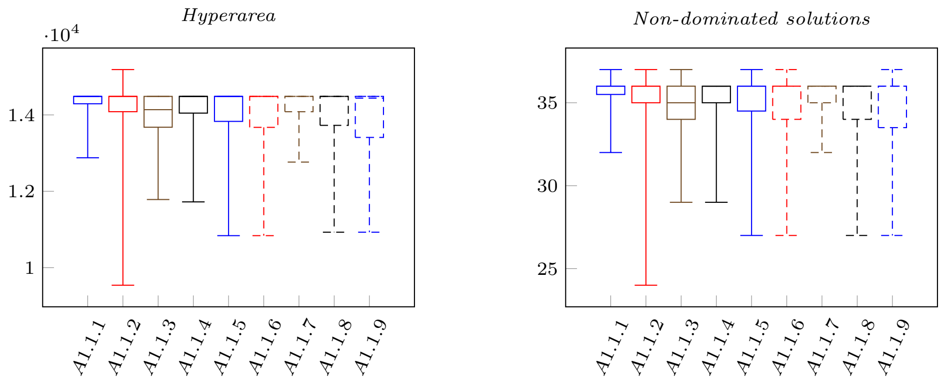


Figure 6.3: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A1.1.1–A1.1.9.*

Table 6.2 shows the number of runs (out of 40) that obtained the true Pareto set, for each hyperparameter combination. For example, for $\alpha = 0.65$ and $p_h = 0.2$, the MOOCEM obtained the true Pareto set 24 out of the 40 times (*i.e.* 60% of the time). Consequently, validating that the MOOCEM algorithm works correctly and will be able to solve the larger problems that follow this discussion. The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.3 and depicted in Figures B.1 and B.2 for A1.1.1–A1.1.9.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.3. The Friedman and Iman-Davenport test statistics for both the adjusted hyperareas (and the number of non-dominated solutions) indicate significance ($\chi^2_F = \{22.11,\ 22.15\} > \chi^2_8,\ p < \alpha$ and $F_{ID} = \{2.89,\ 2.9\} > F_{8,312},\ p < \alpha$). The outcome is to reject H_0, which means that there exists a statistically significant difference between at least one of the groups. Consequently, the Nemenyi *post hoc* test is conducted to determine between which groups a statistically significant difference exists.

Table 6.2: *The number of runs that the MOOCEM obtained the true Pareto set, for the corresponding hyperparameter combinations A1.1.1–A1.1.9.*

A1.1.1	A1.1.2	A1.1.3	A1.1.4	A1.1.5	A1.1.6	A1.1.7	A1.1.8	A1.1.9
24	23	14	27	27	20	26	21	20

Table 6.3: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A1.1.1–A1.1.9.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	22.11	0.01	2.89	0.004	Reject H_0
Non-dominated solutions	22.15	0.01	2.9	0.004	

The Nemenyi *post hoc* test is presented in Tables B.4 and B.5 and represent the upper triangle of the matrix that contains the adjusted p-values of the multiple pairwise comparisons for both the adjusted hyperareas and number of non-dominated solutions, respectively. The Friedman and Iman-Davenport extension both detected a significant difference (for $\alpha = 0.05$) between the groups, however upon further analysis, the Nemenyi *post hoc* test found that there was no statistically significant difference between the groups. Consequently, the hyperparameter combination employed do not influence the performance of the MOOCEM (on the OMP) and consequently the hyperparameter combinations A1.1.1–A1.1.9 are equally desirable.

6.2.2 (s, S) Inventory problem

Note that for the IP, BAP5, BAP10 and BAP16, 100 simulation runs per hyperparameter combination were executed, *i.e.* $n = 100$ observations per group. The critical values for $k = 9$ and $n = 100$ for the Chi-square and F-distributions are $\chi^2_8 = 15.51$ and $F_{8,792} = 1.95$, respectively. Note that the hyperareas are not adjusted for the IP, BAP5, BAP10 or BAP16, since the approximation fronts are not discontinuous. Tables B.6 and B.7 present the hyperareas and number of non-dominated solutions and their ranks for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests.

For example, the values in the columns of Table B.6 represent a snapshot of the hyperareas and their ranks obtained for $\alpha = 0.65$ and $p_h = 0.2$ (or A1.2.1) for *Run 1–10 and 99–100*. The boxplots in Figure 6.4 provide a graphical summary of the results for hyperparameter combinations A1.2.1–A1.2.9, as presented in Tables B.6 and B.7.

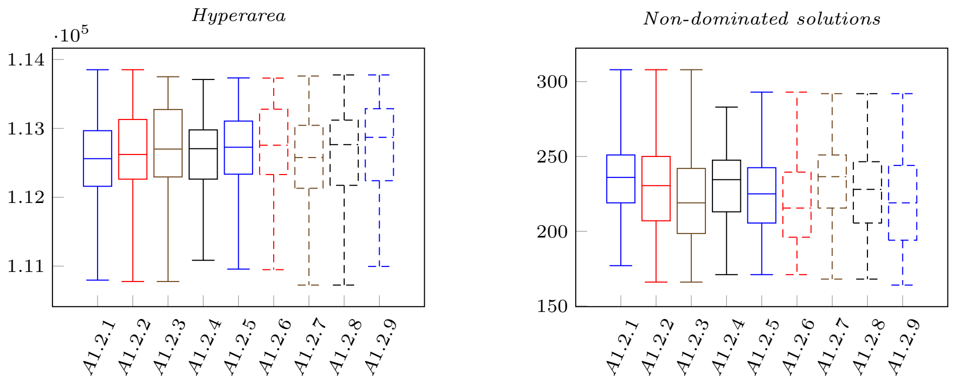


Figure 6.4: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A1.2.1–A1.2.9.*

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.8 and depicted in Figures B.3 and B.4 for A1.2.1–A1.2.9.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.4. The Friedman and Iman-Davenport test statistics do not indicate significance for the hyperareas ($\chi^2_F = 6.76 < \chi^2_8,\ F_{ID} = 0.846 < F_{8,792},\ p > \alpha$). Therefore, do not reject H_0. The tests, however, do indicate significance for the number of non-dominated solutions found ($\chi^2_F = 63.86 > \chi^2_8,\ F_{ID} = 8.59 > F_{8,792},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

Table 6.4: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A1.2.1–A1.2.9.*

	Friedman test	*p*-value	Iman-Davenport test	*p*-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	6.76	0.56	0.84	0.56	Fail to reject H_0
Non-dominated solutions	63.86	0	8.59	0	Reject H_0

The Nemenyi *post hoc* test results are presented in Table 6.5 and contains the adjusted *p*-values of the multiple pairwise comparisons for number of non-dominated solutions. The red *p*-values indicate statistical significance, for example the Nemenyi *post hoc* test found that there are statistically significant differences between A1.2.1 and A1.2.3, A1.2.5–A1.2.6 and A1.2.8–A1.2.9.

Table 6.5: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A1.2.1–A1.2.9.*

	A1.2.2	A1.2.3	A1.2.4	A1.2.5	A1.2.6	A1.2.7	A1.2.8	A1.2.9
A1.2.1	1	0.02	0.72	0	0	1	0.04	0
A1.2.2		0.92	1	0.08	0	1	1	0.01
A1.2.3			1	1	0.17	1	1	1
A1.2.4				1	0	1	1	0.22
A1.2.5					1	0.19	1	1
A1.2.6						0	0.09	1
A1.2.7							1	0.01
A1.2.8								1

6.2.3 Buffer allocation problem: five machines

Tables B.9 and B.10 present the hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.5 provide a graphical summary of the results for hyperparameter combinations A1.3.1–A1.3.9, as presented in Tables B.9 and B.10.

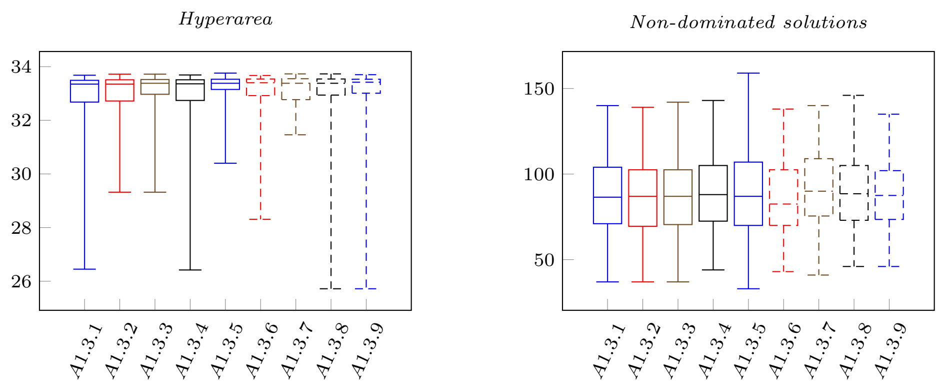


Figure 6.5: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A1.3.1–A1.3.9.*

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.11 and depicted in Figures B.5 and B.6 for A1.3.1–A1.3.9.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.6. The Friedman and Iman-Davenport test statistics do not indicate significance for the hyperareas ($\chi_F^2 = 13.8 < \chi_8^2,\ F_{ID} = 1.74 < F_{8,792},\ p > \alpha$) and therefore, fail to reject H_0. The tests, however, do indicate significance for the number of non-dominated solutions found ($\chi_F^2 = 33.78 > \chi_8^2,\ F_{ID} = 4.37 > F_{8,792},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

Table 6.6: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A1.3.1–A1.3.9.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ_F^2	p	F_{ID}	p	
Hyperarea	13.8	0.09	1.74	0.09	Fail to reject H_0
Non-dominated solutions	33.78	0	4.37	0	Reject H_0

The Nemenyi *post hoc* test is presented in Table 6.7 and contains the adjusted p-values of the multiple pairwise comparisons for number of non-dominated solutions. The red p-values indicate statistical significance, for example the Nemenyi *post hoc* test found that there was only statistically significant differences between A1.3.7 and A1.3.2–A1.3.3 and A1.3.6.

6.2.4 Buffer allocation problem: 10 machines

Tables B.12 and B.13 present the hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 99–100 as well as the sum of the ranks which is used in the

Table 6.7: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A1.3.1–A1.3.9.*

	A1.3.2	A1.3.3	A1.3.4	A1.3.5	A1.3.6	A1.3.7	A1.3.8	A1.3.9
A1.3.1	1	1	1	1	1	0.27	1	1
A1.3.2		1	0.21	1	1	0	0.1	0.98
A1.3.3			0.17	1	1	0	0.08	0.83
A1.3.4				1	0.78	1	1	1
A1.3.5					1	0.15	1	1
A1.3.6						0.01	0.41	1
A1.3.7							1	1
A1.3.8								1

Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.6 provide a graphical summary of the results for hyperparameter combinations A1.4.1–A1.4.9, as presented in Tables B.12 and B.13.

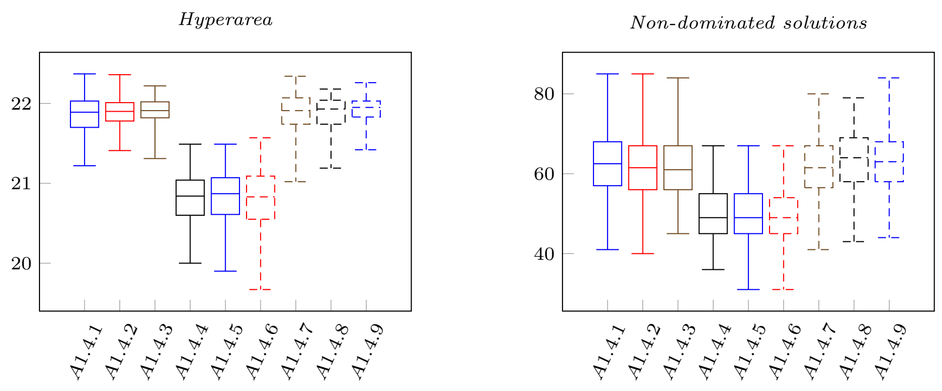


Figure 6.6: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A1.4.1–A1.4.9.*

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are given in Table B.14 and presented in Figures B.7 and B.8 for A1.4.1–A1.4.9.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.8. The Friedman and Iman-Davenport test statistics for both the adjusted hyperareas (and the number of non-dominated solutions) indicate strong significance ($\chi^2_F = \{529.1,\ 368.75\} > \chi^2_8,\ p < \alpha$ and $F_{ID} = \{193.36,\ 84.65\} > F_{8,792},\ p < \alpha$). Therefore, the outcome is to reject H_0, which means that there exists a significance difference between at least one of the groups in terms of both hyperarea and number of non-dominated solutions. Consequently, the Nemenyi *post hoc* test is conducted to determine between which groups a statistically significant difference exists.

The Nemenyi *post hoc* test is presented in Table 6.9 and contains the adjusted p-values of the multiple pairwise comparisons for both hyperareas and number of non-dominated solutions, since the test resulted in the same p-values. For example, the Nemenyi *post hoc* test found that there was a statistically significant difference between A1.4.1 and A1.4.4–A1.4.6 (for both the hyperareas and number of non-dominated solutions).

Table 6.8: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A1.4.1–A1.4.9.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ_F^2	p	F_{ID}	p	
Hyperarea	529.1	0	193.36	0	Reject H_0
Non-dominated solutions	368.75	0	84.65	0	

Table 6.9: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the hyperareas and number of non-dominated solution found of the approximation fronts for the hyperparameter combinations A1.4.1–A1.4.9.*

	A1.4.2	A1.4.3	A1.4.4	A1.4.5	A1.4.6	A1.4.7	A1.4.8	A1.4.9
A1.4.1	1	1	0	0	0	1	1	1
A1.4.2		1	0	0	0	1	1	1
A1.4.3			0	0	0	1	1	1
A1.4.4				1	1	0	0	0
A1.4.5					1	0	0	0
A1.4.6						0	0	0
A1.4.7							1	1
A1.4.8								1

6.2.5 Non-linear buffer allocation problem: 16 machines

Tables B.15 and B.16 present the hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.7 provide a graphical summary of the results for hyperparameter combinations A1.5.1–A1.5.9, as presented in Tables B.15 and B.16. The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.17 and depicted in Figures B.9 and B.10 for A1.5.1–A1.5.9.

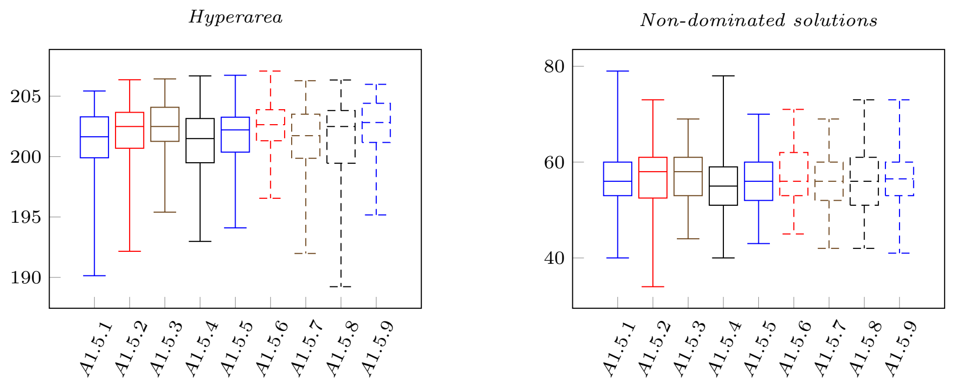


Figure 6.7: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A1.5.1–A1.5.9.*

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.10. The Friedman and Iman-Davenport test statistics indicate significance for the hyperareas ($\chi_F^2 = 41.66 > \chi_8^2$, $F_{ID} = 5.44 > F_{8,792}$, $p < \alpha$). Therefore, the outcome is to reject H_0 for the hyperareas, which means that there exists a significant difference between at least one of the groups in terms of the hyperarea value. Consequently, the Nemenyi *post hoc* test is required to determine between which groups a statistically significant difference exists. The tests, however, do not indicate significance for the number of non-dominated solutions found ($\chi_F^2 = 10.28 < \chi_8^2,\ F_{ID} = 1.29 < F_{8,792},\ p > \alpha$) and therefore, fail to reject H_0.

Table 6.10: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A1.5.1–A1.5.9.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ_F^2	p	F_{ID}	p	
Hyperarea	41.66	0	5.44	0	Reject H_0
Non-dominated solutions	10.28	0.25	1.29	0.25	Fail to reject H_0

The Nemenyi *post hoc* test is presented in Table 6.11 and contains the adjusted p-values of the multiple pairwise comparisons for hyperareas. For example, the Nemenyi *post hoc* test found that there is a statistically significant difference between A1.5.1 and A1.5.3, A1.5.6 and A1.5.9.

Table 6.11: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the hyperareas of the approximation fronts for the hyperparameter combinations A1.5.1–A1.5.9.*

	A1.5.2	A1.5.3	A1.5.4	A1.5.5	A1.5.6	A1.5.7	A1.5.8	A1.5.9
A1.5.1	1	0	1	1	0	1	1	0
A1.5.2		1	1	1	0.73	1	1	1
A1.5.3			0.02	0.33	1	0.21	0.63	1
A1.5.4				1	0.01	1	1	0.02
A1.5.5					0.14	1	1	0.38
A1.5.6						0.08	0.28	1
A1.5.7							1	0.24
A1.5.8								0.73

This concludes the individual (or problem-specific) analysis of the hyperparameter combinations for the MOOCEM. Finally, the single best hyperparameter combination can be determined for the MOOCEM, across all five simulation problems. Note that only the hyperareas are considered, since the hyperarea is the better indicator of the quality of an approximation front. Figure 6.8 represents the average hyperareas obtained for each respective hyperparameter combination (collectively referred to as A1.1–A1.9) for each simulation problem. The average number of non-dominated solutions obtained for hyperparameter combinations A1.1–A1.9 is presented in Figure B.11 for each simulation problem.

Figure 6.9 is used to explain the process followed to determine the single best hyperparameter combination (specifically for BAP10) for the MOOCEM. From the p-values given in Table 6.9 and included in the figure, it can be seen that hyperparameter combinations A1.4.4–A1.4.6 are statistically significantly different to A1.4.1–A1.4.3 and A1.4.7–A1.4.9 which is illustrated by the red rectangles in Figure 6.9. Because A1.4.4–A1.4.6 have the smaller average hyperareas they are eliminated from consideration. Accordingly, A1.4.1–A1.4.3 and A1.4.7–A1.4.9 are not statistically significantly different and are therefore considered. A similar process was followed for each problem and the outcome is presented in Table 6.12, where it is clear that hyperparameter combination A1.2–A1.3 and A1.7–A1.9 are the common denominators across all five simulation problems.

Next, the hyperparameter combination (*i.e.* A1.2–A1.3 and A1.7–A1.9) that obtains the largest average hyperarea (for most of the time) is chosen. The green checkmarks in Table 6.12 correspond to the hyperparameter combinations with the largest average hyperarea. Consequently, because hyperparameter combination A1.9 (or $\alpha = 0.75$ and $p_h = 0.4$) resulted in the largest

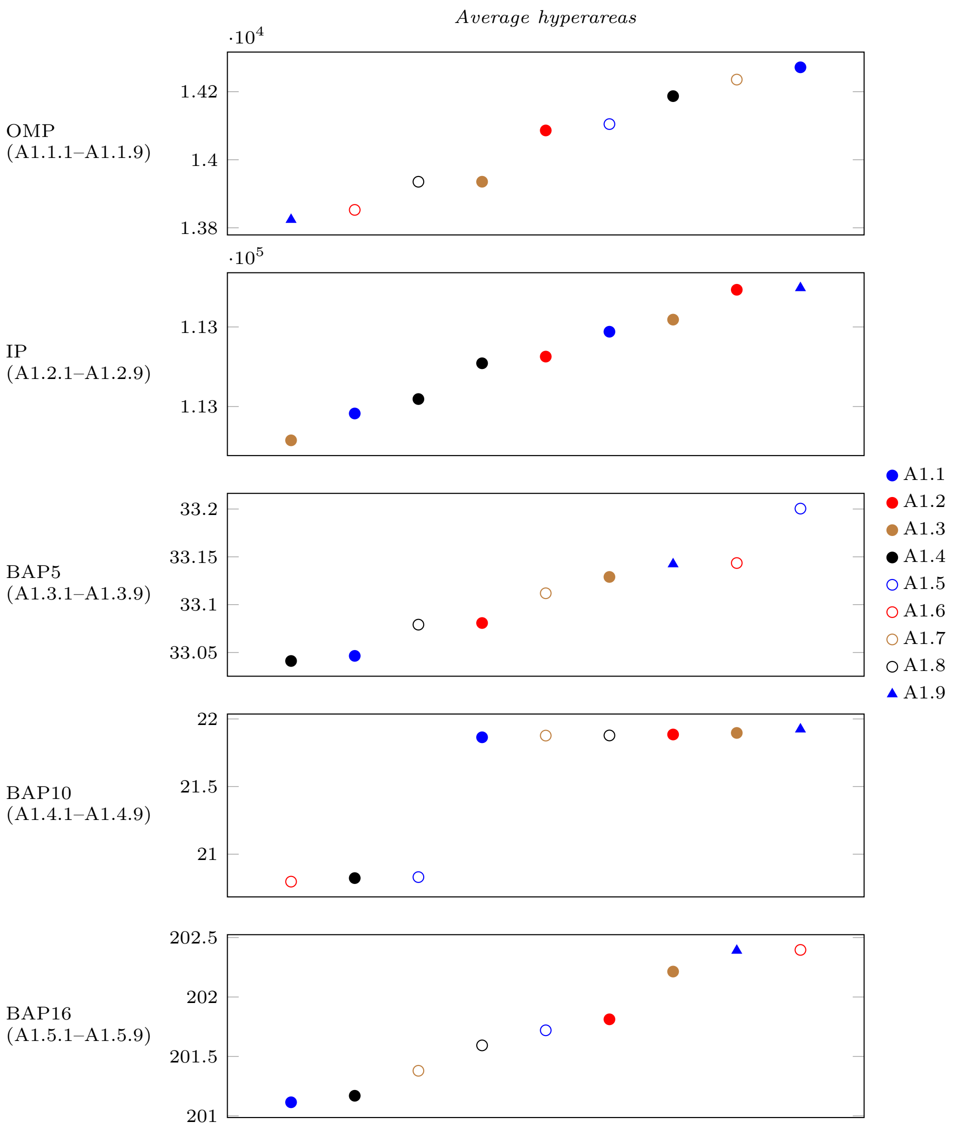


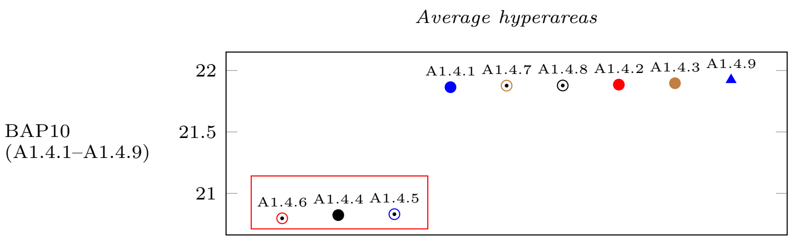
Figure 6.8: *The average hyperareas obtained for hyperparameter combinations A1.1–A1.9 for the respective simulation problems.*

average hyperarea for four out of the five problems, it is employed in the BOCEGAH. Note that hyperparameter combinations A1.2–A1.3 and A1.7–A1.9 are not statistically significantly different, *i.e.* any one of these combinations could be used, however, one had to be chosen.

Interestingly, Bekker [21] found that the probability of inverting the decision variable histograms p_h is typically in the range [0.1, 0.3], however the hyperparameter study conducted for the MOOCEM on the five simulation problems concluded that $p_h = 0.4$ is the common best hyperparameter combination.

This concludes the empirical study determining the hyperparameter combination to be employed for the MOOCEM in the BOCEGAH. The results can be used as a guideline for choosing hyperparameters for the MOOCEM, when deciding on suitable hyperparameter combinations to employ, or is at least a good place to start.

The next section documents the extensive parametric study conducted for the NSGA-II.



	A1.4.2	A1.4.3	A1.4.4	A1.4.5	A1.4.6	A1.4.7	A1.4.8	A1.4.9
A1.4.1	1	1	0	0	0	1	1	1
A1.4.2		1	0	0	0	1	1	1
A1.4.3			0	0	0	1	1	1
A1.4.4				1	1	0	0	0
A1.4.5					1	0	0	0
A1.4.6						0	0	0
A1.4.7							1	1
A1.4.8								1

Figure 6.9: *An example illustrating the hyperparameter selection process followed for BAP10 based on the Friedman and Iman-Davenport extension hypothesis test and Nemenyi post hoc test conducted for the hyperareas obtained for hyperparameter combinations A1.4.1–A1.4.9.*

Table 6.12: *The hyperparameters for the MOOCEM that are considered for implementation in the BOCEGAH.*

	A1.1	A1.2	A1.3	A1.4	A1.5	A1.6	A1.7	A1.8	**A1.9**
OMP	✓	✓	✓	✓	✓	✓	✓	✓	✓
IP	✓	✓	✓	✓	✓	✓	✓	✓	✓
BAP5	✓	✓	✓	✓	✓	✓	✓	✓	✓
BAP10	✓	✓	✓				✓	✓	✓
BAP16		✓	✓		✓	✓	✓	✓	✓

6.3 Determining the NSGA-II hyperparameters

The NSGA-II contains many interacting hyperparameters to be adjusted and consequently, the purpose of this section is to determine the single best hyperparameter combination for the NSGA-II that results in the best algorithmic performance, across all five simulation problems. Ultimately, the results of the Friedman and Iman-Davenport extension tests are combined and used to decide which hyperparameter combination to implement for NSGA-II and employ in the BOCEGAH.

Population size N is typically chosen in the range $[20,\ 100]$ because large populations result in better convergence toward the true Pareto front, however, the time complexity increases linearly with the size of the population [93]. In this study $N = 100$.

Crossover probability p_c typically ranges between $[0.3,\ 0.9]$. In this study the $p_c \in \{0.9,\ 1\}$ is considered, *i.e.* crossover operations are applied to 90% and 100% of the parents.

Mutation probability p_m typically ranges between $[0.001,\ 0.01]$, where large probabilities could disrupt the search. Usually, the $p_m = 1/n_d$, where n_d is the number of decision variables, *i.e.* on average, one variable is mutated [93]. However, because the number of decision variables differs from simulation problem to simulation problem, ranging from 2 to 15, $p_m \in \{0.1,\ 0.2,\ 0.3\}$ is considered.

The NSGA-II parameters that are considered for the hyperparameter study are summarised in Table 6.13. A similar naming convention is adopted as before, where A2.1.1 refers to the hyperparameter combination rank selection, polynomial mutation, $p_c = 0.9$ and $p_m = 0.1$ for the OMP specifically. A search space of 24 combinations per simulation problem is explored, *i.e.* the number of groups being compared is $k = 24$.

The search space explores selection operators, binary tournament selection and rank selection, crossover operator simulated binary crossover, mutation operators polynomial mutation and dynamic mutation and (as mentioned) crossover probabilities $p_c \in \{0.9,\ 1\}$ and mutation probabilities $p_m \in \{0.1,\ 0.2,\ 0.3\}$ with a maximum of 1 000 solutions evaluations and a population size $N = 100$. Again, each hyperparameter combination (or group) is run for 100 individual (simulation) runs (unless stated otherwise), *i.e.* $n = 100$ observations per group resulting in 100 individual hyperareas and number of non-dominated solutions.

Table 6.13: *The hyperparameter search space and corresponding naming convention adopted refer to the specific hyperparameter combinations for the specific simulation problem, resulting in the entire hyperparameter search space considered for the NSGA-II.*

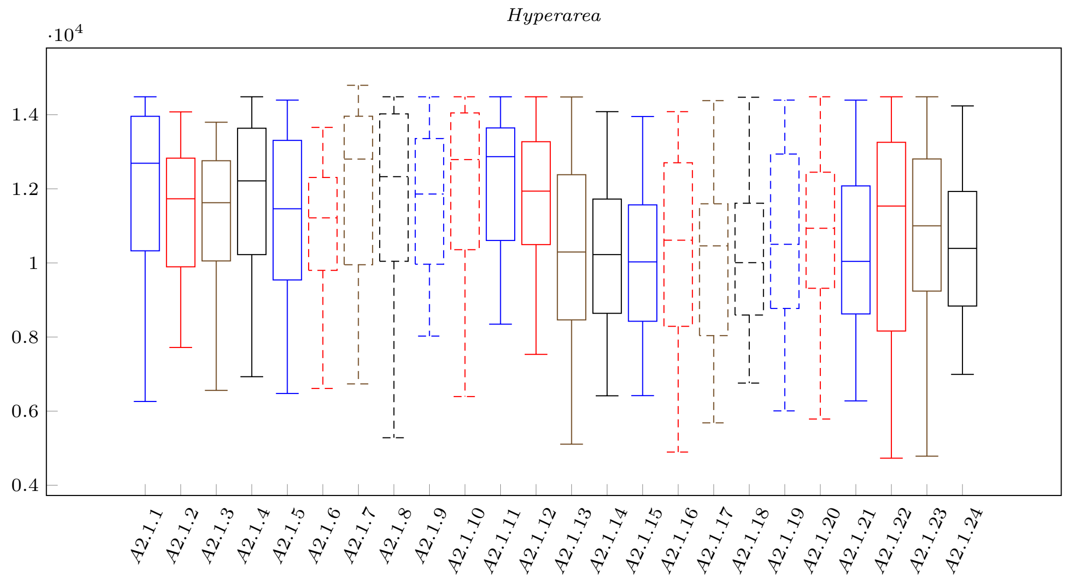
Selection operator	Mutation operator	Crossover probability	Mutation probability	OMP	IP	BAP5	BAP10	BAP16
Rank selection	Polynomial mutation	$p_c = 0.9$	$p_m = 0.1$	A2.1.1	A2.2.1	A2.3.1	A2.4.1	A2.5.1
			$p_m = 0.2$	A2.1.2	A2.2.2	A2.3.2	A2.4.2	A2.5.2
			$p_m = 0.3$	A2.1.3	A2.2.3	A2.3.3	A2.4.3	A2.5.3
		$p_c = 1$	$p_m = 0.1$	A2.1.4	A2.2.4	A2.3.4	A2.4.4	A2.5.4
			$p_m = 0.2$	A2.1.5	A2.2.5	A2.3.5	A2.4.5	A2.5.5
			$p_m = 0.3$	A2.1.6	A2.2.6	A2.3.6	A2.4.6	A2.5.6
	Dynamic mutation	$p_c = 0.9$	$p_m = 0.1$	A2.1.7	A2.5.7	A2.3.7	A2.4.7	A2.5.7
			$p_m = 0.2$	A2.1.8	A2.2.8	A2.3.8	A2.4.8	A2.5.8
			$p_m = 0.3$	A2.1.9	A2.2.9	A2.3.9	A2.4.9	A2.5.9
		$p_c = 1$	$p_m = 0.1$	A2.1.10	A2.2.10	A2.3.10	A2.4.10	A2.5.10
			$p_m = 0.2$	A2.1.11	A2.2.11	A2.3.11	A2.4.11	A2.5.11
			$p_m = 0.3$	A2.1.12	A2.2.12	A2.3.12	A2.4.12	A2.5.12
Binary tournament selection	Polynomial mutation	$p_c = 0.9$	$p_m = 0.1$	A2.1.13	A2.2.13	A2.3.13	A2.4.13	A2.5.13
			$p_m = 0.2$	A2.1.14	A2.2.14	A2.3.14	A2.4.14	A2.5.14
			$p_m = 0.3$	A2.1.15	A2.2.15	A2.3.15	A2.4.15	A2.5.15
		$p_c = 1$	$p_m = 0.1$	A2.1.16	A2.2.16	A2.3.16	A2.4.16	A2.5.16
			$p_m = 0.2$	A2.1.17	A2.2.17	A2.3.17	A2.4.17	A2.5.17
			$p_m = 0.3$	A2.1.18	A2.2.18	A2.3.18	A2.4.18	A2.5.18
	Dynamic mutation	$p_c = 0.9$	$p_m = 0.1$	A2.1.19	A2.2.19	A2.3.19	A2.4.19	A2.5.19
			$p_m = 0.2$	A2.1.20	A2.2.20	A2.3.20	A2.4.20	A2.5.20
			$p_m = 0.3$	A2.1.21	A2.2.21	A2.3.21	A2.4.21	A2.5.21
		$p_c = 1$	$p_m = 0.1$	A2.1.22	A2.2.22	A2.3.22	A2.4.22	A2.5.22
			$p_m = 0.2$	A2.1.23	A2.2.23	A2.3.23	A2.4.23	A2.5.23
			$p_m = 0.3$	A2.1.24	A2.2.24	A2.3.24	A2.4.24	A2.5.24

The hypothesis tests conducted are first discussed for the OMP, then the IP, BAP5, BAP10 and finally for BAP16. Thereafter, the results of the respective tests are combined and a single best hyperparameter combination is chosen. Note that the previous sections (§6.2.1–§6.2.5) wil be used as a template to succinctly present the remainder of the results.

6.3.1 Open mine problem

Recall that for the OMP, only 40 simulation runs per hyperparameter combination were executed, *i.e.* $n = 40$ observations. Accordingly, the critical values for $k = 24$ and $n = 40$ for the Chi-square and F-distributions are $\chi^2_{23} = 35.17$ and $F_{23,897} = 1.54$, respectively. Tables

B.1 and B.2 present the adjusted hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 39–40 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.10 provide a graphical summary of the results for hyperparameter combinations A2.1.1–A2.1.24, as presented in Tables B.18 and B.19.



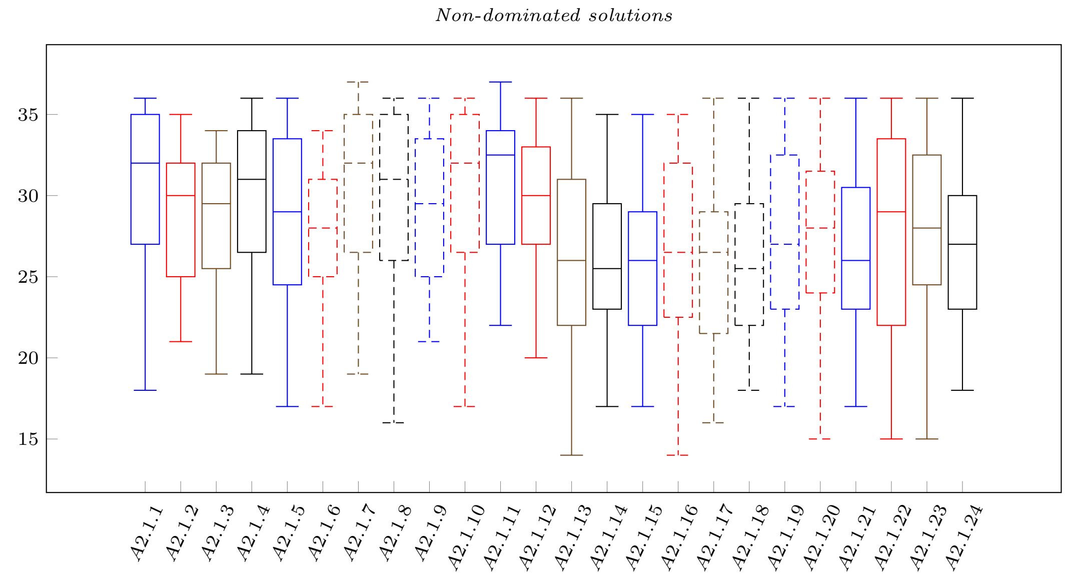


Figure 6.10: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A2.1.1–A2.1.24.*

Table 6.14 shows the number of runs (out of 40) that obtained the true Pareto set, for each hyperparameter combination. For example, for hyperparameter combination A2.1.1, the NSGA-II obtained the true Pareto set 3 out of the 40 times (*i.e.* 7.5% of the time). Consequently, validating that the NSGA-II algorithm works correctly and will be able to solve the larger problems that follow this discussion.

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.20 and depicted in Figures B.12–B.14 for A2.1.1–A2.1.24.

Table 6.14: *The number of runs that the NSGA-II obtained the true Pareto set, for the respective hyperparameter combinations.*

A2.1.1	A2.1.4	A2.1.7	A2.1.8	A2.1.9	A2.1.10	A2.1.11	A2.1.12	A2.1.20	A2.1.22	A2.1.23
3	2	2	4	1	4	1	1	3	1	2

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.15. The Friedman and Iman-Davenport test statistics for both the adjusted hyperareas (and the number of non-dominated solutions) indicate significance ($\chi^2_F = \{304.76,\ 285.67\} > \chi^2_{23},\ p < \alpha$ and $F_{ID} = \{19.32,\ 17.56\} > F_{23,897},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

Table 6.15: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A2.1.1–A2.1.24.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	304.76	0	19.32	0	Reject H_0
Non-dominated solutions	285.67	0	17.56	0	

The Nemenyi *post hoc* test is presented in Tables B.21 and B.22 and contains the adjusted p-values of the multiple pairwise comparisons for both the adjusted hyperareas and number of non-dominated solutions, respectively. The p-values in red indicate statistical significance, for example A1.1.1 and A1.1.13–A1.1.19, A1.1.21–A1.1.22 and A1.1.24.

6.3.2 (s, S) Inventory problem

Recall that for the IP, BAP5, BAP10 and BAP16, 100 simulation runs per hyperparameter combination were executed, *i.e.* $n = 100$ observations per group. The critical values for $k = 24$ and $n = 100$ for the Chi-square and F-distributions are $\chi^2_{23} = 35.17$ and $F_{23,2277} = 1.53$, respectively. Tables B.23 and B.24 present the hyperareas and number of non-dominated solutions and their ranks for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.11 provide a graphical summary of the results for hyperparameter combinations A2.2.1–A2.2.24, as presented in Tables B.23 and B.24.

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are given in Table B.25 and presented in Figures B.15 and B.19 for A2.2.1–A2.2.24.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.16. The Friedman and Iman-Davenport test statistics both indicate significance for both the hyperarea and non-dominated solutions ($\chi^2_F = \{171.73,\ 1030.87\} > \chi^2_{23},\ F_{ID} = \{7.99,\ 80.41\} > F_{23,2277},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

The Nemenyi *post hoc* test is presented in Tables B.26 and B.27 and contains the adjusted p-values of the multiple pairwise comparisons for number of non-dominated solutions. The red p-values indicate statistical significance, for example there are statistically significant differences between A2.2.1 and A2.2.13, A2.2.15, A2.2.18 and A2.2.19.

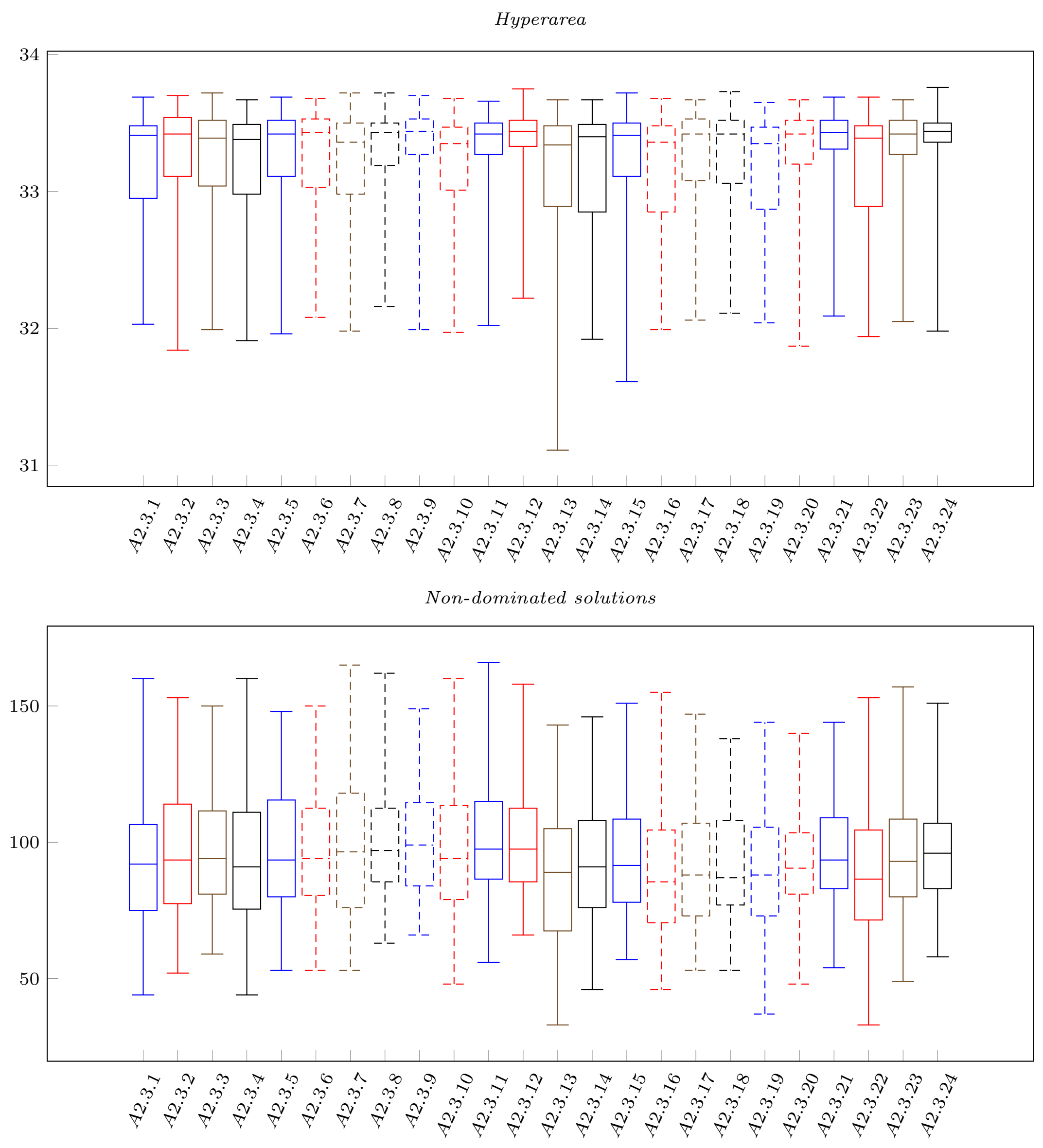


Figure 6.11: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A2.2.1–A2.2.24.*

Table 6.16: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A2.2.1–A2.2.24.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ_F^2	p	F_{ID}	p	
Hyperarea	171.73	0	7.99	0	Reject H_0
Non-dominated solutions	1030.87	0	80.41	0	

6.3.3 Buffer allocation problem: five machines

Tables B.28 and B.29 present the hyperareas and number of non-dominated solutions and their ranks for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.12 provide a graphical summary of the results for hyperparameter combinations A2.3.1–A2.3.24, as presented in Tables B.28 and B.29.

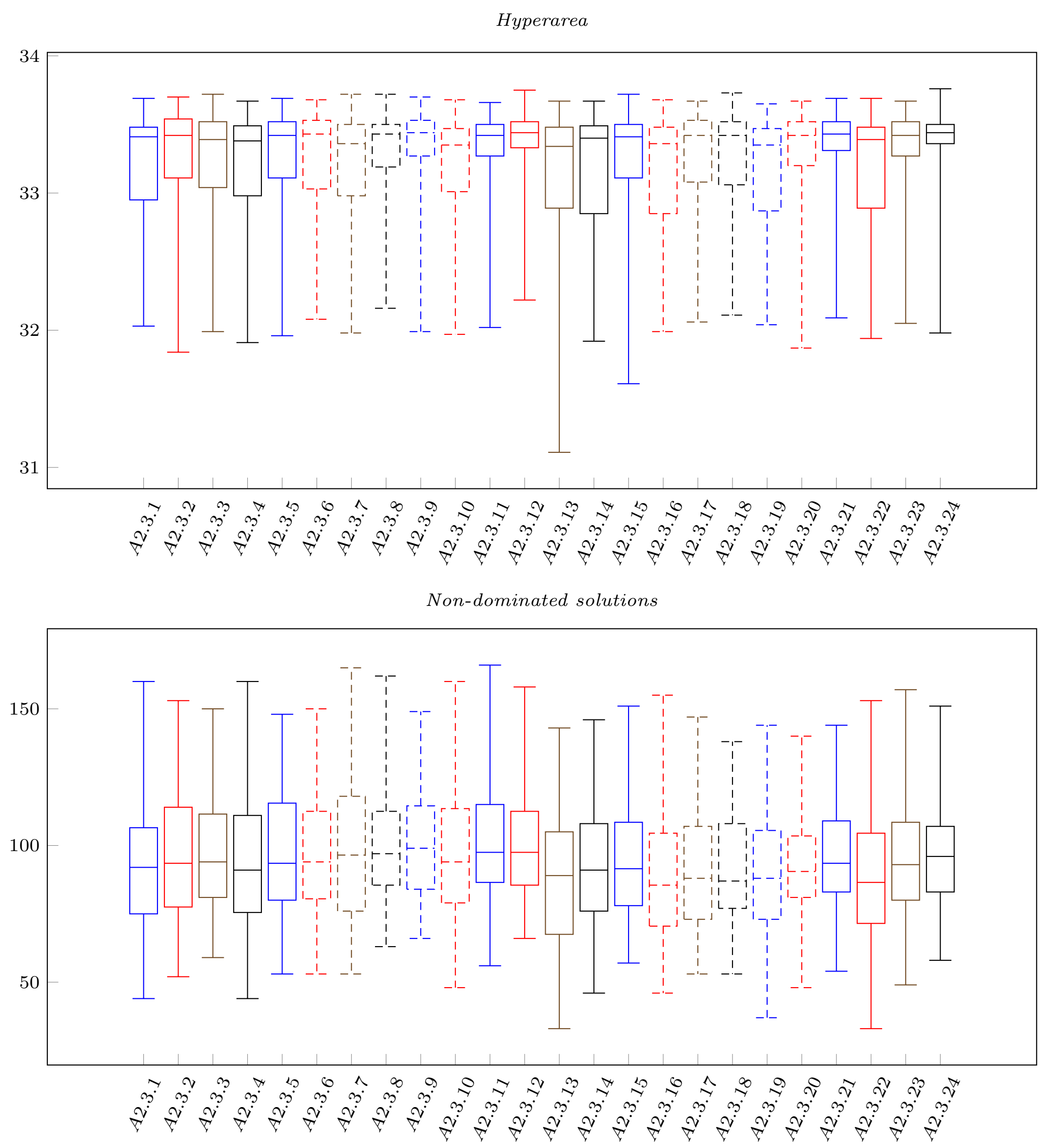


Figure 6.12: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A2.3.1–A2.3.24.*

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are given in Table B.30 and presented in Figures B.20 and B.24 for A2.3.1–A2.3.24.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.17. The Friedman and Iman-Davenport test statistics both indicate significance for both the hyperarea and non-dominated solutions ($\chi^2_F = \{90.47,\ 274.21\} > \chi^2_{23},\ p < \alpha$ and $F_{ID} = \{4.05,\ 13.4\} > F_{23,2277},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

The Nemenyi *post hoc* test is presented in Tables B.31 and B.32 and contains the adjusted p-values of the multiple pairwise comparisons for number of non-dominated solutions. The red

Table 6.17: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A2.3.1–A2.3.24.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	90.47	0	4.05	0	Reject H_0
Non-dominated solutions	274.21	0	13.4	0	

p-values indicate statistical significance, for example there are statistically significant differences between A2.2.1 and A2.2.13, A2.2.15, A2.2.18 and A2.2.19 in terms of hyperarea.

6.3.4 Buffer allocation problem: 10 machines

Tables B.33 and B.34 present the hyperareas and number of non-dominated solutions and their ranks for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.13 provide a graphical summary of the results for hyperparameter combinations A2.4.1–A2.4.24, as presented in Tables B.33 and B.34. The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are given in Table B.35 and presented in Figures B.25 and B.29 for A2.4.1–A2.4.24. The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.17. The Friedman and Iman-Davenport test statistics both indicate significance for both the hyperarea and non-dominated solutions ($\chi^2_F = \{268.26,\ 103.06\} > \chi^2_{23},\ p < \alpha$ and $F_{ID} = \{13.07,\ 4.64\} > F_{23,2277},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

Table 6.18: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A2.4.1–A2.4.24.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	268.26	0	13.07	0	Reject H_0
Non-dominated solutions	103.06	0	4.64	0	

The Nemenyi *post hoc* test is presented in Tables B.36 and B.37 and contains the adjusted p-values of the multiple pairwise comparisons for number of non-dominated solutions. The red p-values indicate statistical significance, for example there are statistically significant differences between A2.4.1 and A2.4.8–A2.4.9, A2.4.11–A2.4.12, A2.4.21 and A2.4.24 in terms of hyperarea.

6.3.5 Non-linear buffer allocation problem: 16 machines

Tables B.38 and B.39 present the hyperareas and number of non-dominated solutions and their ranks for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.14 provide a graphical summary of the results for hyperparameter combinations A2.5.1–A2.5.24, as presented in Tables B.38 and B.39. The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are given in Table B.30 and presented in Figures B.30 and B.34 for A2.5.1–A2.5.24.

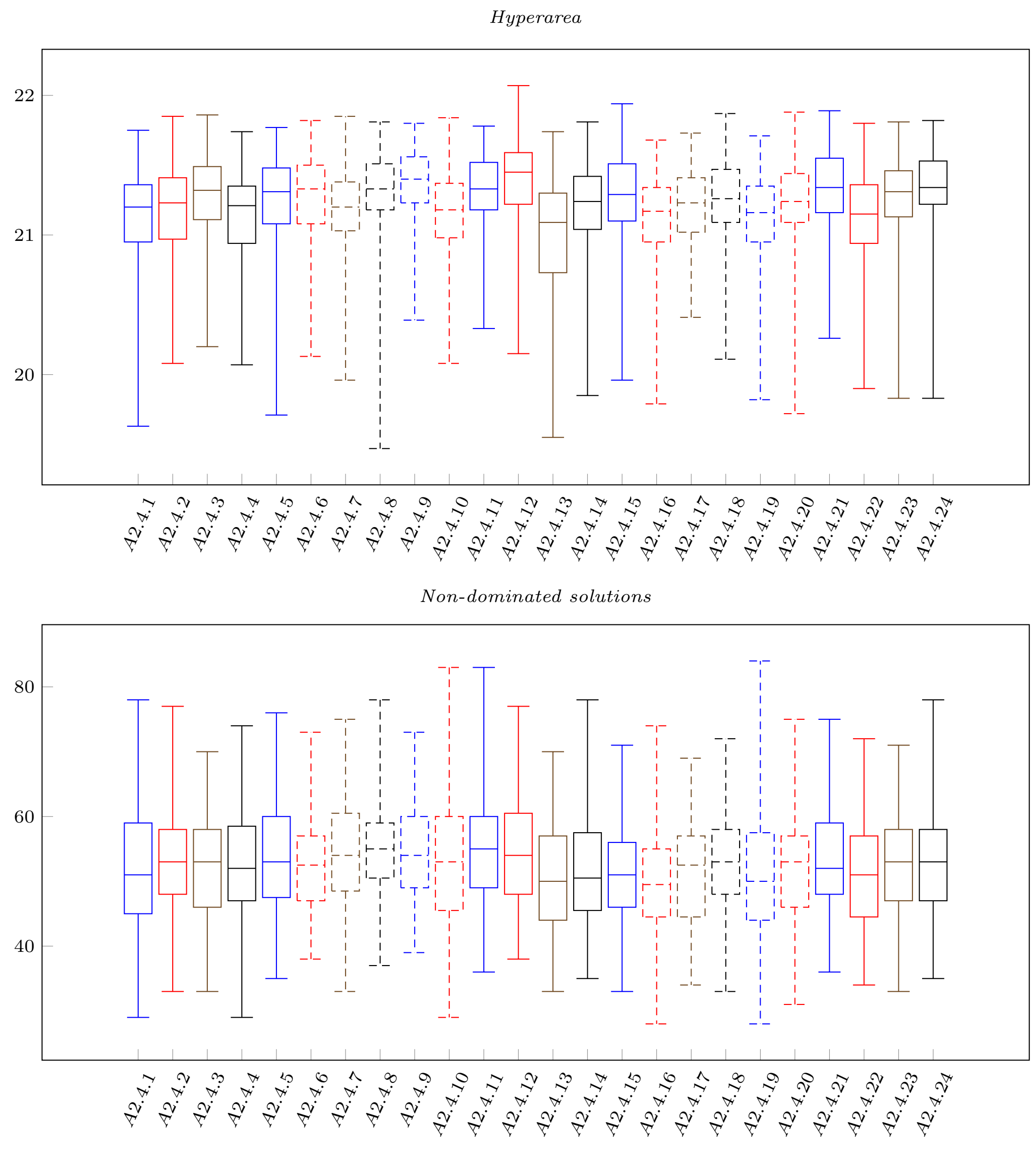
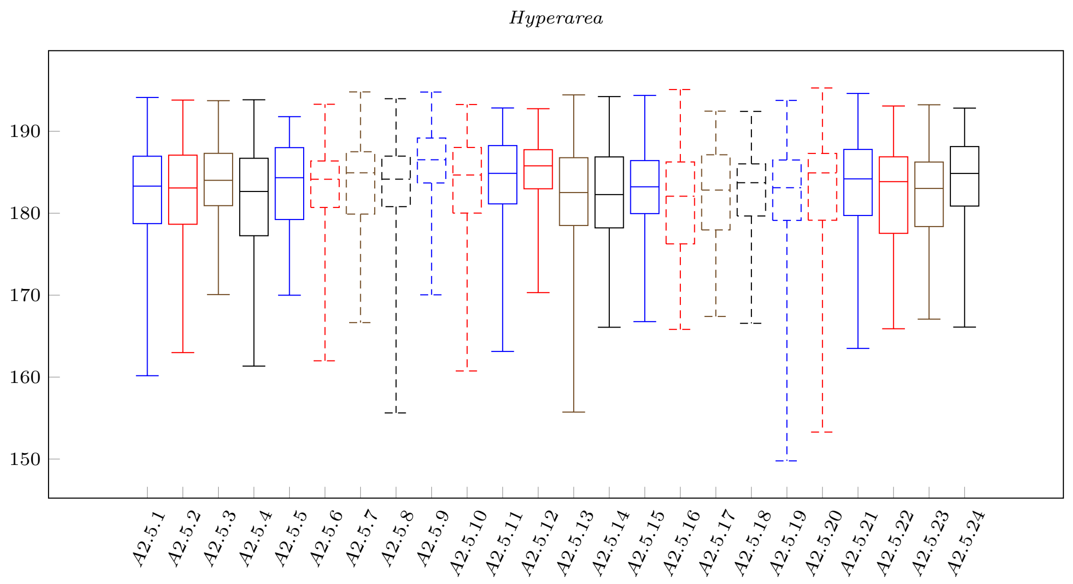


Figure 6.13: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A2.4.1–A2.4.24.*

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.19. The Friedman and Iman-Davenport test statistics both indicate significance for both the hyperarea and non-dominated solutions ($\chi^2_F = \{115.15,\ 73.33\} > \chi^2_{23},\ p < \alpha$ and $F_{ID} = \{5.22,\ 3.26\} > F_{23,2277},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

Table 6.19: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A2.5.1–A2.5.24.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	115.15	0	5.22	0	Reject H_0
Non-dominated solutions	73.33	0	3.26	0	



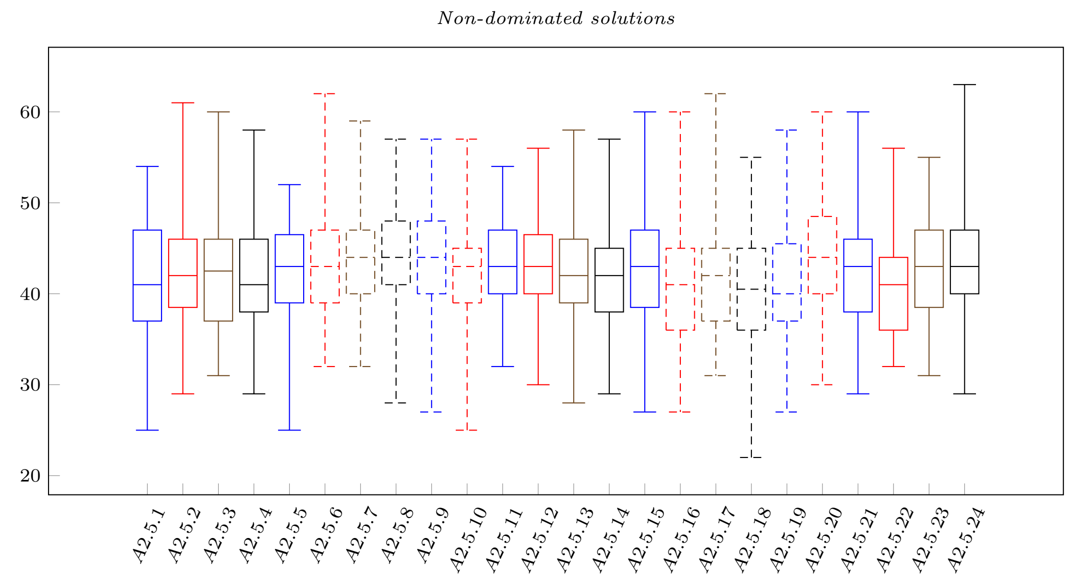


Figure 6.14: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A2.5.1–A2.5.24.*

The Nemenyi *post hoc* test is presented in Tables B.41 and B.42 and contains the adjusted p-values of the multiple pairwise comparisons for number of non-dominated solutions. The red p-values indicate statistical significance, for example there are statistically significant differences between A2.5.1 and A2.5.9 and A2.5.12 in terms of hyperarea.

This concludes the individual (or problem-specific) analysis of the hyperparameter combinations for the NSGA-II. Finally, the single best hyperparameter combination can be determined for the NSGA-II, across all five simulation problems. Again, note that only the hyperareas are considered, since the hyperarea is the better indicator of the quality of an approximation front. Figure 6.15 represents the average hyperareas obtained for each respective hyperparameter combination (collectively referred to as A2.1–A2.24) for the simulation problems. The average number of non-dominated solutions obtained for hyperparameter combinations A2.1–A2.24 is presented in Figure B.35 for each simulation problem.

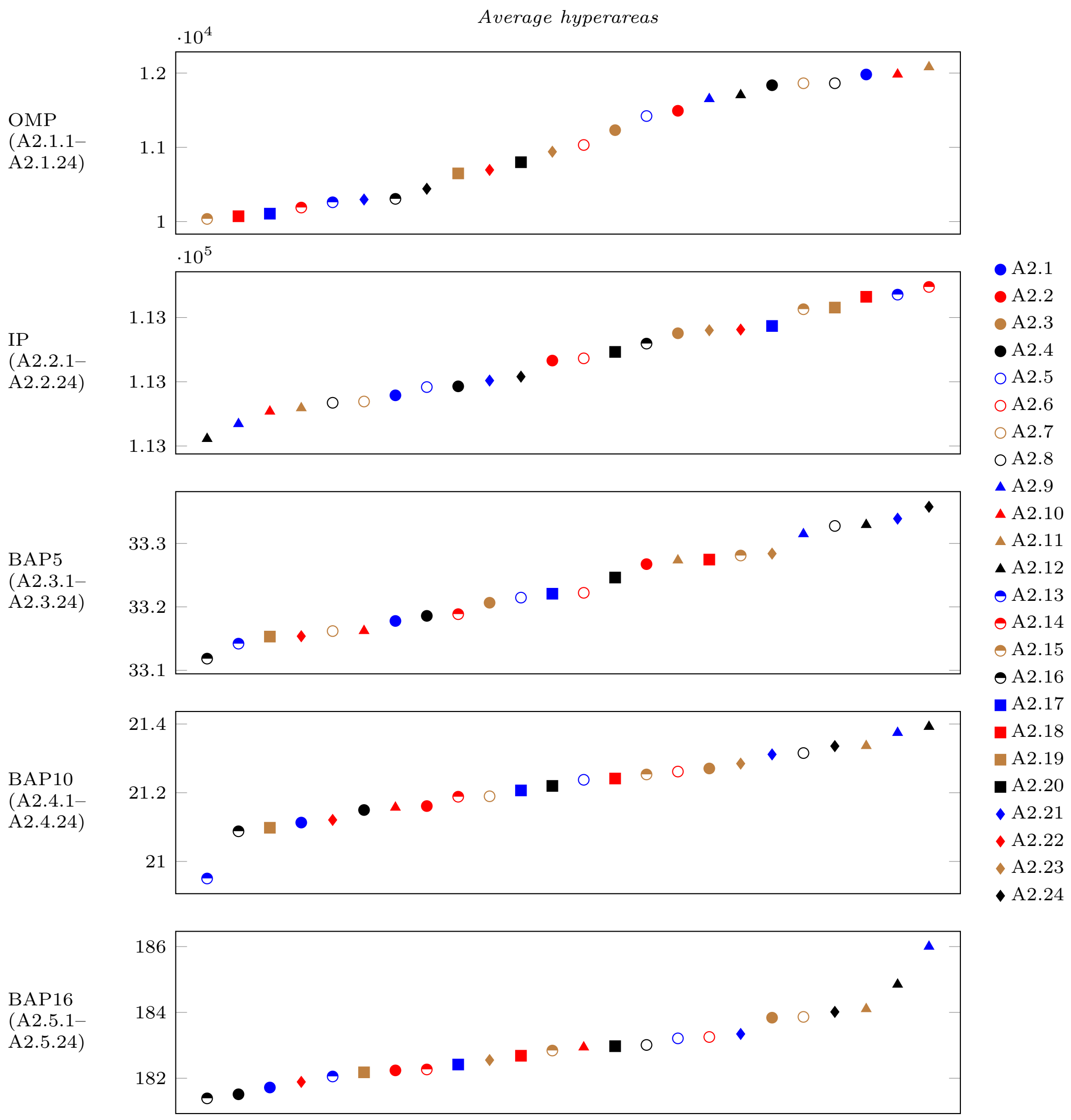


Figure 6.15: *The average hyperareas obtained for each hyperparameter combinations A2.1–A2.24 for the respective simulation optimisation problems.*

Figure 6.16 is used to explain the process followed to determine the single best hyperparameter combination (specifically for BAP16) for the NSGA-II. From the p-values given in Table B.41 and included in the figure, it can be seen that A2.5.12 is statistically significantly different to A2.5.1, A2.5.4, A2.5.13–A2.5.23 which is illustrated by the red rectangle in Figure 6.16. Because A2.5.1, A2.5.4, A2.5.13–A2.5.23 have smaller average hyperareas they are eliminated from consideration.

Accordingly, solutions A2.5.3, A2.5.7, A2.5.9, A2.5.11, A2.5.12 and A2.5.24 are not statistically significantly different and are therefore considered. A similar process was followed for each problem and the outcome is presented in Table 6.20, where it is clear that hyperparameter combination A2.3 is the common denominator across all five simulation problems. Consequently, hyperparameter combination A2.3 (or rank selection, polynomial mutation, $p_c = 0.9$ and $p_m = 0.3$) is employed in BOCEGAH.

Interestingly, the proposed range for p_m is typically between $[0.001,\ 0.01]$, since large probabilities could disrupt the search [93], however the hyperparameter study conducted for the NSGA-II

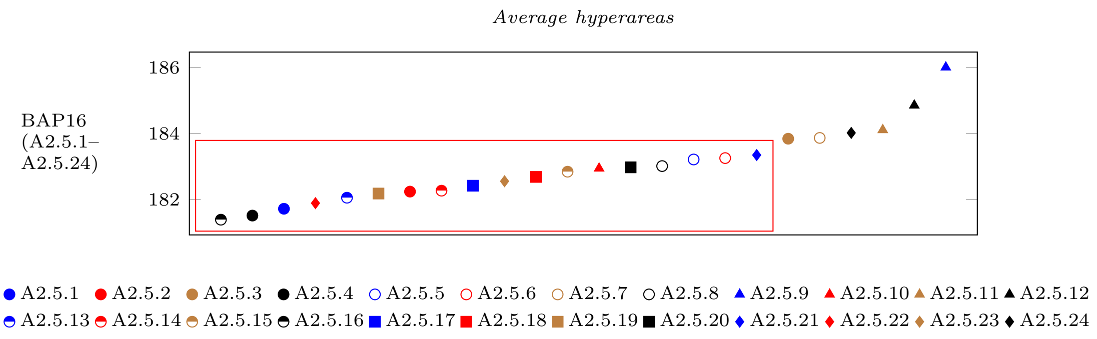


Figure 6.16: *An example illustrating the hyperparameter selection process followed for BAP16 based on the Friedman and Iman-Davenport extension hypothesis test and Nemenyi post hoc test conducted for the hyperareas obtained for hyperparameter combinations A2.5.1–A2.5.24.*

on the five simulation problems concluded that $p_m = 0.3$ is the common best hyperparameter combination.

Table 6.20: *The hyperparameters combinations for the NSGA-II that are considered for implementation in BOCEGAH.*

	OMP	IP	BAP5	BAP10	BAP16
A2.1	✓		✓		
A2.2	✓	✓	✓		
A2.3	✓	✓	✓	✓	✓
A2.4	✓		✓		
A2.5	✓		✓		
A2.6		✓	✓	✓	
A2.7	✓				✓
A2.8	✓		✓	✓	
A2.9	✓		✓	✓	✓
A2.10	✓				
A2.11	✓		✓	✓	✓
A2.12	✓		✓	✓	✓
A2.13		✓			
A2.14		✓	✓		
A2.15		✓	✓	✓	
A2.16		✓			
A2.17		✓	✓		
A2.18		✓	✓		
A2.19		✓			
A2.20		✓	✓		
A2.21			✓	✓	
A2.22		✓			
A2.23		✓	✓	✓	
A2.24			✓	✓	✓

This concludes the empirical study determining the hyperparameter combination to be employed for the NSGA-II in the BOCEGAH. The results can be used a guideline for choosing hyperparameters for the NSGA-II, when deciding on suitable hyperparameter combinations to employ, or is at least a good place to start.

The next section documents the parametric study conducted for the move operators proposed for DBMOSA.

6.4 Determining the DBMOSA algorithm hyperparameters

The aim in this section is to determine which three move operators for the DBMOSA resulted in the best algorithmic performance, across all five simulation problems. Ultimately, the results of the Friedman and Iman-Davenport extension tests are combined and used to decide which three move operators to implement for the DBMOSA to employ in the BOSAH. First, it is necessary to define which parameters are held constant and are summarised in Table 6.21.

Table 6.21: *The hyperparameters that are held constant during the hyperparameter study for DBMOSA.*

Parameter	Value
Initial temperature	10
Maximum iterations	1000
Annealing schedule	Geometric
Cooling parameter	0.75
Heating parameter	1.2
Maximum accepts	10
Maximum attempts	5

It is important to know that for DBMOSA each run was run with a different *initial solution* and *RNS*, an example of which is depicted in Table 6.22 for the IP. For *Run 1* the initial solution is $s = 1$ and $S = 1$ with RNS = 1, *Run 2* the initial solution is $s = 5$ and $S = 5$ with RNS = 2, until *Run 100* with the initial solution is $s = 397$ and $S = 397$ with RNS = 100, applied to all the hyperparameter combinations being evaluated. A similar approach is followed for the other simulation optimisation problems. Moreover, the initial solution for each run remains the same for the different hyperparameter combinations.

Table 6.22: *DBMOSA setup information.*

	Initial solution	RNS
Run 1	{1,1}	1
Run 2	{5,5}	2
⋮	⋮	⋮
Run 100	{397,397}	100

The DBMOSA parameters that are considered for the hyperparameter study are summarised in Table 6.23. A similar naming convention is adopted as before, where A3.1.1 refers to the hyperparameter combination represents *move operator* 1, for the OMP specifically. A search space of four combinations per simulation problem is explored, *i.e.* the number of groups being compared is $k = 4$. Again, each hyperparameter combination (or group) is run for 100 individual (simulation) runs (unless stated otherwise), *i.e.* $n = 100$ observations per group resulting in 100 individual hyperareas and number of non-dominated solutions. Also, each run consists of 1 000 solution evaluations (as mentioned previously) to facilitate fair comparison.

The hypothesis tests conducted are first discussed for the OMP, then the IP, BAP5, BAP10 and finally for BAP16. Thereafter, the results of the respective tests are combined and the three best move operators are chosen.

Table 6.23: *The hyperparameter search space and corresponding naming convention adopted refer to the specific hyperparameter combinations for the specific simulation problem, resulting in the entire hyperparameter search space considered for the DBMOSA.*

Move operator	OMP	IP	BAP5	BAP10	BAP16
1	A3.1.1	A3.2.1	A3.3.1	A3.4.1	A3.5.1
2	A3.1.2	A3.2.2	A3.3.1	A3.4.2	A3.5.2
3	A3.1.3	A3.2.3	A3.3.3	A3.4.3	A3.5.3
4	A3.1.4	A3.2.4	A3.3.1	A3.4.4	A3.5.4

Open mine problem

Recall that for the OMP, only 40 simulation runs per hyperparameter combination were executed, *i.e.* $n = 40$ observations. Accordingly, the critical values for $k = 4$ and $n = 40$ for the Chi-square and F-distributions are $\chi^2_3 = 7.82$ and $F_{3,117} = 2.68$, respectively. Tables B.43 and B.44 present the adjusted hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 39–40 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.17 provide a graphical summary of the results for hyperparameter combinations A3.1.1–A3.1.4, as presented in Tables B.43 and B.44.

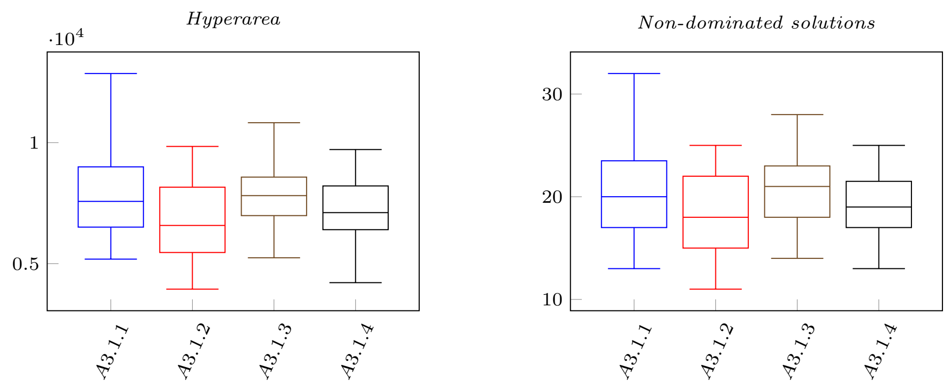


Figure 6.17: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A3.1.1–A3.1.4.*

The DBMOSA was unable to find the true Pareto set for any of the 40 simulation runs for the four different move operators used. However, this does not in-validate the DBMOSA since the hyperparameter search space is very small. The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.3 and depicted in Figure B.36 for A3.1.1–A3.1.4.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.24. The Friedman and Iman-Davenport test statistics for both the adjusted hyperareas (and the number of non-dominated solutions) do not indicate significance ($\chi^2_F = \{7.41,\ 5.29\} < \chi^2_3, p > \alpha$ and $F_{ID} = \{2.57,\ 1.8\} < F_{3,117},\ p > \alpha$). Therefore, the tests fails to reject H_0, which means that there is no statistically significant difference between any of the groups, *i.e.* the move operators are equally desirable. Consequently, no *post hoc* analysis is required.

Table 6.24: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A3.1.1–A3.1.4.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	7.41	0.06	2.57	0.058	Fail to reject H_0
Non-dominated solutions	5.29	0.15	1.8	0.15	

(s, S) Inventory problem

Recall that for the IP, BAP5, BAP10 and BAP16, 100 simulation runs per hyperparameter combination were executed, *i.e.* $n = 100$ observations. Accordingly, the critical values for $k = 4$ and $n = 100$ for the Chi-square and F-distributions are $\chi^2_3 = 7.82$ and $F_{3,297} = 2.64$, respectively. Tables B.46 and B.47 present the hyperareas and number of non-dominated solutions and their ranks for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.18 provide a graphical summary of the results for hyperparameter combinations A3.2.1–A3.2.4, as presented in Tables B.46 and B.47.

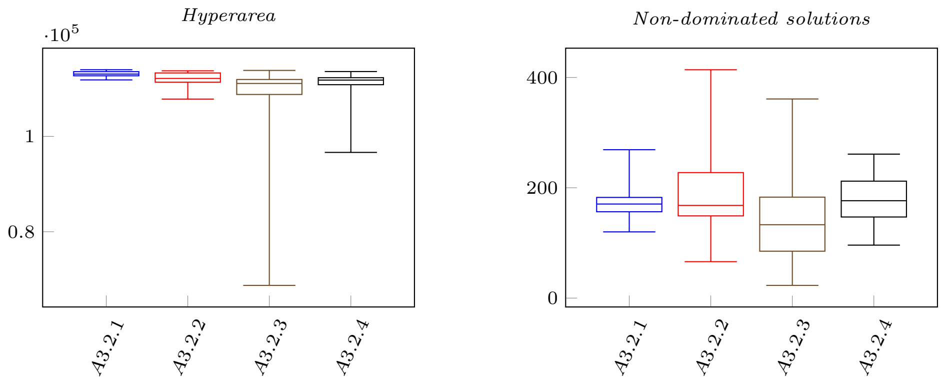


Figure 6.18: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A3.2.1–A3.2.4.*

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.48 and depicted in Figure B.37 for A3.2.1–A3.2.4.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.25. The Friedman and Iman-Davenport test statistics indicate significance for both the hyperareas and number of non-dominated solutions found ($\chi^2_F = \{128.68,\ 28.82\} > \chi^2_3,\ p < \alpha$ and $F_{ID} = \{74.36,\ 10.52\} > F_{3,297},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

The Nemenyi *post hoc* test is presented in Tables 6.26a and 6.26b and contains the adjusted p-values of the multiple pairwise comparisons for both the adjusted hyperareas and number of non-dominated solutions, respectively. The red p-values indicate statistical significance. For example, there are statistically significant differences between A3.2.1 and A3.2.2–A3.2.4 for the hyperareas and number of non-dominated solutions.

Table 6.25: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A3.2.1–A3.2.4.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	128.68	0	74.36	0	Reject H_0
Non-dominated solutions	28.82	0	10.52	0	

Table 6.26: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons for (a) hyperarea and (b) number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A3.2.1–A3.2.4.*

(a) *Hyperarea*

	A3.2.2	A3.2.3	A3.2.4
A3.2.1	0	0	0
A3.2.2		0	0.01
A3.2.3			0.33

(b) *Non-dominated solutions*

	A3.2.2	A3.2.3	A3.2.4
A3.2.1	1	0	1
A3.2.2		0	1
A3.2.3			0

Buffer allocation problem: five machines

Tables B.49 and B.50 present the hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.19 provide a graphical summary of the results for hyperparameter combinations A3.3.1–A3.3.4, as presented in Tables B.49 and B.50.

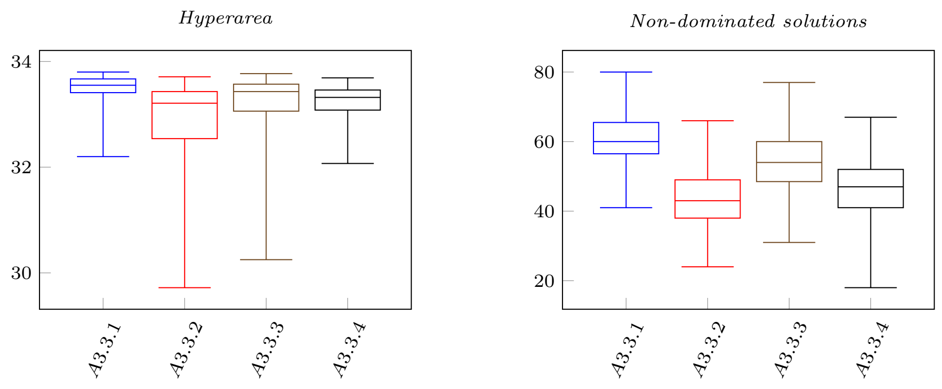


Figure 6.19: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A3.3.1–A3.3.4.*

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.51 and depicted in Figure B.38 for A3.3.1–A3.3.4.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.27. The Friedman and Iman-Davenport test statistics indicate strong significance for both the hyperareas and number of non-dominated solutions found ($\chi^2_F = \{75.16,\ 148.04\} > \chi^2_3,\ p < \alpha$ and $F_{ID} = \{33.09,\ 96.44\} > F_{3,297},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

The Nemenyi *post hoc* test is presented in Tables 6.28a and 6.28b and contains the adjusted

Table 6.27: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A3.3.1–A3.3.4.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ_F^2	p	F_{ID}	p	
Hyperarea	75.16	0	33.09	0	Reject H_0
Non-dominated solutions	148.04	0	96.44	0	

p-values of the multiple pairwise comparisons for both the adjusted hyperareas and number of non-dominated solutions, respectively. The red p-values indicate statistical significance. For example, there are statistically significant differences between A3.3.1 and A3.3.2–A3.3.4 for the hyperareas and number of non-dominated solutions.

Table 6.28: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons for (a) hyperarea and (b) number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A3.3.1–A3.3.4.*

(a) *Hyperarea*

	A3.3.2	A3.3.3	A3.3.4
A3.3.1	0	0	0
A3.3.2		0	0.29
A3.3.3			0.26

(b) *Non-dominated solutions*

	A3.3.2	A3.3.3	A3.3.4
A3.3.1	0	0	0
A3.3.2		0	0.64
A3.3.3			0

Buffer allocation problem: 10 machines

Tables B.52 and B.53 present the hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.20 provide a graphical summary of the results for hyperparameter combinations A3.4.1–A3.4.4, as presented in Tables B.52 and B.53.

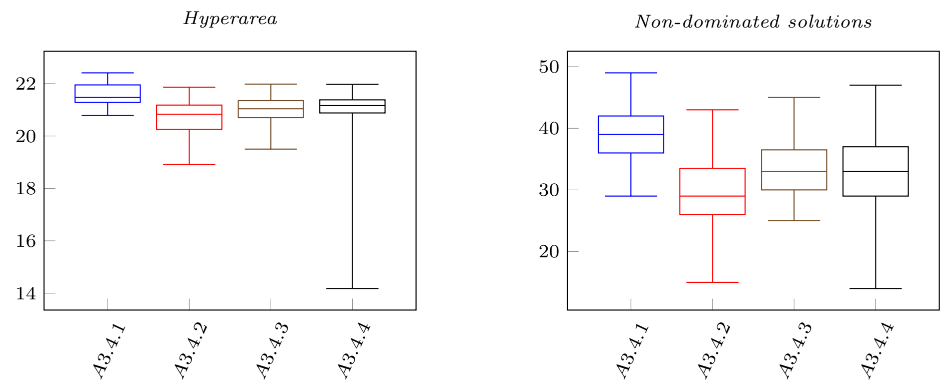


Figure 6.20: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A3.4.1–A3.4.4.*

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.54 and depicted in Figure B.39 for A3.4.1–A3.4.4.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.29. The Friedman and Iman-Davenport test statistics indicate strong significance for both the hyperareas and number of non-dominated solutions found ($\chi^2_F = \{105.28,\ 101.96\} > \chi^2_3,\ p < \alpha$ and $F_{ID} = \{53.52,\ 50.97\} > F_{3,297},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

Table 6.29: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A3.4.1–A3.4.4.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ^2_F	p	F_{ID}	p	
Hyperarea	105.28	0	53.52	0	Reject H_0
Non-dominated solutions	101.96	0	50.97	0	

The Nemenyi *post hoc* test is presented in Tables 6.30a and 6.30b and contains the adjusted p-values of the multiple pairwise comparisons for both the hyperarea and number of non-dominated solutions. For example, there are statistically significant differences between A3.4.1 and A3.4.2–A3.4.4 for both hyperarea and number of non-dominated solutions.

Table 6.30: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons for (a) hyperarea and (b) number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A3.4.1–A3.4.4.*

(a) *Hyperarea*

	A3.4.2	A3.4.3	A3.4.4
A3.4.1	0	0	0
A3.4.2		0.18	0.01
A3.4.3			1

(b) *Non-dominated solutions*

	A3.4.2	A3.4.3	A3.4.4
A3.4.1	0	0	0
A3.4.2		0	0.13
A3.4.3			1

Non-linear buffer allocation problem: 16 machines

Tables B.55 and B.56 present the hyperareas and number of non-dominated solutions and their ranks (respectively) for run 1–10 and 99–100 as well as the sum of the ranks which is used in the Friedman and Iman-Davenport extension hypothesis tests. The boxplots in Figure 6.21 provide a graphical summary of the results for hyperparameter combinations A3.5.1–A3.5.4, as presented in Tables B.55 and B.56.

The runs that correspond to the worst and best approximation fronts obtained (in terms of hyperarea and number of non-dominated solutions) are presented in Table B.57 and depicted in Figure B.40 for A3.5.1–A3.5.4.

The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 6.31. The Friedman and Iman-Davenport test statistics indicate strong statistical significance for both the hyperareas and number of non-dominated solutions found ($\chi^2_F = \{86.96,\ 66.52\} > \chi^2_3,\ p < \alpha$ and $F_{ID} = \{40.41,\ 28.21\} > F_{3,297},\ p < \alpha$). Consequently, H_0 is rejected and the Nemenyi *post hoc* test is conducted.

The Nemenyi *post hoc* tests are presented in Tables 6.32a and 6.32b and contains the adjusted p-values of the multiple pairwise comparisons for both the hyperareas and number of non-dominated solutions, respectively. For example, there was statistically significant differences between A3.5.1 and A3.5.2–A3.5.4 for both hyperarea and number of non-dominated solutions.

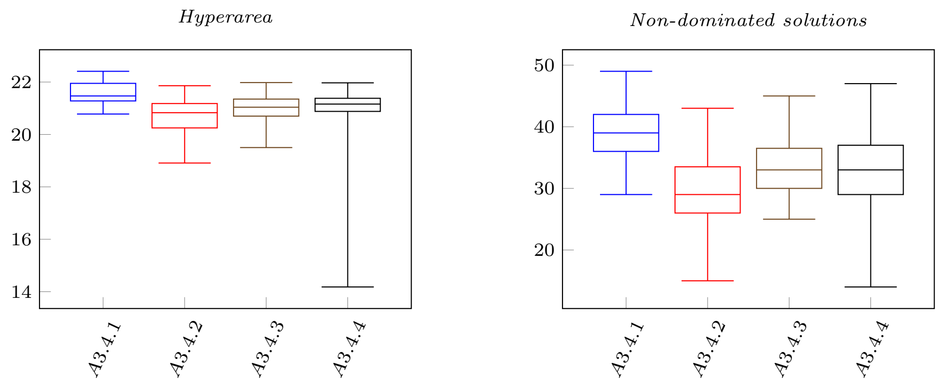


Figure 6.21: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for hyperparameter combinations A3.5.1–A3.5.4.*

Table 6.31: *The Friedman and Iman-Davenport test results based on the ranked hyperareas and non-dominated solutions obtained for each hyperparameter combination A3.5.1–A3.5.4.*

	Friedman test	p-value	Iman-Davenport test	p-value	Outcome
	χ_F^2	p	F_{ID}	p	
Hyperarea	86.96	0	40.41	0	Reject H_0
Non-dominated solutions	66.52	0	28.21	0	

Table 6.32: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons for (a) hyperarea and (b) number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A3.5.1–A3.5.4.*

(a) *Hyperarea*

	A3.5.2	A3.5.3	A3.5.4
A3.5.1	0	0	0
A3.5.2		0.28	0.09
A3.5.3			1

(b) *Non-dominated solutions*

	A3.5.2	A3.5.3	A3.5.4
A3.5.1	0	0	0
A3.5.2		0.77	0.6
A3.5.3			1

This concludes the individual (or problem-specific) analysis of the four move operators proposed for the DBMOSA. Finally, the three best move operators can be determined for the DBMOSA, across all five simulation problems. Recall that only the hyperareas are considered, since the hyperarea is the better indicator of the quality of an approximation front. Figure 6.22 represents the average hyperareas obtained for each respective hyperparameter combination (collectively referred to as A3.1–A3.4) for each simulation problem. The average number of non-dominated solutions obtained for hyperparameter combinations A3.1–A3.4 is presented in Figure B.41 for each simulation problem.

Figure 6.23 is used to explain the process followed to determine the three best move operators (specifically for BAP16) for the DBMOSA. From the p-values given in Table 6.32a and included in the figure, it can be seen that hyperparameter combination A3.5.1 (or move operator 1) is statistically significantly different to A3.5.2–A3.5.4 which is illustrated by the red rectangles in Figure 6.23. Consequently, move operator 1 is considered. Even though move operator 1 is statistically significantly better than move operator 2–4, the hyperheuristic requires three LLHs.

The second and third best move operators need to be determined and because A3.5.2–A3.5.4 are not statistically significantly different ($p > \alpha$), the two move operators with the largest average

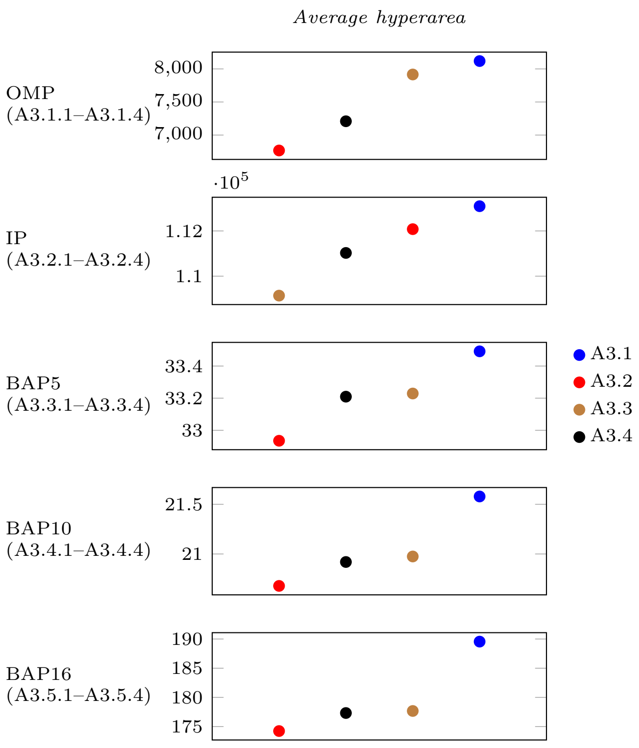
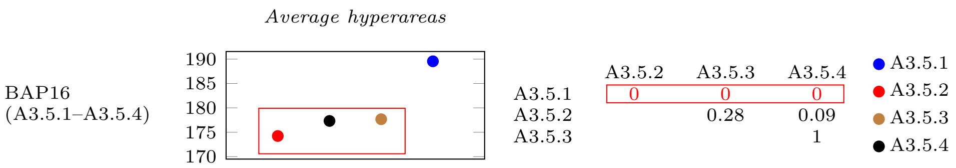


Figure 6.22: *The average hyperareas obtained for hyperparameter combinations A3.1–A3.4 for the respective simulation problems.*

hyperarea are chosen. Accordingly, move operators 1, 3 and 4 are considered for the BAP16. A similar process was followed for each problem and the outcome is presented in Table 6.33, where it is clear that hyperparameter combination A3.1, A3.3 and A3.4 are the common denominators across four of the five simulation problems. Interestingly, move operator 1 (or A3.1) constantly outperforms the other (proposed) move operators.



	A3.5.2	A3.5.3	A3.5.4
A3.5.1	0	0	0
A3.5.2		0.28	0.09
A3.5.3			1

Figure 6.23: *An example illustrating the hyperparameter selection process followed for BAP16 based on the Friedman and Iman-Davenport extension hypothesis test and Nemenyi* post hoc *test conducted for the hyperareas obtained for hyperparameter combinations A3.5.1–A3.5.4.*

Table 6.33: *The move operators that are considered for implementation in BOSAH.*

	A3.1	A3.2	A3.3	A3.4
OMP	✓		✓	✓
IP	✓	✓		✓
BAP5	✓		✓	✓
BAP10	✓		✓	✓
BAP16	✓		✓	✓

This concludes the empirical study determining the three hyperparameter combinations to be employed for the DBMOSA in the BOSAH.

6.5 Conclusion: Chapter 6

The purpose of this chapter was to determine the best hyperparameter combinations, respectively for the MOOCEM, the NSGA-II and the DBMOSA, respectively. Ultimately, the aim was to determine common hyperparameter combinations suitable for the subset of problems considered in the study. The final outcome of all the hyperparameter studies for the BOCEGAH and the BOSAH are summarised in Table 6.34.

Table 6.34: *The final hyperparameter combinations.*

	Hyperparameter combinations
MOOCEM	A1.9
NSGA	A2.3
DBMOSA	A3.1, A3.3, A3.4

The MOOCEM will be employed in the BOCEGAH with hyperparameter combination A1.9, *i.e.* $\alpha = 0.75$ and $p_h = 0.4$. The NSGA-II will also be employed in the BOCEGAH, however with hyperparameter combination A2.3, *i.e.* rank selection, polynomial mutation, $p_c = 0.9$ and $p_m = 0.3$. Finally, the DBMOSA will be implemented in the BOSAH using move operator 1, 3 and 4. The next chapter documents the extensive algorithmic comparative study conducted for the BOCEGAH and the BOSAH.

CHAPTER 7

Algorithm Performance Assessment and Comparison

The previous chapter documented the empirical study that was conducted to determine the best hyperparameter combination for each algorithm, *i.e.* the MOOCEM, the NSGA-II and the DBMOSA.

This chapter serves to fulfil Objective 7, as stated in Chapter 1, evaluating the quality of the BOCEGAH and the BOSAH using the five discrete-event dynamic stochastic simulation optimisation problems, as discussed in Chapter 4. The performances are documented for each hyperheuristic and each simulation problem. Thereafter, the hyperheuristics are compared to their individual LLHs to draw a conclusion on the generalisation capabilities of the proposed hyperheuristics. Finally, the BOCEGAH and the BOSAH are compared to determine whether or not population-based search or single-solution based search performs better, particularly in this study. The chapter closes with a conclusion and summary of its contents.

The Friedman and Iman-Davenport tests are used to compare hyperheuristics with their constituent LLHs. The hypothesis test is first described for the BOCEGAH and its constituent LLHs, *i.e.* the MOOCEM employing hyperparameter combination A1.9 (simply denoted by A1.9) and the NSGA-II employing hyperparameter combination A2.3 (simply denoted by A2.3). Thereafter, a similar process is followed for the BOSAH and the DBMOSA employing move operators A3.1, A3.3 and A3.4. Finally, the tests are used to compare the population-based and the single-solution based search hyperheuristics, the BOCEGAH and the BOSAH.

7.1 BOCEGAH *versus* MOOCEM and NSGA-II

The critical values applicable for the tests conducted in this section, for the χ^2 and F-distributions, are presented in Table 7.1, for $k = 3$ and $\alpha = 0.05$. Again, recall that $n = 40$ for the OMP and $n = 100$ for the IP, BAP5, BAP10 and BAP16. The boxplots in Figure 7.1 provide a graphical summary of the results for A1.9, A2.3 and the BOCEGAH.

Table 7.1: *The critical values for the χ^2 and F-distributions at $k = 3$, $\alpha = 0.05$ but for different n.*

		Friedman	Iman-Davenport
k	n	$\chi^2_{k-1,\alpha}$	$F_{k-1,(k-1)(n-1),\alpha}$
3	40	5.99	3.11
3	100	5.99	3.04

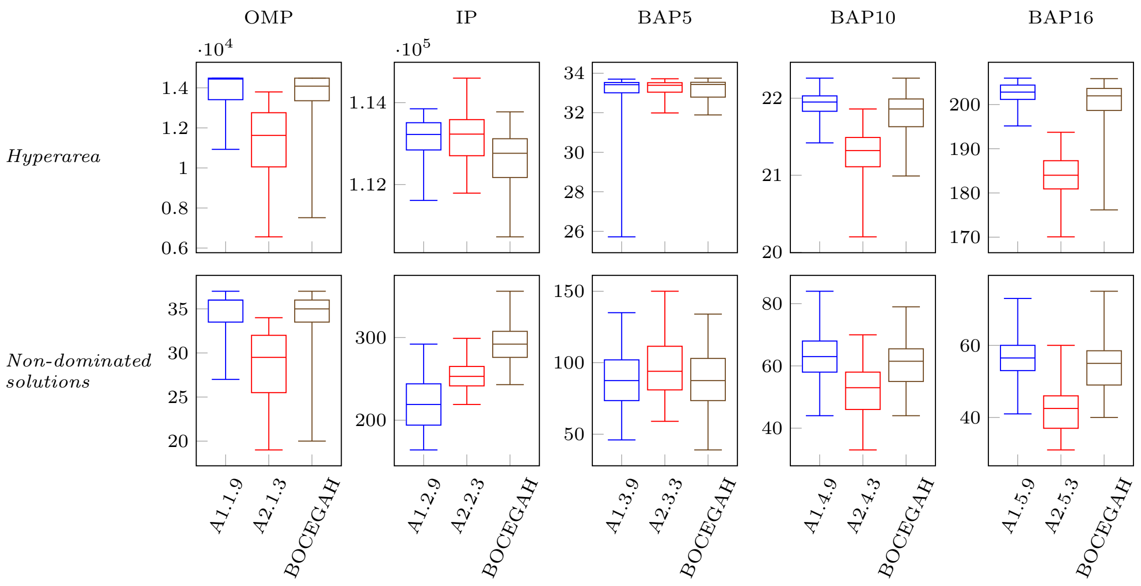


Figure 7.1: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for each simulation problem comparing A1.9, A2.3 and the BOCEGAH.*

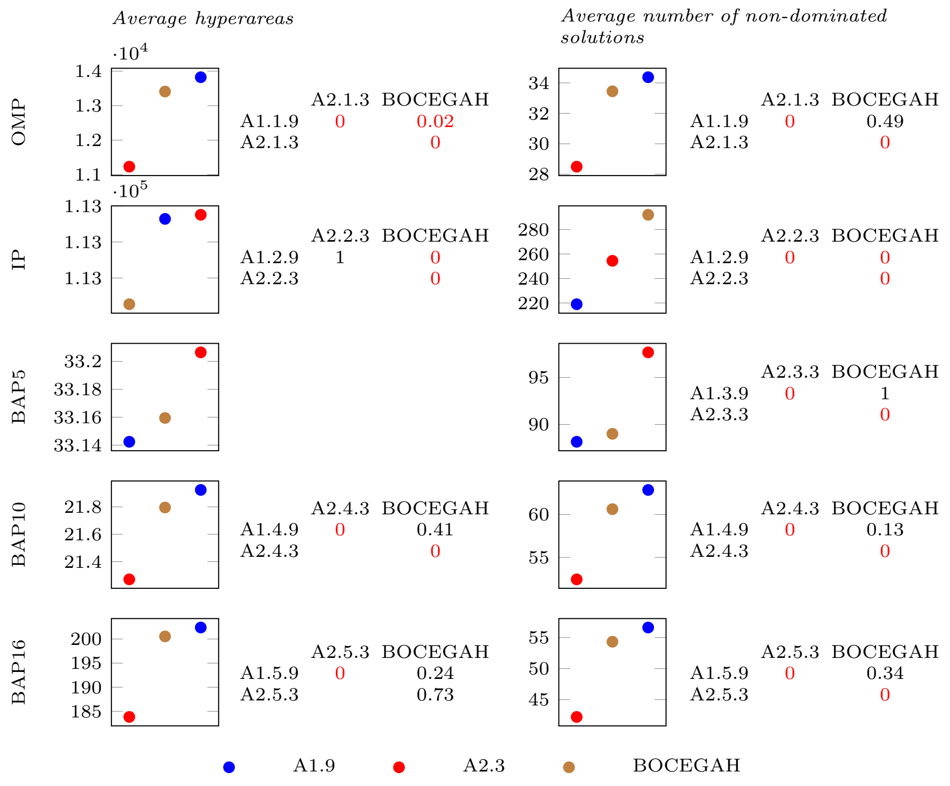
The Friedman and Iman-Davenport extension hypothesis test results are summarised in Table 7.2, for each simulation problem. The outcome of the hypothesis tests, to reject H_0 is indicated by a red checkmark. For example, from Tables 7.1 and 7.2 it can be seen that (for BAP5) the Friedman and Iman-Davenport test statistics indicated statistical significance for the hyperarea ($\chi_F^2 = 0.62 < 5.99,\ p > 0.05$ and $F_{ID} = 0.31 < 3.04,\ p > 0.05$), *i.e.* the tests failed to reject H_0 (denoted by '✗'). However, the tests did not indicate statistical significance for the number of non-dominated solutions ($\chi_F^2 = 31.34 > 5.99,\ p < 0.05$ and $F_{ID} = 18.39 > 3.04,\ p < 0.05$), *i.e.* the outcome is to reject H_0 (denoted by '✓').

Table 7.2: *A summary of the Friedman and Iman-Davenport test results (for each simulation problem) based on the ranked hyperareas and non-dominated solutions obtained by comparing A1.9, A2.3 and the BOCEGAH.*

		Friedman test	p-value	Iman-Davenport test	p-value	Reject H_0
		χ_F^2	p	F_{ID}	p	
OMP	Hyperarea	50.55	0	66.94	0	✓
	Non-dominated solutions	44.04	0	47.76	0	
IP	Hyperarea	42.98	0	27.10	0	✓
	Non-dominated solutions	143.96	0	254.29	0	
BAP5	Hyperarea	0.62	0.73	0.31	0.74	✗
	Non-dominated solutions	31.34	0	18.39	0	✓
BAP10	Hyperarea	137.58	0	218.21	0	✓
	Non-dominated solutions	66.47	0	49.28	0	
BAP16	Hyperarea	155.12	0	342.18	0	✓
	Non-dominated solutions	112.19	0	126.47	0	

Recall that §6.2 is used as a template to succinctly present the results obtained for the hypothesis tests and *post hoc* test. Accordingly, the Nemenyi *post hoc* test is conducted for the OMP, IP, BAP10 and BAP16 for both the hyperarea and number of non-dominated solutions.

However, the *post hoc* test for BAP5 is only conducted for the number of non-dominated solutions. The results are presented (where applicable) by the p-value tables in Figure 7.2, along with the averages of the n hyperareas and number of non-dominated solutions, for each simulation problem comparing A1.9, A2.3 and the BOCEGAH.



	A2.1.3	BOCEGAH
A1.1.9	0	0.02
A2.1.3		0

	A2.2.3	BOCEGAH
A1.2.9	1	0
A2.2.3		0

	A2.4.3	BOCEGAH
A1.4.9	0	0.41
A2.4.3		0

	A2.5.3	BOCEGAH
A1.5.9	0	0.24
A2.5.3		0.73

	A2.1.3	BOCEGAH
A1.1.9	0	0.49
A2.1.3		0

	A2.2.3	BOCEGAH
A1.2.9	0	0
A2.2.3		0

	A2.3.3	BOCEGAH
A1.3.9	0	1
A2.3.3		0

	A2.4.3	BOCEGAH
A1.4.9	0	0.13
A2.4.3		0

	A2.5.3	BOCEGAH
A1.5.9	0	0.34
A2.5.3		0

Figure 7.2: *The p-values obtained by the Nemenyi post hoc test and the averages of the n hyperareas and number of non-dominated solutions for each simulation problem comparing A1.9, A2.3 and the BOCEGAH.*

By combining A1.9 and A3.2 to create the BOCEGAH, the hope was that the ensemble algorithm would enable each LLH to compensate, to some degree, for the weaknesses of other by method of *algorithmic cooperation*. The result, as reported in [142], is a more (or at least as) effective search method that leads to better (or similar) overall performances when compared with its individual LLHs [36]. From Table 7.3 it can be seen that the performances of A1.9 for the OMP and IP are statistically significantly better to that of the BOCEGAH (in terms of hyperarea).

Table 7.3: *A summary indicating the statistically significant differences in terms of hyperareas and number of non-dominated solutions for A1.9, A2.3 and the BOCEGAH for each simulation problem.*

	Hyperarea					Non-dominated solutions				
Problem	OMP	IP	BAP5	BAP10	BAP16	OMP	IP	BAP5	BAP10	BAP16
A1.9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A2.3		✓	✓							
BOCEGAH			✓	✓	✓	✓	✓	✓	✓	✓

A1.9 (or MOOCEM) and the BOCEGAH outperform A3.2 (or NSGA-II) in terms of hyperarea for the OMP, BAP10 and BAP16, and in terms of the number of non-dominated solutions they outperform NSGA-II for all five simulation problems. By combining MOOCEM and NSGA-II to create the BOCEGAH, the hope was that the ensemble algorithm would enable each LLH to compensate, to some degree, for the weaknesses of other by method of algorithmic cooperation.

However, the hypothesis test and post hoc analysis concluded that (next) MOOCEM with hyperparameter combination A1.9 outperforms the BOCEGAH for the OMP and IP. Therefore, the statement that claims hyperheuristics perform better (or at least similar) to its LLHs cannot be claimed for the population-based hyperheuristic the BOCEGAH. This concludes the performance comparisons between BOCEGAH and its LLHs, and concluded that overall the MOOCEM performed better.

7.2 BOSAH *versus* DBMOSAs move operators

The critical values applicable for the tests conducted in this section, for the χ^2 and F-distributions, are presented in Table 7.4, for $k = 4$ and $\alpha = 0.05$. The boxplots in Figure 7.3 provide a graphical summary of the results for A3.1, A3.3–A3.4 and the BOSAH.

Table 7.4: *The critical values for the χ^2 and F-distributions at $k = 4$, $\alpha = 0.05$ but for different n.*

		Friedman	Iman-Davenport
k	n	$\chi^2_{k-1,\alpha}$	$F_{k-1,(k-1)(n-1),\alpha}$
4	40	7.82	2.68
4	100	7.82	2.64

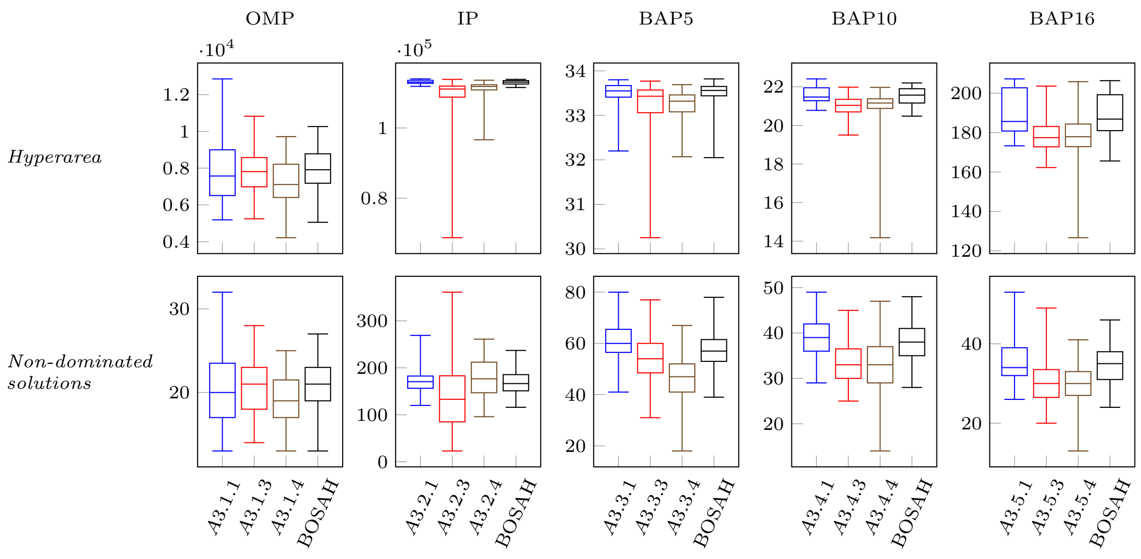


Figure 7.3: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions for each simulation problem comparing A3.1, A3.3–A3.4 and the BOSAH.*

The Friedman and Iman-Davenport hypothesis test results are summarised in Table 7.5, for each simulation problem. From Tables 7.4 and 7.5 it can be seen that (for the OMP) the Friedman and Iman-Davenport test statistics do not indicate statistical significance for either the hyperarea or number of non-dominated solutions ($\{\chi^2_F = \{7.35,\ 6.01\} < 7.82,\ p > 0.05\}$ and $\{F_{ID} = \{2.55,\ 2.06\} < 2.68,\ p > 0.05\}$), *i.e.* the tests failed to reject H_0 (✗). However, for the IP, BAP5, BAP10 and BAP16 the tests indicate strong statistical significance, *i.e.* ✓.

Table 7.5: *A summary of the Friedman and Iman-Davenport test results (for each simulation problem) based on the ranked hyperareas and non-dominated solutions obtained comparing A3.1, A3.3–A3.4 and the BOSAH.*

		Friedman test	p-value	Iman-Davenport test	p-value	Reject H_0
		χ^2_F	p	F_{ID}	p	
OMP	Hyperarea	7.35	0.06	2.55	0.06	×
	Non-dominated solutions	6.01	0.11	2.06	0.11	
IP	Hyperarea	174.84	0	138.3	0	✓
	Non-dominated solutions	25.78	0	9.31	0	
BAP5	Hyperarea	83.86	0	38.41	0	✓
	Non-dominated solutions	128.09	0	73.76	0	
BAP10	Hyperarea	84.23	0	38.65	0	✓
	Non-dominated solutions	82.97	0	37.85	0	
BAP16	Hyperarea	96.14	0	46.69	0	✓
	Non-dominated solutions	70.67	0	30.51	0	

Accordingly, the Nemenyi *post hoc* test is conducted for the IP, BAP5, BAP10 and BAP16 for both the hyperarea and number of non-dominated solutions. The results are presented (where applicable) by the p-value tables in Figure 7.4, along with the averages of the n hyperareas and number of non-dominated solutions, for each simulation problem comparing A3.1, A3.3–A3.4 and the BOSAH.

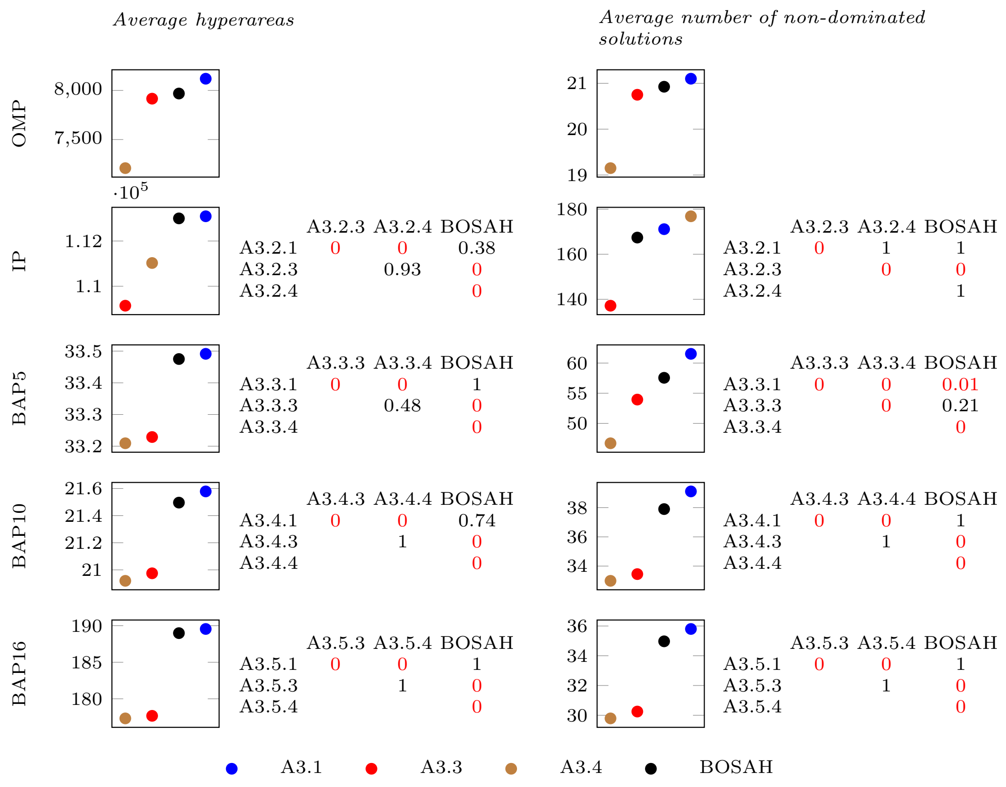
From Table 7.6 it can be seen that the performances of A3.1 and BOSAH are not statistically significantly different, however they are statistically significantly different to A3.3 and A3.4 (for the IP, BAP5, BAP10 and BAP16). Therefore, it can be said that A3.1 and BOSAH outperform A3.3 and A3.4 (in terms of hyperarea).

Interestingly, A3.1 outperforms BOSAH in terms of the number of non-dominated solution, however, not for the hyperarea. For this reason, only the hyperarea is considered, since it provides a better indication of the quality of an approximation front. Consequently, it can be said that BOSAH performs similar to A3.1 and outperforms A3.3 and A3.4 on four of the five simulation problems.

Again, by combining DBMOSA with three move operators to create the BOSAH, the hope was that the ensemble algorithm would enable each LLH to compensate, to some degree, for the weaknesses of the other and consequently, the statement that claims hyperheuristics perform better or at least similar to its LLHs holds true for the single-solution based hyperheuristic the BOSAH. The result is a more (or at least as) effective search method that leads to better (or similar) overall performances when compared with its individual LLHs.

Table 7.6: *A summary indicating the statistically significant differences in terms of hyperareas and number of non-dominated solutions for A3.1, A3.3–A3.4 and the BOSAH for each simulation problem.*

	Hyperarea					Non-dominated solutions				
Problem	OMP	IP	BAP5	BAP10	BAP16	OMP	IP	BAP5	BAP10	BAP16
A3.1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
A3.3	✓					✓				
A3.4	✓					✓	✓			
BOSAH	✓	✓	✓	✓	✓	✓	✓		✓	✓



Average hyperareas:

	A3.2.3	A3.2.4	BOSAH
A3.2.1	0	0	0.38
A3.2.3		0.93	0
A3.2.4			0

	A3.3.3	A3.3.4	BOSAH
A3.3.1	0	0	1
A3.3.3		0.48	0
A3.3.4			0

	A3.4.3	A3.4.4	BOSAH
A3.4.1	0	0	0.74
A3.4.3		1	0
A3.4.4			0

	A3.5.3	A3.5.4	BOSAH
A3.5.1	0	0	1
A3.5.3		1	0
A3.5.4			0

Average number of non-dominated solutions:

	A3.2.3	A3.2.4	BOSAH
A3.2.1	0	1	1
A3.2.3		0	0
A3.2.4			1

	A3.3.3	A3.3.4	BOSAH
A3.3.1	0	0	0.01
A3.3.3		0	0.21
A3.3.4			0

	A3.4.3	A3.4.4	BOSAH
A3.4.1	0	0	1
A3.4.3		1	0
A3.4.4			0

	A3.5.3	A3.5.4	BOSAH
A3.5.1	0	0	1
A3.5.3		1	0
A3.5.4			0

Figure 7.4: *The p-values obtained by the Nemenyi post hoc test and the averages of the n hyperareas and number of non-dominated solutions for each simulation problem comparing A3.1, A3.3–A3.4 and the BOSAH.*

This concludes the performance comparisons between BOSAH and its LLHs, and concluded that overall BOSAH performed similar to or better than its LLHs, but never worse.

7.3 BOCEGAH *versus* BOSAH

Finally, the population-based and single-solution based hyperheuristics BOCEGAH and BOSAH are compared. The critical values applicable for the tests conducted in this section, for the χ^2 and F-distributions, are presented in Table 7.7, for $k = 2$ and $\alpha = 0.05$.

Table 7.7: *The critical values for the χ^2 and F-distributions at $k = 2$, $\alpha = 0.05$ but for different n.*

		Friedman	Iman-Davenport
k	n	$\chi^2_{k-1,\alpha}$	$F_{k-1,(k-1)(n-1),\alpha}$
2	40	3.84	4.09
2	100	3.84	3.94

The boxplots in Figure 7.5 provide a graphical summary of the results for the BOSAH and the BOCEGAH and provides some insight in terms of which hyperheuristic performed better. To further demonstrate the differences between BOSAH and BOCEGAH, the worst and best approximation sets in terms of hyperarea are presented in Figure E.1. The approximation fronts correspond to the approximation sets obtained for the runs listed in Tables E.2 and E.1. In addition, the worst and best approximation sets in terms of the number of non-dominated solutions, also specified in Tables E.2 and E.1, are presented in Figure E.2.

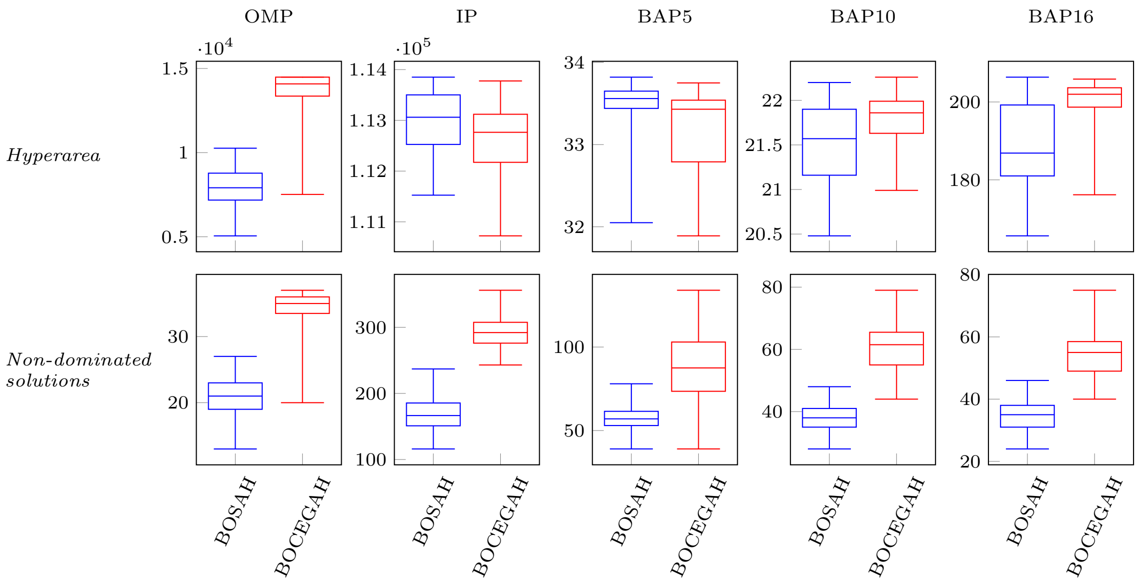


Figure 7.5: *Box plots illustrating the spread of hyperareas and number of non-dominated solutions found for the BOCEGAH and the BOSAH.*

The Friedman and Iman-Davenport hypothesis test results are summarised in Table 7.5, for each simulation problem. Note that the Iman-Davenport test statistic could not be determined, for the number of non-dominated solutions, for the IP, BAP10 and BAP16. This is because $\chi_F^2 = 100$ and $n = 100$ and accordingly (6.2) results in a division of zero. From Tables 7.7 and 7.8 it can be seen that the Friedman and Iman-Davenport test statistics indicate significance for both hyperarea and number of non-dominated solutions for all five simulation problems, *i.e.* H_0 is rejected (✓).

Since only two groups are being compared (*i.e.* $k = 2$), it is deemed unnecessary to perform the Nemenyi *post hoc* tests. The averages of the n hyperareas and number of non-dominated solutions, for each simulation problem comparing the BOSAH and the BOCEGAH are presented in Figure 7.6.

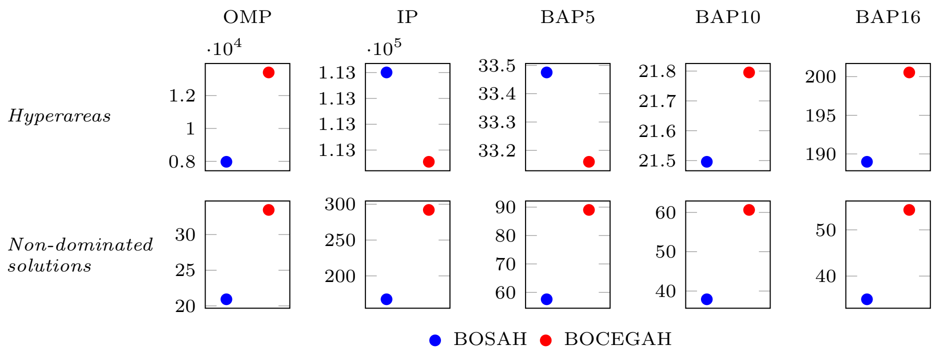


Figure 7.6: *The averages of the n hyperareas and number of non-dominated solutions for the BOSAH and the BOCEGAH.*

From table 7.9 it can be concluded that the BOSAH outperforms the BOCEGAH on the IP and BAP5 simulation problems, whereas, BOCEGAH outperforms the BOSAH on the OMP, BAP10 and BAP16. In addition, the average of the n hyperareas is presented, along with the

Table 7.8: *A summary of the Friedman and Iman-Davenport test results (for each simulation problem) based on the ranked hyperareas and non-dominated solutions obtained comparing the BOSAH and the BOCEGAH.*

		Friedman test	p-value	Iman-Davenport test	p-value	Reject H_0
		χ_F^2	p	F_{ID}	p	
OMP	Hyperarea	36.1	0	361	0	✓
	Non-dominated solutions	36.1	0	361	0	
IP	Hyperarea	9	0	9.79	0	✓
	Non-dominated solutions	100	0	–	–	
BAP5	Hyperarea	14.44	0	16.71	0	✓
	Non-dominated solutions	73.96	0	281.18	0	
BAP10	Hyperarea	11.56	0	12.94	0	✓
	Non-dominated solutions	100	0	–	–	
BAP16	Hyperarea	36	0	55.69	0	✓
	Non-dominated solutions	100	0	–	–	

so-called hyperarea ratio.

Even though their performances are statistically significantly different, their hyperarea ratios are relatively similar (next) except for the OMP. It seems the BOSAH struggled to find the true Pareto front which could be due to the fact that the Pareto set was discontinuous, and that single-solution based search is considered exploitation oriented whereas population-based search is considered exploration oriented as they are powerful in the approximation of the whole Pareto set.

Table 7.9: *A comparison between the BOSAH and the BOCEGAH.*

Problem	OMP	IP	BAP5	BAP10	BAP16
BOSAH		✓	✓		
BOCEGAH	✓			✓	✓
Hyperarea (BOSAH)	7968.49	113 000.49	33.48	21.5	188.98
Hyperarea (BOCEGAH)	13 408.65	112 654.36	33.16	21.78	200.54
True hyperarea	14 485.04	114 442.61	33.84	22.66	214.33
Hyperarea ratio (BOSAH)	0.5501	0.9874	0.9894	0.9488	0.8817
Hyperarea ratio (BOCEGAH)	0.9257	0.9844	0.9799	0.9612	0.9357

7.4 Conclusion: Chapter 7

This chapter opened with an extensive algorithmic comparative study focussed on comparing the algorithmic performance of the BOCEGAH with its constituent LLHs (*i.e.* the MOOCEM and the NSGA-II) and the BOSAH with its constituent LLHs (*i.e.* the A3.1, A3.3 and A3.4 for DBMOSA). Later, the BOCEGAH and the BOSAH were compared. Upon considering the evaluation results for the BOCEGAH and the BOSAH, in §7.1 and §7.2, it can be concluded that both hyperheuristics failed to exhibit superior performance and did not indicate favourable performance improvements relative to *all* the individual applications of the LLHs.

CHAPTER 8

Conclusion

This final chapter comprises three section. In §8.1 a chapter-by-chapter overview of the research documented in this study is provided. This is followed in §8.2 by an appraisal of the contributions made in this study and §8.3, which contains suggestions for seven avenues of further investigation as possible follow-up work, to fulfil Objective 7 as stated in Chapter 1.

8.1 Thesis summary

This study opened with the introductory, Chapter 1, in §1.1 with a general background in which the need for a more generalise purpose optimisation tool, by means of hyperheuristic approaches (*i.e.* BOCEGAH and BOSAH), especially in the context of simulation optimisation, was motivated. Furthermore, the potential utility of an ANN as metamodel was established, motivating the pilot study that was conducted to determine its feasibility. Thereafter, the problem considered in the study was formally described in §1.2. Following this was a delimitation of the study's scope to bi-objective discrete-event, dynamic and stochastic simulation optimisation problems and FNNs in a supervised learning paradigm, with specific focus on regression problems, presented in §1.3. In addition, the optimisation approaches considered for inclusion in the proposed hyperheuristic framework were limited to the MOOCEM, the NSGA-II and the DBMOSA. Accordingly, algorithmic comparisons were limited to the BOCEGAH (and its constituent sub-algorithms) and the BOSAH (and its constituent algorithms), for the five bi-objective discrete-event, dynamic and stochastic simulation optimisation problems. Next, the objectives pursued in the study were outlined in §1.4, followed by the proposed research methodology in §1.5. The first chapter closed in §1.6 with a detailed description of how the remainder of the study was organised into the respective chapters and corresponding appendices.

Apart from the above-mentioned introductory chapter, this study comprised a further seven chapters, a bibliography, and five appendices. Chapter 2 was a literature review in fulfilment of Objective 1 in §1.4, and consisted of six sections. The first section, §2.1, emphasised the need for simulation as a research field and, accordingly, the need for better optimisation approaches to solve complex (combinatorial) simulation optimisation problems. Thereafter, in §2.2, the important notions pertaining to MOO was discussed, which included Pareto dominance, Pareto rank, the quality of MOO algorithms, the notion of archiving and finally MOO performance assessment. This was followed in §2.3 by a discussion of the criteria used to classify metaheuristics and listed the main classes of metaheuristics. Next, in §2.4, the motivations supporting the research on hyperheuristics were discussed and a classification of the hyperheuristic approaches were presented. §2.5, discussed the potential utility of an ANN as metamodel and contained a review of the pertinent literature related to ANNs. The review included the fundamentals of ANNs and the terminology found in the ANN literature, where FNNs were selected to form the basis of discussion. Furthermore, prominent training algorithms and activation functions were discussed.

In Chapter 3, a pilot study was presented to determine the feasibility of an ANN as metamodel in fulfilment of Objective 2 in §1.4, and consisted of five sections. The first section, §3.1, briefly discussed the regression models used to compare to the ANN, which included MLR, PLR, SVR, DTR, RFR and XGBoost. This was followed in §3.2 by the performance measures used to assess the regression models as to facilitate the comparison. Next, in §3.3, Bayesian optimisation was discussed as the proposed method for finding the best hyperparameters for the FNN with one hidden layer. Thereafter, in §3.4, the training, validation and test results for the respective regression models were documented including the results obtained for the FNN employing the hyperparameters found during Bayesian optimisation.

Chapter 4 discussed the simulation models studied in fulfilment of Objective 3 in §1.4, and consisted of three sections. The first section, §4.1, introduced the statistical prerequisites required for stochastic output analysis. Thereafter, §4.2 presented the respective simulation models, discussing each model in terms of their inputs, outputs as well as the sensible upper bounds that were determined by method of statistical inference. In addition, the hypothesis tests conducted to determine whether or not 100 observations per solution were sufficient was documented for each simulation model and concluded in §4.3.

Chapter 5 documented both hyperheuristics and their constituent sub-algorithm in fulfilment of Objective 4 and 5 in §1.4, and consisted of seven sections. The first section, §5.1, introduced some of the main concepts of P- and S-metaheuristics, which included initial solutions, solution representation and stopping conditions. Following that, in §5.2–§5.4, was a discussion for the MOOCEM, the NSGA-II and the DBMOSA in the context of simulation optimisation and its integration in Tecnomatix. Thereafter, §5.5 and §5.6 discussed the hyperheuristics proposed in the study, also in the context of simulation optimisation and its integration in Tecnomatix.

Chapter 6 documented both hyperheuristics and their constituent sub-algorithms in fulfilment of Objective 6 in §1.4, and consisted of five sections. The first section, §6.1, introduced the non-parametric inferential statistical testing employed in this chapter, *i.e.* the Friedman and Iman-Davenport extension with the Nemenyi *post hoc* procedure. These statistical procedures facilitated the algorithmic parameter evaluation and the algorithmic performance comparisons conducted in Chapter 7. These statistical procedures were documented in §6.2–§6.4 for the MOOCEM, the NSGA-II and the DBMOSA, and concluded the common hyperparameter combination for each algorithm in §6.5.

Chapter 7 documented algorithmic performance comparisons in fulfilment of Objective 7 in §1.4, and consisted of four sections. The first section, §7.1, documented the statistical procedure that facilitated the comparison between the BOCEGAH, the MOOCEM and the NSGA-II. Thereafter, §7.2, documented the statistical procedure that facilitated the comparison between the BOSAH and the DBMOSA employing move operators A3.1, A3.3 and A3.4. Finally, §7.3, documented the statistical procedure that facilitated the comparison between the BOCEGAH and the BOSAH and concluded in §7.4.

8.2 Appraisal of thesis contributions

The main contribution of this study is five-fold. This section contains a documentation and appraisal of these contributions.

Contribution 1 *The establishment of a formal pilot study, providing an in-depth investigation* *into the workings of an ANN.*

The novel pilot study proposed some avenues for further experimentation, some of which are listed in §3.5. The study provided valuable work pertaining to the complex interactions within an ANN and from the Python scripts included in Listing A, it is possible to further the study and ascertain more definitive results pertaining to the feasibility of an ANN as metamodel.

Contribution 2 *The addition of five new optimisation tools to solve bi-objective simulation optimisation problems within Tecnomatix.*

The addition of two population-based metaheuristics, the MOOCEM and the NSGA-II and the corresponding hyperheuristic tool BOCEGAH, which combined MOOCEM and the NSGA-II. The addition of a single-solution based metaheuristic, the DBMOSA employing four (proposed) move operators and the hyperheuristic tool BOSAH which combined three of the move operators, thereby enhancing the bi-objective simulation optimisation capabilities of Tecnomatix. This may be of particular interest to industries using Tecnomatix.

Contribution 3 *The modification and furtherance of the AMALGAM hyperheuristic within the context of single-solution based search.*

The application of AMALGAM towards single solutions-based search has been attempts in the literature, see [128], however a new heuristic selection mechanism was proposed and outlined in §5.6.

Contribution 4 *An evaluation of the algorithmic parameters for the MOOCEM and the NSGA-II.*

The novelty of the hyperparameter search space explored is the valuable insights that can be inferred from the hyperparameter combinations when considering the five discrete-event simulation optimisation problems with varying sizes, approximation front forms and complexities. Moreover, the results can be used as a guideline for choosing hyperparameters for the MOOCEM and the NSGA-II, refer to §6.2 and §6.3.

Contribution 5 *The realisation of the inconsistency with the hyperarea calculation for discontinuous approximation fronts.*

A linear adjustment was proposed for the hyperarea calculation, taking into account the discontinuity of the approximation front by considering the number of non-dominated solutions in the front *versus* that of the true front, as documented in §6.2.1.

8.3 Suggestions for future work

Suggestions for further investigation is documented in this section. In each case, the suggestion is stated formally and then elaborated upon.

Suggestion 1 *Consider the evaluation of the optimisation approaches for different problem contexts.*

The problem contexts considered in this study were all (coincidentally) *min–max* problems and therefore *min–min* and *max–max* problems should also be considered.

Suggestion 2 *Design more move operators for single-solution based search for discrete vector representation.*

The parameter evaluations of DBMOSA's move operators concluded that move operator 1 was statistically significantly better than the other proposed move operators, as documented in §6.4.

1. Creating perturbations of x from the Laplacian distribution. The Laplacian distribution is suggested due to its tails that decay relatively slow, ensuring that regions distant from the current solutions are explored [232].
2. Use feature importance techniques to determine which inputs have the most influence on the outputs and possibly use that to guide the search.

Suggestion 3 *Conduct an empirical study for the algorithmic parameters for the DBMOSA algorithm.*

The study should consider different annealing schedules (some of which are listed in §5.4.2), initial temperature (see §5.4.3 for some guidelines) and the effect of different initial solutions.

Suggestion 4 *Design a complexity measure that encapsulates the complexity of a simulation problem.*

Currently, complexity is measured in terms of the number of objectives, number of decision variables, the size of the search space and number of stochastic elements present in the simulation problem. Some have proposed complexity measures [207], however, it is too complex as it takes into account the complexity of the software used, for example the number of objects modeled.

In this study, BAP16 is considered the most complex simulation problem, however no consideration was made for the actual time required to optimise the model. In that case the OMP would have to be considered as it took the longest amount of time to evaluate due to the simulation of the waiting times of the trucks moving to the trains *etc.* The complexity term should also consider the computer used to run the simulations since it influences the time required to optimise a model, *i.e.* consider the central and graphical processing units. Moreover, the time it takes to optimise a model is also influenced by the number of models optimised simultaneously and should therefore also be considered.

Suggestion 5 *Design an improved hyperarea calculation that takes into account the true number of non-dominated solutions found.*

If the true Pareto set is available, then the hyperarea calculation should only consider the area underneath the non-dominated solutions that corresponds to the true non-dominated solutions found in the true Pareto set.

Suggestion 6 *Consolidate the method used to calculate the hyperarea of an approximation front.*

In the literature there are various methods proposed for calculating the hyperarea of an approximation set, namely by method of summing the rectangles under the front (for a *min–max* approximation front), or a more accurate method which includes summing the rectangles and trapezoids (as employed in this study) under the front. Also, some literature suggests that the area under each non-dominated solution with reference to the chosen reference point be summed, however, it does not translate to the dominated area.

Suggestion 7 *Further the work done to determine the feasibility of an ANN.*

As mentioned, the Python scripts included in Listing A can be used as a starting point to further the study and ascertain more definitive results pertaining to the feasibility of an ANN as metamodel. Also, some suggestions for further experimentation is listed in §3.5.

References

[1] Abdel-Basset M, Manogaran G, Rashad H & Zaied ANH, 2018, *A comprehensive review of quadratic assignment problem: variants, hybrids and applications*, Journal of Ambient Intelligence and Humanized Computing, pp. 1–24.

[2] Aiken LS, West SG & Pitts SC, 2003, *Multiple linear regression*, Handbook of Psychology, pp. 481–507.

[3] Amaran S, Sahinidis NV, Sharda B & Bury SJ, 2016, *Simulation optimization: a review of algorithms and applications*, Annals of Operations Research, **240(1)**, pp. 351–380.

[4] Andradóttir S, 2006, *An overview of simulation optimization via random search*, Handbooks in Operations Research and Management Science, **13(1)**, pp. 617–631.

[5] April J, Glover FW, Kelly JP & Laguna M, 2003, *Simulation-based optimization: Practical introduction to simulation optimization*, Proceedings of the 35th Winter Simulation Conference: Driving Innovation, New Orleans (NO), pp. 71–78.

[6] Asta S, Özcan E, Parkes AJ & Etaner-Uyar AŞ, 2013, *Generalizing hyper-heuristics via apprenticeship learning*, Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization, Berlin, Heidelberg, pp. 169–178.

[7] Atashpaz-Gargari E & Lucas C, 2007, *Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition*, Proceedings of the IEEE Congress on Evolutionary Computation, pp. 4661–4667.

[8] Awad M & Khanna R, "Support vector regression", in: *Efficient learning machines*, Springer, 2015, pp. 67–80.

[9] Ayob M & Kendall G, 2003, *A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine*, Proceedings of the International Conference on Intelligent Technologies, pp. 132–141.

[10] Bäck T, 1996, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press, Oxford, UK.

[11] Bäck T, Fogel DB & Michalewicz Z, 1997, *Handbook of Evolutionary Computation, New York (NY)*, Oxford University Press/IOP Publishing, **97(1)**.

[12] Bai R, Blazewicz J, Burke EK, Kendall G & McCollum B, 2012, *A simulated annealing hyper-heuristic methodology for flexible decision support*, A Quarterly Journal of Operations Research (4OR-Q J Oper Res), **10(1)**, pp. 43–66.

[13] Bai R & Kendall G, "An investigation of automated planograms using a simulated annealing based hyper-heuristic", in: *Metaheuristics: Progress as Real Problem Solvers*, Springer, 2005, pp. 87–108.

[14] Baker JE, 1985, *Adaptive selection methods for genetic algorithms*, Proceedings of the International Conference on Genetic Algorithms and their Applications, Pittsburg (PA).

[15] Bamporiki T, Bekker J & Yoon M, 2019, *Using a discrete-event, simulation optimisation optimiser to solve a stochastic multi-objective NP-hard problem*, Proceedings of the International Conference on Competitive Manufacturing, Stellenbosch.

[16] Bangsow S, 2020, *Tecnomatix Plant Simulation*, 1st Edition, Springer.

[17] Banzhaf W, Nordin P, Keller RE & Francone FD, 1998, *Genetic Programming: An Introduction*, Morgan Kaufmann Publishers, San Francisco (SF).

[18] Barton RR & Meckesheimer M, 2006, *Metamodel-based Simulation Optimization*, Handbooks in Operations Research and Management Science, **13(1)**, pp. 535–574.

[19] Bashyam S & Fu M, 1998, *Optimization of (s, S) inventory systems with random lead times and a service level constraint*, Management Science, **44(1)**, pp. 243–256.

[20] Bekker JF, *Introduction to discrete-event simulation*, Stellenbosch, July 2019.

[21] Bekker J, 2012, *Applying the cross-entropy method in multi-objective optimisation of dynamic stochastic systems*, PhD thesis, Stellenbosch University, Stellenbosch.

[22] Bekker J & Aldrich C, 2011, *The cross-entropy method in multi-objective optimisation: An assessment*, European Journal of Operational Research, **211(1)**, pp. 112–121.

[23] Benlic U, Epitropakis MG & Burke EK, 2017, *A hybrid breakout local search and reinforcement learning approach to the vertex separator problem*, European Journal of Operational Research, **261(3)**, pp. 803–818.

[24] Bergmann B & Hommel G, "Improvements of general multiple test procedures for redundant systems of hypotheses", in: *Multiple Hypotheses Testing*, Springer, 1988, pp. 100–115.

[25] Bergstra J & Bengio Y, 2012, *Random search for hyper-parameter optimization.*, Journal of machine learning research, **13(2)**, February, pp. 281–305.

[26] Bergstra J, Yamins D & Cox D, 2013, *Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures*, Proceedings of the 30th International Conference on Machine Learning (ICML), Atlanta, Georgia (GA), pp. 115–123.

[27] Bettonvil B, 1989, *A formal description of discrete event dynamic systems including infinitesimal perturbation analysis*, European Journal of Operational Research, **42(2)**, pp. 213–222.

[28] Bettonvil B, Del Castillo E & Kleijnen JP, 2009, *Statistical testing of optimality conditions in multiresponse simulation-based optimization*, European Journal of Operational Research, **199(2)**, pp. 448–458.

[29] Bilgin B, Özcan E & Korkmaz EE, 2007, *An Experimental Study on Hyper-heuristics and Exam Timetabling*, Proceedings of the International Conference on the Practice and Theory of Automated Timetabling, Berlin, Heidelberg, pp. 394–412.

[30] Bishop CM, 1995, *Neural Networks for Pattern Recognition*, Oxford University Press, Cambridge, UK.

[31] Bjorck J, Gomes C, Selman B & Weinberger KQ, 2018, *Understanding Batch Normalization*, Proceedings of the.

[32] Blickle T & Thiele L, 1996, *A comparison of selection schemes used in evolutionary algorithms*, Evolutionary Computation, **4(4)**, pp. 361–394.

[33] Boute RN, Gijsbrechts J, van Jaarsveld W & Vanvuchelen N, 2022, *Deep reinforcement learning for inventory control: A roadmap*, European Journal of Operational Research, **298(2)**, April, pp. 401–412.

[34] Brownlee J, 2017, *What is the difference between test and validation datasets*, url: `https://machinelearningmastery.com/difference-test-validation-datasets/#:~:text=That%20the%20%E2%80%9Cvalidation%20dataset%E2%80%9D%20is,it%20to%20other%20final%20models`.

[35] Burke EK & Bykov Y, 2008, *A Late Acceptance Strategy in Hill-climbing for Exam Timetabling Problems*, Proceedings of the Conference on the Practice and Theory of Automated Timetabling (PATAT), Montreal, Canada.

[36] Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Özcan 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.

[37] Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E & Qu R, 2009, *A survey of hyper-heuristics*, (Unpublished) Technical Report, School of Computer Science and Information Technology, University of Nottingham, Nottingham, UK.

[38] Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E & Woodward JR, 2010, *A Classification of Hyper-heuristic Approaches*, pp. 449–468 in Gendreau M & Potvin JY (Eds), *Handbook of Metaheuristics*, Springer US, Boston (MA).

[39] Burke EK, Hyde MR, Kendall G, Ochoa G, Özcan E & Woodward JR, 2019, *A Classification of Hyper-heuristic Approaches: Revisited*, pp. 453–477 in Gendreau M & Potvin JY (Eds), *Handbook of Metaheuristics*, Springer, Cham.

[40] Burke EK, Kendall G & Soubeiga E, 2003, *A tabu-search hyperheuristic for timetabling and rostering*, Journal of Heuristics, **9(6)**, pp. 451–470.

[41] Can B & Heavey C, 2012, *A comparison of genetic programming and artificial neural networks in metamodeling of discrete-event simulation models*, Computers & Operations Research, **39(2)**, February, pp. 424–436.

[42] Chakhlevitch K & Cowling P, 2008, *Hyperheuristics: Recent Developments*, pp. 3–29 in Cotta C, Sevaux M & Sörensen K (Eds), *Adaptive and Multilevel Metaheuristics*, Springer, Berlin, Heidelberg.

[43] Chen T & Guestrin C, 2016, *Xgboost: A scalable tree boosting system*, Proceedings of the 22th International Conference on Knowledge Discovery and Data Mining, New York (NY), pp. 785–794.

[44] Clarke SM, Griebsch JH & Simpson TW, 2005, *Analysis of support vector regression for approximation of complex engineering analyses*, Journal of Mechanical Design, **127(6)**, pp. 1077–1087.

[45] Clevert DA, Unterthiner T & Hochreiter S, 2015, *Fast and accurate deep network learning by exponential linear units (elus).*

[46] Coello Coello CA, 2009, *Evolutionary multi-objective optimization: Some current research trends and topics that remain to be explored*, Frontiers of Computer Science in China, **3(1)**, pp. 18–30.

[47] Coello Coello CA, Lamont GB & Van Veldhuizen DA, 2007, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd Edition, Springer, New York (NY).

[48] Conover WJ, 1999, *Practical nonparametric statistics*, 3rd Edition, John Wiley & Sons.

[49] Cortes C & Vapnik V, 1995, *Support-vector networks*, Machine learning, **20(3)**, pp. 273–297.

[50] Cowling P, Kendall G & Soubeiga E, 2001, *A Hyperheuristic Approach to Scheduling a Sales Summit*, Proceedings of the Practice and Theory of Automated Timetabling III, Berlin, Heidelberg, pp. 176–190.

[51] Cowling P, Kendall G & Soubeiga E, 2002, *Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation*, Proceedings of the Workshops on Applications of Evolutionary Computation, Berlin, Heidelberg, pp. 1–10.

[52] Crawford B, Soto R, Astorga G, Garciéa J, Castro C & Paredes F, 2017, *Putting continuous metaheuristics to work in binary search spaces*, Machine Learning Applied to Metaheuristics and Combinatorial Problems, **2017(2)**, pp. 1–19.

[53] Cruz F, Van Woensel T & Smith JM, 2010, *Buffer and throughput trade-offs in M/G/1/K queueing networks: A bi-criteria approach*, International Journal of Production Economics, **125(2)**, June, pp. 224–234.

[54] Cruz FR, Duarte AR & Van Woensel T, 2008, *Buffer allocation in general single-server queueing networks*, Computers & Operations Research, **35(11)**, pp. 3581–3598.

[55] Cutler A, Cutler DR & Stevens JR, 2012, *Random forests*, pp. 157–175 in Zhang C & Ma Y (Eds), *Ensemble Machine Learning: Methods and Applications*, Springer US, Boston (MA).

[56] Cybenko G, 1989, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals and Systems, **2(4)**, pp. 303–314.

[57] Dasgupta D & Michalewicz Z, 1997, *Evolutionary algorithms—an overview*, pp. 3–28 in Dasgupta D & Michalewicz Z (Eds), Springer, Berlin, Heidelberg.

[58] De Boer PT, Kroese DP & Rubinstein RY, 2004, *A fast cross-entropy method for estimating buffer overflows in queueing networks*, Management Science, **50(7)**, pp. 883–895.

[59] Deb K, 2011, *Multi-objective optimisation using evolutionary algorithms: An introduction*, pp. 3–34 in Wang L, Ng AHC & Deb K (Eds), *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*, Springer, London (UK).

[60] Deb K, 2001, *Multi-Objective Optimization Using Evolutionary*, John Wiley & Sons.

[61] Deb K, 2010, *Recent Developments in Evolutionary Multi-Objective Optimization*, pp. 339–368 in Ehrgott M, Figueira JR & Greco S (Eds), Springer, Boston (MA).

[62] Deb K & Agrawal RB, 1995, *Simulated binary crossover for continuous search space*, Complex Systems, **9(2)**, pp. 115–148.

[63] Deb K & Agrawal S, 1999, *A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms*, Proceedings of the Artificial Neural Nets and Genetic Algorithms, Vienna, pp. 235–243.

[64] Deb K & Beyer HG, 2001, *Self-Adaptive Genetic Algorithms with Simulated Binary Crossover*, Evolutionary Computation, **9(2)**, June, pp. 197–221.

[65] Deb K & Deb D, 2014, *Analysing mutation schemes for real-parameter genetic algorithms*, International Journal of Artificial Intelligence and Soft Computing, **4(1)**, February, pp. 1–28.

[66] Deb K & Kumar A, 1995, *Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems*, Complex Systems, **9(6)**, pp. 431–454.

[67] 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)**, April, pp. 182–197.

[68] Deb K & Sundar J, 2006, *Reference Point Based Multi-Objective Optimization Using Evolutionary Algorithms*, Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, Seattle, Washington (WA), pp. 635–642.

[69] Deep K & Thakur M, 2007, *A new crossover operator for real coded genetic algorithms*, Applied Mathematics and Computation, **188(1)**, pp. 895–911.

[70] Deep K & Thakur M, 2007, *A new mutation operator for real coded genetic algorithms*, Applied Mathematics and Computation, **193(1)**, October, pp. 211–230.

[71] Derrac J, Garciéa 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)**, March, pp. 3–18.

[72] Di Pierro F, Khu ST & Savic DA, 2007, *An investigation on preference order ranking scheme for multiobjective evolutionary optimization*, IEEE Transactions on Evolutionary Computation, **11(1)**, pp. 17–45.

[73] Dorigo M & Blum C, 2005, *Ant colony optimization theory: A survey*, Theoretical Computer Science, **344(3)**, November, pp. 243–278.

[74] Dowsland KA, Soubeiga E & Burke E, 2007, *A simulated annealing based hyper-heuristic for determining shipper sizes for storage and transportation*, European Journal of Operational Research, **179(3)**, June, pp. 759–774.

[75] Dozat T, 2016, *Incorporating nesterov momentum into adam*, URL: `https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ`.

[76] Drake JH, Kheiri A, Özcan E & Burke EK, 2020, *Recent advances in selection hyper-heuristics*, European Journal of Operational Research, **285(2)**, September, pp. 405–428.

[77] Dréo J, Pétrowski A, Siarry P & Taillard E, 2006, *Metaheuristics for Hard Optimization*, 1st Edition, Springer, Berlin, Heidelberg.

[78] Duchi J, Hazan E & Singer Y, 2011, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research, **12(7)**, July, pp. 2121–2159.

[79] Dueck G, 1993, *New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel*, Journal of Computational Physics, **104(1)**, January, pp. 86–92.

[80] Edmund K.Burke GK (Ed), 2010, *Search Methodologies*, 1st Edition, Springer, New York (NY).

[81] Eremenko K & de Ponteves H, 2019, *Machine Learning AZ: Hands-On Python & R In Data Science.*

[82] Fang H, Wang Q, Tu YC & Horstemeyer MF, 2008, *An efficient non-dominated sorting method for evolutionary algorithms*, Evolutionary Computation, **16(3)**, September, pp. 355–384.

[83] Fausett LV, 1993, *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*, 1st Edition, Pearson.

[84] Fausett LV, 1994, *Neural Networks Based on Competition*, Proceedings of the Fundamentals of Neural Networks: Architectures, Algorithms and Applications. Englewood Cliffs: Prentice-Hall, pp. 156–217.

[85] Fleischer M, 2003, *The Measure of Pareto Optima Applications to Multi-objective Metaheuristics*, Proceedings of the International Conference on Evolutionary Multi-criterion Optimization, Berlin, Heidelberg, pp. 519–533.

[86] Fonseca DJ & Navaresse D, 2002, *Artificial neural networks for job shop simulation*, Advanced Engineering Informatics, **16(4)**, October, pp. 241–246.

[87] Fonseca DJ, Navaresse DO & Moynihan GP, 2003, *Simulation metamodeling through artificial neural networks*, Engineering Applications of Artificial Intelligence, **16(3)**, April, pp. 177–183.

[88] Friedman JH, 1991, *Multivariate adaptive regression splines*, The Annals of Statistics, **19(1)**, March, pp. 1–67.

[89] Friedman M, 1937, *The use of ranks to avoid the assumption of normality implicit in the analysis of variance*, Journal of the American Statistical Association, **32(200)**, May, pp. 675–701.

[90] Fu MC, 1994, *Optimization via simulation: A review*, Annals of Operations Research, **53(1)**, December, pp. 199–247.

[91] Geem ZW, Kim JH & Loganathan GV, 2001, *A New Heuristic Optimization Algorithm: Harmony Search*, Simulation: Transactions of The Society for Modeling and Simulation International, **76(2)**, February, pp. 60–68.

[92] Geman S & Geman D, 1984, *Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), **6(6)**, November, pp. 721–741.

[93] El-Ghazali T, 2009, *Metaheuristics from Design to Implementation*, John Wiley & Sons.

[94] Glasserman P & Ho Y.-C, 1991, *Gradient Estimation via Perturbation Analysis*, 1st Edition, Springer Science & Business Media, New York (NY).

[95] Glorot X & Bengio Y, 2010, *Understanding the Difficulty of Training Deep Feedforward Neural Networks*, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (PMLR), Sandinia, Italy, pp. 249–256.

[96] Glorot X, Bordes A & Bengio Y, 2011, *Deep Sparse Rectifier Neural Networks*, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Fort Lauderdale (FL), pp. 315–323.

[97] Glover F, 1986, *Future paths for integer programming and links to artificial intelligence*, Computers & Operations Research, **13(5)**, pp. 533–549.

[98] Glover F, 1977, *Heuristics for integer programming using surrogate constraints*, Decision Sciences, **8(1)**, January, pp. 156–166.

[99] Glover F, Kelly JP & Laguna M, 1996, *New Advances and Applications of Combining Simulation and Optimization*, Proceedings of the 28th Conference on Winter Simulation, California (CA), pp. 144–152.

[100] Glover F & Laguna M, 1997, *Tabu Search Principles*, 1st Edition, Springer, Boston (MA).

[101] Glover F, Laguna M & Martié R, 2015, *Scatter Search*, pp. 519–537 in Ghosh A & Tsutsui S (Eds), *Advances in Evolutionary Computing*, Springer, Berlin, Heidelberg.

[102] Glynn PW, 1987, *Likelilood ratio gradient estimation: an overview*, Proceedings of the 19th Conference on Winter Simulation, Georgia (GA), pp. 366–375.

[103] Goldberg DE & Deb K, 1991, *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms*, Foundations of Genetic Algorithms, **1**, pp. 69–93.

[104] Goldberg DE, Korb B & Deb K, 1989, *Messy genetic algorithms: Motivation, analysis, and first results*, Complex Systems, **3(5)**, pp. 493–530.

[105] Goldsman D & Nelson BL, 1994, *Ranking, selection and multiple comparisons in computer simulation*, Proceedings of the Winter Simulation Conference, Lake Buena Vista (FL), pp. 192–199.

[106] Goodfellow I, Bengio Y, Courville A & Bengio Y, 2016, *Deep learning*, MIT press, Cambridge (UK).

[107] Grefenstette JJ, 1992, *Genetic algorithms for changing environments.*

[108] Guizzo G, Vergilio SR, Pozo AT & Fritsche GM, 2017, *A Multi-objective and Evolutionary Hyper-heuristic Applied to the Integration and Test Order Problem*, Applied Soft Computing, **56**, March, pp. 331–344.

[109] Hachicha W, 2011, *A simulation metamodelling based neural networks for lot-sizing problem in MTO sector*, International Journal of Simulation Modelling, **10(4)**, pp. 191–203.

[110] Hajek B & Sasaki G, 1989, *Simulated annealing—to cool or not*, Systems & Control Letters, **12(5)**, June, pp. 443–447.

[111] Hansen P & Mladenović N, 2001, *Variable neighborhood search: Principles and applications*, European Journal of Operational Research, **130(3)**, pp. 449–467.

[112] He K, Zhang X, Ren S & Sun J, 2015, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 1026–1034.

[113] Heiberger RM & Neuwirth E, 1998, *Polynomial regression*, pp. 235–268 in Rawlings JO, Pantula SG & Dickey DA (Eds), *Applied Regression Analysis: A Research Tool*, Springer, New York (NY).

[114] Hertz A & de Werra D, 1990, *The tabu search metaheuristic: How we used it*, Annals of Mathematics and Artificial Intelligence, **1(1)**, September, pp. 111–121.

[115] Hill NJ & Parks GT, 2015, *Pressurized water reactor in-core nuclear fuel management by tabu search*, Annals of Nuclear Energy, **75**, January, pp. 64–71.

[116] Hinton G, Srivastava N & Swersky K, 2012, *Neural Networks for Machine Learning*, Lecture 6a: Overview of mini-batch gradient descent.

[117] Ho YC & Cao X.-R, 1991, *Introduction to Discrete Event Dynamic Systems*, pp. 1–13 in , *Perturbation Analysis of Discrete Event Dynamic Systems*, Springer, Boston (MA).

[118] Hochberg Y & Tamhane AC, 1987, *Multiple Comparison Procedures*, John Wiley & Sons.

[119] Holland JH, 1992, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.*

[120] Holland JH, 1992, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT press.

[121] Hollander M, Wolfe DA & Chicken E, 2015, *Nonparametric Statistical Methods*, 3rd Edition, John Wiley & Sons.

[122] Holm S, 1979, *A Simple Sequentially Rejective Multiple Test Procedure*, Scandinavian Journal of Statistics, **6(2)**, pp. 65–70.

[123] Hoos HH & Stützle T, 2004, *Stochastic Local Search: Foundations and Applications*, Elsevier, San Francisco (CA).

[124] Hornik K, 1991, *Approximation capabilities of multilayer feedforward networks*, Neural Networks, **4(2)**, pp. 251–257.

[125] Hornik K, Stinchcombe M & White H, 1989, *Multilayer feedforward networks are universal approximators*, Neural Networks, **2(5)**, pp. 359–366.

[126] Hsiao PK, Jiang S.-J, Sahayam A & Hsiao TC, 2012, *Determination of trace elements in silicon powders by inductively coupled plasma quadrupole mass spectrometry with a membrane desolvation sample introduction system*, Atomic Spectroscopy, **33(1)**, January, pp. 1–8.

[127] Huang M, 2004, *An efficient general cooling schedule for simulated annealing*, Proceedings of the Computational Science and Its Applications, Berlin, Heidelberg, pp. 396–404.

[128] Hüsselmann G, 2022, *Bus route design and frequency setting for public transit systems*, PhD thesis, Stellenbosch University, Stellenbosch.

[129] Iman RL & Davenport JM, 1980, *Approximations of the critical region of the friedman statistic*, Communications in Statistics-Theory and Methods, **9(6)**, pp. 571–595.

[130] James G, Witten D, Hastie T & Tibshirani R, 2013, *An Introduction to Statistical Learning*, 1st Edition, Springer, New York (NY).

[131] Johnson DS, Papadimitriou CH & Yannakakis M, 1988, *How easy is local search?*, Journal of Computer and System Sciences, **37(1)**, September, pp. 79–100.

[132] Kaelbling LP, Littman ML & Moore AW, 1996, *Reinforcement learning: A survey*, Journal of Artificial Intelligence Research, **4**, May, pp. 237–285.

[133] Karaboga D, 2005, *An idea based on honey bee swarm for numerical optimization*, (Unpublished) Technical Report, Erciyes University.

[134] Karimi-Mamaghan M, Mohammadi M, Meyer P, Karimi-Mamaghan AM & Talbi E.-G, 2022, *Machine Learning at the service of Meta-heuristics for solving Combinatorial Optimization Problems: A state-of-the-art*, European Journal of Operational Research, **296(2)**, January, pp. 393–422.

[135] Karpathy A, 2016, *CS231n: Convolutional Neural Networks for Visual Recognition*, Stanford University.

[136] Kelton WD, 1997, *Statistical analysis of simulation output*, Proceedings of the 29th Conference on Winter Simulation, Cincinnati, Ohio (OH), pp. 23–30.

[137] Kendall G & Hussin NM, 2005, *An investigation of a tabu-search-based hyper-heuristic for examination timetabling*, Proceedings of the Multidisciplinary Scheduling: Theory and Applications, Boston (MA), pp. 309–328.

[138] Kendall G & Mohamad M, 2004, *Channel assignment in cellular communication using a great deluge hyper-heuristic*, Proceedings of the 12th IEEE International Conference on Networks (ICON), pp. 769–773.

[139] Kennedy J, 2006, *Swarm intelligence*, pp. 187–219 in Zomaya AY (Ed), *Handbook of Nature-Inspired and Innovative Computing*, Springer.

[140] Khan W, Salhi A, Asif M, Adeeb R & Sulaiman M, 2015, *Enhanced version of multi-algorithm genetically adaptive for multiobjective optimization*, International Journal of Advanced Computer Science and Applications, **6(12)**, pp. 279–287.

[141] Kingma DP & Ba J, 2015, *Adam: A method for stochastic optimization*, Proceedings of the 3th International Conference for Learning Representations, San Diego (CA).

[142] Kiraz B, Etaner-Uyar AŞ & Özcan E, 2013, *Selection hyper-heuristics in dynamic environments*, Journal of the Operational Research Society, **64(12)**, pp. 1753–1769.

[143] Kirkpatrick S, Gelatt CD & Vecchi MP, 1983, *Optimization by simulated annealing*, American Association for the Advancement of Science, **220(4598)**, pp. 671–680.

[144] Kleijnen JP, 2009, *Kriging metamodeling in simulation: A review*, European Journal of Operational Research, **192(3)**, February, pp. 707–716.

[145] Kleijnen JP & Van Beers WC, 2004, *Application-driven sequential designs for simulation experiments: Kriging metamodelling*, Journal of the Operational Research Society, **55(8)**, pp. 876–883.

[146] Kleijnen JP & Van Beers WC, 2005, *Robustness of Kriging when interpolating in random simulation with heterogeneous variances: Some experiments*, European Journal of Operational Research, **165(3)**, September, pp. 826–834.

[147] Knowles JD, Thiele L & Zitzler E, 2005, *A tutorial on the performance assessment of stochastic multiobjective optimizers*, 3rd International Conference on Evolutionary Multi-Criterion Optimization (EMO), **214**, February.

[148] Koç Ç, Bektaş T, Jabali O & Laporte G, 2016, *Thirty years of heterogeneous vehicle routing*, European Journal of Operational Research, **249(1)**, February, pp. 1–21.

[149] Konak A, Coit DW & Smith AE, 2006, *Multi-objective optimization using genetic algorithms: A tutorial*, Reliability Engineering & System Safety, **91(9)**, September, pp. 992–1007.

[150] Korte BH, Vygen J, Korte B & Vygen J, 2012, *Combinatorial Optimization*, Springer, Berlin, Heidelberg.

[151] Koutsoukas A, Monaghan KJ, Li X & Huan J, 2017, *Deep-learning: Investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data*, Journal of Cheminformatics, **9(42)**, pp. 1–13.

[152] Krogh A & Hertz JA, 1991, *A simple weight decay can improve generalization*, Proceedings of the Advances in Neural Information Processing Systems (NIPS), pp. 950–957.

[153] Kubat M, 1998, *Neural networks: A comprehensive foundation by Simon Haykin, Macmillan, 1994, ISBN 0-02-352781-7.*, The Knowledge Engineering Review, **13(4)**, pp. 409–412.

[154] Kuhn M & Johnson K, 2013, *Applied Predictive Modeling*, 1st Edition, Springer, New York (NY).

[155] Kuo Y, Yang T, Peters BA & Chang I, 2007, *Simulation metamodel development using uniform design and neural networks for automated material handling systems in semiconductor wafer fabrication*, Simulation Modelling Practice and Theory, **15(8)**, September, pp. 1002–1015.

[156] Laguna M, *OptQuest*, Opttek Systems, Inc., 2011.

[157] Law AM, Kelton WD & Kelton WD, 2007, *Simulation Modeling and Analysis*, 5th Edition, Mc Graw Hill Education, New York (NY).

[158] Le QV, 2015, *A tutorial on deep learning part 1: Nonlinear classifiers and the backpropagation algorithm*, Google Inc., Mountain View, CA, December.

[159] LeCun YA, Bottou L, Orr GB & Müller K.-R, 2012, *Efficient backprop*, pp. 9–48 in Montavon G, Orr GB & Müller K.-R (Eds), *Neural Networks: Tricks of the Trade*, Springer, Berlin, Heidelberg.

[160] Lee LH, Chew EP, Teng S & Goldsman D, 2010, *Finding the non-dominated Pareto set for multi-objective simulation models*, IIE Transactions, **42(9)**, pp. 656–674.

[161] Lehrbaum A & Musliu N, 2012, *A New Hyperheuristic Algorithm for Cross-domain Search Problems*, Proceedings of the International Conference on Learning and Intelligent Optimization (LION), Berlin, Heidelberg, pp. 437–442.

[162] Li M & Yao X, 2020, *Quality evaluation of solution sets in multiobjective optimisation: A survey*, ACM Computing Surveys (CSUR), **52(26)**, March, pp. 1–38.

[163] Li W, Özcan E & John R, 2017, *A learning automata-based multiobjective hyper-heuristic*, IEEE Transactions on Evolutionary Computation, **23(1)**, pp. 59–73.

[164] Li W, Özcan E & John R, 2017, *Multi-objective evolutionary algorithms and hyper-heuristics for wind farm layout optimisation*, Renewable Energy, **105**, May, pp. 473–482.

[165] Li YF, Ng SH, Xie M & Goh T, 2010, *A systematic comparison of metamodeling techniques for simulation optimization in decision support systems*, Applied Soft Computing, **10(4)**, September, pp. 1257–1273.

[166] Liagkouras K & Metaxiotis K, 2013, *An elitist polynomial mutation operator for improved performance of MOEAs in computer networks*, Proceedings of the 22nd International Conference on Computer Communication and Networks (ICCCN), Nassau, Bahamas, pp. 1–5.

[167] Lin C, Yang JI, Lin KJ & Wang ZD, 1998, *Pressurized water reactor loading pattern design using the simple tabu search*, Nuclear Science and Engineering, **129(1)**, pp. 61–71.

[168] Liu Y, Wang Y & Zhang J, 2012, *New machine learning algorithm: Random forest*, Proceedings of the International Conference on Information Computing and Applications (ICICA), Berlin, Heidelberg, pp. 246–252.

[169] Lotter DP, 2017, *Design of a weapon assignment subsystem within a ground-based air defence environment*, PhD thesis, Stellenbosch University, Stellenbosch.

[170] Lourenço HR, Martin OC & Stützle T, "Iterated Local Search", in: *Handbook of Metaheuristics*, ed. by Fred Glover and Gary A. Kochenberger, Boston (MA): Springer, 2003, pp. 320–353.

[171] Lozano M, Herrera F & Cano JR, 2008, *Replacement strategies to preserve useful diversity in steady-state genetic algorithms*, Information Sciences, **178(23)**, pp. 4421–4433.

[172] Maas AL, Hannun AY & Ng AY, 2013, *Rectifier nonlinearities improve neural network acoustic models*, Proceedings of the, (CA).

[173] Maashi M, Özcan E & Kendall G, 2014, *A multi-objective hyper-heuristic based on choice function*, Expert Systems with Applications, **41(9)**, July, pp. 4475–4493.

[174] MacGregor Smith J & Cruz FR, 2005, *The buffer allocation problem for general finite buffer queueing networks*, IIE Transactions, **37(4)**, pp. 343–365.

[175] Mahfoud SW, 1995, *Niching methods for genetic algorithms*, PhD thesis, University of Illinois, Urbana, Illinois (IL).

[176] Mattera D, Palmieri F & Haykin S, 1999, *An explicit algorithm for training support vector machines*, IEEE Signal Processing Letters, **6(9)**, September, pp. 243–245.

[177] Mazur M, *A step by step backpropagation example*, 2015.

[178] Meghabghab G & Kandel A, 2004, *Stochastic simulations of web search engines: RBF versus second-order regression models*, Information Sciences, **159(1-2)**, January, pp. 1–28.

[179] Mehrotra K, Mohan CK & Ranka S, 1996, *Elements of Artificial Neural Networks*, MIT press.

[180] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH & Teller E, 1953, *Equation of state calculations by fast computing machines*, The Journal of Chemical Physics, **21(6)**, pp. 1087–1092.

[181] Miettinen K, 1998, *Nonlinear Multiobjective Optimization*, 1st Edition, Springer Science & Business Media, New York (NY).

[182] Misir M, Vancroonenburg W, Verbeeck K & Berghe GV, 2011, *A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete*, Proceedings of the 9th Meta-heuristics International Conference (MIC 2011), Udine, Italy, pp. 289–298.

[183] Misir M, Verbeeck K, De Causmaecker P & Berghe GV, 2010, *Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem*, Proceedings of the Congress on Evolutionary Computation, pp. 1–8.

[184] Mitchell TM, 1997, *Machine Learning*, McGraw-Hill, New York (NY).

[185] Mladenović N & Hansen P, 1997, *Variable neighborhood search*, Computers & Operations Research, **24(11)**, November, pp. 1097–1100.

[186] Mornati F, 2013, *Pareto Optimality in the work of Pareto*, European Journal of Social Sciences, **(51-2)**, pp. 65–82.

[187] Moscato P, 2000, *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*, Caltech Concurrent Computation Program, October.

[188] Mueller JP & Massaron L, 2019, *Deep Learning*, For Dummies.

[189] Nair V & Hinton GE, 2010, *Rectified linear units improve restricted boltzmann machines*, Proceedings of the International Conference on Machine Learning (ICML), pp. 807–814.

[190] Nakayama H, Arakawa M & Sasaki R, 2002, *Simulation-based optimization using computational intelligence*, Optimization and Engineering, **3(2)**, June, pp. 201–214.

[191] Negahban A & Smith JS, 2014, *Simulation for manufacturing system design and operation: Literature review and analysis*, Journal of Manufacturing Systems, **33(2)**, pp. 241–261.

[192] Nel GS, 2021, *A hyperheuristic approach towards the training of artificial neural networks*, PhD thesis, Stellenbosch University, Stellenbosch.

[193] Nesterov Y, 1983, *A method for unconstrained convex minimization problem with the rate of convergence*, Mathematics.

[194] Nielsen M, 2015, *How the backpropagation algorithm works*, Proceedings of the Neural Networks and Deep Learning.

[195] Nielsen MA, 2019, *Neural Networks and Deep Learning*, San Francisco (CA).

[196] Osman IH & Kelly JP, 1997, *Meta-heuristics theory and applications*, Journal of the Operational Research Society, **48(6)**, pp. 657–657.

[197] Osman IH & Kelly JP, 1996, *Meta-heuristics: An overview*, pp. 1–21 in Osman IH & Kelly JP (Eds), Springer, Boston (MA).

[198] Osman IH & Laporte G, 1996, *Metaheuristics: A bibliography*, Annals of Operations Research, **63**, pp. 511–623.

[199] Özcan E, Bilgin B & Korkmaz EE, 2008, *A comprehensive analysis of hyper-heuristics*, Intelligent Data Analysis, **12(1)**, pp. 3–23.

[200] Özcan E, Bykov Y, Birben M & Burke EK, 2009, *Examination timetabling using late acceptance hyper-heuristics*, Proceedings of the Congress on Evolutionary Computation, pp. 997–1004.

[201] Özcan E, Misir M, Ochoa G & Burke EK, 2012, *A reinforcement learning: Great-deluge hyper-heuristic for examination timetabling*, pp. 34–55 in , *Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends*, IGI Global.

[202] Papadopoulos HT & Heavey C, 1996, *Queueing theory in manufacturing systems analysis and design: A classification of models for production and transfer lines*, European Journal of Operational Research, **92(1)**, July, pp. 1–27.

[203] Parkinson AR, Balling R & Hedengren JD, 2013, *Optimization Methods for Engineering Design*, Brigham Young University, **5(11)**.

[204] Pereira I, Madureira A, de Moura Oliveira PB & Abraham A, "Tuning meta-heuristics using multi-agent learning in a scheduling system", in: *Transactions on Computational Science XXI*, ed. by Marina L. Gavrilova, C.J.Kenneth Tan, and Ajith Abraham, vol. 8160, Lecture Notes in Computer Science book series, Berlin, Heidelberg: Springer, 2013, pp. 190–210.

[205] Podgorelec V & Zorman M, 2016, *Decision tree learning*, pp. 1–28 in , *Encyclopedia of Complexity and Systems Science*, Springer, Berlin, Heidelberg.

[206] Pohlert T, *The pairwise multiple comparison of mean ranks package (PMCMR)*, 2019, R package, 2014, p. 9.

[207] Popovics G & Monostori L, 2016, *An approach to Determine Simulation Model Complexity*, The Sixth International Conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV), **(257–261)**.

[208] Qian N, 1999, *On the momentum term in gradient descent learning algorithms*, Neural Networks, **12(1)**, January, pp. 145–151.

[209] Razali NM & Geraghty J, 2011, *Genetic algorithm performance with different selection strategies in solving TSP*, Proceedings of the 1th World Congress on Engineering, London (UK), pp. 1–6.

[210] Riquelme N, Von Lücken C & Baran B, 2015, *Performance metricsi n multi-objective optimization*, Proceedings of the Latin American Computing Conference (CLEI), Arequipa, Peru, pp. 1–11.

[211] Rojas R, 2013, *Neural networks: A systematic introduction*, 1st Edition, Springer Science & Business Media, Berlin, Heidelberg.

[212] Rokach L & Maimon O, 2005, *Decision trees*, pp. 165–192 in Maimon O & Rokach L (Eds), *Data Mining and Knowledge Discovery Handbook*, Springer, Boston (MA).

[213] Ross P, 2005, *Hyper-heuristics*, pp. 529–556 in Burke EK & Kendall G (Eds), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer.

[214] Rostami S, 2019, *Hypervolume Indicator*, Accessed: 2022-09-27, URL: `https://datacrayon.com/posts/search%20and%20optimisation/practical%20evolutionary%20algorithms/hypervolume%20indicator`.

[215] Rubinstein R, 1990, *How to optimize discrete-event systems from a single sample path by the score function method*, Annals of Operations Research, **27(1)**, pp. 175–212.

[216] Rubinstein RY & Shapiro A, 1993, *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*, 1st Edition, Wiley.

[217] Rubinstein RY & Kroese DP, 2004, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*, Springer.

[218] Russell S & Norvig P, 2009, *Artificial Intelligence: A Modern Approach*, 3$^{\text{rd}}$ Edition, Pearson.

[219] Sabuncuoglu I & Touhami S, 2002, *Simulation metamodelling with neural networks: An experimental investigation*, International Journal of Production Research, **40(11)**, pp. 2483–2505.

[220] Sacks J, Welch WJ, Mitchell TJ & Wynn HP, 1989, *Design and analysis of computer experiments*, Statistical Science, **4(4)**, pp. 409–423.

[221] Salman A, Ahmad I & Al-Madani S, 2002, *Particle swarm optimization for task assignment problem*, Microprocessors and Microsystems, **26(8)**, November, pp. 363–371.

[222] Schlunz EB, 2016, *Multiobjective in-core fuel management optimisation for nuclear research reactors*, PhD thesis, Stellenbosch University, Stellenbosch.

[223] Schölkopf B, Smola AJ & Bach F, 2002, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT press.

[224] Schriber TJ, Brunner DT & Smith JS, 2013, *Inside discrete-event simulation software: How it works and why it matters*, Proceedings of the 2013 Winter Simulations Conference (WSC), pp. 424–438.

[225] Schwefel HP & Rudolph G, 1995, pp. 7–45 in , *Contemporary Evolution Strategies*, Springer, Berlin, Heidelberg.

[226] Shaffer JP, 1986, *Modified sequentially rejective multiple test procedures*, Journal of the American Statistical Association, **81(395)**, pp. 826–831.

[227] Shah T, 2020, *About Train, Validation and Test Sets in Machine Learning.*

[228] Shaw P, 1998, *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, pp. 417–431.

[229] Sheskin DJ, 2011, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman & Hall.

[230] Shin M, Sargent RG & Goel AL, 2002, *Gaussian radial basis functions for simulation metamodeling*, Proceedings of the Proceedings of the Winter Simulation Conference, San Diego (CA), pp. 483–488.

[231] Sidney S, 1957, *Nonparametric statistics for the behavioral sciences*, **125(3)**.

[232] Smith KI, Everson RM & Fieldsend JE, 2004, *Dominance measures for multi-objective simulated annealing*, Proceedings of the Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753), pp. 23–30.

[233] Smith KI, Everson RM, Fieldsend JE, Murphy C & Misra R, 2008, *Dominance-based multiobjective simulated annealing*, IEEE Transactions on Evolutionary computation, **12(3)**, pp. 323–342.

[234] Smola AJ & Schölkopf B, 2004, *A tutorial on support vector regression*, Statistics and Computing, **14(3)**, pp. 199–222.

[235] Snoek J, Larochelle H & Adams RP, 2012, *Practical bayesian optimization of machine learning algorithms*, Advances in Neural Information Processing Systems (NIPS), **25**.

[236] Soliés JF, Fraire HJ, Soto-Monterrubio JC & Pazos-Rangel R, "Multi-Objective Simulated Annealing Algorithms for General Problems", in: *Handbook of Research on Military, Aeronautical, and Maritime Logistics and Operations*, IGI Global, 2016, pp. 280–292.

[237] Sørngård B, 2014, *Information Theory for Analyzing Neural Networks*, Proceedings of the.

[238] Srinivas N & Deb K, 1994, *Muiltiobjective optimization using nondominated sorting in genetic algorithms*, Evolutionary Computation, **2(3)**, pp. 221–248.

[239] Steurer M, Hill RJ & Pfeifer N, 2021, *Metrics for evaluating the performance of machine learning based automated valuation models*, Journal of Property Research, **38(2)**, pp. 1–31.

[240] 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.

[241] Sutskever I, Martens J, Dahl G & Hinton G, 2013, *On the importance of initialization and momentum in deep learning*, Proceedings of the 3^{th} 30th International Conference on Machine Learning (ICML), Atlanta, Georgia (GA), pp. 1139–1147.

[242] Sutton RS & Barto AG, 2018, *Reinforcement Learning: An Introduction*, 2nd Edition, MIT press, Cambridge (MA).

[243] Svensén M & Bishop CM, 2007, *Pattern Recognition and Machine Learning*, Springer.

[244] Svozil D, Kvasnicka V & Pospichal J, 1997, *Introduction to multi-layer feed-forward neural networks*, omputer Science Chemometrics and Intelligent Laboratory Systems, **39(1)**, November, pp. 43–62.

[245] Tieleman T & Hinton G, 2012, *Neural Networks for Machine Learning*, (Unpublished) Technical Report, University of Toronto, Toronto.

[246] Tsang E & Voudouris C, 1997, *Fast local search and guided local search and their application to British Telecom's workforce scheduling problem*, Operations Research Letters, **20(3)**, March, pp. 119–127.

[247] Tsutsui S, Yamamura M & Higuchi T, 1999, *Multi-parent recombination with simplex crossover in real coded genetic algorithms*, Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, pp. 657–664.

[248] Van Gelder L, Das P, Janssen H & Roels S, 2014, *Comparative study of metamodelling techniques in building energy simulation: Guidelines for practitioners*, Simulation Modelling Practice and Theory, **49(1)**, December, pp. 245–257.

[249] Van Laarhoven PJ & Aarts EH, "Simulated Annealing", in: *Simulated Annealing: Theory and Applications*, Springer, June 1987, pp. 7–15.

[250] Van Luong T, Melab N & Talbi E.-G, 2011, *GPU computing for parallel local search metaheuristic algorithms*, IEEE Transactions on Computers, **62(1)**, January, pp. 173–185.

[251] Vapnik V & Chervonenkis AY, 1974, *The method of ordered risk minimization, I*, Stochastic Systems, **8**, pp. 21–30.

[252] Venter G & Sobieszczanski Sobieski J, 2004, *Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization*, Structural and Multidisciplinary Optimization, **26(1)**, January, pp. 121–131.

[253] Voudouris C & Tsang E, 1999, *Guided local search and its application to the traveling salesman problem*, European Journal of Operational Research, **113(2)**, March, pp. 469–499.

[254] Vrugt JA & Robinson BA, 2007, *Improved evolutionary optimization from genetically adaptive multimethod search*, Proceedings of the National Academy of Sciences, **104(3)**, pp. 708–711.

[255] Whitley LD, 1989, *The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best.*, Proceedings of the Junth Icga, Fairfax (VA), pp. 116–123.

[256] Winston WL, 2004, *Operations Research: Applications and Algorithms*, 4th Edition, Duxbury Press.

[257] Wolpert DH & Macready WG, 1997, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation, **1(1)**, pp. 67–82.

[258] Xu J, Huang E, Chen C.-H & Lee LH, 2015, *Simulation optimization: A review and exploration in the new era of cloud computing and big data*, Asia-Pacific Journal of Operational Research, **32(03)**.

[259] Yang S, Wu C & Hu SJ, 2000, *Modeling and analysis of multi-stage transfer lines with unreliable machines and finite buffers*, Annals of Operations Research, **93(1)**, pp. 405–421.

[260] Yoon M & Bekker J, 2017, *Single-and multi-objective ranking and selection procedures in simulation: A historical review*, South African Journal of Industrial Engineering, **28(2)**, pp. 37–45.

[261] Yoshida H, Kawata K, Fukuyama Y, Takayama S & Nakanishi Y, 2000, *A particle swarm optimization for reactive power and voltage control considering voltage security assessment*, IEEE Transactions on Power Systems, **15(4)**, pp. 1232–1239.

[262] Zaremba W, Sutskever I & Vinyals O, 2014, *Recurrent neural network regularization*, Neural and Evolutionary Computing.

[263] Zeiler MD, 2012, *Adadelta: An adaptive learning rate method*, Machine Learning.

[264] Zheng YJ, 2015, *Water wave optimization: A new nature-inspired metaheuristic*, Computers & Operations Research, **55(1)**, March, pp. 1–11.

[265] Zhou Y, Hao J.-K & Duval B, 2016, *Reinforcement learning based local search for grouping problems: A case study on graph coloring*, Expert Systems with Applications, **64**, December, pp. 412–422.

[266] Zitzler E, 1999, *Evolutionary algorithms for multiobjective optimization: Methods and applications*, PhD thesis, Swiss Federal Institute of Technology, Zurich.

[267] Zitzler E, Knowles J & Thiele L, 2008, *Quality Assessment of Pareto Set Approximations*, pp. 373–404 in Branke J, Deb K, Miettinen K & Słowiński R (Eds), *Multiobjective Optimization: Interactive and Evolutionary Approaches*, Springer, Berlin, Heidelberg.

[268] Zitzler E & Künzli S, 2004, *Indicator-Based Selection in Multiobjective Search*, Proceedings of the Parallel Problem Solving from Nature, Berlin, Heidelberg, pp. 832–842.

[269] Zitzler E & Thiele L, 1999, *Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach*, IEEE Transactions on Evolutionary Computation, **3(4)**, November, pp. 257–271.

[270] Zitzler E, Thiele L, Laumanns M, Fonseca CM & Da Fonseca VG, 2003, *Performance assessment of multiobjective optimizers: An analysis and review*, IEEE Transactions on Evolutionary Computation, **7(2)**, April, pp. 117–132.

APPENDIX A

Metamodel Pilot Study Results

This appendix contains additional results obtained during the pilot study conducted in Chapter 3 to determine the feasibility of an ANN metamodel. The results obtained for the 30 trails conducted for each regression model is presented in Tables A.1–A.5 and the Python script used for MLR is given in Listing A.2. Thereafter, the results for the 30 trials conducted for hyperparameter combinations A–B is given in Tables A.6–A.9 and the Python script for hyperparameter combinations *C* is given in Listing A.4. The Python script illustrating the process followed for Bayesian optimisation is given in Listing A.3.

Listing A.1: The Python script used for the data preprocessing phase.

```python
#Build regressors:

# 1. Multiple Linear Regression
from sklearn.linear_model import LinearRegression
mlr_regressor = LinearRegression()

# 2. Support Vector Machine Regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.svm import SVR
SVM_regressor = SVR(kernel = 'rbf')
svm_regressor = MultiOutputRegressor(SVM_regressor)

# 3. Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
dt_regressor = DecisionTreeRegressor()

# 4. Random Forest Regression
from sklearn.ensemble import RandomForestRegressor
rfr_regressor = RandomForestRegressor(n_estimators = 15, random_state = 0)

# 5. K-Nearest Neighbour Regression
from sklearn.neighbors import KNeighborsRegressor
knn_regressor = KNeighborsRegressor()

```

Listing A.2: The Python script used to run 30 independent trials for the MLR model.

```python
# Multiple Linear Regression
for i in range(1, 31):
  dataset = pd.read_csv('ssIP_dataset.csv')
  X = dataset.iloc[: , 0:2].values # Independent variables or inputs
  y = dataset.iloc[: , 2:4].values # Dependent variables or outputs

  X_train, X_test, y_train, y_test = train_test_split(X, y,
                                            test_size = 0.2,
                                            random_state = i)
  sc_X = StandardScaler()
  sc_y = StandardScaler()
  X_train = sc_X.fit_transform(X_train) # Normalise X_train
  X_test = sc_X.transform(X_test)       # Normalise X_test
  y_train = sc_y.fit_transform(y_train) # Normalise y_train
  y_test = sc_y.transform(y_test)       # Normalise y_test

  mlr_regressor = LinearRegression()    # Build regressor
  mlr_regressor.fit(X_train, y_train)
  t_y_pred = mlr_regressor.predict(X_train)

```

```
      v_mses = []
      v_maes = []
      test_maes = []
      test_mses = []
      t_mse = mean_squared_error(y_train,t_y_pred)
      t_mae = mean_absolute_error(y_train,t_y_pred)

      # 10-Fold Cross Validation
      v_mse = -np.round(cross_val_score(mlr_regressor, X_train,
                                        y_train,
                                        scoring='neg_mean_squared_error',
                                        cv = 10), 4)
      v_mses.append(v_mse)
      v_mse_avg = np.round(v_mse.mean(), 4)
      v_mae = -np.round(cross_val_score(mlr_regressor, X_train,
                                        y_train,
                                        scoring='neg_mean_absolute_error',
                                        cv = 10), 4)
      v_maes.append(v_mae)
      v_mae_avg = np.round(v_mae.mean(), 4)

      # Evaluate regressor on Test set
      test_mse = np.round(mean_squared_error(y_test, y_pred), 4)
      test_mses.append(test_mse)
      test_mse_avg = np.round(test_mse.mean(), 4)
      test_mae = np.round(mean_absolute_error(y_test, y_pred), 4)
      test_maes.append(test_mae)
      test_mae_avg = np.round(test_mae.mean(), 4)

      # Populate dataframe
      trial_performance_row = [t_mse, t_mae,
                               v_mse_avg, v_mae_avg,
                               test_mse_avg, test_mae_avg]
      MLR_performance_df.loc['Run_' + str(i)] = trial_performance_row
```

Listing A.3: The Python script used to run the Bayesian optimisation and determine suitable hyperparameters for the FNN.

```
def model(hp):

# Hyperparameter search space
hp_optimizer = hp.Choice("optimizer", values = ["sgd","adam", "nadam"])
hp_learning_rate = hp.Choice("learning_rate", values = [1e-2, 1e-3, 1e-4])
hp_regularizer = hp.Choice("kernel_regularizer", values = ["l1", "l2", "l1_l2"])
hp_activation = hp.Choice("activation", values = ["sigmoid", "relu", "elu"])
hp_initializer = hp.Choice("kernel_initializer",
values=["HeUniform", "HeNormal", "GlorotUniform", "GlorotNormal"])
hp_units = hp.Int("units", min_value = 2, max_value = 20, step = 1)

# Build ANN
model = Sequential()

# Hidden layer
model.add(Dense(units = hp_units,
              kernel_regularizer = hp_regularizer,
              activation = hp_activation,
              kernel_initializer = hp_initializer))

model.add(BatchNormalization()) # Comment out if not applicable

# Output layer
model.add(tf.keras.layers.Dense(units = 2, activation = hp_activation))

model.compile(optimizer = hp_optimizer,
              loss = "mean_squared_error",
              metrics = ["mean_squared_error"]
)

return model
```

Listing A.4: The Python script used to run 30 independent trails for the hyperparameter combination C found during Bayesian optimisation for the ANN.

```python
# Import dataset
dataset = pd.read_csv('ssIP_dataset.csv')
X = dataset.iloc[: , 0:2].values # Independent variables, i.e. inputs
y = dataset.iloc[: , 2:4].values # Dependent variables, i.e. outputs

for i in range(1, 31):
# Split dataset using random state i
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.2,
                                                    random_state = i)

# Normalise
sc_X = StandardScaler()
sc_y = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
y_train = sc_y.fit_transform(y_train)
y_test = sc_y.transform(y_test)

# Build ANN using best hyperparameters
bo_model = Sequential()

# Hidden layer
bo_model.add(Dense(units=14,
                   kernel_regularizer = "l2",
                   kernel_initializer = "HeNormal",
                   activation='elu'))

bo_model.add(BatchNormalization()) # Comment out if not applicable

# Output layer
bo_model.add(Dense(units=2, activation='elu'))

bo_optimizer = tf.keras.optimizers.Nadam(learning_rate = 0.001)

bo_model.compile(optimizer = bo_optimizer,
                 loss = "mean_squared_error",
                 metrics=["mean_squared_error"])

history_bo = bo_model.fit(X_train,
                          y_train,
                          batch_size = batch_size,
                          epochs = max_epochs,
                          validation_split = 0.25,
                          callbacks=[early_stopping])

# Evaluate model on Test set
eval_result = bo_model.evaluate(X_test, y_test)
```

Table A.1: *Summary of the performance of MLR in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0.1479	0.3109	0.1479	0.3109	0.1480	0.3111
Trial 2	0.1481	0.3110	0.1481	0.3110	0.1474	0.3103
Trial 3	0.1479	0.3109	0.1479	0.3109	0.1470	0.3100
Trial 4	0.1482	0.3112	0.1482	0.3112	0.1463	0.3092
Trial 5	0.1480	0.3108	0.1480	0.3108	0.1487	0.3118
Trial 6	0.1480	0.3109	0.1480	0.3109	0.1476	0.3106
Trial 7	0.1481	0.3111	0.1481	0.3111	0.1476	0.3105
Trial 8	0.1480	0.3109	0.1480	0.3109	0.1486	0.3118
Trial 9	0.1482	0.3111	0.1482	0.3111	0.1486	0.3117
Trial 10	0.1479	0.3108	0.1479	0.3108	0.1483	0.3113
Trial 11	0.1481	0.3110	0.1481	0.3110	0.1481	0.3109
Trial 12	0.1481	0.3111	0.1481	0.3111	0.1477	0.3105
Trial 13	0.1477	0.3107	0.1478	0.3107	0.1486	0.3114
Trial 14	0.1478	0.3107	0.1478	0.3107	0.1485	0.3116
Trial 15	0.1480	0.3111	0.1480	0.3111	0.1483	0.3108
Trial 16	0.1479	0.3109	0.1479	0.3109	0.1484	0.3111
Trial 17	0.1480	0.3108	0.1480	0.3109	0.1482	0.3114
Trial 18	0.1478	0.3107	0.1478	0.3107	0.1475	0.3107
Trial 19	0.1480	0.3109	0.1480	0.3109	0.1491	0.3120
Trial 20	0.1481	0.3110	0.1481	0.3110	0.1473	0.3107
Trial 21	0.1480	0.3110	0.1480	0.3110	0.1484	0.3111
Trial 22	0.1481	0.3110	0.1481	0.3110	0.1485	0.3115
Trial 23	0.1480	0.3108	0.1480	0.3108	0.1497	0.3128
Trial 24	0.1479	0.3109	0.1480	0.3110	0.1473	0.3103
Trial 25	0.1478	0.3106	0.1478	0.3106	0.1489	0.3120
Trial 26	0.1476	0.3106	0.1476	0.3106	0.1486	0.3116
Trial 27	0.1480	0.3108	0.1480	0.3108	0.1480	0.3113
Trial 28	0.1479	0.3109	0.1479	0.3109	0.1487	0.3117
Trial 29	0.1479	0.3109	0.1479	0.3110	0.1476	0.3103
Trial 30	0.1481	0.3111	0.1481	0.3111	0.1472	0.3101
Average	**0.1480**	**0.3109**	**0.1480**	**0.3109**	**0.1481**	**0.3111**

Table A.2: *Summary of the performance of SVM in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0.0026	0.0438	0.0026	0.0439	0.0026	0.0438
Trial 2	0.0026	0.0437	0.0026	0.0438	0.0026	0.0437
Trial 3	0.0026	0.0439	0.0026	0.0439	0.0026	0.0439
Trial 4	0.0026	0.0439	0.0026	0.0439	0.0026	0.0439
Trial 5	0.0026	0.0439	0.0026	0.0439	0.0026	0.0439
Trial 6	0.0026	0.0440	0.0026	0.0439	0.0026	0.0440
Trial 7	0.0026	0.0439	0.0026	0.0439	0.0026	0.0439
Trial 8	0.0026	0.0438	0.0026	0.0438	0.0026	0.0438
Trial 9	0.0026	0.0439	0.0026	0.0439	0.0026	0.0439
Trial 10	0.0026	0.0439	0.0026	0.0438	0.0026	0.0439
Trial 11	0.0026	0.0438	0.0026	0.0439	0.0026	0.0438
Trial 12	0.0026	0.0437	0.0026	0.0437	0.0026	0.0437
Trial 13	0.0026	0.0438	0.0026	0.0437	0.0026	0.0438
Trial 14	0.0026	0.0438	0.0026	0.0440	0.0026	0.0438
Trial 15	0.0026	0.0439	0.0026	0.0439	0.0026	0.0439
Trial 16	0.0026	0.0437	0.0026	0.0438	0.0026	0.0437
Trial 17	0.0026	0.0437	0.0026	0.0436	0.0026	0.0437
Trial 18	0.0026	0.0437	0.0026	0.0438	0.0026	0.0437
Trial 19	0.0026	0.0436	0.0026	0.0438	0.0026	0.0436
Trial 20	0.0026	0.0438	0.0026	0.0438	0.0026	0.0438
Trial 21	0.0026	0.0440	0.0026	0.0438	0.0026	0.0440
Trial 22	0.0026	0.0440	0.0026	0.0439	0.0026	0.0440
Trial 23	0.0026	0.0438	0.0026	0.0439	0.0026	0.0438
Trial 24	0.0026	0.0440	0.0026	0.0439	0.0026	0.0440
Trial 25	0.0026	0.0436	0.0026	0.0438	0.0026	0.0436
Trial 26	0.0026	0.0437	0.0026	0.0438	0.0026	0.0437
Trial 27	0.0026	0.0438	0.0026	0.0438	0.0026	0.0438
Trial 28	0.0026	0.0438	0.0026	0.0439	0.0026	0.0438
Trial 29	0.0026	0.0436	0.0026	0.0436	0.0026	0.0436
Trial 30	0.0026	0.0438	0.0026	0.0439	0.0026	0.0438
Average	**0.0026**	**0.0438**	**0.0026**	**0.0438**	**0.0026**	**0.0438**

Table A.3: *Summary of the performance of DTR in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0	0	0	0.0025	0	0.0024
Trial 2	0	0	0	0.0025	0	0.0024
Trial 3	0	0	0	0.0025	0	0.0024
Trial 4	0	0	0	0.0025	0	0.0024
Trial 5	0	0	0	0.0025	0	0.0024
Trial 6	0	0	0	0.0025	0	0.0024
Trial 7	0	0	0	0.0025	0	0.0024
Trial 8	0	0	0	0.0025	0	0.0024
Trial 9	0	0	0	0.0025	0	0.0024
Trial 10	0	0	0	0.0025	0	0.0024
Trial 11	0	0	0	0.0025	0	0.0024
Trial 12	0	0	0	0.0025	0	0.0024
Trial 13	0	0	0	0.0025	0	0.0024
Trial 14	0	0	0	0.0025	0	0.0024
Trial 15	0	0	0	0.0025	0	0.0024
Trial 16	0	0	0	0.0025	0	0.0024
Trial 17	0	0	0	0.0025	0	0.0024
Trial 18	0	0	0	0.0025	0	0.0024
Trial 19	0	0	0	0.0025	0	0.0024
Trial 20	0	0	0	0.0025	0	0.0024
Trial 21	0	0	0	0.0025	0	0.0024
Trial 22	0	0	0	0.0025	0	0.0024
Trial 23	0	0	0	0.0025	0	0.0024
Trial 24	0	0	0	0.0025	0	0.0024
Trial 25	0	0	0	0.0025	0	0.0024
Trial 26	0	0	0	0.0025	0	0.0024
Trial 27	0	0	0	0.0025	0	0.0024
Trial 28	0	0	0	0.0025	0	0.0024
Trial 29	0	0	0	0.0025	0	0.0024
Trial 30	0	0	0	0.0025	0	0.0024
Average	**0**	**0**	**0**	**0.0025**	**0**	**0.0024**

Table A.4: *Summary of the performance of RFR in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0	0.00151	0	0.0015	0	0.0015
Trial 2	0	0.00149	0	0.0015	0	0.0015
Trial 3	0	0.00150	0	0.0015	0	0.0015
Trial 4	0	0.00151	0	0.0015	0	0.0015
Trial 5	0	0.00149	0	0.0016	0	0.0015
Trial 6	0	0.00149	0	0.0016	0	0.0015
Trial 7	0	0.00150	0	0.0015	0	0.0015
Trial 8	0	0.00149	0	0.0016	0	0.0015
Trial 9	0	0.00151	0	0.0016	0	0.0015
Trial 10	0	0.00151	0	0.0015	0	0.0015
Trial 11	0	0.00148	0	0.0015	0	0.0015
Trial 12	0	0.00149	0	0.0015	0	0.0015
Trial 13	0	0.00150	0	0.0015	0	0.0015
Trial 14	0	0.00149	0	0.0016	0	0.0015
Trial 15	0	0.00150	0	0.0015	0	0.0015
Trial 16	0	0.00150	0	0.0016	0	0.0015
Trial 17	0	0.00151	0	0.0016	0	0.0015
Trial 18	0	0.00149	0	0.0015	0	0.0015
Trial 19	0	0.00151	0	0.0015	0	0.0015
Trial 20	0	0.00151	0	0.0015	0	0.0015
Trial 21	0	0.00149	0	0.0015	0	0.0015
Trial 22	0	0.00151	0	0.0015	0	0.0015
Trial 23	0	0.00151	0	0.0015	0	0.0015
Trial 24	0	0.00149	0	0.0015	0	0.0015
Trial 25	0	0.00150	0	0.0016	0	0.0015
Trial 26	0	0.00149	0	0.0015	0	0.0015
Trial 27	0	0.00149	0	0.0016	0	0.0015
Trial 28	0	0.00149	0	0.0016	0	0.0015
Trial 29	0	0.00149	0	0.0015	0	0.0015
Trial 30	0	0.00150	0	0.0015	0	0.0015
Average	**0**	**0.00150**	**0**	**0.0015**	**0**	**0.0015**

Table A.5: *Summary of the performance of XGBoost in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0.0001	0.0072	0.0001	0.0071	0.0001	0.0072
Trial 2	0.0001	0.0070	0.0001	0.0071	0.0001	0.0070
Trial 3	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 4	0.0001	0.0072	0.0001	0.0071	0.0001	0.0072
Trial 5	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 6	0.0001	0.0070	0.0001	0.0070	0.0001	0.0070
Trial 7	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 8	0.0001	0.0070	0.0001	0.0070	0.0001	0.0070
Trial 9	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 10	0.0001	0.0072	0.0001	0.0071	0.0001	0.0072
Trial 11	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 12	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 13	0.0001	0.0070	0.0001	0.0072	0.0001	0.0070
Trial 14	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 15	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 16	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 17	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 18	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 19	0.0001	0.0072	0.0001	0.0071	0.0001	0.0072
Trial 20	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 21	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 22	0.0001	0.0072	0.0001	0.0071	0.0001	0.0072
Trial 23	0.0001	0.0070	0.0001	0.0071	0.0001	0.0070
Trial 24	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 25	0.0001	0.0071	0.0001	0.0072	0.0001	0.0071
Trial 26	0.0001	0.0070	0.0001	0.0070	0.0001	0.0070
Trial 27	0.0001	0.0070	0.0001	0.0071	0.0001	0.0070
Trial 28	0.0001	0.0071	0.0001	0.0071	0.0001	0.0071
Trial 29	0.0001	0.0070	0.0001	0.0071	0.0001	0.0070
Trial 30	0.0001	0.0070	0.0001	0.0071	0.0001	0.0070
Average	**0.0001**	**0.0071**	**0.0001**	**0.0071**	**0.0001**	**0.0071**

Table A.6: *Summary of the performance of the ANN with hyperparameter combination A in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0.052	0.146	0.052	0.146	0.053	0.145
Trial 2	0.052	0.145	0.052	0.145	0.076	0.211
Trial 3	0.054	0.149	0.054	0.149	0.054	0.149
Trial 4	0.054	0.150	0.054	0.150	0.056	0.156
Trial 5	0.052	0.144	0.052	0.144	0.152	0.340
Trial 6	0.052	0.146	0.052	0.146	0.058	0.168
Trial 7	0.053	0.146	0.053	0.146	0.062	0.176
Trial 8	0.054	0.148	0.054	0.148	0.054	0.147
Trial 9	0.053	0.146	0.053	0.146	0.062	0.175
Trial 10	0.055	0.151	0.055	0.151	0.054	0.143
Trial 11	0.055	0.150	0.055	0.150	0.058	0.162
Trial 12	0.055	0.151	0.055	0.151	0.209	0.396
Trial 13	0.054	0.150	0.054	0.150	0.054	0.145
Trial 14	0.055	0.151	0.055	0.151	0.053	0.144
Trial 15	0.054	0.147	0.054	0.147	0.059	0.161
Trial 16	0.054	0.148	0.054	0.148	0.055	0.155
Trial 17	0.054	0.149	0.054	0.149	0.061	0.172
Trial 18	0.052	0.146	0.052	0.146	0.058	0.169
Trial 19	0.053	0.146	0.053	0.146	0.056	0.151
Trial 20	0.052	0.145	0.052	0.145	0.050	0.138
Trial 21	0.053	0.143	0.053	0.143	0.051	0.135
Trial 22	0.054	0.148	0.054	0.148	0.055	0.155
Trial 23	0.053	0.147	0.053	0.147	0.054	0.149
Trial 24	0.054	0.148	0.054	0.148	0.060	0.173
Trial 25	0.054	0.148	0.054	0.148	0.057	0.160
Trial 26	0.052	0.145	0.052	0.145	0.078	0.218
Trial 27	0.049	0.137	0.049	0.137	0.079	0.213
Trial 28	0.052	0.144	0.052	0.144	0.074	0.206
Trial 29	0.051	0.140	0.051	0.140	0.052	0.147
Trial 30	0.053	0.147	0.053	0.147	0.055	0.155
Average	**0.053**	**0.147**	**0.053**	**0.147**	**0.067**	**0.177**

Table A.7: *Summary of the performance of the ANN with hyperparameter combination B in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0.049	0.130	0.049	0.130	0.049	0.129
Trial 2	0.049	0.131	0.049	0.131	0.050	0.131
Trial 3	0.049	0.133	0.049	0.133	0.049	0.133
Trial 4	0.049	0.133	0.049	0.133	0.050	0.134
Trial 5	0.049	0.132	0.049	0.132	0.050	0.132
Trial 6	0.050	0.133	0.050	0.133	0.051	0.133
Trial 7	0.050	0.134	0.050	0.134	0.049	0.134
Trial 8	0.050	0.134	0.050	0.134	0.051	0.135
Trial 9	0.049	0.133	0.049	0.133	0.050	0.133
Trial 10	0.049	0.131	0.049	0.131	0.050	0.133
Trial 11	0.049	0.132	0.049	0.132	0.049	0.132
Trial 12	0.050	0.135	0.050	0.135	0.050	0.135
Trial 13	0.049	0.131	0.049	0.131	0.049	0.130
Trial 14	0.049	0.131	0.049	0.131	0.049	0.131
Trial 15	0.051	0.136	0.051	0.136	0.051	0.136
Trial 16	0.048	0.129	0.048	0.129	0.049	0.130
Trial 17	0.049	0.131	0.049	0.131	0.049	0.132
Trial 18	0.049	0.130	0.049	0.130	0.049	0.131
Trial 19	0.049	0.133	0.049	0.133	0.049	0.131
Trial 20	0.049	0.133	0.049	0.133	0.050	0.133
Trial 21	0.050	0.135	0.050	0.135	0.050	0.135
Trial 22	0.049	0.132	0.049	0.132	0.049	0.131
Trial 23	0.049	0.132	0.049	0.132	0.049	0.132
Trial 24	0.049	0.132	0.049	0.132	0.049	0.132
Trial 25	0.050	0.135	0.050	0.135	0.050	0.134
Trial 26	0.046	0.122	0.046	0.122	0.047	0.123
Trial 27	0.049	0.131	0.049	0.131	0.049	0.131
Trial 28	0.049	0.132	0.049	0.132	0.050	0.132
Trial 29	0.049	0.132	0.049	0.132	0.049	0.131
Trial 30	0.049	0.131	0.049	0.131	0.050	0.131
Average	**0.049**	**0.132**	**0.049**	**0.132**	**0.049**	**0.132**

Table A.8: *Summary of the performance of the ANN with hyperparameter combination C in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0.054	0.148	0.054	0.148	0.053	0.143
Trial 2	0.051	0.139	0.051	0.139	0.050	0.138
Trial 3	0.050	0.137	0.050	0.137	0.049	0.136
Trial 4	0.050	0.134	0.050	0.134	0.050	0.134
Trial 5	0.049	0.133	0.049	0.133	0.049	0.133
Trial 6	0.051	0.138	0.051	0.138	0.052	0.137
Trial 7	0.050	0.137	0.050	0.137	0.049	0.132
Trial 8	0.052	0.142	0.052	0.142	0.053	0.140
Trial 9	0.051	0.139	0.051	0.139	0.053	0.140
Trial 10	0.050	0.135	0.050	0.135	0.049	0.134
Trial 11	0.051	0.140	0.051	0.140	0.053	0.148
Trial 12	0.049	0.134	0.049	0.134	0.049	0.129
Trial 13	0.052	0.142	0.052	0.142	0.051	0.138
Trial 14	0.051	0.139	0.051	0.139	0.050	0.136
Trial 15	0.051	0.140	0.051	0.140	0.050	0.138
Trial 16	0.054	0.148	0.054	0.148	0.055	0.150
Trial 17	0.050	0.136	0.050	0.136	0.050	0.138
Trial 18	0.051	0.137	0.051	0.137	0.051	0.135
Trial 19	0.050	0.135	0.050	0.135	0.049	0.132
Trial 20	0.053	0.143	0.053	0.143	0.053	0.140
Trial 21	0.055	0.150	0.055	0.150	0.054	0.147
Trial 22	0.051	0.139	0.051	0.139	0.050	0.136
Trial 23	0.050	0.137	0.050	0.137	0.051	0.139
Trial 24	0.052	0.141	0.052	0.141	0.050	0.138
Trial 25	0.051	0.138	0.051	0.138	0.050	0.134
Trial 26	0.053	0.145	0.053	0.145	0.054	0.146
Trial 27	0.050	0.135	0.050	0.135	0.049	0.131
Trial 28	0.051	0.139	0.051	0.139	0.050	0.137
Trial 29	0.050	0.136	0.050	0.136	0.048	0.130
Trial 30	0.050	0.134	0.050	0.134	0.051	0.139
Average	**0.051**	**0.139**	**0.051**	**0.139**	**0.051**	**0.138**

Table A.9: *Summary of the performance of the ANN with hyperparameter combination D in terms of MSE and MAE for the 30 trails on the IP dataset.*

	Training set		Validation set		Test set	
	MSE	MAE	MSE	MAE	MSE	MAE
Trial 1	0.060	0.150	0.060	0.150	0.060	0.150
Trial 2	0.060	0.154	0.060	0.154	0.061	0.155
Trial 3	0.060	0.151	0.060	0.151	0.060	0.151
Trial 4	0.060	0.153	0.060	0.153	0.060	0.153
Trial 5	0.061	0.154	0.061	0.154	0.062	0.154
Trial 6	0.059	0.146	0.059	0.146	0.060	0.147
Trial 7	0.062	0.153	0.062	0.153	0.061	0.153
Trial 8	0.061	0.153	0.061	0.153	0.062	0.153
Trial 9	0.059	0.148	0.059	0.148	0.059	0.149
Trial 10	0.062	0.155	0.062	0.155	0.063	0.155
Trial 11	0.060	0.152	0.060	0.152	0.060	0.152
Trial 12	0.057	0.146	0.057	0.146	0.058	0.146
Trial 13	0.059	0.150	0.059	0.150	0.059	0.150
Trial 14	0.061	0.154	0.061	0.154	0.061	0.154
Trial 15	0.060	0.150	0.060	0.150	0.060	0.150
Trial 16	0.059	0.151	0.059	0.151	0.060	0.151
Trial 17	0.060	0.151	0.060	0.151	0.061	0.151
Trial 18	0.058	0.145	0.058	0.145	0.058	0.146
Trial 19	0.058	0.145	0.058	0.145	0.057	0.145
Trial 20	0.059	0.148	0.059	0.148	0.059	0.148
Trial 21	0.060	0.152	0.060	0.152	0.059	0.151
Trial 22	0.058	0.146	0.058	0.146	0.058	0.146
Trial 23	0.062	0.155	0.062	0.155	0.062	0.155
Trial 24	0.059	0.150	0.059	0.150	0.058	0.149
Trial 25	0.060	0.151	0.060	0.151	0.060	0.150
Trial 26	0.060	0.151	0.060	0.151	0.061	0.152
Trial 27	0.062	0.154	0.062	0.154	0.061	0.153
Trial 28	0.060	0.152	0.060	0.152	0.061	0.153
Trial 29	0.060	0.149	0.060	0.149	0.059	0.149
Trial 30	0.060	0.150	0.060	0.150	0.060	0.151
Average	**0.060**	**0.151**	**0.060**	**0.151**	**0.060**	**0.151**

APPENDIX B

Algorithmic Parameter Evaluation Results

This appendix contains additional results obtained during the hyperparameter search space evaluation for the respective algorithms as described in Chapter 6 which focussed on determining good parameter values for the BOCEGAH and BOSAH.

The results documented in this appendix were omitted from the respective sections (in Chapter 6) so as to enhance the understanding of the main text. After determining good parameter values for the respective sub-algorithms (or LLHs) they are employed in the BOCEGAH and the BOSAH, respectively.

B.1 MOOCEM

This section presents the results for the hyperparameter study conducted for the MOOCEM.

B.1.1 Open mine problem

Tables B.1 and B.2 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–40 for hyperparameter combinations A1.1.1–A1.1.9. Figures B.1 and B.2 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A1.1.1–A1.1.9 that correspond to the runs given in Table B.3.

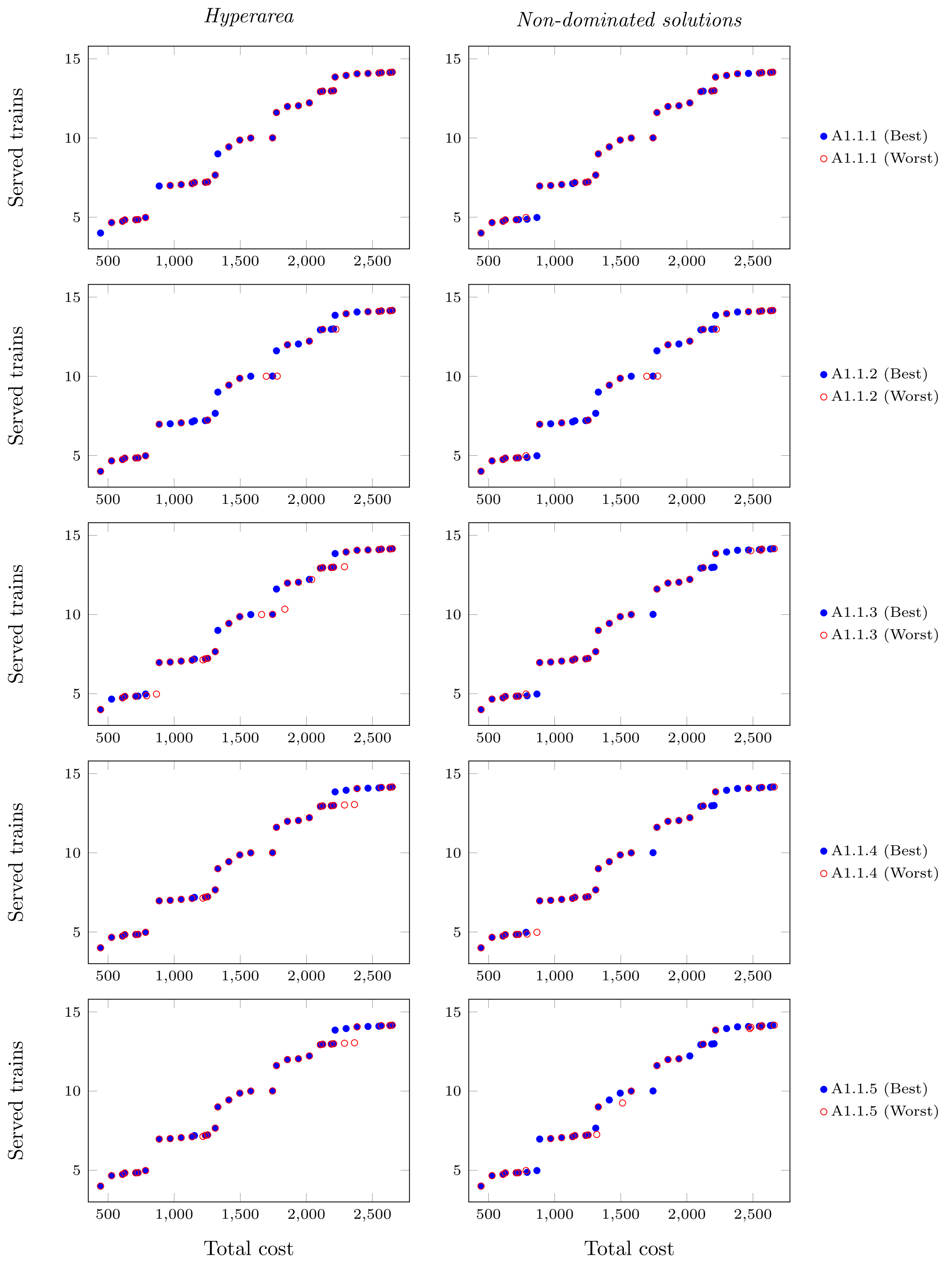


Figure B.1: *The best and worst best approximation fronts for hyperparameter combinations A1.1.1–A1.1.5.*

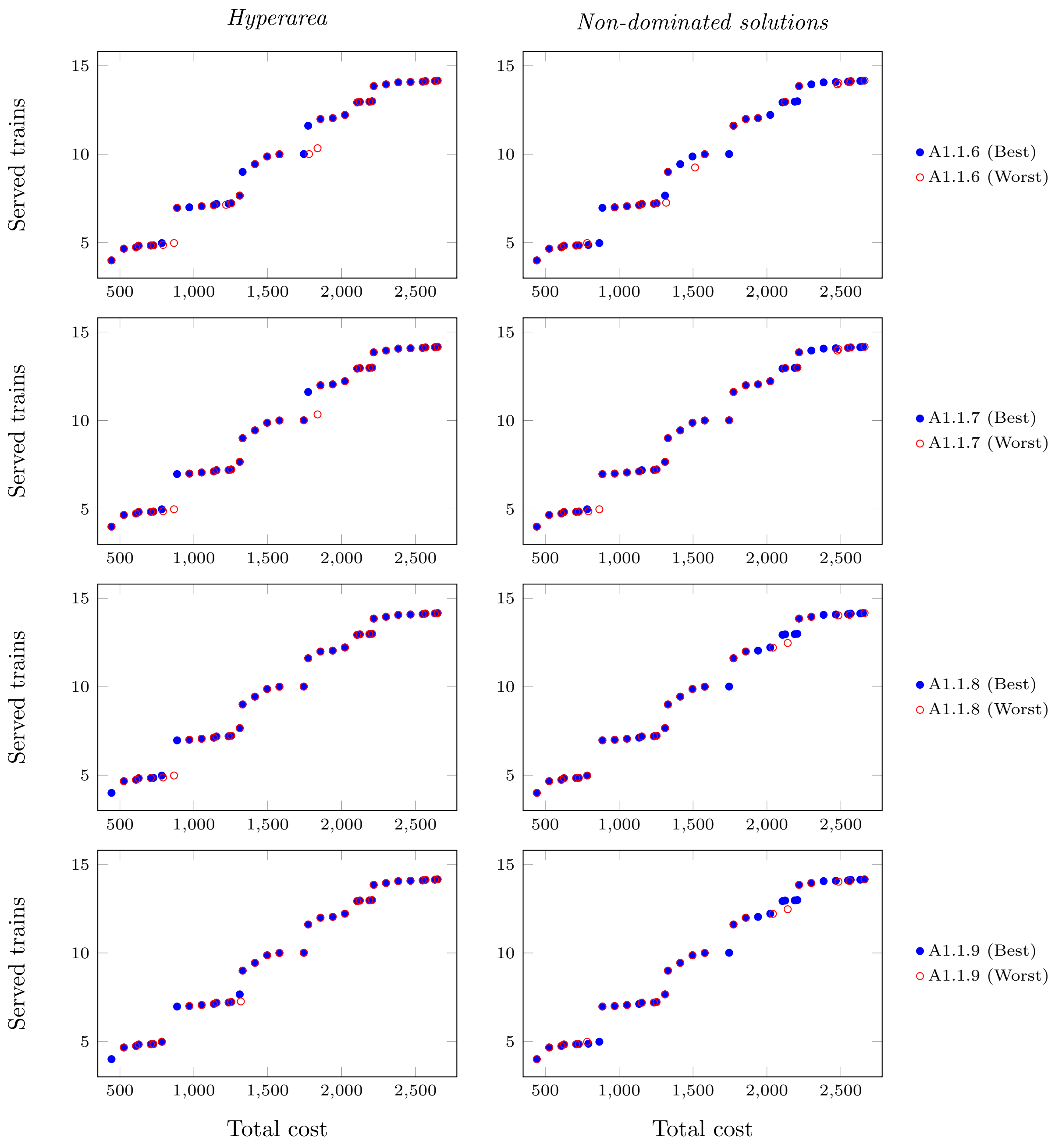


Figure B.2: *The best and worst best approximation fronts for hyperparameter combinations A1.1.6–A1.1.9.*

B.1.2 (s, S) Inventory problem

Tables B.6 and B.7 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A1.2.1–A1.2.9. Figures B.3 and B.4 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A1.2.1–A1.2.9 that correspond to the runs given in Table B.8.

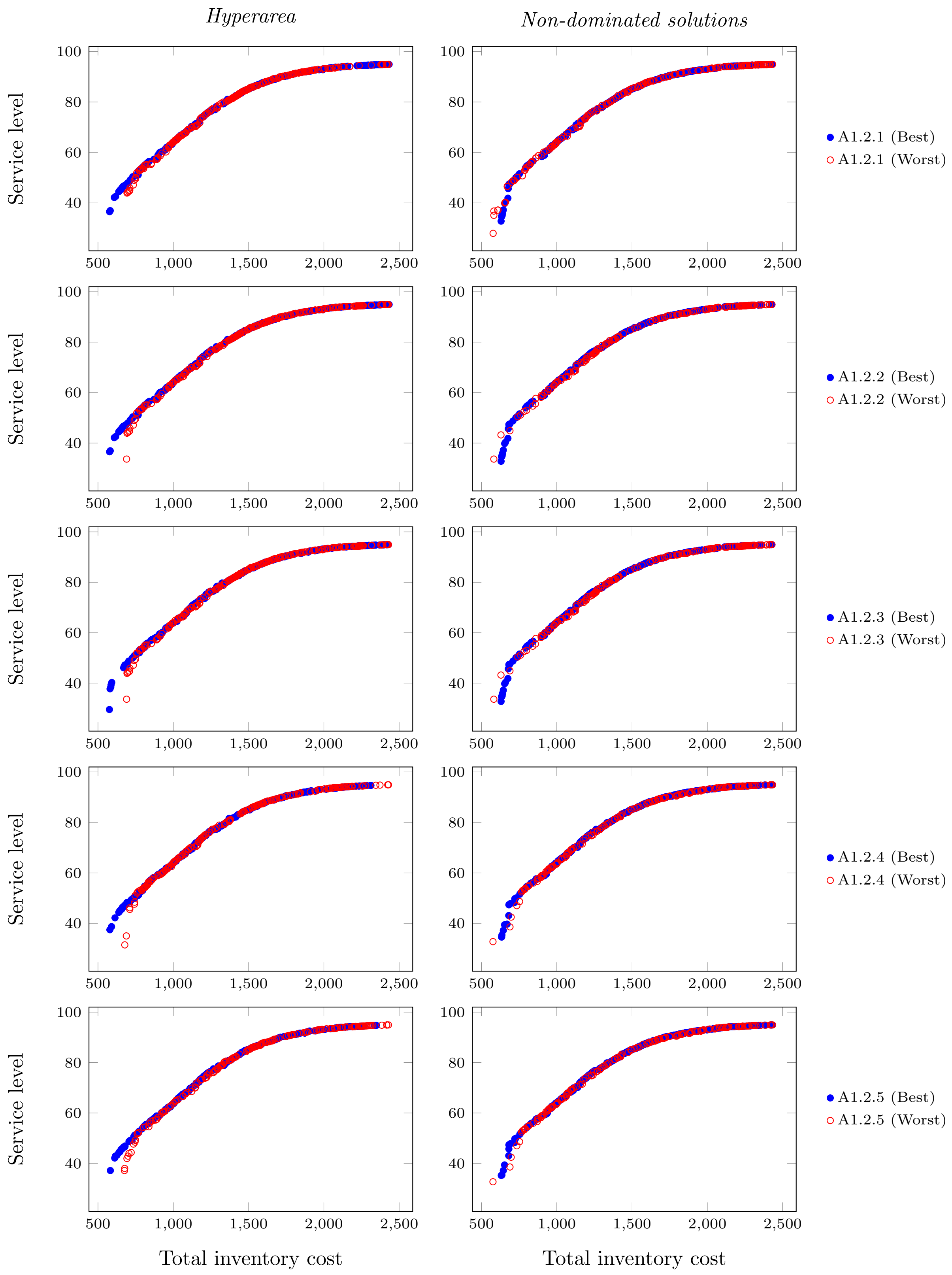


Figure B.3: *The best and worst best approximation fronts for hyperparameter combinations A1.2.1–A1.2.5.*

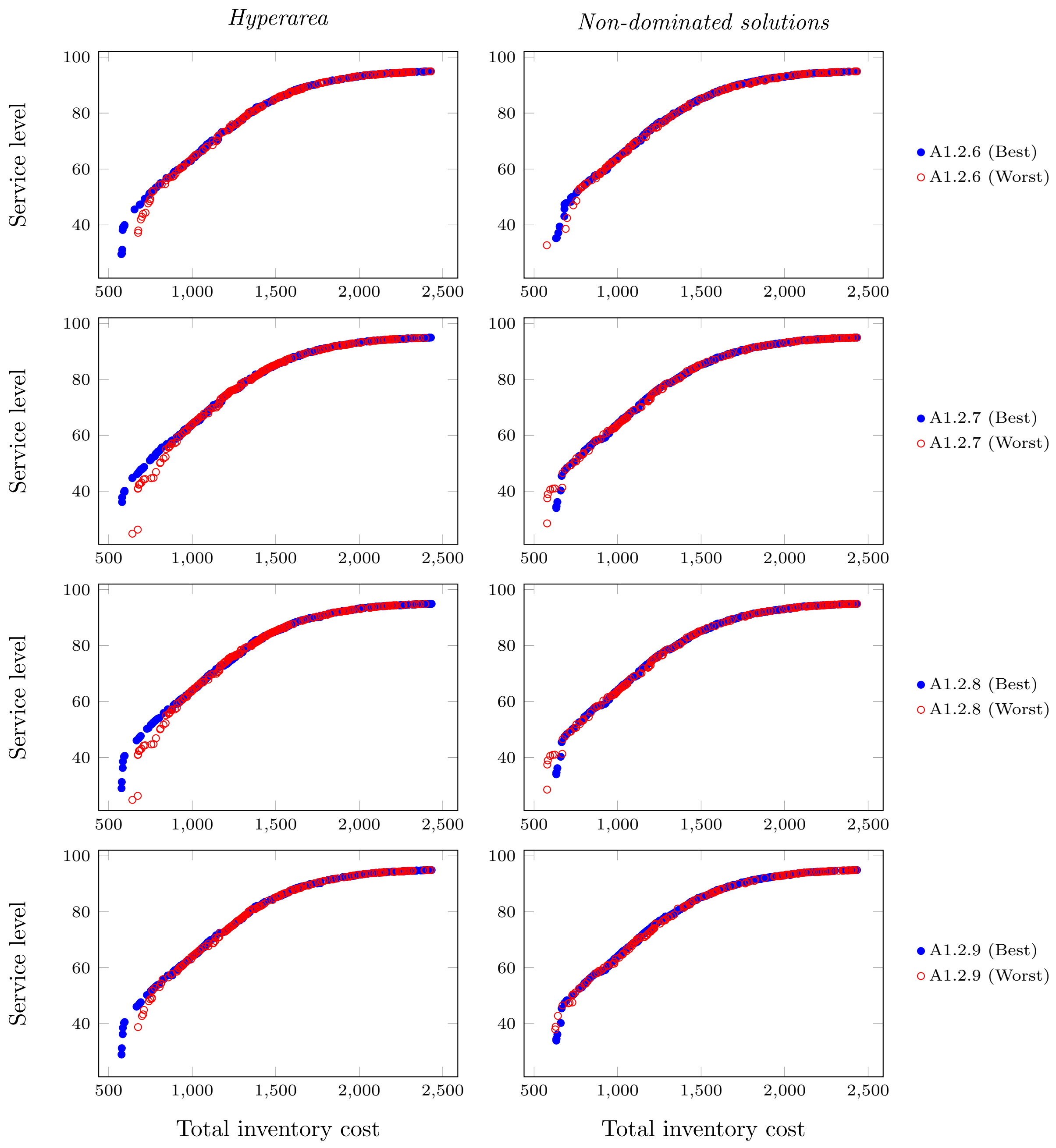


Figure B.4: *The best and worst best approximation fronts for hyperparameter combinations A1.2.6–A1.2.9.*

B.1.3 Buffer allocation problem: five machines

Tables B.9 and B.10 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A1.3.1–A1.3.9. Figures B.5 and B.6 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A1.3.1–A1.3.9 that correspond to the runs given in Table B.11.

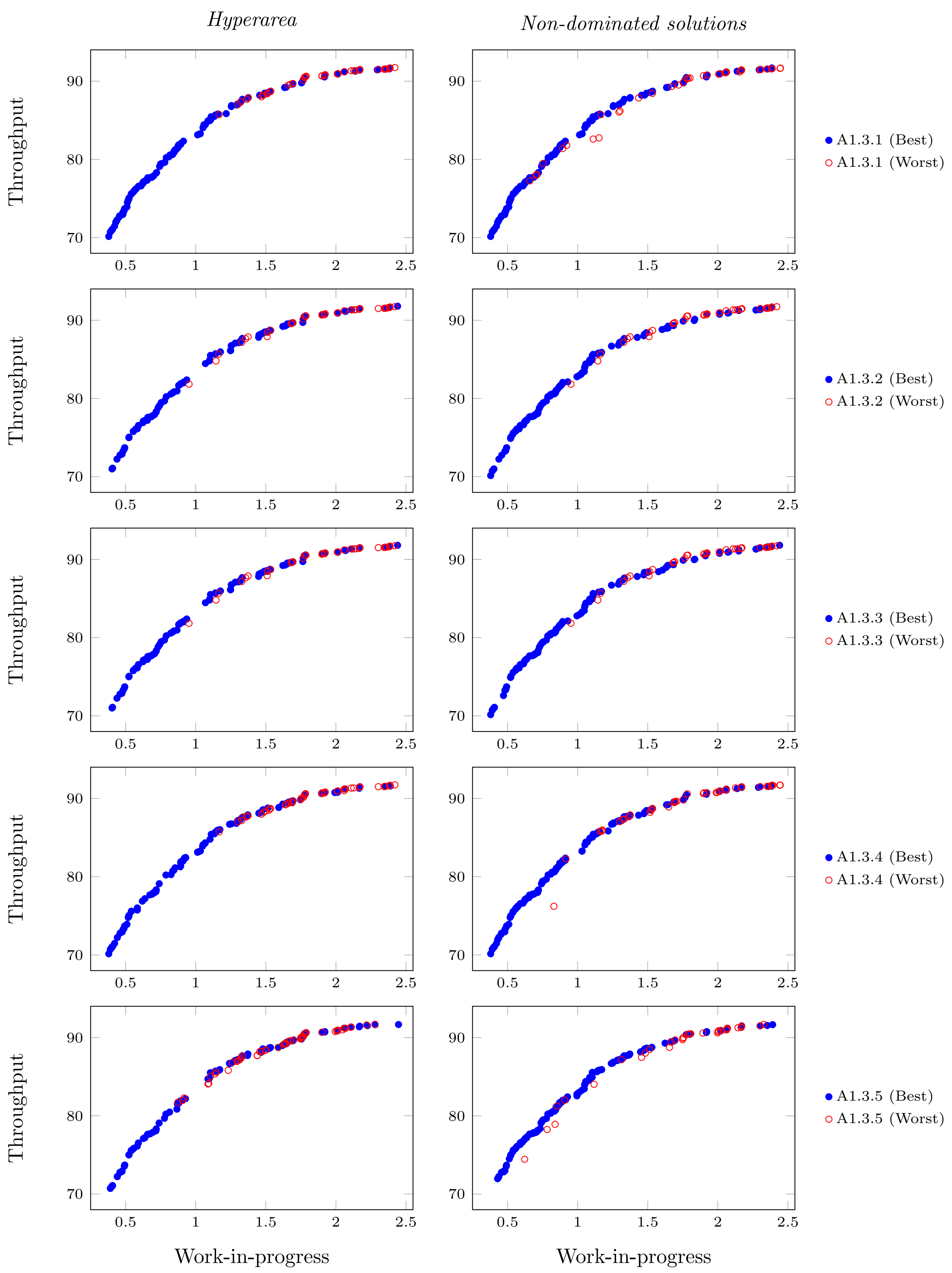


Figure B.5: *The best and worst best approximation fronts for hyperparameter combinations A1.3.1–A1.3.5.*

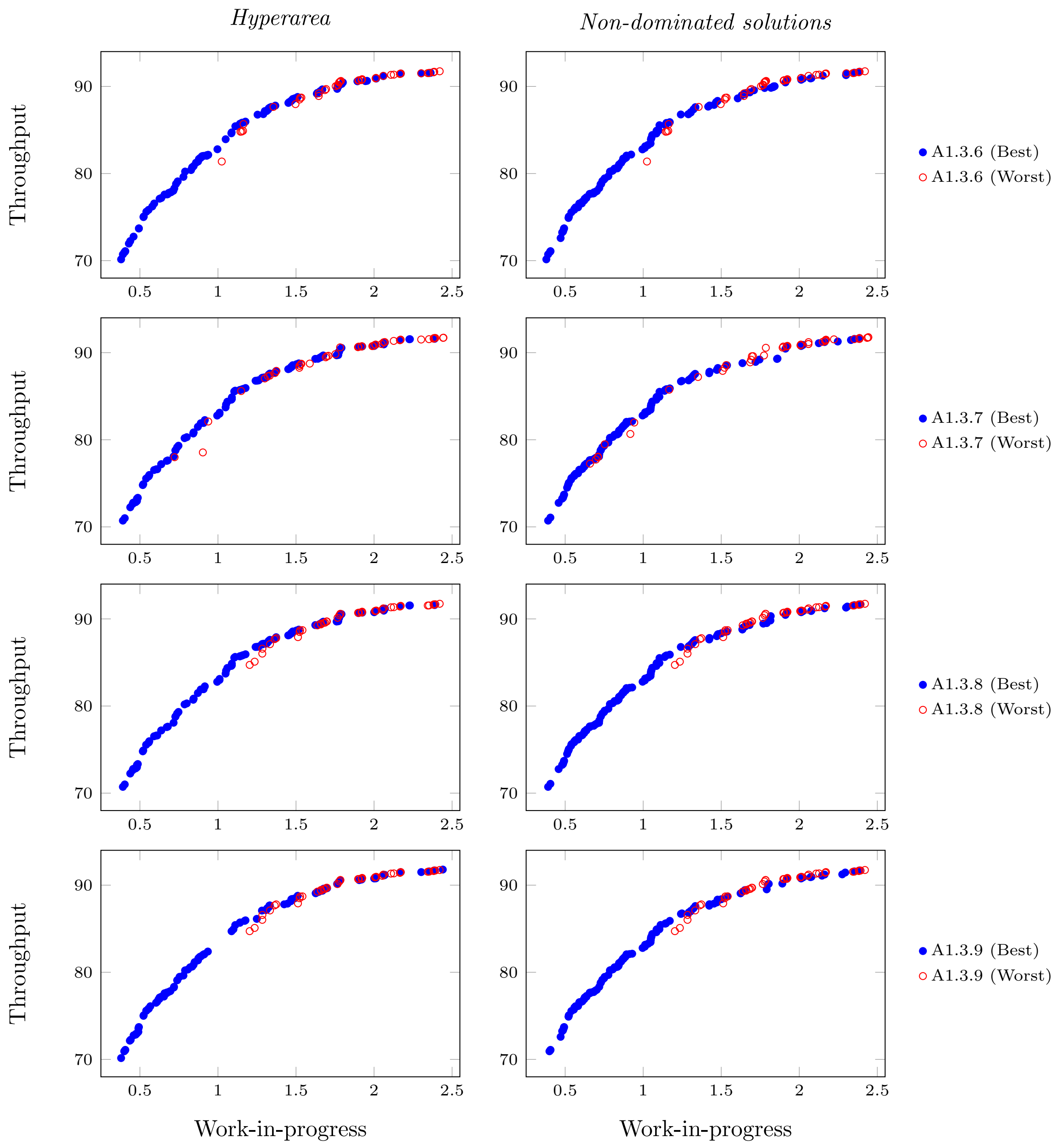


Figure B.6: *The best and worst best approximation fronts for hyperparameter combinations A1.3.6–A1.3.9.*

B.1.4 Buffer allocation problem: 10 machines

Tables B.12 and B.13 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A1.4.1–A1.4.9. Figures B.7 and B.8 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A1.4.1–A1.4.9 that correspond to the runs given in Table B.14.

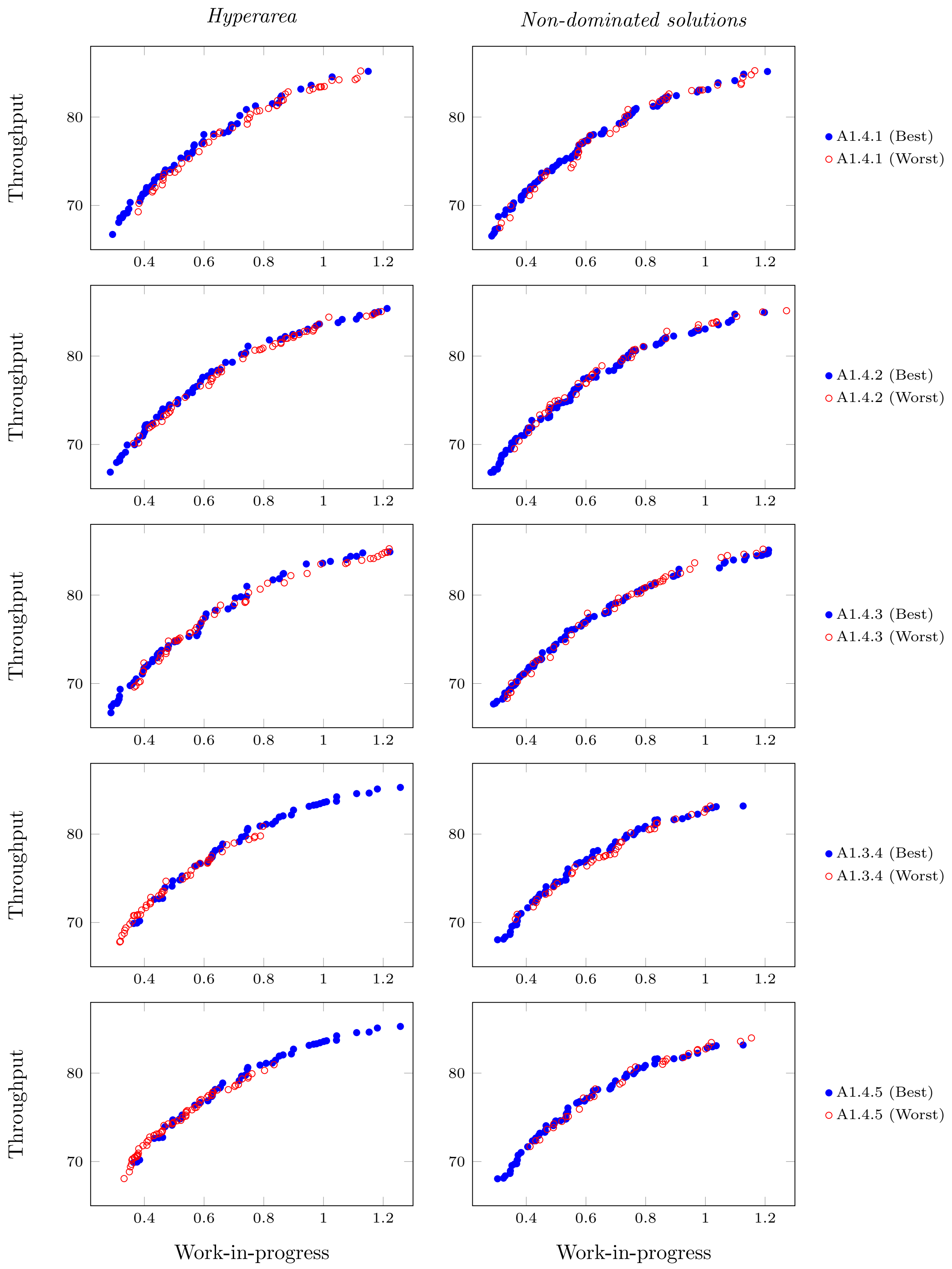


Figure B.7: *The best and worst best approximation fronts for hyperparameter combinations A1.4.1–A1.4.5.*

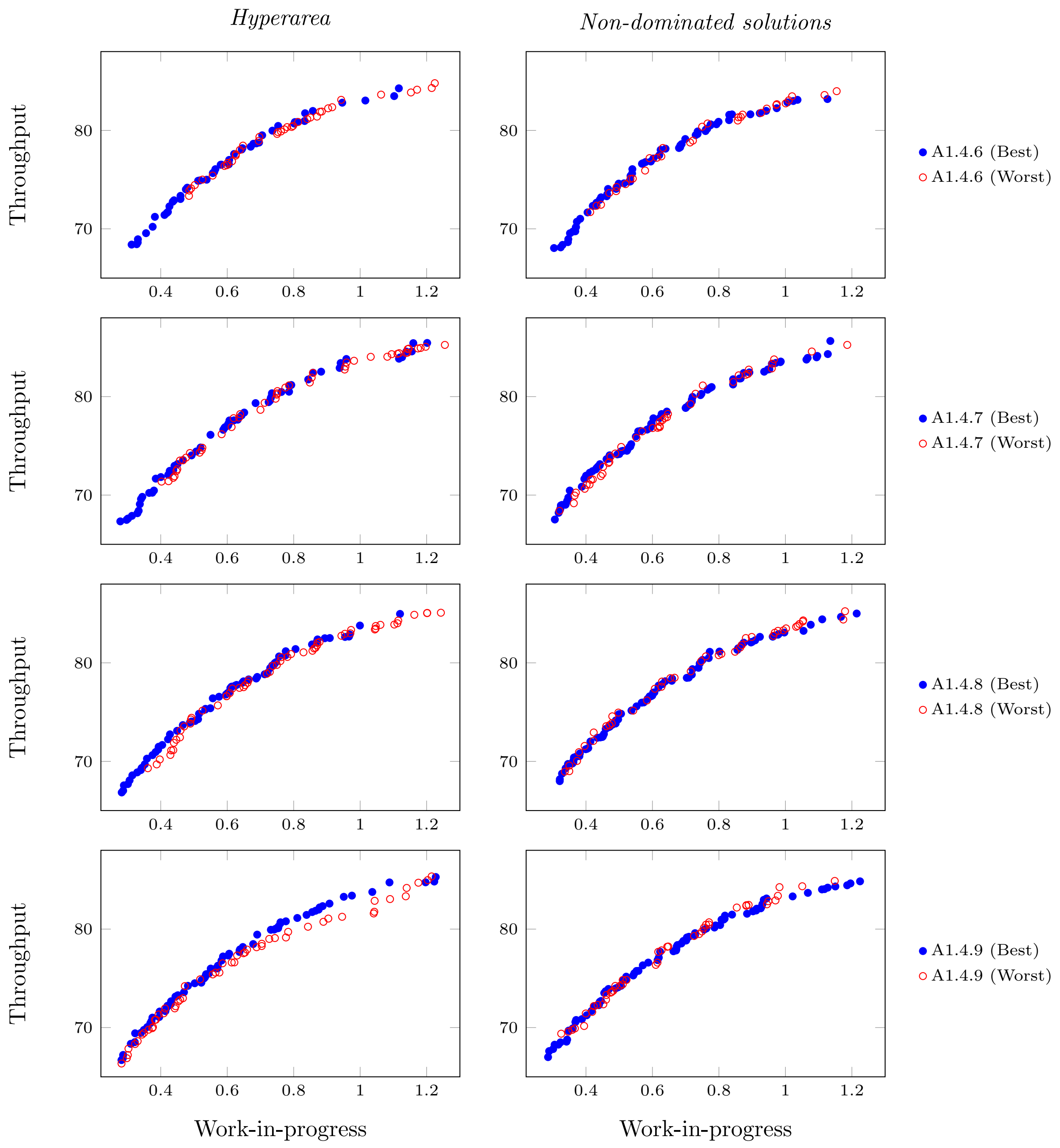


Figure B.8: *The best and worst best approximation fronts for hyperparameter combinations A1.4.6–A1.4.9.*

B.1.5 Non-linear buffer allocation problem: 16 machines

Tables B.15 and B.16 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A1.5.1–A1.5.9. Figures B.9 and B.10 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A1.5.1–A1.5.9 that correspond to the runs given in Table B.17. In Figure B.11 the average number of non-dominated solutions obtained for hyperparameter combinations A1.1–A1.9 is illustrated for the respective simulation problems.

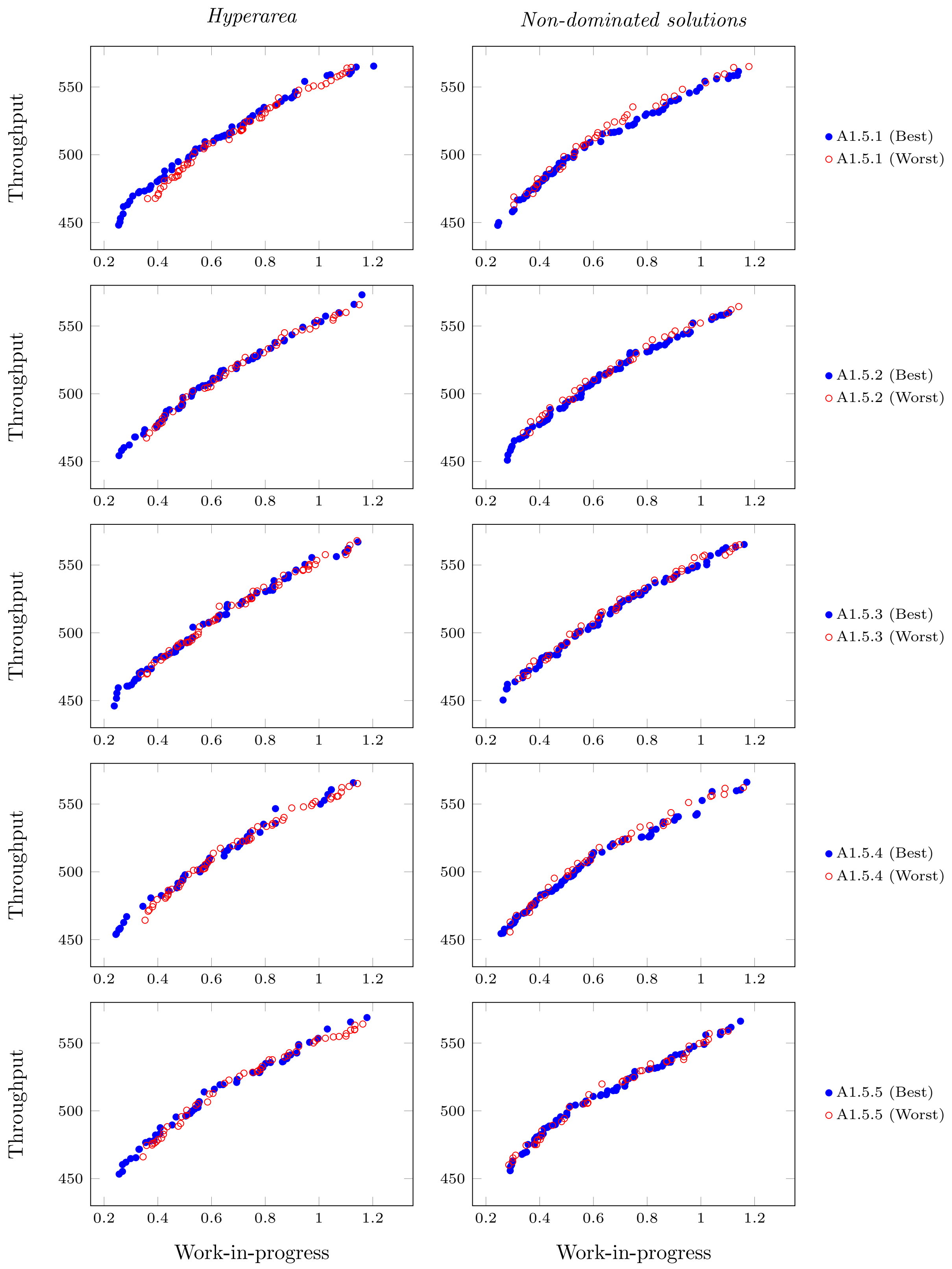


Figure B.9: *The best and worst best approximation fronts for hyperparameter combinations A1.5.1–A1.5.5.*

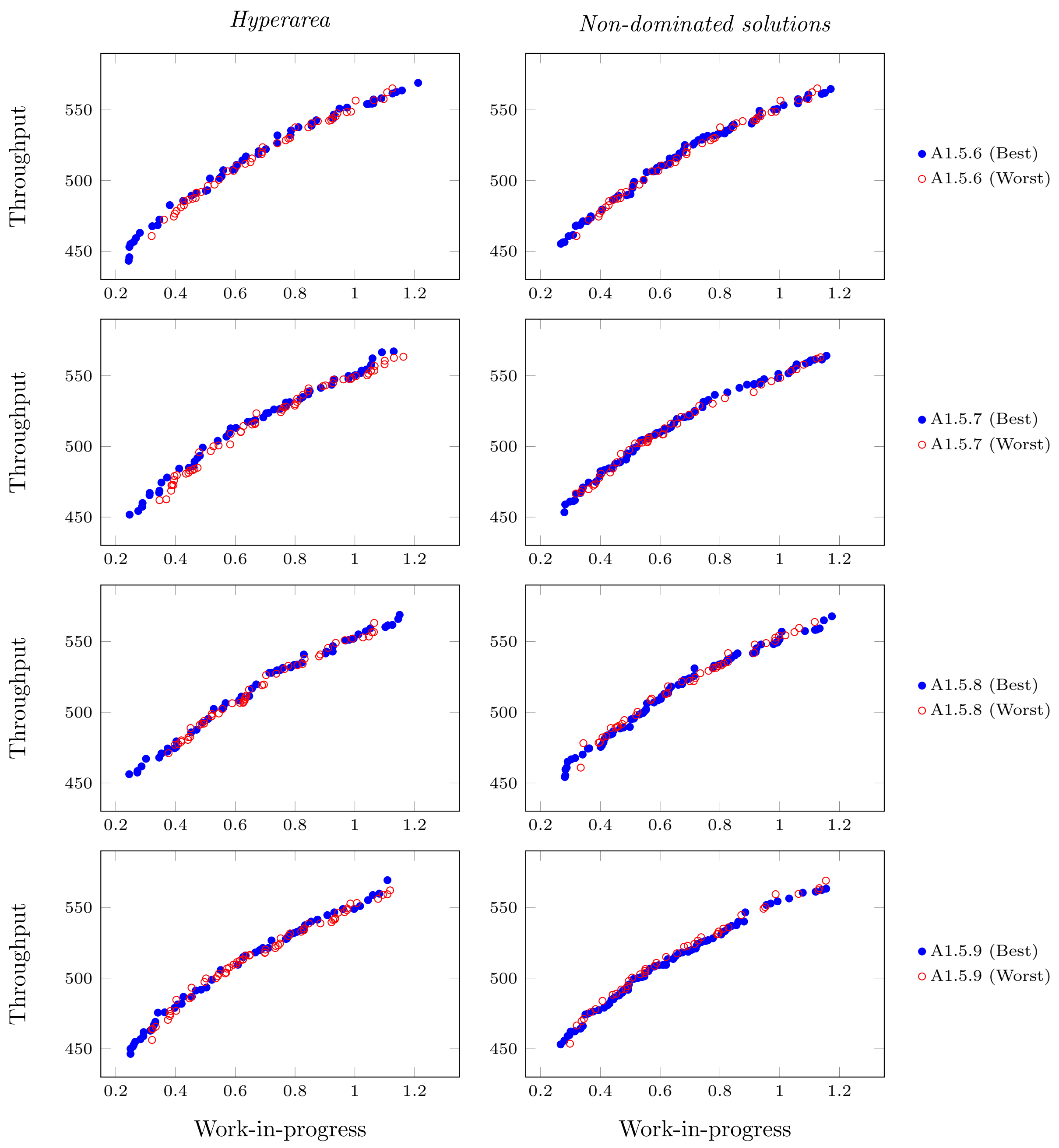


Figure B.10: *The best and worst best approximation fronts for hyperparameter combinations A1.5.6–A1.5.9.*

Table B.1: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–40 for hyperparameter combinations A1.1.1–A1.1.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.1.1	A1.1.2	A1.1.3	A1.1.4	A1.1.5	A1.1.6	A1.1.7	A1.1.8	A1.1.9
1	13 937.94 (3.5)	13 937.94 (3.5)	13 561.84 (6)	13 639.30 (5)	14 485.04 (1)	13 996.08 (2)	13 475.37 (8.5)	13 475.37 (8.5)	13 541.07 (7)
2	14 394.87 (6)	14 485.04 (3)	14 210.87 (7)	14 485.04 (3)	14 485.04 (3)	14 485.04 (3)	14 485.04 (3)	14 082.68 (8)	14 081.71 (9)
3	14 485.04 (2)	14 075.36 (4)	14 061.31 (6)	14 485.04 (2)	13 667.82 (7)	14 069.39 (5)	14 485.04 (2)	12 840.98 (8)	12 048.31 (9)
4	14 485.04 (4.5)	14 485.04 (4.5)	14 484.93 (9)	14 485.04 (4.5)	14 485.04 (4.5)	14 485.04 (4.5)	14 485.04 (4.5)	14 485.04 (4.5)	14 485.04 (4.5)
5	14 476.29 (1)	13 431.18 (6.5)	13 431.18 (6.5)	13 774.89 (3)	13 681.22 (4.5)	13 681.22 (4.5)	14 083.22 (2)	12 585.41 (8.5)	12 585.41 (8.5)
6	14 485.04 (1.5)	14 085.60 (4)	14 085.60 (4)	14 085.60 (4)	13 679.91 (6.5)	13 679.91 (6.5)	14 485.04 (1.5)	12 995.81 (8.5)	12 995.81 (8.5)
7	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)
8	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)
9	13 805.37 (3)	12 067.12 (4.5)	12 067.12 (4.5)	14 083.23 (2)	10 925.83 (8.5)	10 925.83 (8.5)	14 210.87 (1)	10 928.97 (6.5)	10 928.97 (6.5)
10	14 394.87 (8)	14 485.04 (4)	14 485.04 (4)	13 681.50 (9)	14 485.04 (4)	14 485.04 (4)	14 485.04 (4)	14 485.04 (4)	14 485.04 (4)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
39	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)	14 485.04 (5)
40	14485.04 (3.5)	13680.34 (3.5)	14485.04 (8)	14485.04 (3.5)	14485.04 (3.5)	12881.52 (9)	14485.04 (3.5)	14083.51 (7)	14485.04 (3.5)
$\sum R_{Ci}$	176.5	181.5	241	175	186.5	227.5	164.5	216.5	231
$\sum R_{Ci}^2$	31 152.25	32 942.25	58 081	30 625	34 782.25	51 756.25	27 060.25	46 872.25	53 361

Table B.2: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–40 for hyperparameter combinations A1.1.1–A1.1.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.1.1	A1.1.2	A1.1.3	A1.1.4	A1.1.5	A1.1.6	A1.1.7	A1.1.8	A1.1.9
1	35 (3)	35 (3)	34 (7)	34 (7)	36 (1)	35 (3)	34 (7)	34 (7)	34 (7)
2	37 (1)	36 (4)	35 (8)	36 (4)	36 (4)	36 (4)	36 (4)	35 (8)	35 (8)
3	36 (2)	35 (5)	35 (5)	36 (2)	34 (7)	35 (5)	36 (2)	32 (8)	30 (9)
4	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)
5	36 (1)	33 (6.5)	33 (6.5)	34 (4)	34 (4)	34 (4)	35 (2)	31 (8.5)	31 (8.5)
6	36 (1.5)	35 (4)	35 (4)	35 (4)	34 (6.5)	34 (6.5)	36 (1.5)	32 (8.5)	32 (8.5)
7	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)
8	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)
9	34 (3)	30 (4.5)	30 (4.5)	35 (1.5)	27 (7.5)	27 (7.5)	35 (1.5)	27 (7.5)	27 (7.5)
10	37 (1)	36 (5)	36 (5)	34 (9)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
39	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)	36 (5)
40	36 (3.5)	34 (3.5)	36 (8)	36 (3.5)	36 (3.5)	32 (9)	36 (3.5)	35 (7)	36 (3.5)
$\sum R_{Ci}$	160.5	179.5	224	182.5	184.5	228.5	228.5	229.5	235.5
$\sum R_{Ci}^2$	25 760.25	32 220.25	50 176	33 306.25	34 040.25	52 212.25	52 212.25	52 670.25	55 460.25

Table B.3: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the MOOCEM) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A1.1.1–A1.1.9.*

	Worst		Best	
	HA	NDS	HA	NDS
A1.1.1	Run 37	Run 24	Run 3	Run 2
A1.1.2	Run 18	Run 18	Run 2	Run 38
A1.1.3	Run 30	Run 33	Run 7	Run 25
A1.1.4	Run 34	Run 33	Run 2	Run 2
A1.1.5	Run 34	Run 33	Run 1	Run 19
A1.1.6	Run 30	Run 33	Run 2	Run 19
A1.1.7	Run 32	Run 33	Run 2	Run 2
A1.1.8	Run 1	Run 9	Run 4	Run 4
A1.1.9	Run 1	Run 9	Run 4	Run 20

Table B.4: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the adjusted hyperareas of the approximation fronts for the hyperparameter combinations A1.1.1–A1.1.9.*

	A1.1.2	A1.1.3	A1.1.4	A1.1.5	A1.1.6	A1.1.7	A1.1.8	A1.1.9
A1.1.1	1	0.3	1	1	1	1	1	0.94
A1.1.2		0.54	1	1	1	1	1	1
A1.1.3			0.25	0.94	1	0.06	1	1
A1.1.4				1	1	1	1	0.8
A1.1.5					1	1	1	1
A1.1.6						1	1	1
A1.1.7							1	0.24
A1.1.8								1

Table B.5: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A1.1.1–A1.1.9.*

	A1.1.2	A1.1.3	A1.1.4	A1.1.5	A1.1.6	A1.1.7	A1.1.8	A1.1.9
A1.1.1	1	0.34	1	1	0.2	1	0.17	0.08
A1.1.2		1	1	1	1	1	1	0.8
A1.1.3			1	1	1	1	1	1
A1.1.4				1	1	1	1	1
A1.1.5					1	1	1	1
A1.1.6						1	1	1
A1.1.7							0.99	0.52
A1.1.8								1

Table B.6: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A1.2.1–A1.2.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.2.1	A1.2.2	A1.2.3	A1.2.4	A1.2.5	A1.2.6	A1.2.7	A1.2.8	A1.2.9
1	112 268.32 (2)	112 268.32 (2)	112 268.32 (2)	112 225.68 (5)	112 225.68 (5)	112 225.68 (5)	111 814.83 (8)	111 814.83 (8)	111 814.83 (8)
2	113 556.22 (8)	113 556.22 (8)	113 556.22 (8)	113 692.95 (2)	113 692.95 (2)	113 692.95 (2)	113 581.61 (5)	113 581.61 (5)	113 581.61 (5)
3	111 086.70 (6.5)	111 086.70 (6.5)	113 532.90 (3)	111 599.52 (4)	111 559.62 (5)	113 535.94 (2)	110 724.25 (8.5)	110 724.25 (8.5)	113 724.64 (1)
4	112 335.51 (8.5)	112 335.51 (8.5)	112 698.88 (2)	112 375.57 (6.5)	112 375.57 (6.5)	112 613.20 (3)	112 395.19 (4.5)	112 395.19 (4.5)	113 696.57 (1)
5	112 583.02 (8)	112 583.02 (8)	112 583.02 (8)	112 650.31 (5.5)	112 650.51 (4)	112 650.31 (5.5)	112 659.39 (2)	112 659.39 (2)	112 659.39 (2)
6	112 076.32 (1)	111 646.63 (6.5)	111 646.63 (6.5)	111 894.18 (3)	111 663.01 (4.5)	111 663.01 (4.5)	111 996.09 (2)	111 558.46 (8.5)	111 558.46 (8.5)
7	112 408.44 (8)	112 408.44 (8)	112 408.44 (8)	112 460.96 (5)	112 460.96 (5)	112 460.96 (5)	112 463.91 (2)	112 463.91 (2)	112 463.91 (2)
8	113 000.86 (1.5)	113 000.86 (1.5)	112 934.65 (6)	112 974.55 (3.5)	112 974.55 (3.5)	112 962.09 (5)	112 898.83 (8.5)	112 898.83 (8.5)	112 914.42 (7)
9	113 055.46 (2)	112 894.53 (5.5)	112 894.53 (5.5)	112 689.93 (7)	112 569.16 (9)	112 571.87 (8)	113 191.16 (1)	113 034.35 (3.5)	113 034.35 (3.5)
10	112 461.86 (5)	112 461.86 (5)	112 461.86 (5)	113 178.06 (1.5)	113 084.96 (3)	113 178.06 (1.5)	112 395.88 (8)	112 395.88 (8)	112 395.88 (8)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	112 256.71 (2)	112 256.71 (2)	112 256.71 (2)	112 082.21 (8)	112 082.21 (8)	112 082.21 (8)	112 153.73 (5)	112 153.73 (5)	112 153.73 (5)
100	111 929.60 (2)	111 929.60 (2)	111 929.60 (2)	111 808.64 (5)	111 808.64 (5)	111 808.64 (5)	111 508.13 (8)	111 508.13 (8)	111 508.13 (8)
$\sum R_{Ci}$	522.50	525.50	515.00	515.50	481.00	491.50	516.50	481.50	451.00
$\sum R_{Ci}^2$	273 006.25	276 150.25	265 225.00	265 740.25	231 361.00	241 572.25	266 772.25	231 842.25	203 401.00

Table B.7: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A1.2.1–A1.2.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.2.1	A1.2.2	A1.2.3	A1.2.4	A1.2.5	A1.2.6	A1.2.7	A1.2.8	A1.2.9
1	208 (5)	208 (5)	208 (5)	206 (8)	206 (8)	206 (8)	214 (2)	214 (2)	214 (2)
2	254 (3.5)	254 (3.5)	254 (3.5)	241 (8)	241 (8)	241 (8)	254 (3.5)	254 (3.5)	254 (3.5)
3	233 (5.5)	233 (5.5)	242 (3)	223 (9)	231 (7)	225 (8)	242 (3)	242 (3)	250 (1)
4	256 (4.5)	256 (4.5)	235 (9)	272 (1.5)	272 (1.5)	269 (3)	248 (7.5)	248 (7.5)	255 (6)
5	183 (5)	183 (5)	183 (5)	190 (1.5)	189 (3)	190 (1.5)	181 (8)	181 (8)	181 (8)
6	249 (3)	199 (4.5)	199 (4.5)	251 (1)	198 (6.5)	198 (6.5)	250 (2)	190 (8.5)	190 (8.5)
7	233 (8)	233 (8)	233 (8)	240 (2)	240 (2)	240 (2)	238 (5)	238 (5)	238 (5)
8	250 (1.5)	250 (1.5)	209 (3)	204 (5.5)	204 (5.5)	207 (4)	195 (8)	195 (8)	195 (8)
9	221 (3)	191 (6)	191 (6)	230 (1.5)	191 (6)	201 (4)	230 (1.5)	179 (8.5)	179 (8.5)
10	224 (2)	224 (2)	224 (2)	210 (7.5)	193 (9)	210 (7.5)	219 (5)	219 (5)	219 (5)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	273 (5)	273 (5)	273 (5)	262 (8)	262 (8)	262 (8)	287 (2)	287 (2)	287 (2)
100	219 (3.5)	219 (3.5)	219 (3.5)	219 (3.5)	219 (3.5)	219 (3.5)	213 (8)	213 (8)	213 (8)
$\sum R_{Ci}$	381.5	430	516.5	471.5	548	626	440	509	577.5
$\sum R_{Ci}^2$	145 542.25	184 900	266 772.25	222 312.25	300304	391 876	193 600	259 081	333 506.25

Table B.8: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the MOOCEM) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A1.2.1–A1.2.9.*

	Worst		Best	
	HA	NDS	HA	NDS
A1.2.1	Run 94	Run 89	Run 93	Run 79
A1.2.2	Run 94	Run 62	Run 93	Run 79
A1.2.3	Run 94	Run 62	Run 36	Run 79
A1.2.4	Run 30	Run 23	Run 48	Run 79
A1.2.5	Run 72	Run 23	Run 50	Run 79
A1.2.6	Run 72	Run 23	Run 69	Run 79
A1.2.7	Run 72	Run 21	Run 69	Run 79
A1.2.8	Run 3	Run 89	Run 69	Run 79
A1.2.9	Run 3	Run 89	Run 36	Run 79

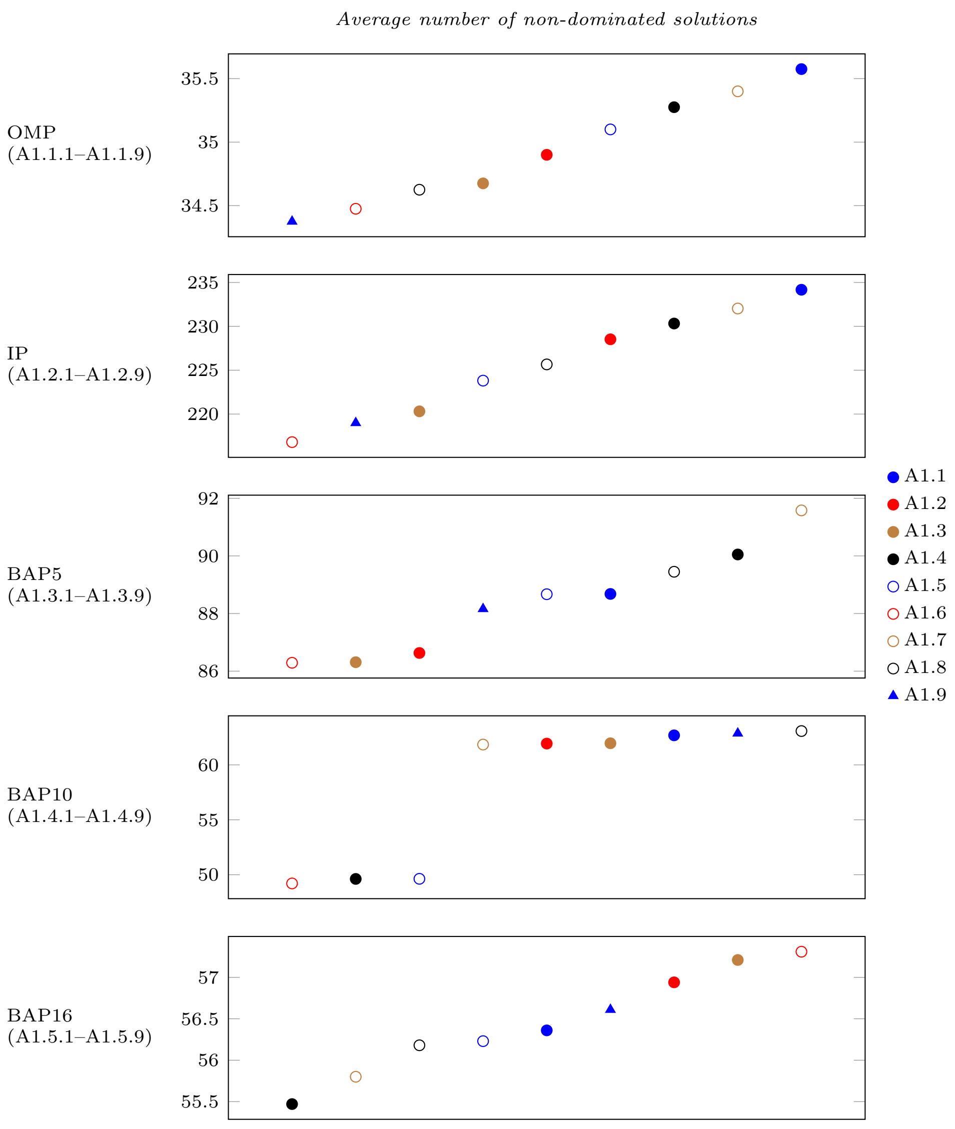


Figure B.11: *The average number of non-dominated solutions obtained for hyperparameter combinations A1.1–A1.9 for the respective simulation problems.*

Table B.9: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A1.3.1–A1.3.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.3.1	A1.3.2	A1.3.3	A1.3.4	A1.3.5	A1.3.6	A1.3.7	A1.3.8	A1.3.9
1	31.78 (9)	32.13 (7)	32.18 (6)	32.30 (3)	32.51 (1)	32.25 (4)	32.04 (8)	32.39 (2)	32.24 (5)
2	33.43 (7)	33.72 (2.5)	33.72 (2.5)	32.80 (9)	33.76 (1)	33.64 (6)	33.38 (8)	33.70 (4.5)	33.70 (4.5)
3	32.37 (3)	32.22 (6)	32.29 (4)	32.21 (8)	33.59 (1)	32.71 (2)	32.22 (7)	32.15 (9)	32.28 (5)
4	33.50 (2.5)	33.50 (2.5)	33.26 (9)	33.42 (6)	33.60 (1)	33.27 (8)	33.47 (4.5)	33.47 (4.5)	33.29 (7)
5	33.42 (7.5)	33.42 (7.5)	33.41 (9)	33.50 (1)	33.45 (5)	33.45 (6)	33.45 (2.5)	33.45 (2.5)	33.45 (4)
6	33.25 (4)	33.34 (2)	33.09 (6)	33.09 (7)	33.33 (3)	32.59 (9)	33.20 (5)	33.37 (1)	32.93 (8)
7	32.22 (5)	32.22 (5)	32.22 (5)	32.40 (2.5)	33.50 (1)	32.40 (2.5)	32.12 (8)	32.12 (8)	32.12 (8)
8	33.57 (6)	33.57 (6)	33.57 (6)	33.56 (8.5)	33.59 (4)	33.56 (8.5)	33.62 (2)	33.62 (2)	33.62 (2)
9	33.01 (6)	33.19 (3)	33.20 (2)	33.02 (5)	33.34 (1)	32.76 (9)	32.92 (8)	32.99 (7)	33.03 (4)
10	33.65 (2)	33.58 (5)	33.62 (3)	33.57 (6)	33.53 (9)	33.67 (1)	33.54 (8)	33.56 (7)	33.62 (4)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	33.02 (3.5)	33.02 (3.5)	32.58 (6)	32.55 (7)	33.46 (1)	33.06 (2)	32.52 (8.5)	32.52 (8.5)	32.99 (5)
100	33.22 (6.5)	33.22 (6.5)	33.29 (5)	33.18 (9)	33.33 (3)	33.30 (4)	33.51 (1.5)	33.51 (1.5)	33.18 (8)
$\sum R_{Ci}$	536.50	525	508.50	555	479	504.00	485	465.50	441.50
$\sum R_{Ci}^2$	287 832.25	275 625	258 572.25	308 025	229 441	254 016	235 225	216 690.25	194 922.25

Table B.10: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A1.3.1–A1.3.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.3.1	A1.3.2	A1.3.3	A1.3.4	A1.3.5	A1.3.6	A1.3.7	A1.3.8	A1.3.9
1	37 (9)	55 (4)	52 (5)	46 (7)	59 (2)	49 (6)	41 (8)	58 (3)	65 (1)
2	73 (8)	105 (5.5)	105 (5.5)	63 (9)	112 (3)	107 (4)	76 (7)	117 (1.5)	117 (1.5)
3	78 (3)	69 (8)	76 (4.5)	71 (7)	111 (1)	64 (9)	81 (2)	73 (6)	76 (4.5)
4	113 (5.5)	113 (5.5)	86 (9)	115 (4)	128 (1)	88 (8)	117 (2.5)	117 (2.5)	94 (7)
5	83 (8.5)	83 (8.5)	105 (1)	95 (3.5)	94 (5)	98 (2)	87 (6.5)	87 (6.5)	95 (3.5)
6	71 (3)	53 (9)	57 (7.5)	72 (2)	69 (4)	57 (7.5)	79 (1)	65 (5)	62 (6)
7	68 (5)	68 (5)	68 (5)	74 (2.5)	92 (1)	74 (2.5)	63 (8)	63 (8)	63 (8)
8	126 (2)	126 (2)	126 (2)	108 (7.5)	79 (9)	108 (7.5)	109 (5)	109 (5)	109 (5)
9	64 (3)	51 (9)	62 (4)	55 (8)	87 (1)	60 (5)	59 (6)	56 (7)	71 (2)
10	107 (5)	106 (6)	94 (9)	111 (3)	120 (1)	104 (7)	113 (2)	108 (4)	99 (8)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	98 (1.5)	98 (1.5)	97 (3)	87 (7)	96 (4)	94 (6)	86 (8.5)	86 (8.5)	95 (5)
100	92 (4.5)	92 (4.5)	71 (8)	106 (1)	70 (9)	80 (6)	93 (2.5)	93 (2.5)	77 (7)
$\sum R_{Ci}$	508	566	568.5	459	515.5	548	404.5	450	480.5
$\sum R_{Ci}^2$	258 064	320 356	323 192.25	210 681	265 740	300 304	163 620	202 500	230 880.25

Table B.11: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the MOOCEM) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A1.3.1–A1.3.9.*

	Worst		Best	
	HA	NDS	HA	NDS
A1.3.1	Run 75	Run 1	Run 50	Run 50
A1.3.2	Run 75	Run 75	Run 2	Run 18
A1.3.3	Run 75	Run 75	Run 2	Run 18
A1.3.4	Run 75	Run 27	Run 92	Run 50
A1.3.5	Run 79	Run 98	Run 2	Run 58
A1.3.6	Run 75	Run 75	Run 10	Run 18
A1.3.7	Run 27	Run 1	Run 60	Run 18
A1.3.8	Run 75	Run 75	Run 60	Run 18
A1.3.9	Run 75	Run 75	Run 2	Run 18

Table B.12: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A1.4.1–A1.4.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.4.1	A1.4.2	A1.4.3	A1.4.4	A1.4.5	A1.4.6	A1.4.7	A1.4.8	A1.4.9
1	22.10 (2)	21.85 (6)	21.92 (4)	20.24 (8)	20.24 (8)	20.24 (8)	21.86 (5)	22.16 (1)	22.03 (3)
2	21.96 (3)	21.96 (4)	22.08 (1)	21.33 (8)	21.33 (8)	21.33 (8)	21.99 (2)	21.93 (5)	21.91 (6)
3	21.29 (5)	21.62 (3)	21.72 (2)	21.13 (7)	21.13 (7)	21.13 (7)	21.05 (9)	21.42 (4)	22.07 (1)
4	21.98 (3)	21.96 (4)	21.65 (5)	20.68 (8)	20.68 (8)	20.68 (8)	22.10 (1)	22.07 (2)	21.60 (6)
5	22.14 (2)	21.78 (6)	22.15 (1)	21.20 (8)	21.20 (8)	21.20 (8)	22.05 (3)	21.82 (5)	22.03 (4)
6	21.36 (6)	21.98 (4)	22.08 (2)	21.09 (8)	21.09 (8)	21.09 (8)	21.53 (5)	22.17 (1)	22.07 (3)
7	21.71 (4)	21.73 (2)	21.66 (5)	20.94 (8)	20.94 (8)	20.94 (8)	21.57 (6)	21.88 (1)	21.71 (3)
8	21.92 (4)	21.91 (5)	21.93 (3)	21.22 (8)	21.22 (8)	21.22 (8)	21.86 (6)	22.05 (2)	22.14 (1)
9	22.05 (1)	21.99 (3)	21.89 (5)	20.64 (8)	20.64 (8)	20.64 (8)	21.99 (2)	21.89 (4)	21.83 (6)
10	21.72 (6)	22.06 (1)	21.87 (5)	21.03 (8)	21.03 (8)	21.03 (8)	21.95 (2)	21.93 (4)	21.93 (3)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	21.89 (3)	21.73 (4)	22.03 (2)	20.89 (8.5)	20.89 (8.5)	21.37 (7)	21.72 (5)	21.71 (6)	22.07 (1)
100	22.03 (4)	22.15 (1)	21.90 (6)	20.83 (7.5)	20.83 (7.5)	20.80 (9)	22.12 (3)	22.15 (2)	22.00 (5)
$\sum R_{Ci}$	371	359	347	799.5	789	799.5	350	373	312
$\sum R_{Ci}^2$	137 641	128 881	120 409	639 200.25	622 521	639 200.25	122 500	139 129	97 344

Table B.13: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A1.4.1–A1.4.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.4.1	A1.4.2	A1.4.3	A1.4.4	A1.4.5	A1.4.6	A1.4.7	A1.4.8	A1.4.9
1	61 (4)	50 (6)	57 (5)	42 (8)	42 (8)	42 (8)	63 (3)	69 (2)	70 (1)
2	64 (2.5)	60 (5.5)	63 (4)	46 (8)	46 (8)	46 (8)	64 (2.5)	65 (1)	60 (5.5)
3	60 (1)	59 (2.5)	59 (2.5)	51 (7)	51 (7)	51 (7)	58 (4)	56 (5)	47 (9)
4	66 (3)	71 (1)	63 (4)	53 (7.5)	53 (7.5)	53 (7.5)	67 (2)	53 (7.5)	57 (5)
5	59 (6)	69 (1)	61 (4)	49 (8)	49 (8)	49 (8)	68 (2.5)	60 (5)	68 (2.5)
6	64 (4.5)	82 (1)	74 (2)	40 (8)	40 (8)	40 (8)	52 (6)	64 (4.5)	71 (3)
7	50 (4)	55 (3)	49 (6.5)	49 (6.5)	49 (6.5)	49 (6.5)	60 (1)	48 (9)	56 (2)
8	80 (2)	68 (6)	84 (1)	67 (8)	67 (8)	67 (8)	75 (4)	75 (4)	75 (4)
9	64 (3)	65 (1.5)	54 (6)	39 (8)	39 (8)	39 (8)	62 (4)	65 (1.5)	59 (5)
10	63 (4.5)	66 (3)	56 (6)	47 (8)	47 (8)	47 (8)	68 (1)	67 (2)	63 (4.5)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	61 (4)	58 (5)	72 (2)	57 (7)	57 (7)	57 (7)	69 (3)	56 (9)	73 (1)
100	56 (6)	58 (5)	65 (2)	46 (8.5)	46 (8.5)	50 (7)	60 (3.5)	60 (3.5)	67 (1)
$\sum R_{Ci}$	368	384.5	393.5	744.5	735	762.5	386	365	361
$\sum R_{Ci}^2$	135 424	147 840.25	154 842.25	554 280.25	540 225	581 406.25	148 996	133 225	130 321

Table B.14: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the MOOCEM) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A1.4.1–A1.4.9.*

	Worst		Best	
	HA	NDS	HA	NDS
A1.4.1	Run 75	Run 55	Run 37	Run 92
A1.4.2	Run 75	Run 76	Run 37	Run 49
A1.4.3	Run 22	Run 91	Run 35	Run 8
A1.4.4	Run 89	Run 35	Run 75	Run 8
A1.4.5	Run 65	Run 44	Run 75	Run 8
A1.4.6	Run 88	Run 44	Run 60	Run 8
A1.4.7	Run 75	Run 82	Run 88	Run 37
A1.4.8	Run 32	Run 58	Run 62	Run 40
A1.4.9	Run 47	Run 58	Run 95	Run 90

Table B.15: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A1.5.1–A1.5.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.5.1	A1.5.2	A1.5.3	A1.5.4	A1.5.5	A1.5.6	A1.5.7	A1.5.8	A1.5.9
1	203.81 (4)	206.29 (3)	206.33 (2)	206.68 (1)	202.53 (7)	203.7 (5)	202.38 (8)	203.29 (6)	201.28 (9)
2	202.7 (5)	204.25 (1)	203.57 (3)	203.63 (2)	201.16 (7)	200.09 (9)	203.38 (4)	201.25 (6)	200.59 (8)
3	202.19 (6)	203.39 (4)	201.42 (8)	202.55 (5)	200.46 (9)	204.71 (1)	201.71 (7)	204.07 (3)	204.47 (2)
4	200.24 (8)	201.65 (4)	195.51 (9)	201.5 (6)	204.48 (1)	200.68 (7)	201.52 (5)	201.65 (3)	204.3 (2)
5	194.67 (9)	201.67 (4)	200.73 (5)	195.99 (8)	202.68 (3)	203.63 (2)	199.85 (7)	200.47 (6)	204.46 (1)
6	200.89 (7)	202.62 (1)	201.86 (2)	196.34 (9)	201.84 (3)	201.78 (4)	197.98 (8)	201.08 (6)	201.31 (5)
7	203.06 (3)	205.08 (1)	202.76 (4)	201.71 (5)	201.08 (9)	201.69 (6)	204.74 (2)	201.14 (7)	201.09 (8)
8	199.21 (9)	201.94 (6)	201.72 (7)	204.56 (1)	201.37 (8)	203.18 (2)	202.27 (4)	202.98 (3)	202.1 (5)
9	196.32 (7)	199.6 (4)	202.13 (1)	193.71 (8)	201.15 (2)	199.49 (5)	191.98 (9)	199.06 (6)	200.25 (3)
10	203.4 (8)	204.09 (4)	204.1 (3)	204.44 (2)	202.61 (9)	203.96 (6)	205.27 (1)	204.06 (5)	203.87 (7)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	200.86 (9)	202.16 (7)	203.53 (2)	202.4 (6)	202.65 (4)	201.5 (8)	202.52 (5)	205.8 (1)	203.28 (3)
100	202.61 (6)	202.26 (9)	203.17 (3)	202.77 (5)	202.47 (7)	202.94 (4)	202.32 (8)	203.8 (2)	205.31 (1)
$\sum R_{Ci}$	585	505	426	563	527	415	533	518	428
$\sum R_{Ci}^2$	342 225	255 025	181 476	316 969	277 729	172 225	284 089	268 324	183 184

Table B.16: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A1.5.1–A1.5.9. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A1.5.1	A1.5.2	A1.5.3	A1.5.4	A1.5.5	A1.5.6	A1.5.7	A1.5.8	A1.5.9
1	63 (3)	58 (4)	50 (8)	44 (9)	64 (2)	70 (1)	53 (7)	54 (6)	56 (5)
2	65 (1)	61 (4)	63 (2)	50 (9)	54 (7)	57 (6)	62 (3)	58 (5)	53 (8)
3	58 (8)	60 (6.5)	63 (2.5)	54 (9)	63 (2.5)	68 (1)	60 (6.5)	62 (4.5)	62 (4.5)
4	50 (8)	57 (3.5)	54 (6)	49 (9)	52 (7)	55 (5)	69 (1)	64 (2)	57 (3.5)
5	61 (4)	66 (1)	48 (9)	64 (2)	51 (6)	62 (3)	49 (8)	60 (5)	50 (7)
6	55 (4.5)	62 (1)	51 (7.5)	43 (9)	61 (2)	55 (4.5)	54 (6)	59 (3)	51 (7.5)
7	46 (9)	60 (2)	48 (7)	52 (4)	49 (6)	51 (5)	54 (3)	47 (8)	65 (1)
8	51 (7)	66 (1)	63 (2)	59 (4)	48 (8)	56 (5)	46 (9)	52 (6)	61 (3)
9	56 (5.5)	58 (3)	63 (1)	56 (5.5)	54 (8.5)	60 (2)	57 (4)	55 (7)	54 (8.5)
10	55 (8)	60 (6)	62 (4.5)	63 (3)	66 (1.5)	66 (1.5)	55 (8)	62 (4.5)	55 (8)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	64 (2)	57 (6.5)	47 (9)	61 (4)	57 (6.5)	66 (1)	62 (3)	50 (8)	59 (5)
100	59 (4.5)	61 (3)	57 (6.5)	52 (9)	57 (6.5)	62 (2)	66 (1)	54 (8)	59 (4.5)
$\sum R_{Ci}$	511.5	479	467	555.5	510	460	532.5	499	485.5
$\sum R_{Ci}^2$	261 632	229 441	218 089	308 580	260100	211 600	283 556.25	249 001	235 710.25

Table B.17: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the MOOCEM) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A1.5.1–A1.5.9.*

	Worst		Best	
	HA	NDS	HA	NDS
A1.5.1	Run 75	Run 24	Run 19	Run 89
A1.5.2	Run 75	Run 83	Run 19	Run 29
A1.5.3	Run 63	Run 60	Run 43	Run 57
A1.5.4	Run 71	Run 56	Run 1	Run 50
A1.5.5	Run 83	Run 29	Run 49	Run 39
A1.5.6	Run 75	Run 75	Run 68	Run 77
A1.5.7	Run 9	Run 93	Run 35	Run 4
A1.5.8	Run 75	Run 28	Run 32	Run 77
A1.5.9	Run 93	Run 52	Run 72	Run 35

B.2 NSGA-II

This section presents the results for the hyperparameter study conducted for the NSGA-II.

B.2.1 Open mine problem

Tables B.18 and B.19 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–40 for hyperparameter combinations A2.1.1–A2.1.24. Figures B.12 and B.14 combines the approximation fronts for which the runs where found to be the same, whereas Figure B.13 illustrate the worst and best approximation fronts found (in terms of hyperareas and number non-dominated solutions), for the respective hyperparameter combinations A2.1.1–A2.1.24, all corresponding to the runs given in Table B.20.

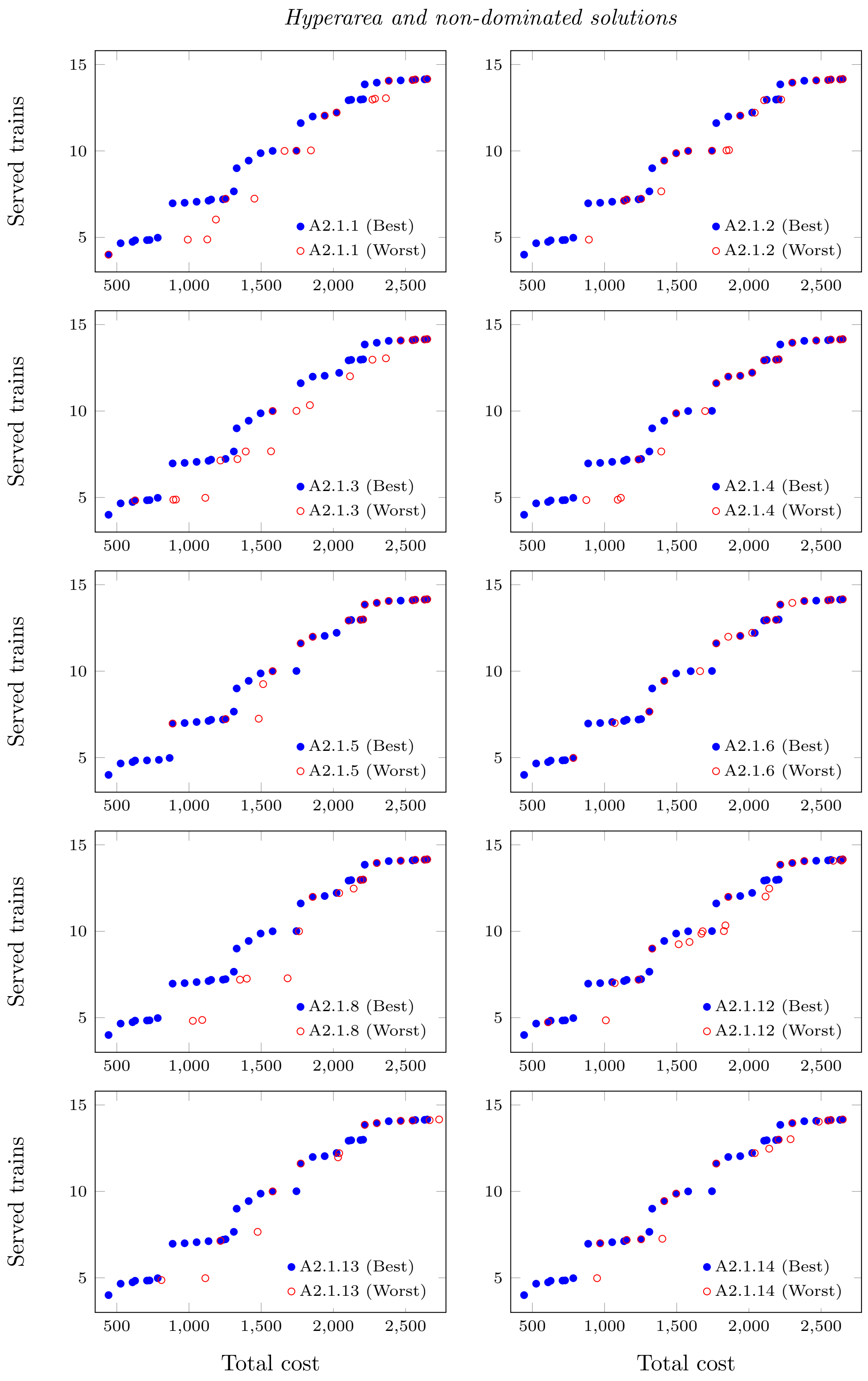


Figure B.12: *The best and worst best approximation fronts for hyperparameter combinations A2.1.1–A2.1.6, A2.1.8, A2.1.12–A2.1.14.*

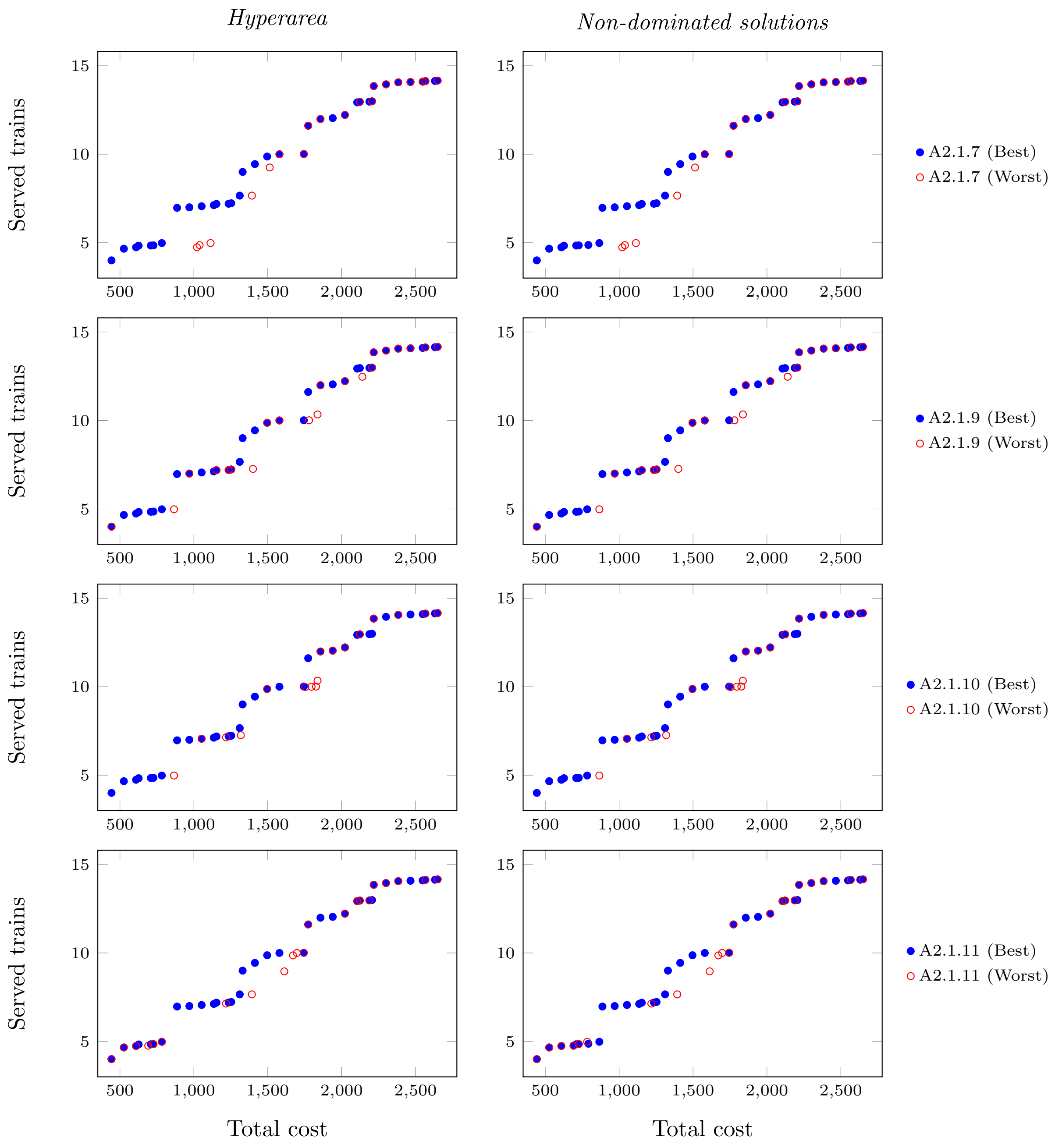


Figure B.13: *The best and worst best approximation fronts for hyperparameter combinations A2.1.7, A2.1.9–A2.1.11.*

B.2.2 (s, S) Inventory problem

Tables B.23 and B.24 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A2.2.1–A2.2.24. Figures B.15–B.19 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A2.2.1–A2.2.24 that correspond to the runs given in Table B.25.

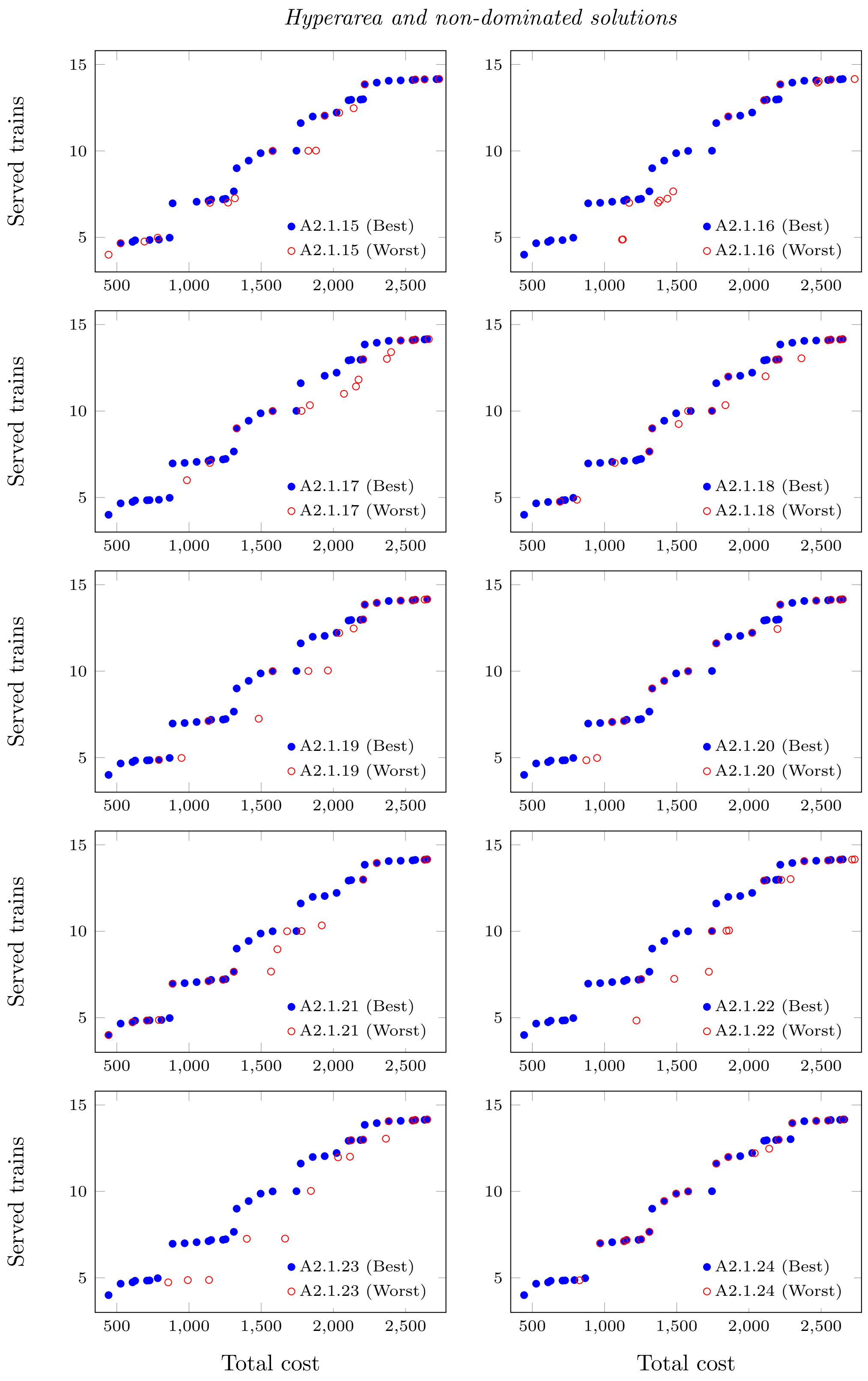


Figure B.14: *The best and worst best approximation fronts for hyperparameter combinations A2.2.16–A2.2.24.*

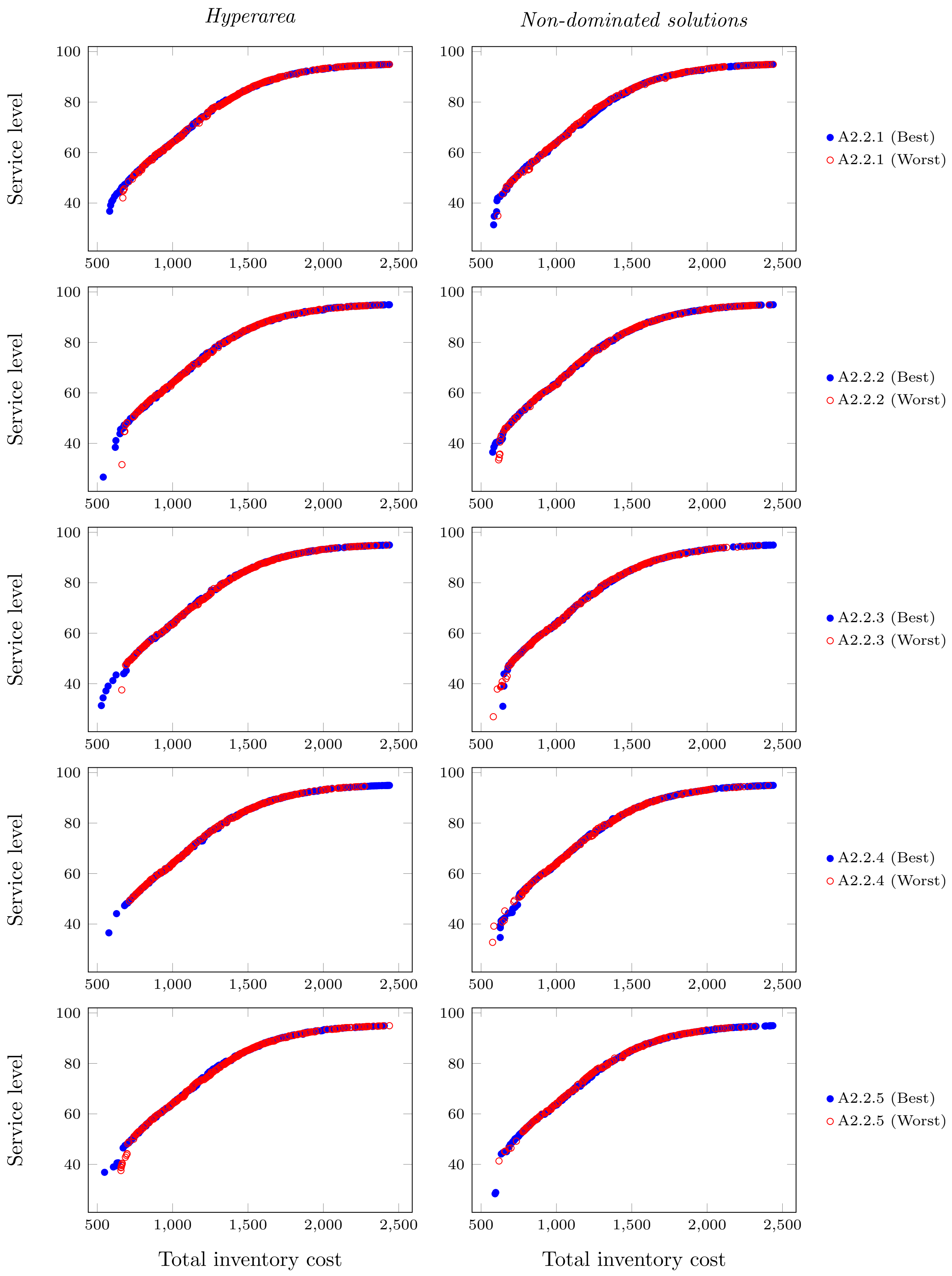


Figure B.15: *The best and worst best approximation fronts for hyperparameter combinations A2.2.1–A2.2.5.*

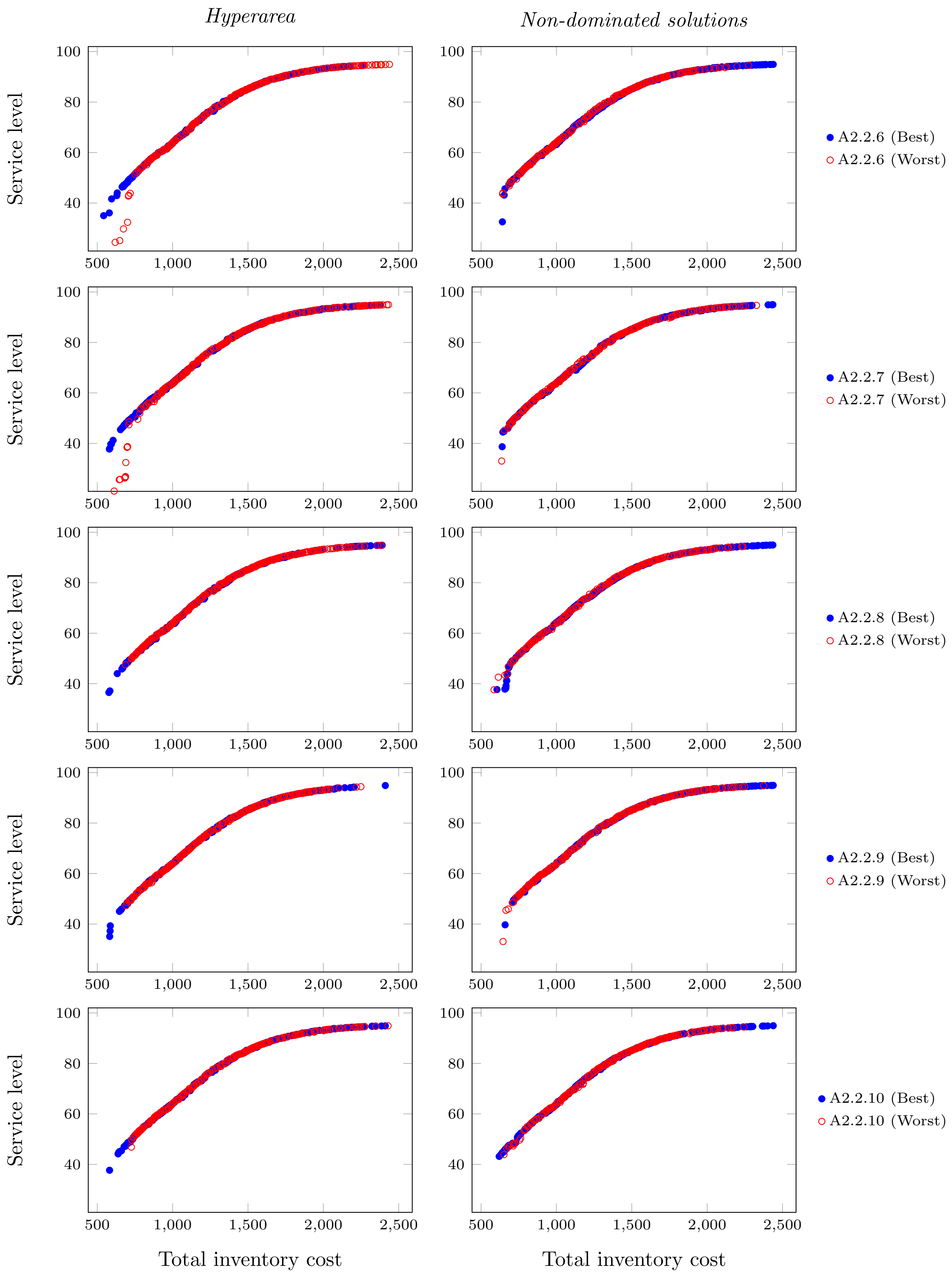


Figure B.16: *The best and worst best approximation fronts for hyperparameter combinations A2.2.6–A2.2.10.*

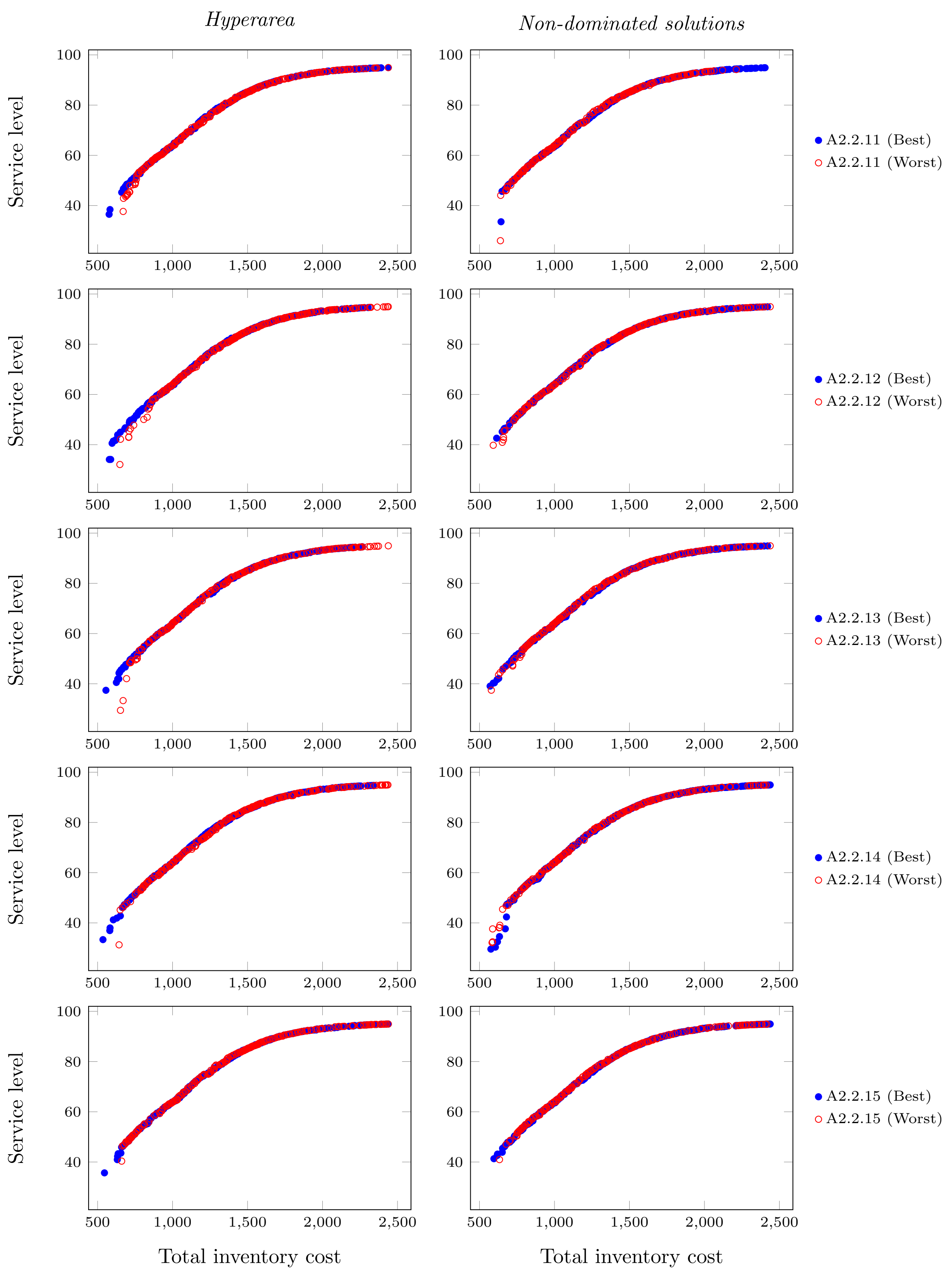


Figure B.17: *The best and worst best approximation fronts for hyperparameter combinations A2.2.11–A2.2.15.*

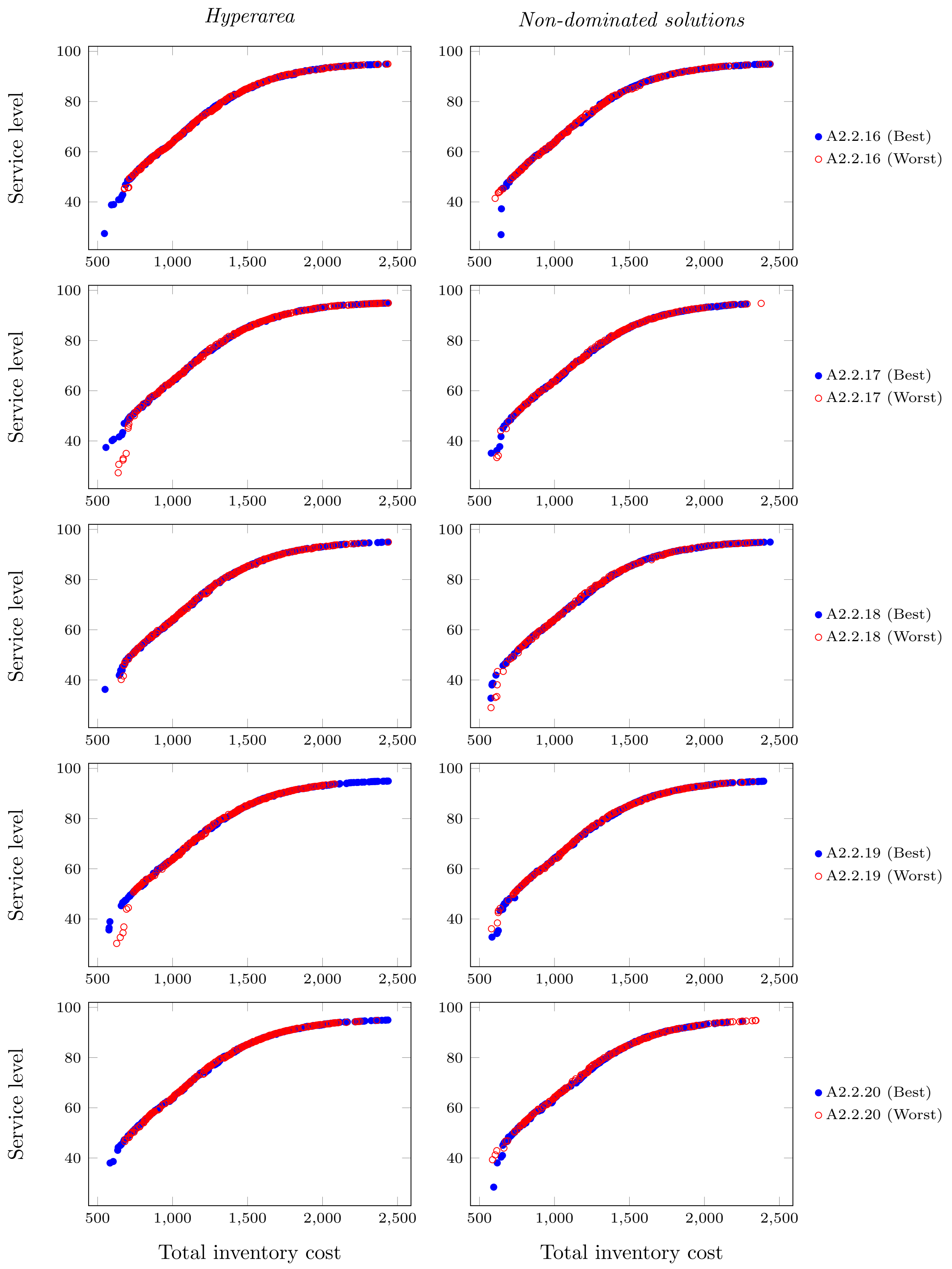


Figure B.18: *The best and worst best approximation fronts for hyperparameter combinations A2.2.16–A2.2.20.*

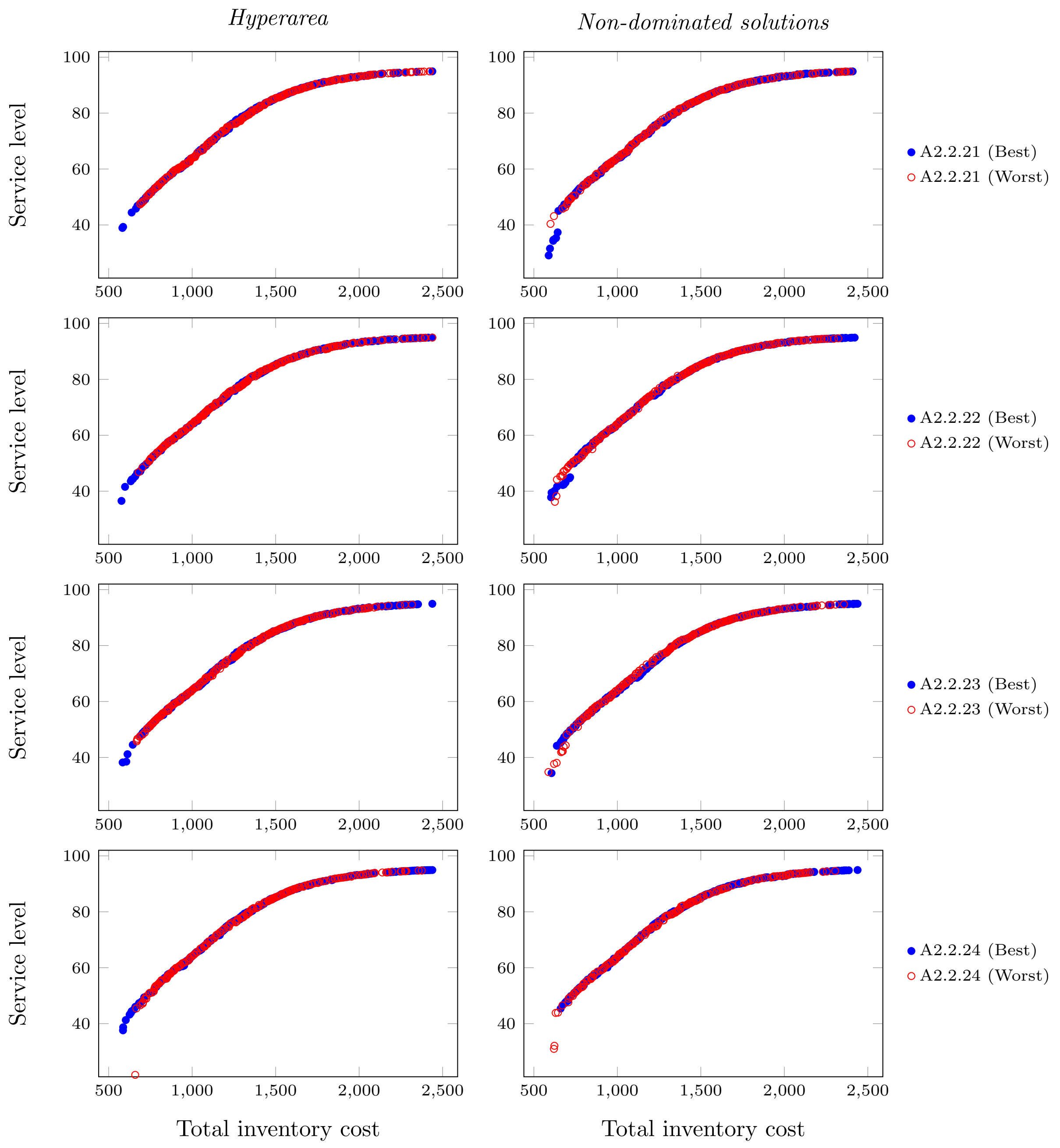


Figure B.19: *The best and worst best approximation fronts for hyperparameter combinations A2.2.21–A2.2.24.*

B.2.3 Buffer allocation problem: five machines

Tables B.28 and B.29 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A2.3.1–A2.3.24. Figures B.20–B.24 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A2.3.1–A2.3.24 that correspond to the runs given in Table B.30.

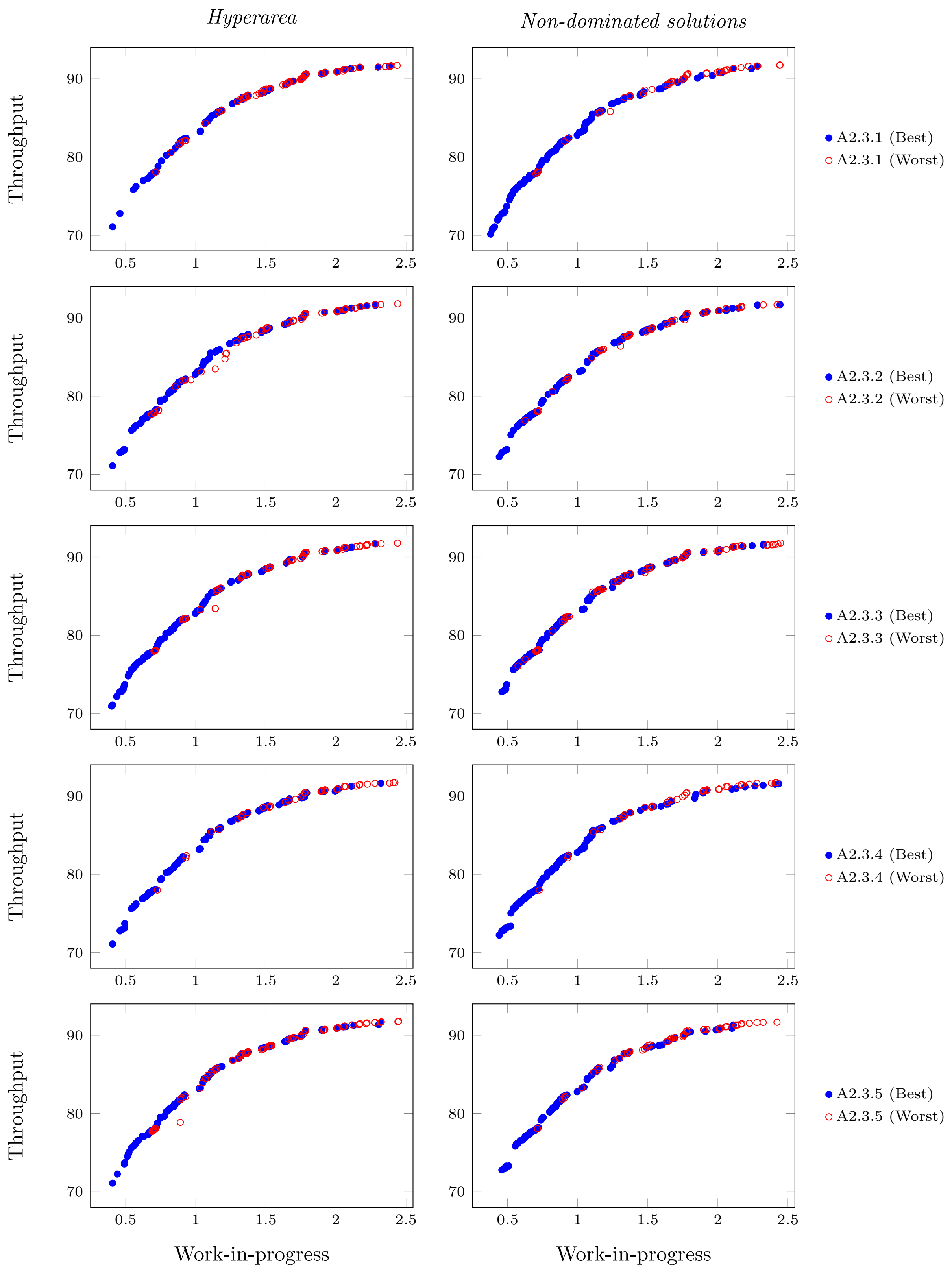


Figure B.20: *The best and worst best approximation fronts for hyperparameter combinations A2.3.1–A2.3.5.*

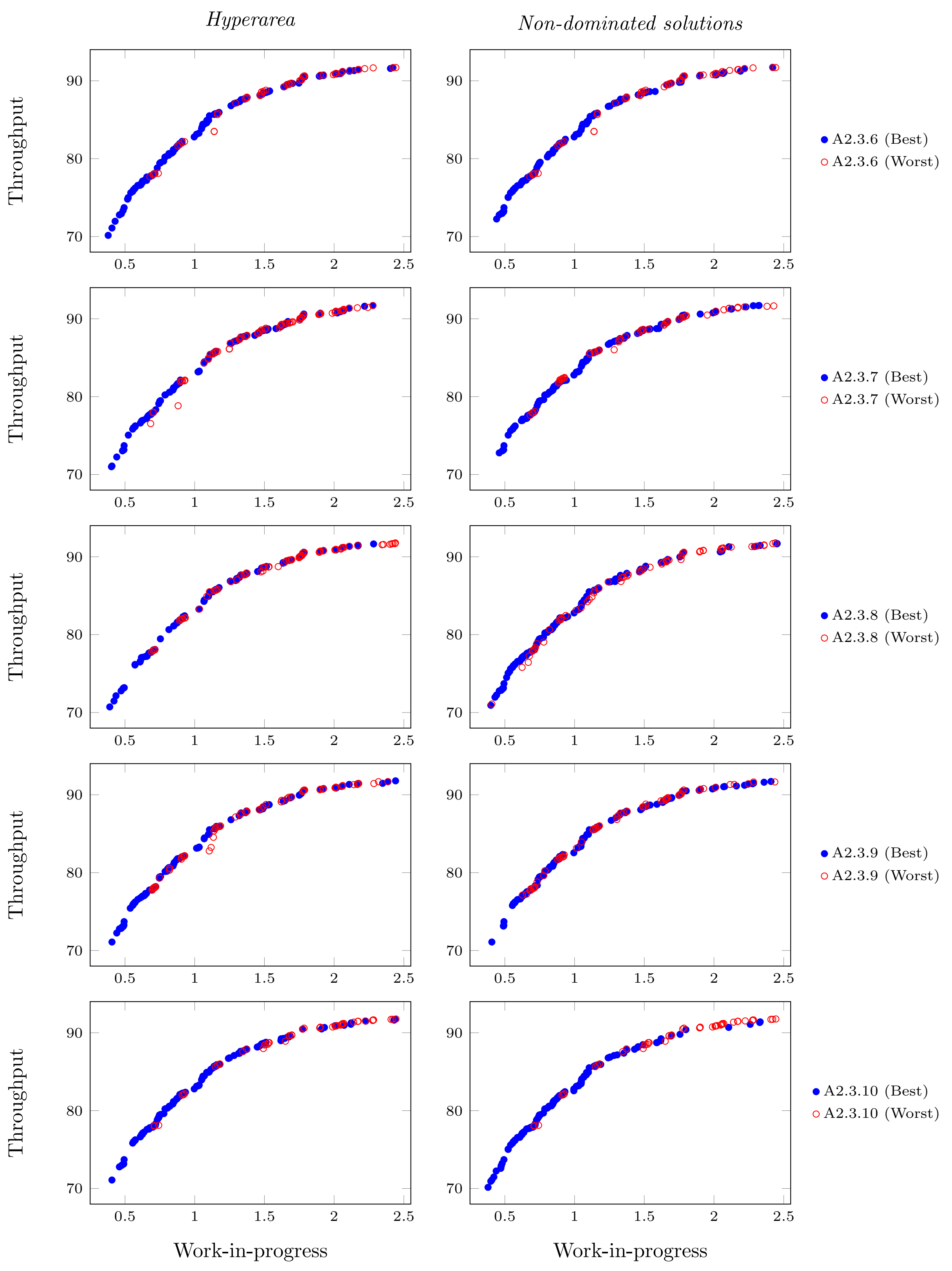


Figure B.21: *The best and worst best approximation fronts for hyperparameter combinations A2.3.6–A2.3.10.*

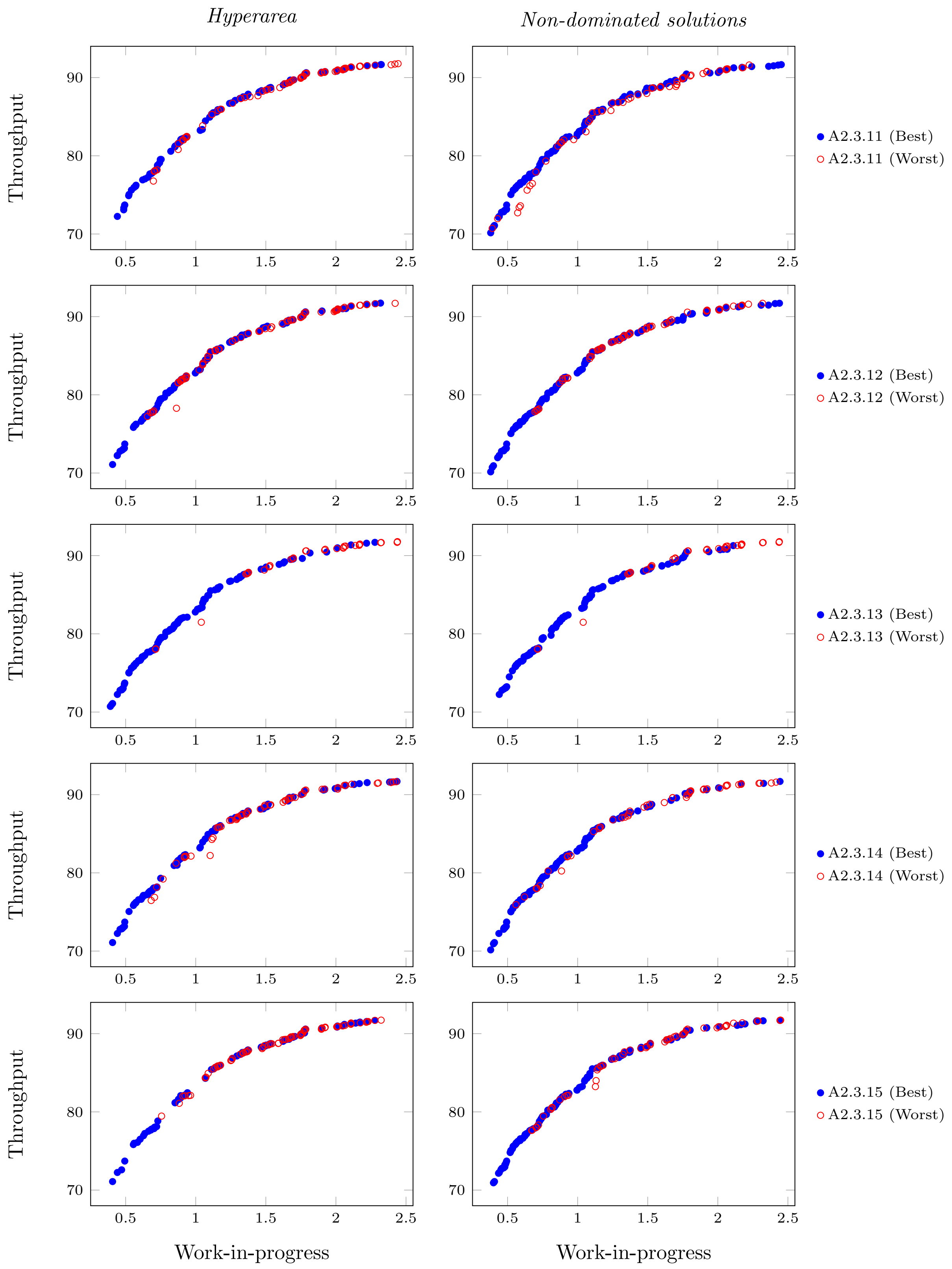


Figure B.22: *The best and worst best approximation fronts for hyperparameter combinations A2.3.11–A2.3.15.*

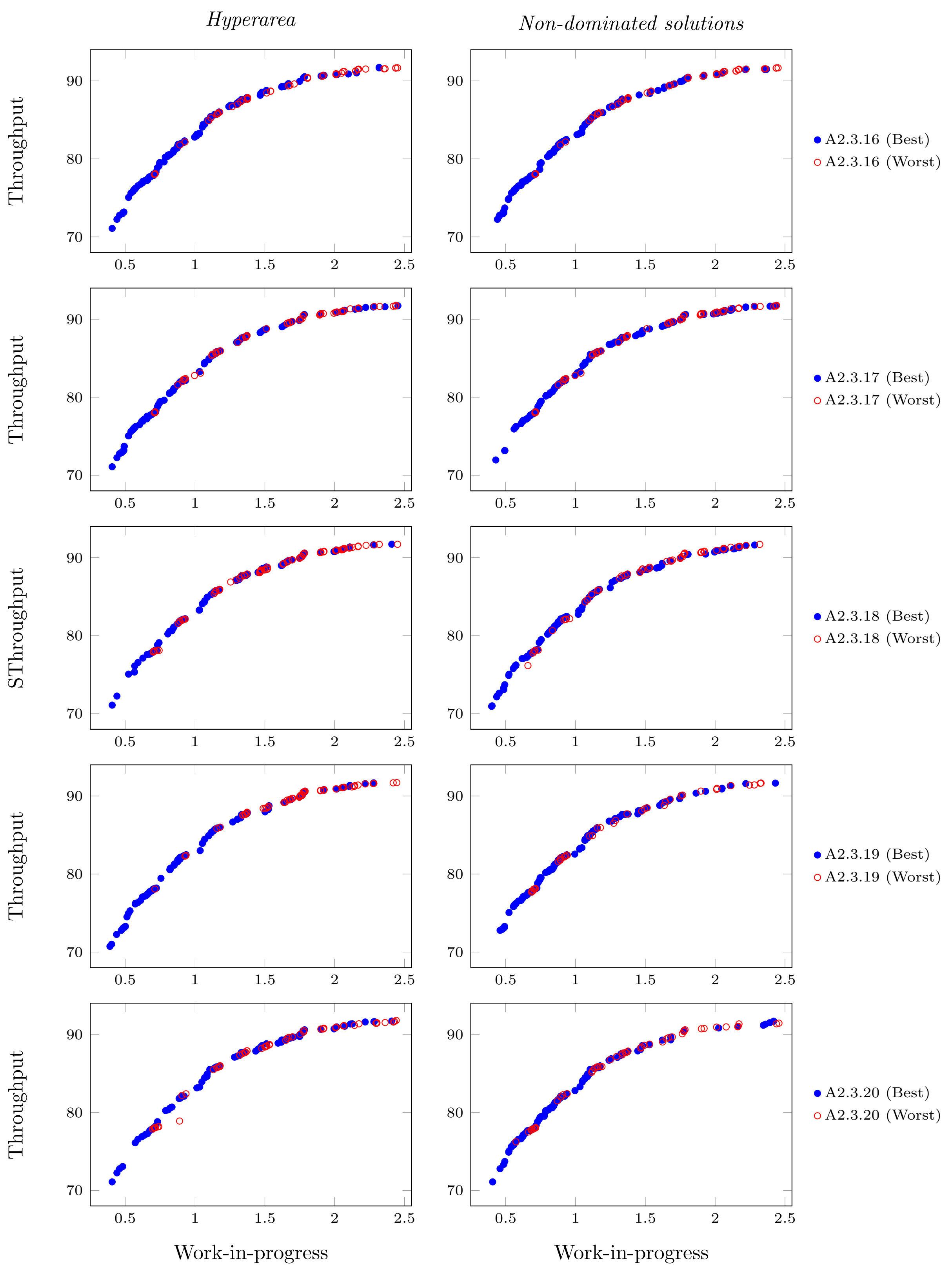


Figure B.23: *The best and worst best approximation fronts for hyperparameter combinations A2.3.16–A2.3.20.*

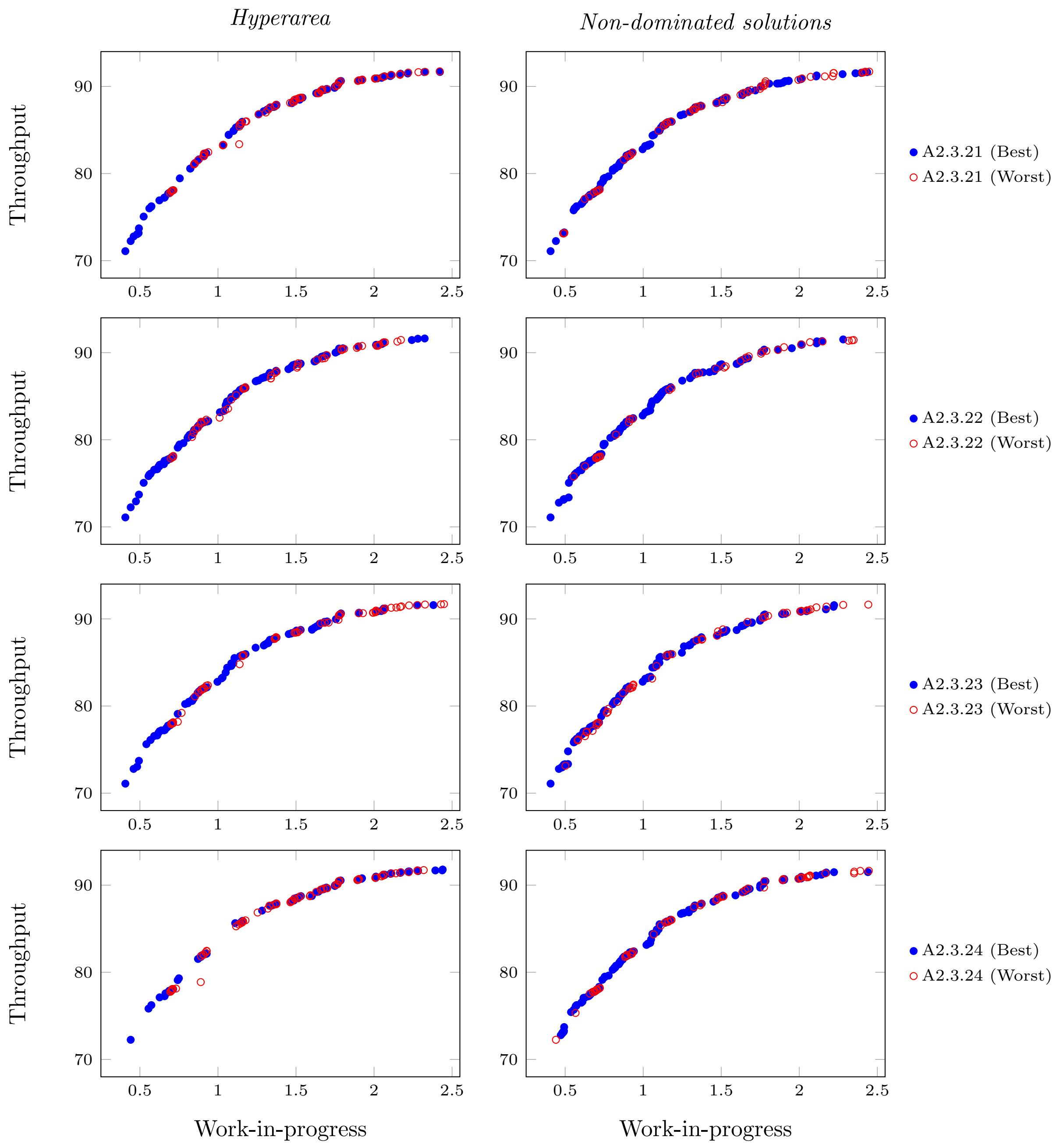


Figure B.24: *The best and worst best approximation fronts for hyperparameter combinations A2.3.21–A2.3.24.*

B.2.4 Buffer allocation problem: 10 machines

Tables B.33 and B.34 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A2.4.1–A2.4.24. Figures B.25–B.29 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A2.4.1–A2.4.24 that correspond to the runs given in Table B.35.

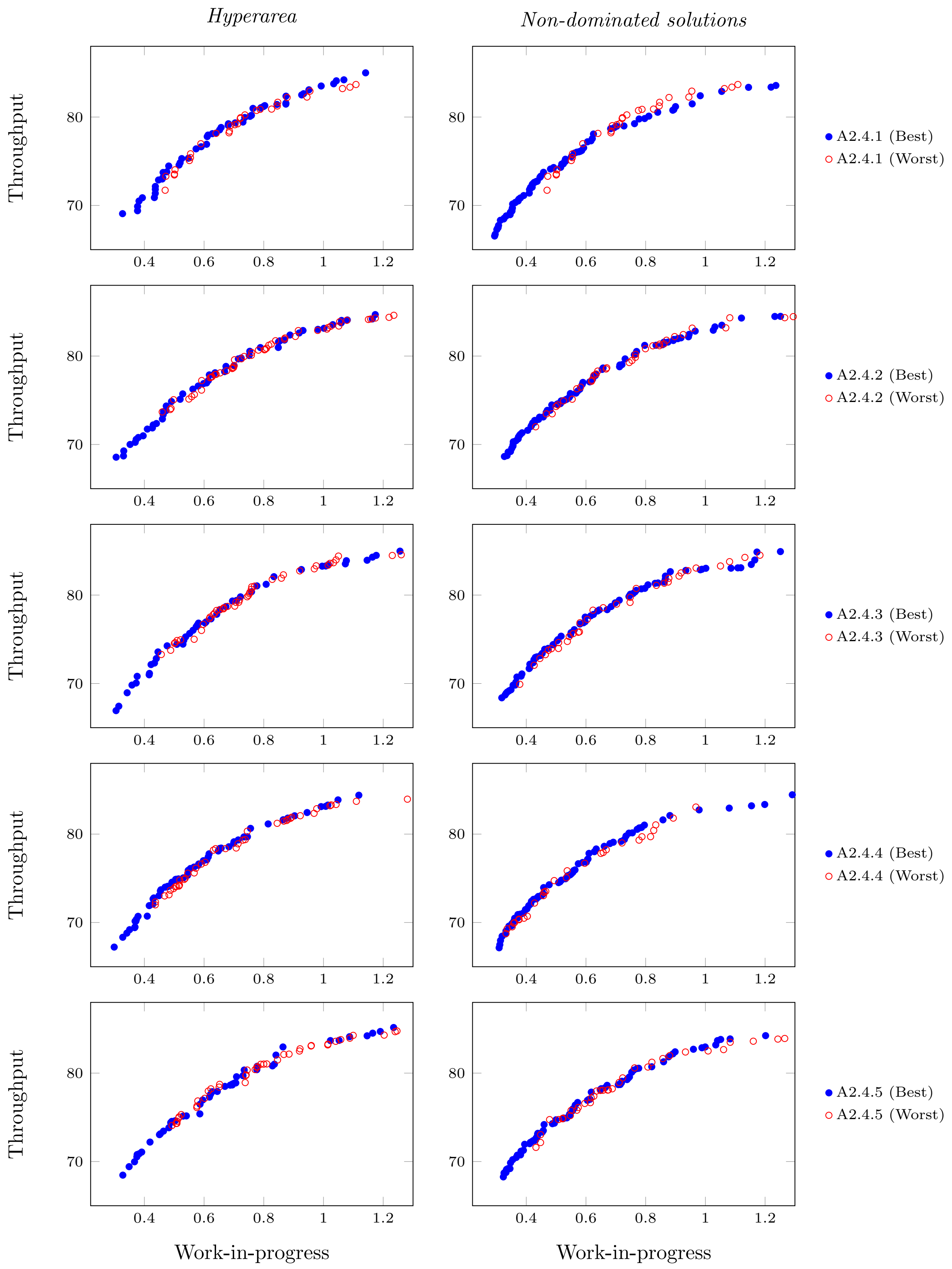


Figure B.25: *The best and worst best approximation fronts for hyperparameter combinations A2.4.1–A2.4.5.*

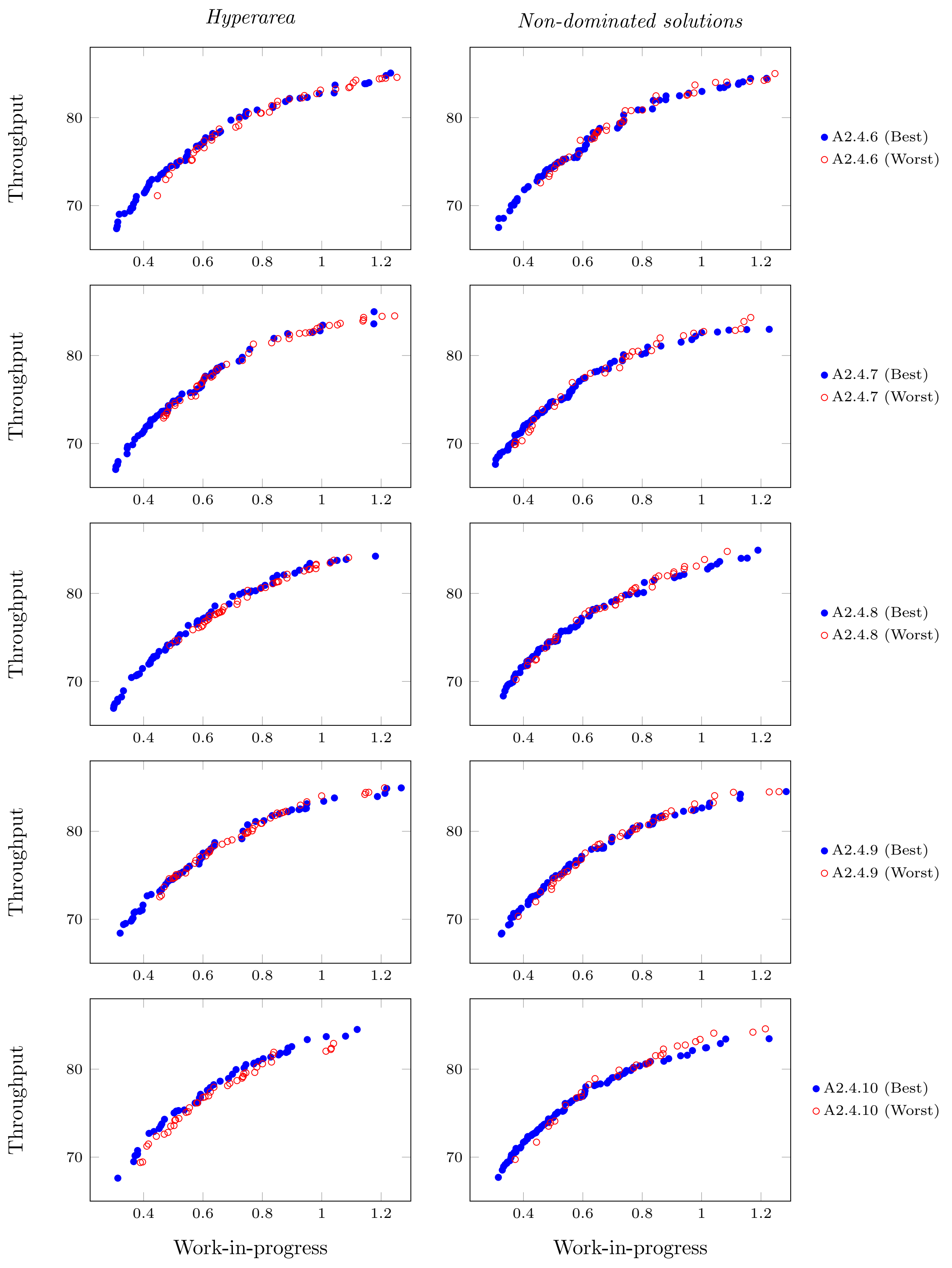


Figure B.26: *The best and worst best approximation fronts for hyperparameter combinations A2.4.6–A2.4.10.*

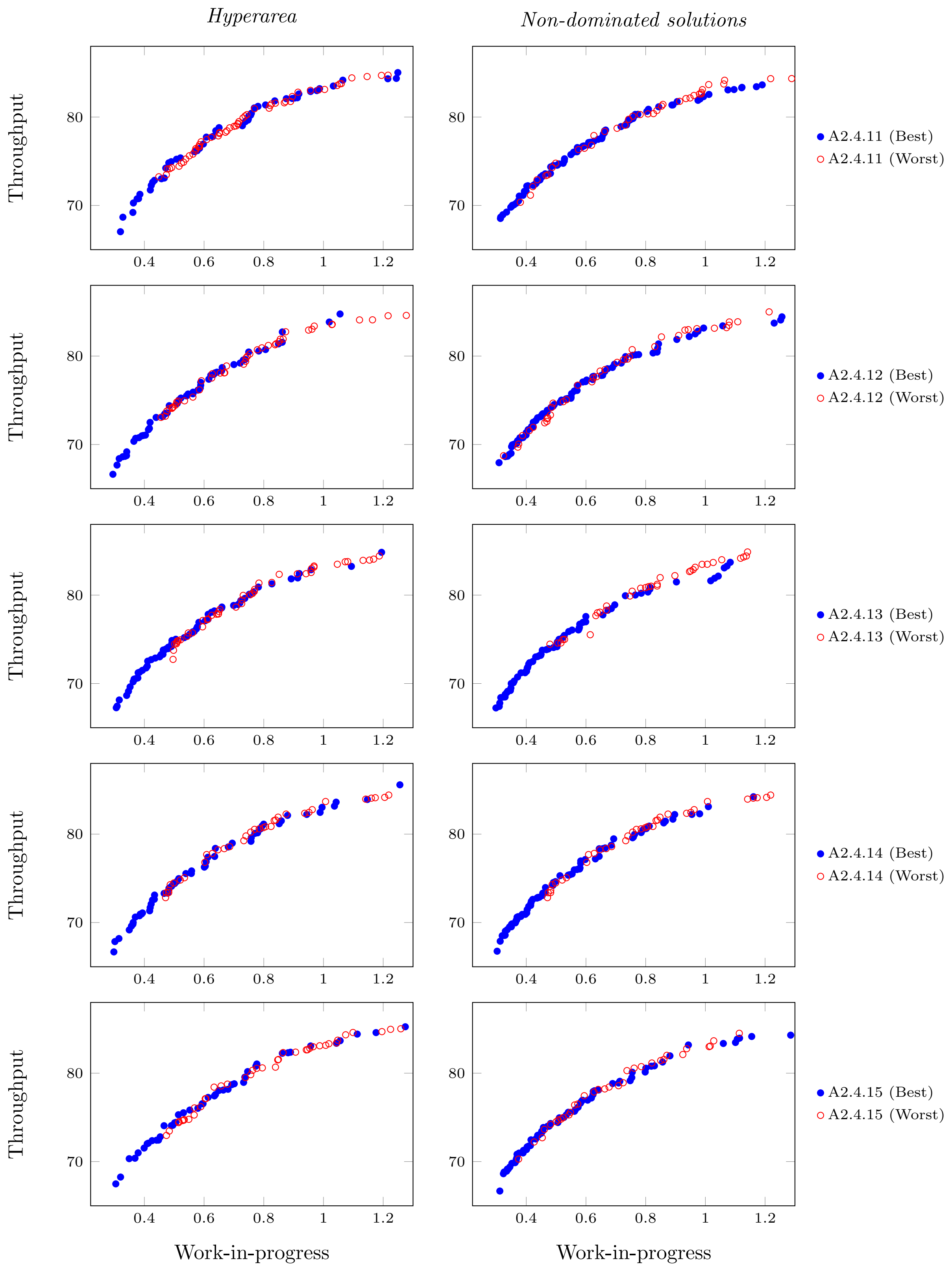


Figure B.27: *The best and worst best approximation fronts for hyperparameter combinations A2.4.11–A2.4.15.*

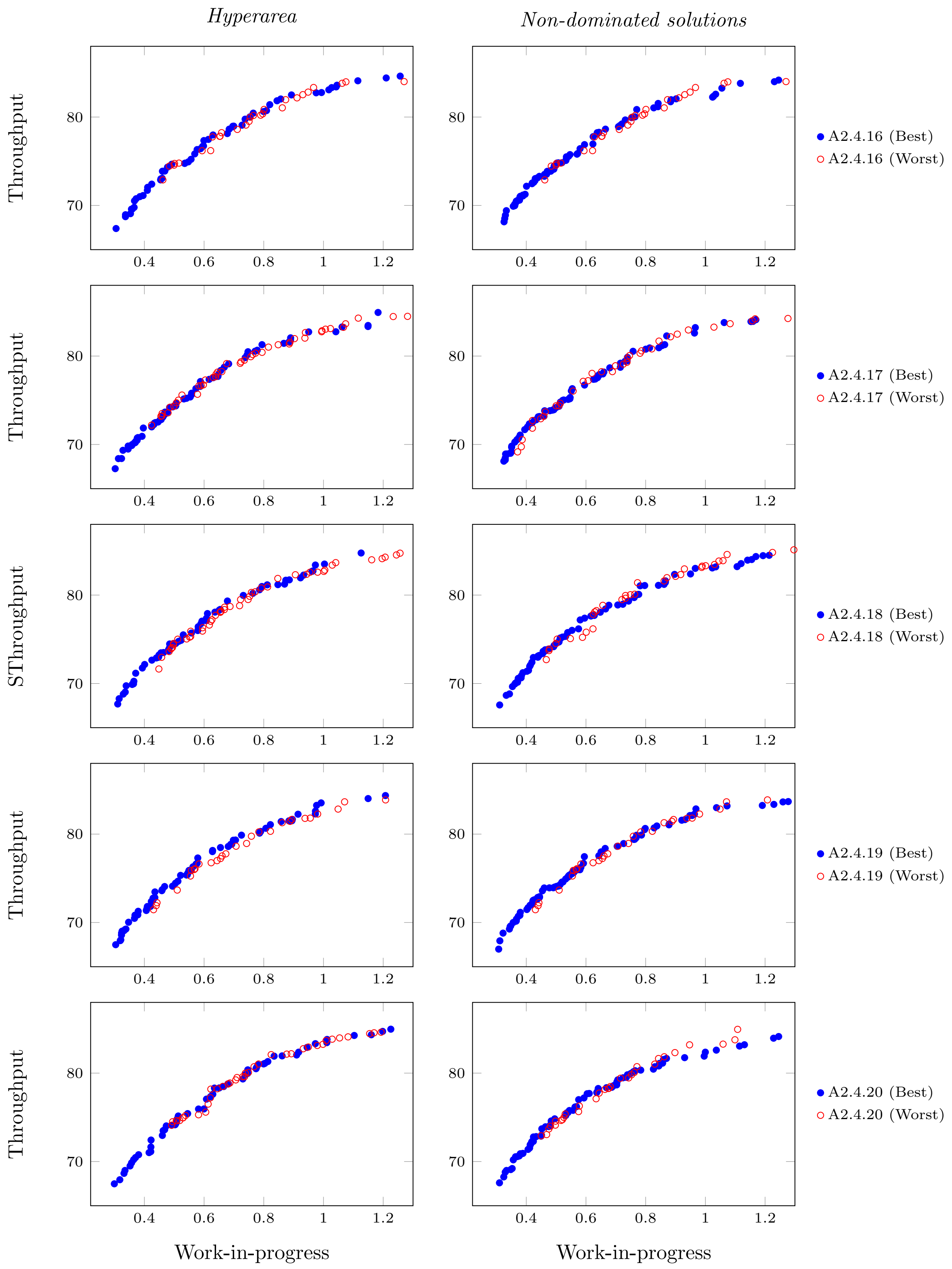


Figure B.28: *The best and worst best approximation fronts for hyperparameter combinations A2.4.16–A2.4.20.*

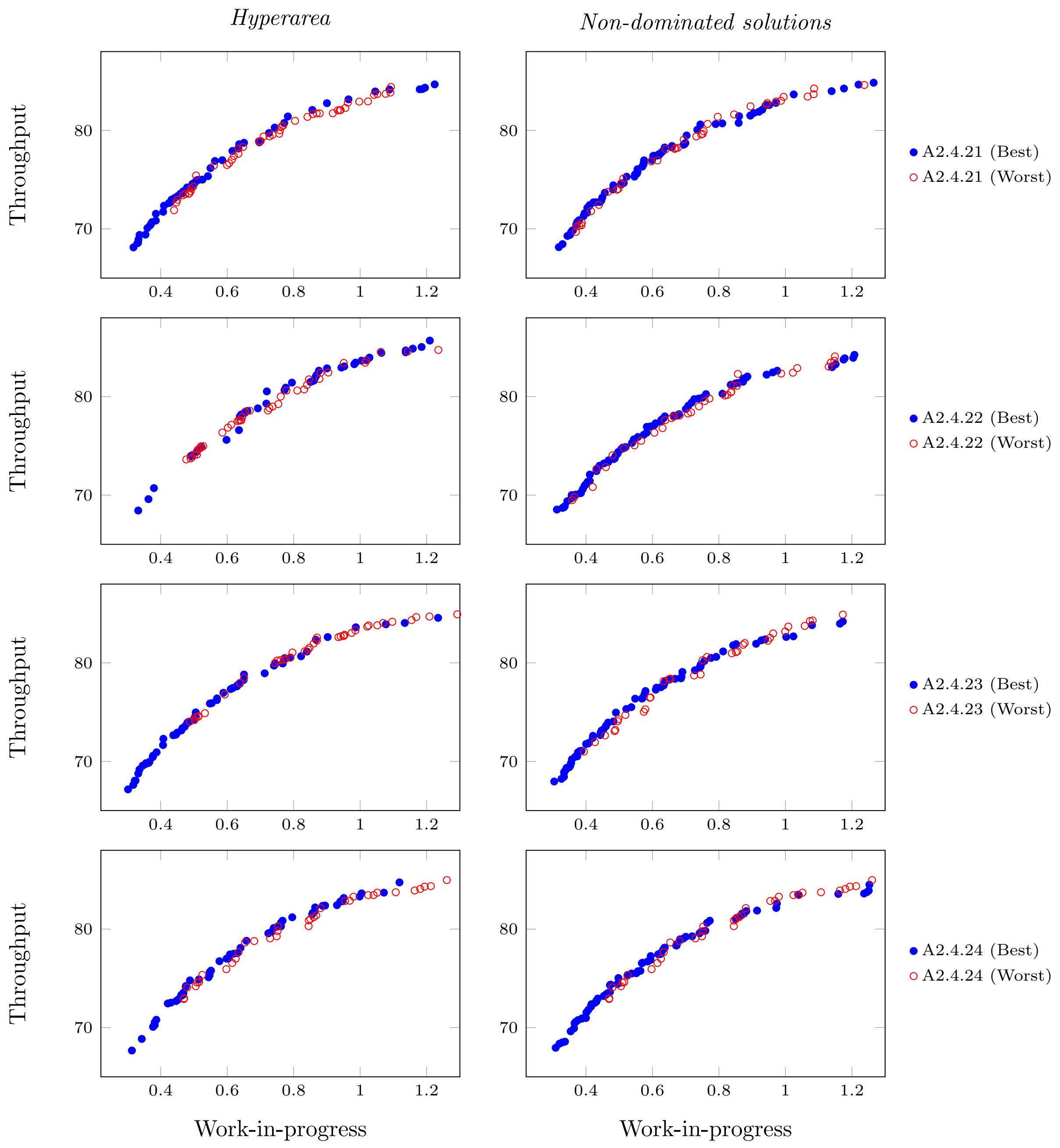


Figure B.29: *The best and worst best approximation fronts for hyperparameter combinations A2.4.21–A2.4.24.*

B.2.5 Non-linear buffer allocation problem: 16 machines

Tables B.38 and B.39 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A2.5.1–A2.5.24. Figures B.30–B.34 illustrate the worst and best approximation fronts found, in terms of hyperareas and number non-dominated solutions, for the respective hyperparameter combinations A2.5.1–A2.5.24 that correspond to the runs given in Table B.40. In Figure B.35 the average number of non-dominated solutions obtained for hyperparameter combinations A2.1–A2.24 is illustrated for the respective simulation problems.

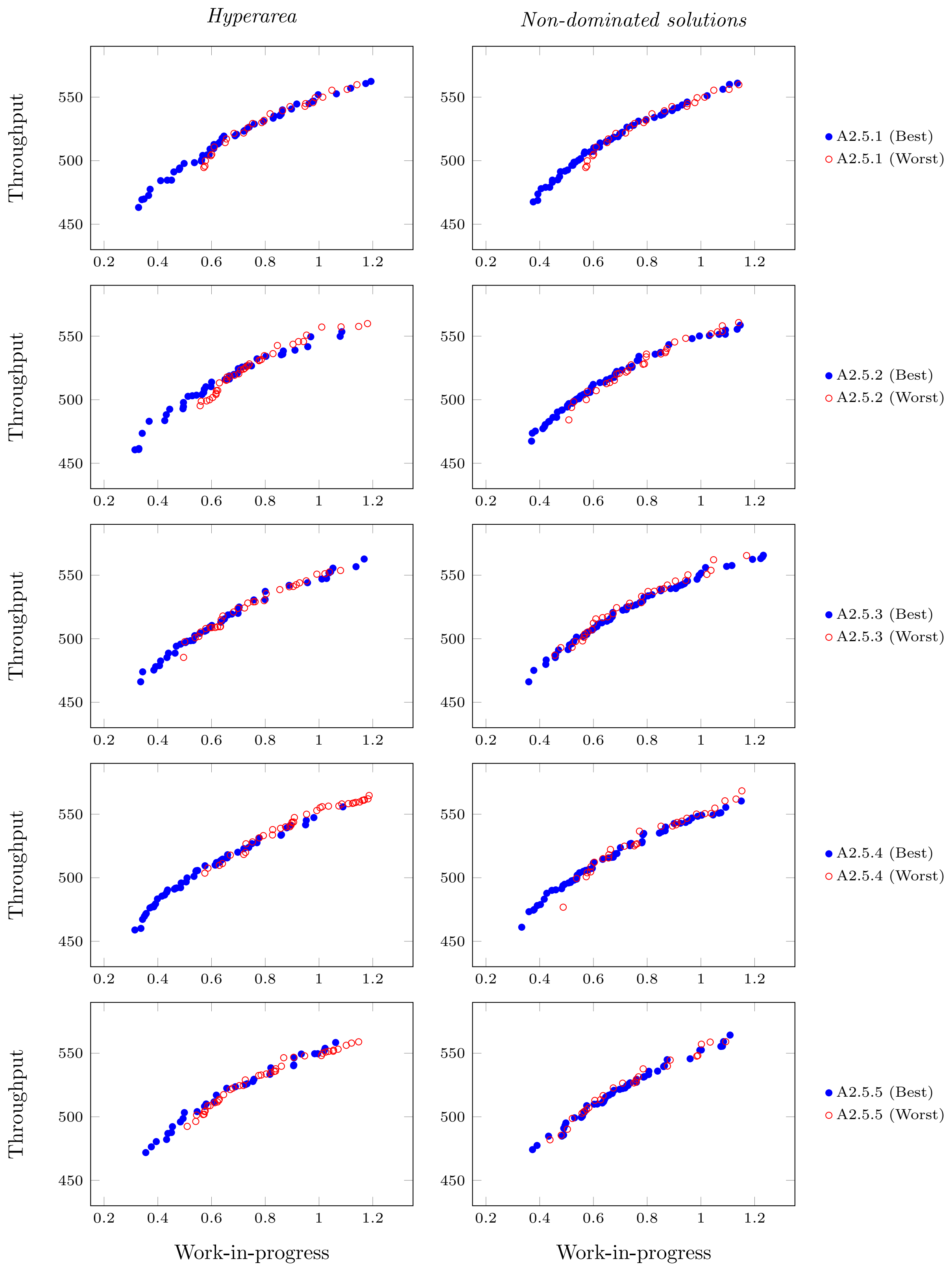


Figure B.30: *The best and worst best approximation fronts for hyperparameter combinations A2.5.1–A2.5.5.*

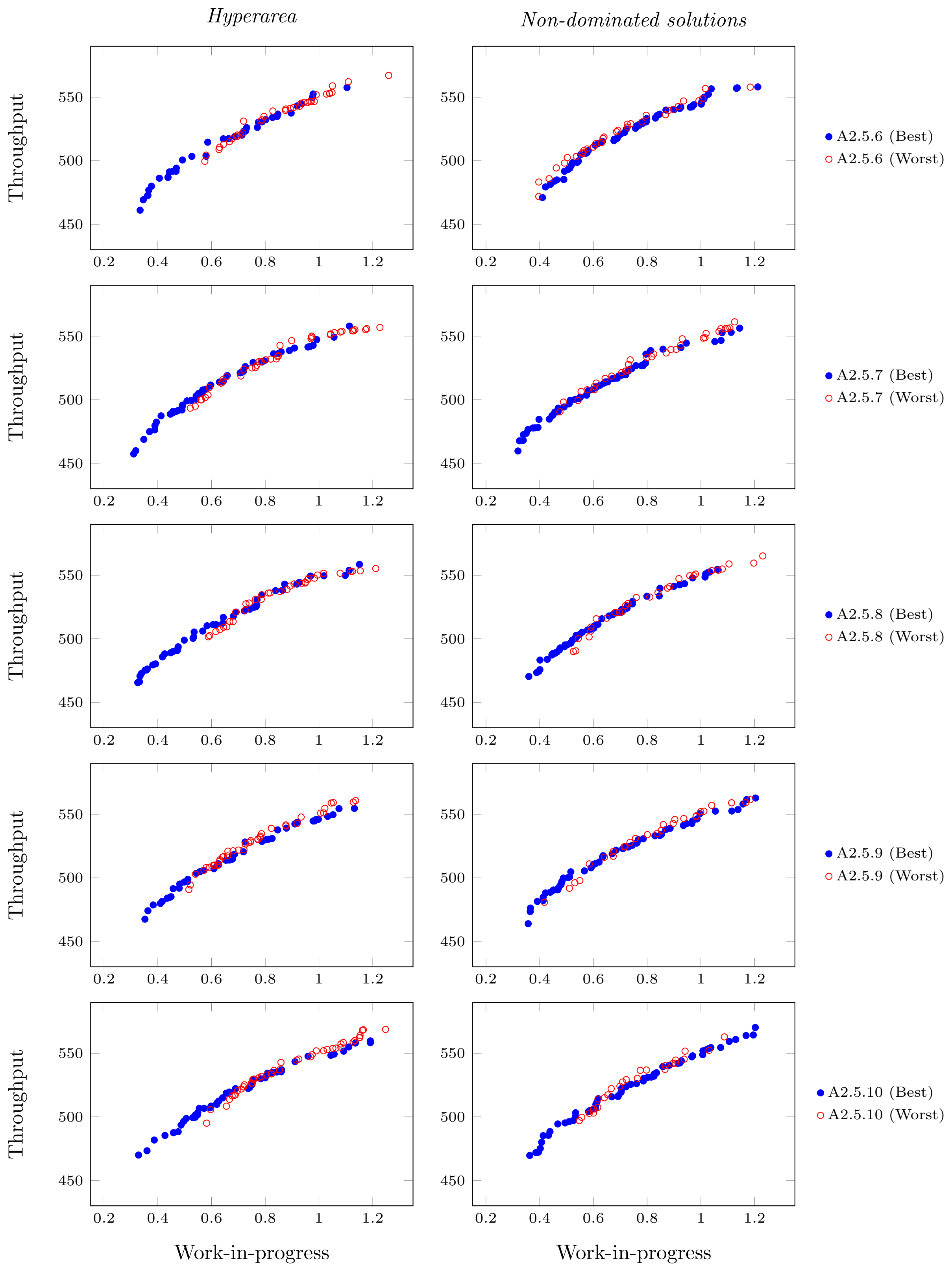


Figure B.31: *The best and worst best approximation fronts for hyperparameter combinations A2.5.6–A2.5.10.*

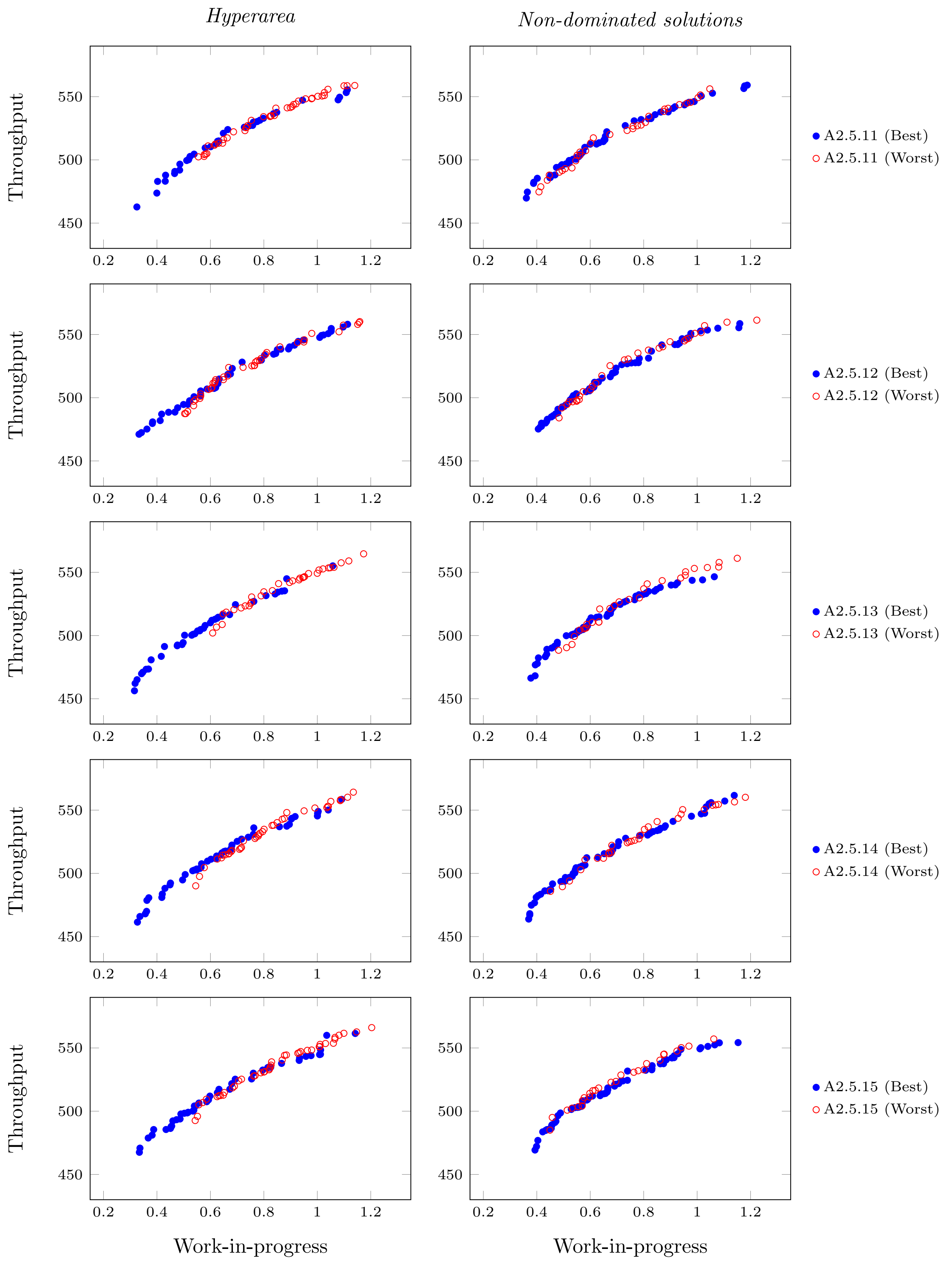


Figure B.32: *The best and worst best approximation fronts for hyperparameter combinations A2.5.11–A2.5.15.*

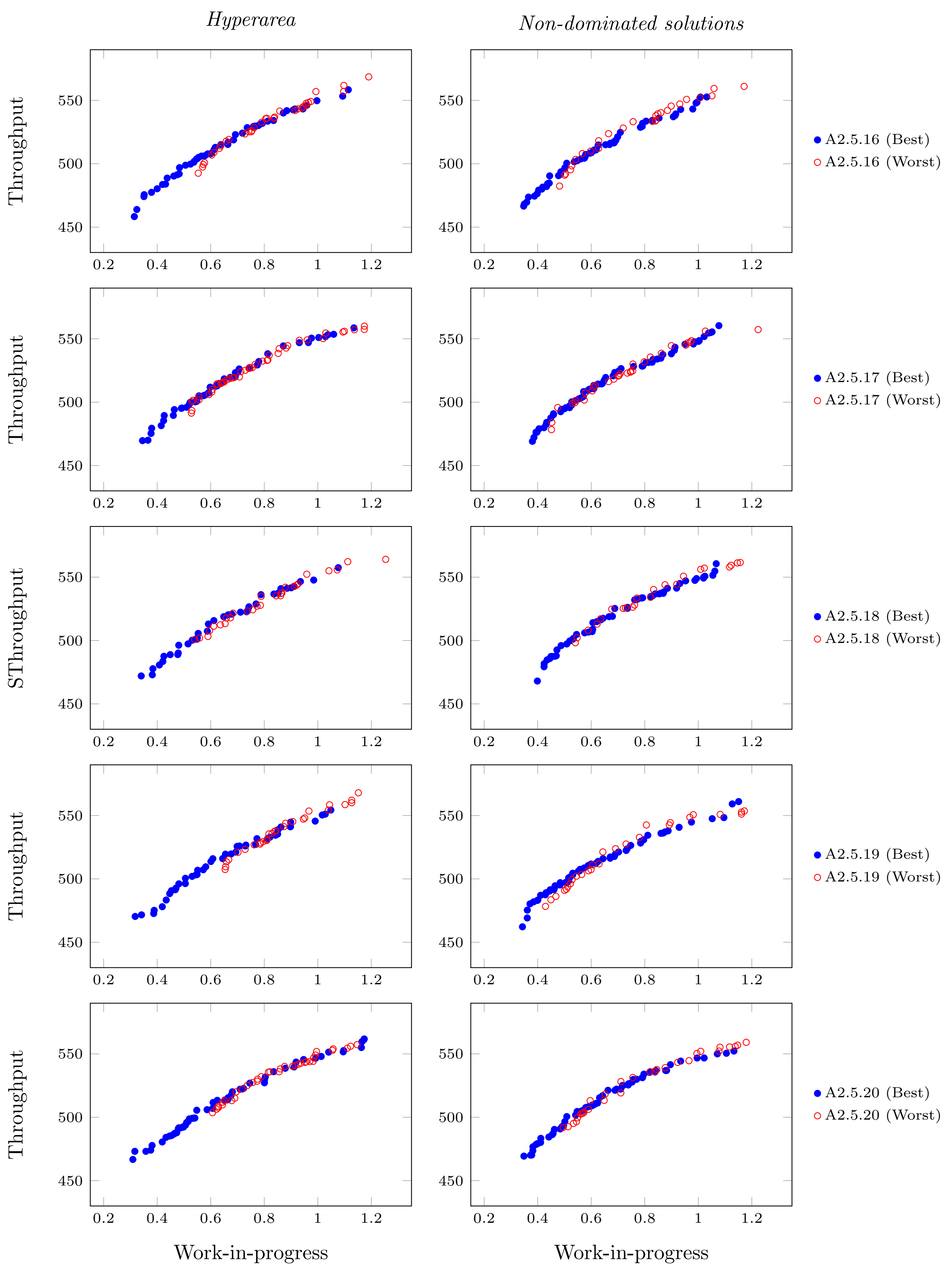


Figure B.33: *The best and worst best approximation fronts for hyperparameter combinations A2.5.16–A2.5.20.*

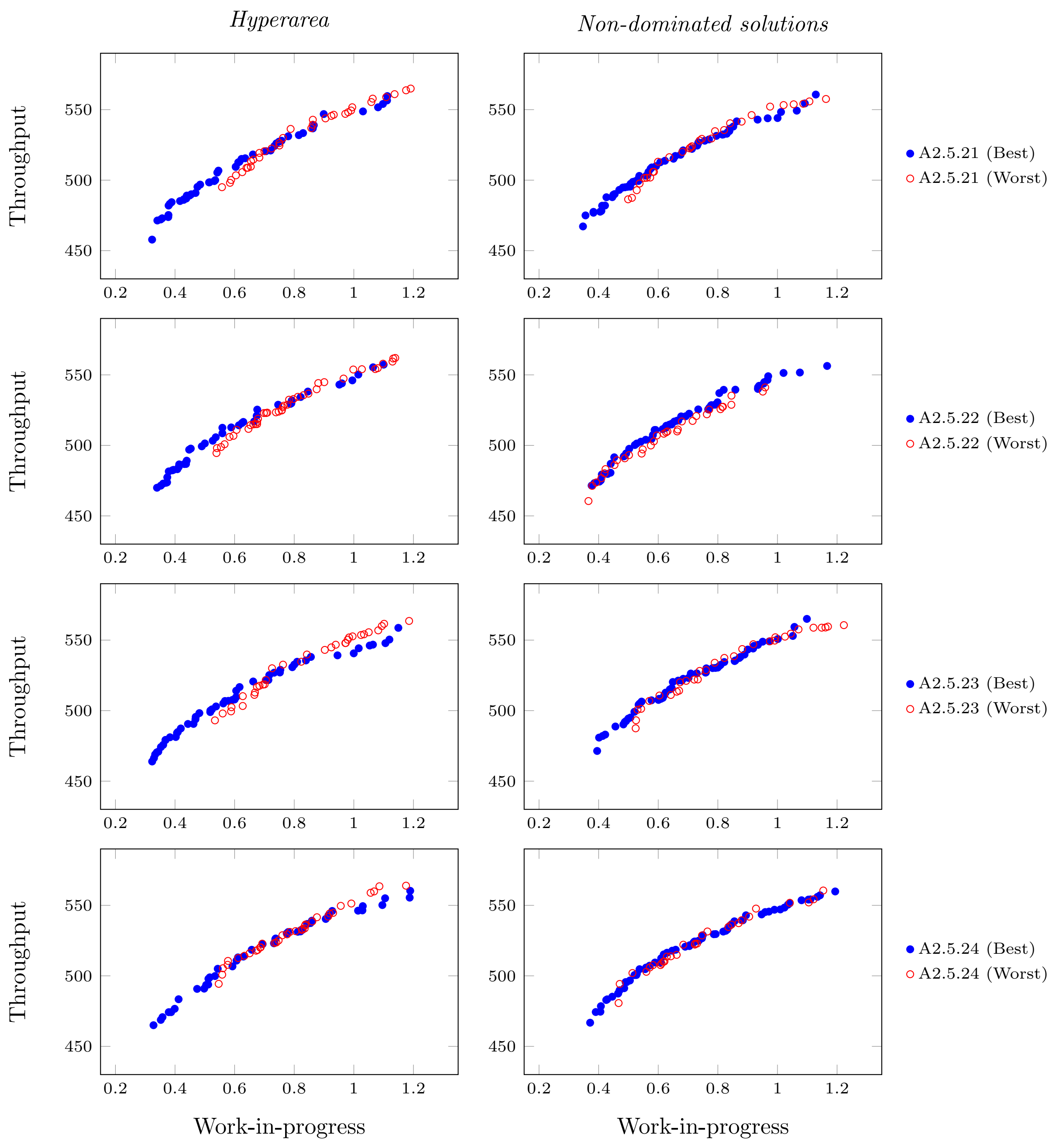


Figure B.34: *The best and worst best approximation fronts for hyperparameter combinations A2.5.21–A2.5.24.*

Table B.18: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–40 for hyperparameter combinations A2.1.1–A2.1.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.1.1	A2.1.2	A2.1.3	A2.1.4	A2.1.5	A2.1.6	...	A2.1.19	A2.1.20	A2.1.21	A2.1.22	A2.1.23	A2.1.24
1	12 630.37 (2)	12 412.64 (3)	9 353.48 (13)	10 415.26 (11)	11 188.10 (8)	8 861.30 (17)	...	10 892.46 (9)	10 456.37 (10)	9 342.55 (14)	6 603.35 (24)	8 912.77 (16)	6 991.79 (23)
2	9 886.77 (16)	12 626.15 (6)	12 056.15 (8)	11 843.21 (9)	8 423.78 (20)	6 612.05 (24)	...	10 110.91 (13)	11 408.44 (11)	8 528.62 (19)	12 680.96 (5)	12 401.58 (7)	13 181.44 (4)
3	9 315.35 (13)	9 668.67 (10)	10 182.95 (6)	9 489.43 (12)	10 537.10 (3)	9 522.34 (11)	...	7 470.06 (22)	8 774.43 (15)	10 466.60 (4)	6 957.33 (24)	10 922.96 (1)	7 295.40 (23)
4	14 485.04 (1.5)	13 685.30 (6)	13 260.07 (10)	13 240.51 (12)	13 683.96 (7)	13 658.39 (8)	...	13 205.74 (13)	12 081.01 (22)	12 255.68 (20)	12 395.18 (19)	12 772.71 (17)	11 419.97 (23)
5	8 939.84 (11)	7 718.24 (16)	8 303.61 (14)	9 065.74 (9)	9 192.94 (8)	11 773.17 (1)	...	7 262.17 (18)	5 786.88 (24)	9 323.92 (6)	6 992.16 (21)	9 283.66 (7)	8 975.24 (10)
6	9 788.53 (4)	9 803.30 (3)	10 535.92 (2)	7 577.52 (18)	9 412.66 (8)	9 361.31 (9)	...	6 007.41 (24)	7 148.81 (19)	7 722.37 (17)	6 700.57 (23)	7 916.77 (15)	9 090.19 (10)
7	10 633.44 (15)	11 711.32 (6)	8 741.69 (21)	11 939.68 (5)	12 278.55 (4)	10 762.40 (13)	...	9 432.93 (18)	10 852.26 (12)	9 409.07 (19)	11 603.16 (9)	10 867.03 (11)	8 585.68 (23)
8	13 928.72 (7)	13 241.53 (12)	13 195.36 (13)	14 485.04 (1.5)	11 736.77 (23)	13 106.97 (15)	...	13 281.77 (10)	12 667.80 (18)	11 499.04 (24)	13 661.53 (8)	13 133.24 (14)	12 450.48 (20)
9	11 258.29 (5)	9 970.75 (12)	10 680.67 (10)	12 402.05 (2)	12 069.08 (4)	10 756.01 (8)	...	9 560.30 (16)	10 047.21 (11)	8 635.27 (23)	8 824.23 (22)	9 182.61 (20)	9 858.45 (14)
10	12 878.16 (2)	11 752.39 (4)	9 448.65 (16)	11 504.02 (6)	10 460.52 (11)	11 331.51 (7)	...	9 547.95 (15)	11 122.05 (8)	8 391.34 (20)	9 604.07 (14)	8 017.31 (22)	9 158.01 (18)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
39	14 350.58 (5)	13 275.53 (14)	11 792.45 (23)	14 059.46 (9)	13 645.07 (11)	12 778.12 (17)	...	12 756.59 (18)	14 485.04 (2)	12 374.41 (20)	13 683.68 (10)	13 257.19 (15)	11 961.04 (22)
40	11 789.72 (7)	9 600.27 (21)	12 083.51 (5)	11 128.37 (12)	13 992.47 (1)	11 196.93 (10)	...	9 787.24 (20)	9 355.47 (22)	8 854.36 (24)	10 034.00 (19)	11 434.59 (9)	8 900.87 (23)
$\sum R_{Ci}$	289	423	473	303	391	522	...	595	543	658	594	517	656
$\sum R_{Ci}^2$	83 521	178 929	223 729	91 809	152 881	272 484	...	354 025	294 849	432 964	352 836	266 772	430 336

Table B.19: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–40 for hyperparameter combinations A2.1.1–A2.1.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.1.1	A2.1.2	A2.1.3	A2.1.4	A2.1.5	A2.1.6	A2.1.7	A2.1.8	...	A2.1.17	A2.1.18	A2.1.19	A2.1.20	A2.1.21	A2.1.22	A2.1.23	A2.1.24
1	32 (2)	31 (3.5)	24 (13.5)	26 (12)	28 (8.5)	23 (17.5)	23 (17.5)	21 (21.5)	...	30 (5.5)	23 (17.5)	28 (8.5)	27 (10.5)	23 (17.5)	21 (21.5)	23 (17.5)	18 (24)
2	27 (12.5)	32 (6)	30 (9)	30 (9)	21 (21.5)	17 (24)	34 (2)	34 (2)	...	19 (23)	22 (19.5)	25 (15.5)	29 (11)	22 (19.5)	32 (6)	32 (6)	33 (4)
3	24 (13)	25 (10)	26 (6.5)	25 (10)	27 (3)	24 (13)	21 (21.5)	26 (6.5)	...	21 (21.5)	25 (10)	21 (21.5)	23 (16.5)	27 (3)	20 (24)	28 (1)	21 (21.5)
4	36 (1.5)	34 (7.5)	33 (13)	33 (13)	34 (7.5)	34 (7.5)	33 (13)	36 (1.5)	...	33 (13)	31 (20)	33 (13)	31 (20)	31 (20)	31 (20)	32 (17)	29 (23)
5	24 (7.5)	21 (17.5)	23 (12)	23 (12)	24 (7.5)	30 (1)	29 (2)	22 (15)	...	18 (22)	24 (7.5)	21 (17.5)	15 (24)	23 (12)	20 (20)	24 (7.5)	25 (4.5)
6	25 (5)	25 (5)	27 (1.5)	20 (18.5)	24 (9)	25 (5)	22 (13)	25 (5)	...	22 (13)	19 (21.5)	17 (23.5)	20 (18.5)	20 (18.5)	17 (23.5)	21 (15.5)	24 (9)
7	28 (12)	30 (6)	23 (21.5)	30 (6)	31 (3.5)	28 (12)	33 (1.5)	31 (3.5)	...	33 (1.5)	28 (12)	24 (19)	28 (12)	26 (17.5)	29 (8.5)	27 (15.5)	22 (24)
8	35 (5.5)	33 (12.5)	33 (12.5)	36 (2)	30 (22)	33 (12.5)	34 (8.5)	32 (17.5)	...	35 (5.5)	32 (17.5)	33 (12.5)	32 (17.5)	29 (24)	34 (8.5)	33 (12.5)	31 (20)
9	28 (6)	25 (14)	27 (8.5)	31 (2)	30 (4)	27 (8.5)	28 (6)	31 (2)	...	21 (24)	25 (14)	25 (14)	25 (14)	22 (23)	23 (21)	23 (21)	25 (14)
10	32 (2)	30 (3.5)	24 (15.5)	29 (6)	26 (11.5)	29 (6)	29 (6)	30 (3.5)	...	21 (21.5)	26 (11.5)	24 (15.5)	28 (8.5)	23 (19)	24 (15.5)	21 (21.5)	23 (19)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
39	36 (3)	33 (15)	30 (22.5)	35 (7.5)	34 (11.5)	32 (17.5)	36 (3)	36 (3)	...	31 (19.5)	30 (22.5)	32 (17.5)	36 (3)	31 (19.5)	34 (11.5)	33 (15)	30 (22.5)
40	30 (7.5)	24 (22.5)	31 (4.5)	28 (14)	35 (1)	29 (10.5)	30 (7.5)	27 (17)	...	28 (14)	31 (4.5)	26 (19)	24 (22.5)	24 (22.5)	26 (19)	30 (7.5)	24 (22.5)
$\sum R_{Ci}$	285	446.5	479	313	391	531.5	310.5	321.5	...	685.5	716.5	578.5	545	657.5	580	509.5	634.5
$\sum R_{Ci}^2$	81 225	199 362.25	229 441	97 969	152 881	282 492.25	96 410.25	103 362.25	...	469 910.25	513 372.25	334 662.25	297 025	432 306.25	336 400	259 590.25	402 590.25

Table B.20: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the NSGA-II) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A2.1.1–A2.1.24.*

	Worst		Best	
	HA	NDS	HA	NDS
A2.1.1	Run 27	Run 27	Run 4	Run 4
A2.1.2	Run 5	Run 5	Run 16	Run 16
A2.1.3	Run 21	Run 21	Run 18	Run 18
A2.1.4	Run 31	Run 31	Run 8	Run 8
A2.1.5	Run 37	Run 37	Run 16	Run 16
A2.1.6	Run 2	Run 2	Run 4	Run 4
A2.1.7	Run 21	Run 21	Run 20	Run 16
A2.1.8	Run 27	Run 27	Run 4	Run 4
A2.1.9	Run 1	Run 6	Run 19	Run 19
A2.1.10	Run 21	Run 34	Run 8	Run 8
A2.1.11	Run 27	Run 27	Run 16	Run 36
A2.1.12	Run 34	Run 34	Run 19	Run 19
A2.1.13	Run 27	Run 27	Run 16	Run 16
A2.1.14	Run 5	Run 5	Run 39	Run 39
A2.1.15	Run 27	Run 27	Run 15	Run 15
A2.1.16	Run 32	Run 32	Run 39	Run 39
A2.1.17	Run 21	Run 21	Run 16	Run 16
A2.1.18	Run 32	Run 32	Run 36	Run 36
A2.1.19	Run 6	Run 6	Run 19	Run 19
A2.1.20	Run 5	Run 5	Run 16	Run 16
A2.1.21	Run 27	Run 27	Run 19	Run 19
A2.1.22	Run 27	Run 27	Run 25	Run 25
A2.1.23	Run 27	Run 27	Run 16	Run 16
A2.1.24	Run 1	Run 1	Run 16	Run 16

Table B.21: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the hyperareas of the approximation fronts for the hyperparameter combinations A2.1.1–A2.1.24.*

	A2.1.2	A2.1.3	A2.1.4	A2.1.5	A2.1.6	A2.1.7	A2.1.8	A2.1.9	A2.1.10	A2.1.11	A2.1.12	A2.1.13	A2.1.14	A2.1.15	A2.1.16	A2.1.17	A2.1.18	A2.1.19	A2.1.20	A2.1.21	A2.1.22	A2.1.23	A2.1.24
A2.1.1	1	1	1	1	0.06	1	1	1	1	1	1	0	0	0	0	0	0	0	0.02	0	0	0.09	0
A2.1.2		1	1	1	1	1	1	1	1	1	1	0.03	0.01	0	0.08	0.01	0	1	1	0.06	1	1	0.06
A2.1.3			1	1	1	1	1	1	0.39	0.46	1	0.52	0.18	0.01	1	0.13	0.07	1	1	0.95	1	1	1
A2.1.4				1	0.15	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0.15	0.04	0	1
A2.1.5					1	1	1	1	1	1	1	0	0	0	0.01	0	0	0.35	1	0.01	0.37	1	0.01
A2.1.6						0.36	0.18	1	0.02	0.02	1	1	1	0.17	1	1	1	1	1	1	1	1	1
A2.1.7							1	1	1	1	1	0	0	0	0	0	0	0	0.11	0	0	0.48	0
A2.1.8								1	1	1	1	0	0	0	0	0	0	0.001	0.05	0	0	0.25	0
A2.1.9									1	1	1	0.001	0	0	0	0	0	0.09	1	0	0.1	1	0
A2.1.10										1	1	0	0	0	0	0	0	0	0.01	0	0	0.03	0
A2.1.11											1	0	0	0	0	0	0	0	0.01	0	0	0.04	0
A2.1.12												0	0	0	0	0	0	0.03	0.63	0	0.03	1	0
A2.1.13													1	1	1	1	1	1	1	1	1	1	1
A2.1.14														1	1	1	1	1	1	1	1	1	1
A2.1.15															1	1	1	1	0.54	1	1	0.12	1
A2.1.16																1	1	1	1	1	1	1	1
A2.1.17																	1	1	1	1	1	1	1
A2.1.18																		1	1	1	1	0.79	1
A2.1.19																			1	1	1	1	1
A2.1.20																				1	1	1	1
A2.1.21																					1	1	1
A2.1.22																						1	1
A2.1.23																							1

Table B.22: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A2.1.1–A2.1.24.*

	A2.1.2	A2.1.3	A2.1.4	A2.1.5	A2.1.6	A2.1.7	A2.1.8	A2.1.9	A2.1.10	A2.1.11	A2.1.12	A2.1.13	A2.1.14	A2.1.15	A2.1.16	A2.1.17	A2.1.18	A2.1.19	A2.1.20	A2.1.21	A2.1.22	A2.1.23	A2.1.24
A2.1.1	1	0.596	1	1	0.027	1	1	1	1	1	1	0	0	0	0	0	0	0	0.01	0	0	0.11	0
A2.1.2		1	1	1	1	1	1	1	1	1	1	0.19	0.07	0	0.25	0.04	0.01	1	1	0.23	1	1	0.82
A2.1.3			1	1	1	1	1	1	0.31	0.54	1	1	0.46	0.02	1	0.30	0.048	1	1	1	1	1	1
A2.1.4				1	0.152	1	1	1	1	1	1	0	0	0	0	0	0	0.01	1	0.15	0.07	0	1
A2.1.5					1	1	1	1	1	1	1	0.01	0	0	0.01	0	0	0.84	1	0.007	0.77	1	0.03
A2.1.6						0.13	0.25	1	0.01	0.02	1	1	1	0.55	1	1	0.95	1	1	1	1	1	1
A2.1.7							1	1	1	1	1	0	0	0	0	0	0	0.006	0.06	0	0.01	0.46	0
A2.1.8								1	1	1	1	0	0	0	0	0	0	0.013	0.11	0	0.01	0.82	0
A2.1.9									1	1	1	0	0	0	0	0	0	0.35	1	0.002	0.32	1	0.01
A2.1.10										1	1	0	0	0	0	0	0	0	0.01	0	0	0.05	0
A2.1.11											1	0	0	0	0	0	0	0.001	0.01	0	0	0.09	0
A2.1.12												0	0	0	0.001	0	0	0.17	1	0	0.15	1	0
A2.1.13													1	1	1	1	1	1	1	1	1	1	1
A2.1.14														1	1	1	1	1	1	1	1	1	1
A2.1.15															1	1	1	1	1	1	1	0.16	1
A2.1.16																1	1	1	1	1	1	1	1
A2.1.17																	1	1	1	1	1	1	1
A2.1.18																		1	1	1	1	0.29	1
A2.1.19																			1	1	1	1	1
A2.1.20																				1	1	1	1
A2.1.21																					1	1	1
A2.1.22																						1	1
A2.1.23																							1

Table B.23: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A2.2.1–A2.2.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.1.1	A2.1.2	A2.1.3	A2.1.4	A2.1.5	A2.1.6	…	A2.1.19	A2.1.20	A2.1.21	A2.1.22	A2.1.23	A2.1.24
1	113 172.34 (15)	113 205.32 (13)	112 949.35 (21)	113 196.28 (9)	113 336.88 (10)	112 556.15 (17)	…	113 441.34 (8)	113 489.94 (6)	112 916.79 (18)	113 793.91 (1)	113 557.65 (4)	112 856.48 (20)
2	113 034.52 (20)	113 037.17 (19)	113 056.01 (1)	113 360.78 (13)	113 491.98 (21)	113 038.56 (17)	…	113 588.63 (4)	113 462.16 (10)	113 656.77 (3)	113 569.84 (5)	113 221.15 (16)	112 871.14 (23)
3	112 020.51 (17)	113 575.27 (1)	111 414.30 (14)	111 310.29 (7)	112 711.92 (6)	111 573.53 (23)	…	113 039.14 (4)	112 618.62 (11)	111 905.69 (19)	112 941.50 (5)	111 958.43 (18)	112 041.66 (16)
4	112 933.80 (18)	112 779.82 (23)	112 791.85 (17)	113 089.44 (3)	113 236.64 (4)	113 528.05 (22)	…	113 001.64 (16)	113 110.37 (12)	113 055.55 (15)	113 770.76 (1)	113 691.94 (2)	113 169.66 (11)
5	113 629.22 (4)	113 091.38 (21)	114 308.77 (11)	113 547.77 (20)	113 184.41 (6)	112 795.53 (1)	…	112 914.68 (22)	113 288.02 (14)	113 598.40 (5)	113 193.81 (17)	113 265.31 (15)	113 479.88 (10)
6	112 608.83 (20)	112 926.97 (11)	112 605.40 (10)	113 079.65 (13)	112 911.65 (18)	112 488.35 (21)	…	113 147.45 (7)	112 657.58 (17)	113 148.26 (6)	113 163.14 (5)	112 874.42 (14)	112 524.92 (22)
7	113 879.95 (2)	113 843.53 (3)	112 873.46 (6)	113 729.51 (18)	113 064.75 (23)	112 924.05 (19)	…	112 553.75 (22)	113 198.92 (11)	113 276.46 (10)	113 052.77 (16)	113 550.25 (8)	113 674.67 (5)
8	113 593.58 (1)	112 891.69 (13)	112 430.82 (23)	112 851.99 (11)	112 219.23 (22)	112 312.20 (20)	…	113 503.85 (6)	113 391.77 (9)	112 732.03 (18)	113 590.56 (2)	112 761.56 (17)	112 862.19 (14)
9	112 187.71 (24)	112 437.25 (21)	113 924.77 (7)	112 420.69 (15)	112 780.57 (13)	112 745.01 (2)	…	113 015.79 (11)	113 225.71 (9)	113 691.46 (3)	112 858.34 (14)	112 790.05 (17)	113 369.66 (8)
10	113 279.50 (11)	112 924.08 (18)	114 166.38 (8)	113 185.08 (4)	112 725.41 (19)	113 099.21 (1)	…	113 240.60 (12)	112 772.18 (21)	113 299.70 (9)	113 214.77 (13)	113 454.45 (6)	113 096.03 (16)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	112 349.01 (19)	113 753.36 (1)	112 707.15 (15)	112 213.91 (23)	111 999.81 (11)	112 360.70 (12)	…	112 883.13 (10)	113 205.36 (7)	112 564.14 (14)	113 219.10 (6)	112 437.39 (17)	113 388.75 (4)
100	112 805.26 (18)	113 084.00 (11)	113 199.71 (3)	113 360.72 (2)	112 169.98 (21)	113 855.00 (9)	…	113 306.14 (8)	113 155.69 (10)	112 590.76 (19)	112 877.87 (15)	112 805.27 (17)	112 834.03 (16)
$\sum R_{Ci}$	1 410	1 227	1 117	1 349	1 371	1 262	…	961	1 214	1 336	1 122	1 133	1 340
$\sum R_{Ci}^2$	1 988 100	1 505 529	1 247 689	1 819 801	1 879 641	1 592 644	…	923 521	1 473 796	1 784 896	1 258 884	1 283 689	1 795 600

Table B.24: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A2.2.1–A2.2.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.2.1	A2.2.2	A2.2.3	A2.2.4	A2.2.5	A2.2.6	A2.2.7	A2.2.8	…	A2.2.16	A2.2.17	A2.2.18	A2.2.19	A2.2.20	A2.2.21	A2.2.22	A2.2.23	A2.2.24
1	276 (3)	277 (1.5)	229 (18)	270 (4.5)	271 (8)	263 (19.5)	241 (7)	246 (22)	…	229 (19.5)	233 (16.5)	220 (23)	277 (1.5)	233 (16.5)	238 (15)	273 (4.5)	242 (13)	226 (21)
2	286 (1)	242 (14)	242 (14)	267 (9)	251 (10.5)	250 (14)	251 (3)	221 (2)	…	247 (10.5)	240 (16)	223 (19)	213 (23)	245 (12)	219 (22)	232 (18)	238 (17)	220 (21)
3	244 (13.5)	250 (7.5)	264 (7.5)	266 (5)	243 (6)	243 (3)	216 (1)	246 (17)	…	218 (23)	235 (18)	225 (22)	245 (12)	228 (21)	247 (9)	265 (2)	244 (13.5)	229 (20)
4	263 (8)	279 (2)	258 (14)	237 (3)	258 (5.5)	264 (9.5)	253 (15)	230 (5.5)	…	221 (22)	229 (18)	227 (19.5)	253 (12.5)	216 (24)	225 (21)	257 (11)	227 (19.5)	235 (16)
5	241 (14)	263 (7.5)	232 (3)	266 (1)	233 (23.5)	272 (20)	223 (6)	263 (4.5)	…	234 (18)	240 (15)	228 (21)	245 (12)	242 (13)	238 (16)	254 (10)	236 (17)	217 (23.5)
6	273 (2.5)	276 (1)	251 (8.5)	267 (14)	250 (10)	260 (11)	238 (4.5)	222 (8.5)	…	222 (22.5)	227 (21)	203 (24)	273 (2.5)	229 (20)	245 (13)	231 (19)	236 (17)	235 (18)
7	271 (7)	300 (1)	238 (4.5)	288 (12.5)	264 (14)	261 (19)	258 (4.5)	263 (6)	…	244 (15)	216 (24)	217 (22.5)	217 (22.5)	262 (10)	243 (16)	239 (18)	219 (21)	231 (20)
8	257 (1)	241 (7.5)	213 (10)	251 (6)	249 (16.5)	241 (21.5)	199 (3.5)	213 (3.5)	…	231 (14)	225 (18.5)	216 (20)	234 (11)	229 (15)	233 (12)	225 (18.5)	226 (16.5)	232 (13)
9	276 (2.5)	242 (17)	257 (11)	253 (4)	261 (5.5)	259 (10)	252 (13)	216 (14.5)	…	207 (24)	232 (19)	249 (16)	222 (22)	254 (12)	260 (8)	224 (20.5)	235 (18)	279 (1)
10	251 (8)	256 (6)	241 (4)	275 (9)	238 (16)	257 (10)	224 (1)	225 (7)	…	233 (13.5)	226 (18.5)	220 (23)	231 (16)	231 (16)	225 (20.5)	233 (13.5)	234 (12)	226 (18.5)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	240 (22)	268 (8.5)	290 (3)	254 (14.5)	251 (6)	265 (2)	237 (14.5)	244 (4)	…	268 (8.5)	262 (13)	251 (16.5)	225 (24)	266 (10)	244 (20.5)	278 (5)	272 (7)	248 (18.5)
100	270 (2)	262 (4)	255 (8)	251 (5)	257 (18)	249 (7)	227 (10.5)	252 (1)	…	232 (19.5)	232 (19.5)	207 (24)	264 (3)	234 (17)	249 (13.5)	237 (16)	250 (12)	212 (23)
$\sum R_{Ci}$	653	803.5	1 042	764	939.5	1 075	654	598	…	1 655.5	1 789	1 902	1 485.5	1 522.5	1 697.5	1 523.5	1 652	1 835.5
$\sum R_{Ci}^2$	426 409	645 612.25	1 085 764	583 696	882 660.25	1 155 625	427 716	357 604	…	2 740 680.25	3 200 521	3 617 604	2 206 710.25	2 318 006.25	2 881 506.25	2321052.25	2 729 104	3 369 060.25

Table B.25: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the NSGA-II) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A2.2.1–A2.2.24.*

	Worst		Best	
	HA	NDS	HA	NDS
A2.2.1	Run 17	Run 30	Run 7	Run 26
A2.2.2	Run 39	Run 60	Run 92	Run 7
A2.2.3	Run 72	Run 78	Run 88	Run 90
A2.2.4	Run 57	Run 23	Run 29	Run 76
A2.2.5	Run 17	Run 65	Run 62	Run 68
A2.2.6	Run 3	Run 8	Run 14	Run 85
A2.2.7	Run 3	Run 11	Run 79	Run 26
A2.2.8	Run 12	Run 1	Run 44	Run 28
A2.2.9	Run 12	Run 39	Run 25	Run 17
A2.2.10	Run 57	Run 88	Run 62	Run 79
A2.2.11	Run 72	Run 11	Run 79	Run 48
A2.2.12	Run 3	Run 22	Run 100	Run 37
A2.2.13	Run 3	Run 44	Run 9	Run 16
A2.2.14	Run 39	Run 44	Run 7	Run 79
A2.2.15	Run 83	Run 65	Run 54	Run 43
A2.2.16	Run 57	Run 60	Run 15	Run 26
A2.2.17	Run 3	Run 44	Run 21	Run 94
A2.2.18	Run 100	Run 44	Run 70	Run 71
A2.2.19	Run 94	Run 35	Run 74	Run 79
A2.2.20	Run 12	Run 38	Run 11	Run 71
A2.2.21	Run 17	Run 43	Run 73	Run 45
A2.2.22	Run 57	Run 61	Run 63	Run 27
A2.2.23	Run 57	Run 29	Run 36	Run 50
A2.2.24	Run 97	Run 89	Run 79	Run 24

Table B.26: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the hyperareas of the approximation fronts for the hyperparameter combinations A2.2.1–A2.2.24.*

	A2.2.2	A2.2.3	A2.2.4	A2.2.5	A2.2.6	A2.2.7	A2.2.8	A2.2.9	A2.2.10	A2.2.11	A2.2.12	A2.2.13	A2.2.14	A2.2.15	A2.2.16	A2.2.17	A2.2.18	A2.2.19	A2.2.20	A2.2.21	A2.2.22	A2.2.23	A2.2.24
A2.2.1	1	0.94	1	1	1	1	1	1	1	1	1	0.01	0	0.04	1	1	0	0	1	1	1	1	1
A2.2.2		1	1	1	1	1	1	1	1	1	0.01	1	1	1	1	1	1	1	1	1	1	1	1
A2.2.3			1	1	1	0.31	1	0.045	0.14	0.25	0	1	1	1	1	1	1	1	1	1	1	1	1
A2.2.4				1	1	1	1	1	1	1	1	0.11	0.04	0.41	1	1	0.02	0.03	1	1	1	1	1
A2.2.5					1	1	1	1	1	1	1	0.047	0.02	0.19	1	1	0.01	0.01	1	1	1	1	1
A2.2.6						1	1	1	1	1	0.045	1	0.94	1	1	1	0.44	0.72	1	1	1	1	1
A2.2.7							1	1	1	1	1	0	0	0.01	1	0.37	0	0	1	1	0.37	0.53	1
A2.2.8								1	1	1	1	0.01	0	0.05	1	1	0	0	1	1	1	1	1
A2.2.9									1	1	1	0	0	0	1	0.05	0	0	1	1	0.05	0.08	1
A2.2.10										1	1	0	0	0	1	0.17	0	0	1	1	0.17	0.26	1
A2.2.11											1	0	0	0.01	1	0.3	0	0	1	1	0.3	0.44	1
A2.2.12												0	0	0	0	0	0	0	0.01	0.67	0	0	0.77
A2.2.13													1	1	1	1	1	1	1	0.18	1	1	0.15
A2.2.14														1	1	1	1	1	1	0.07	1	1	0.06
A2.2.15															1	1	1	1	1	0.63	1	1	0.55
A2.2.16																1	1	1	1	1	1	1	1
A2.2.17																	1	1	1	1		1	1
A2.2.18																		1	1	0.03	1	1	0.02
A2.2.19																			1	0.049	1	1	0.04
A2.2.20																				1	1	1	1
A2.2.21																					1	1	1
A2.2.22																						1	1
A2.2.23																							1

Table B.27: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A2.2.1–A2.2.24.*

	A2.2.2	A2.2.3	A2.2.4	A2.2.5	A2.2.6	A2.2.7	A2.2.8	A2.2.9	A2.2.10	A2.2.11	A2.2.12	A2.2.13	A2.2.14	A2.2.15	A2.2.16	A2.2.17	A2.2.18	A2.2.19	A2.2.20	A2.2.21	A2.2.22	A2.2.23	A2.2.24
A2.2.1	1	0.03	1	1	0.01	1	1	1	1	1	0.53	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.2		1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.3			1	1	1	0.03	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.4				1	0.52	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0.52	0	0	0
A2.2.5					1	1	0.18	1	0.04	1	1	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.6						0.01	0	1	0	1	1	0	0	0	0	0	0	0.01	0	0	0	0	0
A2.2.7							1	1	1	1	0.55	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.8								1	1	1	0.07	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.9									1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.10										0.81	0.02	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.11											1	0	0	0	0	0	0	0	0	0	0	0	0
A2.2.12												0	0	0	0	0	0	0	0	0	0	0	0
A2.2.13													1	1	1	1	1	1	1	1	1	1	1
A2.2.14														1	1	1	1	1	1	1	1	1	1
A2.2.15															1	1	1	0.75	1	1	1	1	1
A2.2.16																1	1	1	1	1	1	1	1
A2.2.17																	1	0.66	1	1	1	1	1
A2.2.18																		0.01	0.04	1	0.04	1	1
A2.2.19																			1	1	1	1	0.13
A2.2.20																				1	1	1	0.48
A2.2.21																					1	1	1
A2.2.22																						1	0.5
A2.2.23																							1

Table B.28: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A2.3.1–A2.3.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.2.1	A2.2.2	A2.2.3	A2.2.4	A2.2.5	A2.2.6	A2.2.7	A2.2.8	...	A2.2.17	A2.2.18	A2.2.19	A2.2.20	A2.2.21	A2.2.22	A2.2.23	A2.2.24
1	33.57 (13)	33.63 (10)	33.54 (1)	33.64 (8)	33.62 (17)	33.67 (14)	33.67 (6)	33.63 (7)	...	32.79 (22)	22 (33.54)	33.58 (12)	33.54 (16)	32.82 (21)	31.96 (24)	33.26 (20)	33.38 (19)
2	33.01 (13)	33.31 (8)	33.45 (12)	32.97 (5)	33.24 (19)	32.26 (3)	32.95 (14)	32.25 (10)	...	33.3 (9)	33.38 (6)	32.23 (22)	32.95 (16)	33.54 (1)	32.75 (18)	33.46 (2)	33.37 (7)
3	32.18 (24)	32.73 (15)	33.45 (14)	32.38 (12)	33.46 (18)	33.44 (5)	32.56 (20)	33.44 (11)	...	32.33 (21)	32.75 (13)	33.39 (9)	32.32 (22)	32.72 (16)	33.47 (3)	33.39 (8)	33.5 (1)
4	33.61 (5)	33.65 (3)	33.65 (14)	33.47 (23)	33.65 (12)	33.48 (2)	33.5 (17)	33.54 (6)	...	33.46 (18)	33.55 (9)	33.4 (22)	33.53 (11)	33.44 (20)	33.44 (19)	33.64 (4)	33.52 (13)
5	32.3 (21)	32.21 (24)	33.42 (3)	33.3 (8)	32.97 (22)	33.27 (6)	33.39 (13)	33.23 (2)	...	33.31 (12)	32.61 (20)	33.43 (5)	32.69 (19)	33.42 (7)	33.34 (11)	33.35 (10)	33.66 (1)
6	32.03 (24)	33 (10)	32.64 (1)	32.14 (21)	33.31 (12)	33.43 (14)	32.74 (23)	33.4 (13)	...	33.16 (9)	32.42 (17)	32.44 (16)	32.26 (20)	32.33 (19)	32.18 (22)	32.4 (18)	33.4 (4)
7	33.45 (5)	32.56 (20)	32.26 (22)	33.38 (21)	33.32 (18)	33.47 (24)	33.34 (8)	33.37 (7)	...	33.48 (3)	33.07 (14)	33.04 (15)	33.01 (16)	32.7 (19)	32.95 (17)	33.56 (1)	33.49 (2)
8	33.37 (23)	33.68 (1)	33.42 (24)	33.47 (15)	33.54 (7)	33.63 (21)	33.67 (17)	33.57 (20)	...	33.54 (12)	33.56 (10)	33.64 (4)	33.53 (13)	33.47 (16)	33.64 (5)	33.46 (18)	33.67 (3)
9	32.94 (16)	33.02 (13)	33.23 (1)	32.26 (22)	33.25 (12)	33.52 (10)	32.53 (23)	32.69 (9)	...	32.79 (18)	32.63 (20)	32.14 (24)	33.21 (11)	33.33 (6)	32.97 (15)	33.36 (5)	33.43 (3)
10	33.6 (2)	33.44 (7)	33.36 (1)	33.44 (12)	32.22 (17)	33.34 (15)	33.36 (8)	32.73 (22)	...	33.41 (11)	33.43 (9)	33.32 (19)	33.53 (5)	33.42 (10)	32.4 (23)	33.37 (14)	33.56 (4)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	32.37 (22)	33.13 (10)	33.04 (16)	32.71 (1)	33.43 (23)	33.43 (14)	33.33 (18)	33.07 (9)	...	32.21 (24)	33.36 (6)	33.39 (4)	32.84 (15)	33.05 (13)	32.52 (19)	33.38 (5)	32.49 (20)
100	33.33 (11)	32.14 (23)	32.24 (3)	32.12 (14)	33.08 (6)	33.4 (22)	33.27 (24)	33.37 (16)	...	33.59 (1)	33.05 (17)	32.81 (20)	33.32 (12)	32.82 (19)	33.38 (9)	33.45 (4)	33.44 (5)
$\sum R_{Ci}$	1 372	1 132	1 265	1 319	1 229	1 188	1 337	1 109	...	1 252	1 107	1 508	1 269	1 042	1 421	1 111	1 075
$\sum R_{Ci}^2$	1 882 384	1 281 424	1 600 225	1 739 761	1 510 441	1 411 344	1 787 569	1 229 881	...	1 567 504	1 225 449	2 274 064	1 610 361	1 085 764	2 019 241	1 234 321	1 155 625

Table B.29: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A2.3.1–A2.3.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.3.1	A2.3.2	A2.3.3	A2.2.4	A2.2.5	A2.2.6	A2.2.7	A2.2.8	A2.2.9	...	A2.2.17	A2.2.18	A2.2.19	A2.2.20	A2.2.21	A2.2.22	A2.2.23	A2.2.24
1	160 (4.5)	152 (9)	143 (14)	118 (17)	103 (18)	128 (16)	164 (2)	162 (3)	144 (12.5)	...	88 (22)	83 (24)	88 (22)	88 (22)	89 (20)	153 (8)	157 (7)	144 (12.5)
2	92 (4)	73 (15)	99 (2)	69 (19)	68 (20)	80 (11)	87 (6)	84 (10)	103 (1)	...	85 (8)	85 (8)	74 (14)	90 (5)	93 (3)	71 (18)	85 (8)	72 (16.5)
3	98 (4.5)	92 (8.5)	106 (1.5)	89 (13)	93 (7)	98 (4.5)	82 (22)	92 (8.5)	106 (1.5)	...	88 (15)	97 (6)	77 (24)	84 (19.5)	89 (13)	85 (17.5)	80 (23)	91 (10)
4	123 (5)	121 (7)	110 (17.5)	111 (15.5)	114 (13)	117 (9)	144 (1)	122 (6)	125 (4)	...	127 (2.5)	111 (15.5)	120 (8)	115 (10.5)	99 (24)	127 (2.5)	106 (19)	102 (22)
5	69 (19.5)	78 (9)	71 (16)	75 (12.5)	64 (23)	79 (8)	66 (22)	92 (2)	95 (1)	...	85 (6)	70 (17.5)	56 (24)	70 (17.5)	89 (4)	68 (21)	74 (14)	91 (3)
6	72 (12.5)	78 (7)	84 (4)	76 (8.5)	74 (10)	54 (23)	57 (21)	76 (8.5)	81 (5.5)	...	72 (12.5)	53 (24)	69 (16.5)	64 (18)	71 (14)	62 (19)	73 (11)	81 (5.5)
7	100 (4)	81 (17.5)	79 (19)	81 (17.5)	104 (2)	90 (9)	102 (3)	105 (1)	87 (12.5)	...	65 (24)	77 (20.5)	76 (22.5)	96 (6)	87 (12.5)	95 (7)	89 (10)	83 (15.5)
8	120 (3.5)	99 (18)	87 (21)	131 (2)	113 (6.5)	82 (24)	105 (15)	107 (11)	134 (1)	...	106 (13)	96 (19)	86 (22)	115 (5)	120 (3.5)	83 (23)	109 (9)	104 (16.5)
9	56 (17)	66 (9)	87 (2)	55 (19)	56 (17)	68 (8)	53 (21)	63 (11)	90 (1)	...	54 (20)	75 (4.5)	37 (23)	57 (14.5)	76 (3)	33 (24)	61 (12.5)	75 (4.5)
10	76 (20)	66 (23)	89 (5)	88 (6.5)	84 (13.5)	86 (11)	82 (15.5)	87 (8.5)	79 (17)	...	73 (22)	87 (8.5)	65 (24)	88 (6.5)	84 (13.5)	74 (21)	82 (15.5)	100 (3)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	87 (18.5)	103 (6)	96 (10.5)	111 (2)	81 (23)	106 (5)	91 (15)	91 (15)	77 (24)	...	85 (21.5)	100 (7)	97 (9)	112 (1)	86 (20)	85 (21.5)	98 (8)	89 (17)
100	75 (16.5)	60 (23)	70 (20.5)	82 (9.5)	92 (2.5)	76 (15)	72 (19)	82 (9.5)	75 (16.5)	...	92 (2.5)	77 (14)	57 (24)	84 (6)	83 (7.5)	70 (20.5)	90 (4)	79 (12)
$\sum R_{Ci}$	1 313.50	1 144.50	1 050	1 276.50	1 131	1 190.50	1 086.50	906.50	879	...	1 423.50	1 447	1 604	1 374	1 159.50	1 563.50	1 224	1 132.50
$\sum R_{Ci}^2$	1 725 282.25	1 309 880.25	1 102 500	1 629 452.25	1 279 161	1 417 290.25	1 180 482.25	821 742.25	772 641	...	2 026 352.30	2 093 809	2 572 816	1 887 876	1 344 440.30	2 444 532.25	1 498 176	1 282 556

Table B.30: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the NSGA-II) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A2.3.1–A2.3.24.*

	Worst		Best	
	HA	NDS	HA	NDS
A2.3.1	Run 6	Run 44	Run 17	Run 1
A2.3.2	Run 34	Run 71	Run 89	Run 48
A2.3.3	Run 27	Run 97	Run 1	Run 39
A2.3.4	Run 44	Run 44	Run 95	Run 39
A2.3.5	Run 27	Run 82	Run 73	Run 39
A2.3.6	Run 34	Run 34	Run 96	Run 89
A2.3.7	Run 59	Run 9	Run 50	Run 89
A2.3.8	Run 21	Run 9	Run 76	Run 1
A2.3.9	Run 44	Run 12	Run 16	Run 39
A2.3.10	Run 34	Run 34	Run 89	Run 1
A2.3.11	Run 34	Run 98	Run 13	Run 1
A2.3.12	Run 66	Run 2	Run 89	Run 1
A2.3.13	Run 44	Run 44	Run 1	Run 15
A2.3.14	Run 39	Run 44	Run 7	Run 79
A2.3.14	Run 12	Run 60	Run 84	Run 1
A2.3.15	Run 28	Run 2	Run 63	Run 1
A2.3.16	Run 44	Run 44	Run 89	Run 15
A2.3.17	Run 32	Run 32	Run 16	Run 48
A2.3.18	Run 45	Run 6	Run 25	Run 65
A2.3.19	Run 21	Run 9	Run 78	Run 39
A2.3.20	Run 71	Run 12	Run 25	Run 18
A2.3.21	Run 87	Run 12	Run 94	Run 39
A2.3.22	Run 97	Run 9	Run 96	Run 15
A2.3.23	Run 44	Run 1	Run 49	Run 39
A2.3.24	Run 21	Run 97	Run 74	Run 15

Table B.31: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the hyperareas of the approximation fronts for the hyperparameter combinations A2.3.1–A2.3.24.*

	A2.3.2	A2.3.3	A2.3.4	A2.3.5	A2.3.6	A2.3.7	A2.3.8	A2.3.9	A2.3.10	A2.3.11	A2.3.12	A2.3.13	A2.3.14	A2.3.15	A2.3.16	A2.3.17	A2.3.18	A2.3.19	A2.3.20	A2.3.21	A2.3.22	A2.3.23	A2.3.24
A2.3.1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.27	1	1	0.82
A2.3.2		1	1	1	1	1	1	1	0.38	1	1	0.26	1	1	0.53	1	1	0.047	1	1	1	1	1
A2.3.3			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.3.4				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.3.5					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.3.6						1	1	1	1	1	1	1	1	1	1	1	1	0.38	1	1	1	1	1
A2.3.7							1	1	1	1	1	1	1	1	1	1	1	1	1	0.88	1	1	1
A2.3.8								1	0.17	1	1	0.11	1	1	0.24	1	1	0.02	1	1	0.5	1	1
A2.3.9									0.14	1	1	0.09	1	1	0.2	1	1	0.02	1	1	0.42	1	1
A2.3.10										1	0.13	1	1	1	1	1	0.16	1	1	0.01	1	0.18	0.045
A2.3.11											1	1	1	1	1	1	1	0.998	1	1	1	1	1
A2.3.12												0.09	1	1	0.19	1	1	0.01	1	1	0.39	1	1
A2.3.13													1	1	1	1	0.10	1	1	0.01	1	0.12	0.03
A2.3.14														1	1	1	1	1	1	1	1	1	1
A2.3.15															1	1	1	0.22	1	1	1	1	1
A2.3.16																1	0.22	1	1	0.02	1	0.26	0.07
A2.3.17																	1	1	1	1	1	1	1
A2.3.18																		0.02	1	1	0.47	1	1
A2.3.19																			1	0	1	0.02	0
A2.3.20																				1	1	1	1
A2.3.21																					0.04	1	1
A2.3.22																						0.53	0.15
A2.3.23																							1

Table B.32: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A2.3.1–A2.3.24.*

	A2.3.2	A2.3.3	A2.3.4	A2.3.5	A2.3.6	A2.3.7	A2.3.8	A2.3.9	A2.3.10	A2.3.11	A2.3.12	A2.3.13	A2.3.14	A2.3.15	A2.3.16	A2.3.17	A2.3.18	A2.3.19	A2.3.20	A2.3.21	A2.3.22	A2.3.23	A2.3.24
A2.3.1	1	1	1	1	1	1	0.013	0	1	0.004	0.048	0.01	1	1	0.21	1	1	1	1	1	1	1	1
A2.3.2		1	1	1	1	1	1	1	1	1	1	0	0.7	1	0	1	0.7	0.001	1	1	0.01	1	1
A2.3.3			1	1	1	1	1	1	1	1	1	0	0.02	0.98	0	0.05	0.02	0	0.33	1	0	1	1
A2.3.4				1	1	1	0.06	0.02	1	0.02	0.2	0	1	1	0.05	1	1	0.29	1	1	1	1	1
A2.3.5					1	1	1	1	1	1	1	0	0.44	1	0	0.95	0.44	0	1	1	0	1	1
A2.3.6						1	1	0.51	1	0.54	1	0	1	1	0	1	1	0.01	1	1	0.05	1	1
A2.3.7							1	1	1	1	1	0	0.09	1	0	0.21	0.09	0	1	1	0	1	1
A2.3.8								1	1	1	1	0	0	0	0	0	0	0	0	1	0	0.41	1
A2.3.9									1	1	1	0	0	0	0	0	0	0	0	1	0	0.16	1
A2.3.10										1	1	0	0.231	1	0	0.57	0.23	0	1	1	0.002	1	1
A2.3.11											1	0	0	0	0	0	0	0	0	1	0	0.17	1
A2.3.12												0	0	0.02	0	0	0	0	0	1	0	1	1
A2.3.13													1	0.04	1	0.72	1	1	0.13	0	1	0	0
A2.3.14														1	1	1	1	1	1	1	1	1	0.47
A2.3.15															0.56	1	1	1	1	1	1	1	1
A2.3.16																1	1	1	1	0	1	0.01	0
A2.3.17																	1	1	1	1	1	1	1
A2.3.18																		1	1	1	1	1	0.458
A2.3.19																			1	0	1	0.04	0
A2.3.20																				1	1	1	1
A2.3.21																					0.02	1	1
A2.3.22																						0.19	0.01
A2.3.23																							1

Table B.33: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A2.4.1–A2.4.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.4.1	A2.4.2	A2.4.3	A2.4.4	A2.4.5	A2.4.6	A2.4.7	A2.4.8	...	A2.4.17	A2.4.18	A2.4.19	A2.4.20	A2.4.21	A2.4.22	A2.4.23	A2.4.24
1	21.14 (10)	20.91 (21)	20.91 (20)	21.01 (17)	21.17 (9)	21.47 (1)	21.1 (12)	21.07 (13)	...	21 (19)	21.3 (6)	21 (18)	21.17 (8)	21.36 (4)	20.62 (23)	21.01 (16)	21.03 (14)
2	21.49 (3)	21.08 (13)	21.33 (6)	20.85 (18)	21.15 (10)	21.36 (5)	20.98 (15)	21.68 (2)	...	20.74 (20)	21.17 (9)	20.88 (17)	21.28 (8)	20.36 (21)	19.95 (22)	20.9 (16)	21.29 (7)
3	20.4 (22)	20.66 (21)	21.21 (10)	21.46 (4)	21.15 (15)	20.8 (20)	21.16 (13)	21.26 (7)	...	20.97 (19)	21.18 (12)	21.25 (8)	21.58 (3)	21.19 (11)	21.15 (14)	21.32 (5)	21.82 (1)
4	21.15 (21)	21.7 (3)	21.58 (4)	21.43 (10)	21.5 (8)	21.41 (12)	21.21 (20)	21.3 (17)	...	21.73 (2)	21.56 (5)	21.34 (16)	21.3 (18)	21.52 (7)	20.97 (24)	21.1 (23)	21.41 (13)
5	21.69 (4)	21.47 (8)	21.46 (9)	20.92 (21)	21.77 (2)	21.64 (5)	21.3 (13)	21.32 (11)	...	21.28 (14)	21.26 (16)	20.59 (22)	21.24 (17)	21.84 (1)	21.53 (7)	21.02 (20)	21.14 (19)
6	21.75 (1)	20.46 (20)	21.37 (5)	21.21 (10)	21.29 (7)	21.38 (4)	21.12 (13)	21.24 (8)	...	21.12 (14)	21.03 (15)	20.57 (19)	19.72 (24)	20.58 (18)	21.2 (11)	21.46 (2)	20.93 (16)
7	21.17 (11)	20.86 (24)	21.3 (6)	20.99 (20)	21.18 (9)	21.26 (8)	20.97 (21)	21.37 (4)	...	21.17 (10)	21.01 (18)	21.08 (14)	21.5 (2)	21.02 (17)	21.05 (15)	20.93 (22)	21.54 (1)
8	21.61 (6)	21.43 (18)	21.31 (21)	21.16 (24)	21.75 (2)	21.58 (10)	21.8 (1)	21.48 (16)	...	21.51 (15)	21.4 (19)	21.57 (11)	21.6 (8)	21.71 (3)	21.48 (17)	21.6 (7)	21.27 (22)
9	21.53 (5)	21.34 (10)	20.44 (24)	20.89 (20)	21.64 (3)	21.57 (4)	21.34 (9)	21.48 (6)	...	20.7 (22)	21.74 (1)	21.33 (11)	21.2 (17)	21.13 (18)	21.2 (15)	21.05 (19)	21.38 (8)
10	20.53 (23)	21.21 (10)	20.94 (20)	21.33 (5)	21 (17)	21.06 (16)	21.2 (12)	20.64 (22)	...	20.95 (18)	21.12 (15)	21.29 (6)	21.6 (1)	21.2 (11)	20.66 (21)	21.44 (3)	21.27 (7)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	21.51 (5)	21.41 (8)	21.07 (23)	21.1 (22)	21.21 (17)	21.43 (7)	21.39 (9)	21.33 (13)	...	21.61 (2)	20.91 (24)	21.54 (4)	21.37 (11)	21.14 (20)	21.1 (21)	21.45 (6)	21.57 (3)
100	20.94 (22)	21.45 (6)	21.63 (2)	21.29 (9)	21.05 (17)	21.08 (16)	21.18 (11)	21.64 (1)	...	20.95 (20)	21.1 (14)	20.96 (18)	21.15 (13)	20.95 (21)	20.87 (23)	21.09 (15)	21.4 (8)
$\sum R_{Ci}$	1 457	1 376	1 120	1 476	1 152	1 109	1 395	1 008	...	1 336	1 201	1 539	1 248	969	1 558	1 146	993
$\sum R_{Ci}^2$	2 122 849	1 893 376	1 254 400	2 178 576	1 327 104	1 229 881	1 946 025	1 016 064	...	1 784 896	1 442 401	2 368 521	1 557 504	938 961	2 427 364	1 313 316	986 049

Table B.34: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A2.4.1–A2.4.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.4.1	A2.4.2	A2.4.3	A2.4.4	A2.4.5	A2.4.6	A2.4.7	A2.4.8	A2.4.9	...	A2.4.16	A2.4.17	A2.4.18	A2.4.19	A2.4.20	A2.4.21	A2.4.22	A2.4.23	A2.4.24
1	57 (2)	49 (11.5)	45 (16.5)	56 (3)	52 (8)	54 (5.5)	61 (1)	43 (21)	50 (9.5)	...	44 (18.5)	43 (21)	47 (14.5)	50 (9.5)	53 (7)	49 (11.5)	48 (13)	38 (24)	43 (21)
2	45 (13.5)	50 (6)	34 (24)	45 (13.5)	43 (15.5)	54 (2.5)	48 (9)	47 (11)	42 (17.5)	...	38 (21)	47 (11)	54 (2.5)	42 (17.5)	62 (1)	49 (7.5)	39 (20)	52 (5)	47 (11)
3	56 (7)	49 (16)	51 (13)	55 (8.5)	63 (1)	46 (21)	54 (10)	52 (11)	47 (18.5)	...	59 (4)	59 (4)	57 (6)	46 (21)	43 (24)	51 (13)	59 (4)	51 (13)	55 (8.5)
4	53 (15.5)	58 (8.5)	58 (8.5)	50 (18.5)	55 (13)	49 (20.5)	52 (17)	67 (2.5)	73 (1)	...	47 (23)	67 (2.5)	53 (15.5)	57 (11)	58 (8.5)	58 (8.5)	45 (24)	66 (4)	64 (6)
5	50 (15)	53 (12)	53 (12)	52 (14)	46 (20.5)	57 (5)	54 (9.5)	47 (19)	56 (6)	...	55 (7.5)	48 (17.5)	62 (1.5)	48 (17.5)	40 (24)	61 (3)	43 (22)	55 (7.5)	53 (12)
6	52 (3)	51 (6)	42 (16.5)	43 (15)	51 (6)	40 (19.5)	35 (24)	47 (12)	49 (10)	...	40 (19.5)	44 (14)	47 (12)	40 (19.5)	39 (22)	52 (3)	47 (12)	52 (3)	42 (16.5)
7	56 (3)	47 (17)	47 (17)	49 (13.5)	49 (13.5)	49 (13.5)	40 (23)	60 (1.5)	54 (5)	...	50 (10.5)	47 (17)	49 (13.5)	50 (10.5)	52 (7)	39 (24)	52 (7)	55 (4)	60 (1.5)
8	60 (11)	65 (3.5)	62 (7)	61 (9)	52 (20.5)	50 (22.5)	60 (11)	64 (5)	57 (15)	...	66 (2)	57 (15)	56 (17.5)	52 (20.5)	59 (13)	65 (3.5)	60 (11)	62 (7)	56 (17.5)
9	50 (6)	38 (23)	51 (4.5)	44 (14)	60 (1)	46 (10.5)	51 (4.5)	57 (2)	49 (7)	...	41 (18.5)	45 (12)	47 (8.5)	41 (18.5)	46 (10.5)	41 (18.5)	39 (22)	40 (21)	47 (8.5)
10	41 (20)	55 (5)	43 (17)	54 (6)	38 (23.5)	45 (11.5)	57 (3.5)	43 (17)	57 (3.5)	...	44 (14)	41 (20)	48 (8.5)	43 (17)	45 (11.5)	41 (20)	39 (22)	48 (8.5)	61 (2)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	62 (4.5)	60 (8)	65 (3)	58 (10.5)	48 (22)	52 (18)	50 (20.5)	61 (6.5)	57 (13)	...	50 (20.5)	56 (15)	59 (9)	58 (10.5)	47 (23)	67 (1)	46 (24)	55 (16.5)	57 (13)
100	51 (5.5)	47 (16.5)	49 (13)	51 (5.5)	44 (20.5)	56 (1)	48 (15)	52 (3.5)	53 (2)	...	45 (19)	50 (9)	50 (9)	49 (13)	44 (20.5)	50 (9)	52 (3.5)	43 (22)	50 (9)
$\sum R_{Ci}$	1 270	1 182	1 262	1 235	1 124	1 262	1 083	966	1 078	...	1 473	1 381	1 262	1 456	1 278	1 218	1 542	1 207	1 178
$\sum R_{Ci}^2$	1 611 630.25	1 395 942.3	1 591 382.25	1 523 990.3	1 263 376	1 592 644	1 172 889	932 190	1 161 006	...	2 168 256.25	1 907 161	1 591 382.25	2 119 936	1 632 006.25	1 483 524	2 377 764	1 456 849	1 387 684

Table B.35: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the NSGA-II) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A2.4.1–A2.4.24.*

	Worst		Best	
	HA	NDS	HA	NDS
A2.4.1	Run 66	Run 66	Run 6	Run 39
A2.4.2	Run 79	Run 26	Run 25	Run 16
A2.4.3	Run 83	Run 58	Run 98	Run 15
A2.4.4	Run 44	Run 42	Run 77	Run 50
A2.4.5	Run 32	Run 12	Run 5	Run 96
A2.4.6	Run 59	Run 21	Run 15	Run 84
A2.4.7	Run 79	Run 82	Run 24	Run 18
A2.4.8	Run 82	Run 68	Run 38	Run 15
A2.4.9	Run 83	Run 58	Run 76	Run 4
A2.4.10	Run 12	Run 64	Run 63	Run 15
A2.4.11	Run 28	Run 9	Run 76	Run 73
A2.5.12	Run 71	Run 60	Run 70	Run 19
A2.5.13	Run 68	Run 27	Run 15	Run 18
A2.5.14	Run 2	Run 2	Run 4	Run 36
A2.5.15	Run 74	Run 82	Run 47	Run 18
A2.5.16	Run 26	Run 26	Run 46	Run 16
A2.5.17	Run 58	Run 26	Run 4	Run 39
A2.5.18	Run 21	Run 71	Run 81	Run 65
A2.4.19	Run 13	Run 13	Run 50	Run 39
A2.4.20	Run 6	Run 44	Run 63	Run 15
A2.4.21	Run 59	Run 82	Run 96	Run 15
A2.5.22	Run 74	Run 12	Run 75	Run 73
A2.5.23	Run 74	Run 55	Run 33	Run 96
A2.5.24	Run 44	Run 44	Run 3	Run 15

Table B.36: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the hyperareas of the approximation fronts for the hyperparameter combinations A2.4.1–A2.4.24.*

	A2.4.2	A2.4.3	A2.4.4	A2.4.5	A2.4.6	A2.4.7	A2.4.8	A2.4.9	A2.4.10	A2.4.11	A2.4.12	A2.4.13	A2.4.14	A2.4.15	A2.4.16	A2.4.17	A2.4.18	A2.4.19	A2.4.20	A2.4.21	A2.4.22	A2.4.23	A2.4.24
A2.4.1	1	0.207	1	0.63	0.138	1	0.002	0	1	0.003	0	0.88	1	0.52	1	1	1	1	1	0	1	0.52	0
A2.4.2		1	1	1	1	1	0.06	0	1	0.10	0	0.047	1	1	1	1	1	1	1	0.01	1	1	0.035
A2.4.3			0.102	1	1	1	1	1	0.193	1	0.7	0	1	1	0.01	1	1	0.01	1	1	0.003	1	1
A2.4.4				0.33	0.067	1	0	0	1	0	0	1	1	0.27	1	1	1	1	0.33	0.07	1	0	1
A2.4.5					1	1	1	1	0.591	1	0.23	0	1	1	0.04	1	1	0.03	1	1	0.01	1	1
A2.4.6						1	1	1	0.128	1	0.998	0	1	1	0.01	1	1	0.005	1	1	0.002	1	1
A2.4.7							0.03	0	1	0.049	0	0.1	1	1	1	1	1	1	1	0.01	1	1	0.016
A2.4.8								1	0	1	1	0	0.97	1	0	0.29	1	0	1	1	0	1	1
A2.4.9									0	1	1	0	0.01	1	0	0	0.47	0	0.09	1	0	1	1
A2.4.10										0.003	0	0.94	1	0.48	1	1	1	1	1	0	1	0.48	0.001
A2.4.11											1	0	1	1	0	0.44	1	0	1	1	0	1	1
A2.4.12												0	0	0.29	0	0	0.04	0	0.01	1	0	0.29	1
A2.4.13													0.002	0	1	0.009	0	1	0	0	1	0	0
A2.4.14														1	1	1	1	1	1	0.26	1	1	0.59
A2.4.15															0.03	1	1	0.02	1	1	0.01		1
A2.4.16																1	0.23	1	1	0	1	0.028	0
A2.4.17																	1	1	1	0.07	1	1	0.17
A2.4.18																		0.2	1	1	0.1	1	1
A2.4.19																			0.998	0	1	0.02	0
A2.4.20																				1	0.53	1	1
A2.4.21																					0	1	1
A2.4.22																						0.01	0
A2.4.23																							1

Table B.37: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A2.4.1–A2.4.24.*

	A2.4.2	A2.4.3	A2.4.4	A2.4.5	A2.4.6	A2.4.7	A2.4.8	A2.4.9	A2.4.10	A2.4.11	A2.4.12	A2.4.13	A2.4.14	A2.4.15	A2.4.16	A2.4.17	A2.4.18	A2.4.19	A2.4.20	A2.4.21	A2.4.22	A2.4.23	A2.4.24
A2.1	1	1	1	1	1	1	0.65	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.2		1	1	1	1	1	1	1	1	1	1	0.78	1	1	1	1	1	1	1	1	0.09	1	1
A2.3			1	1	1	1	0.85	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1
A2.4				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5					1	1	1	1	1	1	1	0.1	1	1	0.14	1	1	0.25	1	1	0.01	1	1
A2.6						1	0.84	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.7							1	1	1	1	1	0.02	1	0.71	0.03	0.8	1	0.05	1	1	0	1	1
A2.8								1	0.8	1	1	0	0.15	0.01	0	0.01	0.85	0	0.5	1	0	1	1
A2.9									1	1	1	0.02	1	0.59	0.02	0.66	1	0.04	1	1	0	1	1
A2.10										1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.11											1	0.01	1	0.5	0.02	0.56	1	0.03	1	1	0	1	1
A2.12												0	1	0.14	0	0.16	1	0.01	1	1	0	1	1
A2.13													1	1	1	1	1	1	1	1	1	1	0.70
A2.14														1	1	1	1	1	1	1	1	1	1
A2.15															1	1	1	1	1	1	1	1	1
A2.16																1	1	1	1	1	1	1	0.89
A2.17																	1	1	1	1	1	1	1
A2.18																		1	1	1	1	1	1
A2.19																			1	1	1	1	1
A2.20																				1	1	1	1
A2.21																					0.33	1	1
A2.22																						0.22	0.08
A2.23																							1

Table B.38: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A2.5.1–A2.5.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.5.1	A2.5.2	A2.5.3	A2.5.4	A2.5.5	A2.5.6	A2.5.7	...	A2.5.18	A2.5.19	A2.5.20	A2.5.21	A2.5.22	A2.5.23	A2.5.24
1	189.3 (4)	182.59 (18)	190.3 (2)	184.15 (17)	187.22 (8)	186.92 (10)	187.32 (7)	...	171.01 (24)	188.02 (6)	185.09 (12)	184.28 (16)	173.39 (23)	184.79 (14)	185.03 (13)
2	184.32 (6)	185.1 (4)	187 (1)	184.21 (7)	186.72 (2)	182.03 (10)	178.3 (16)	...	177.46 (18)	170.46 (23)	171.33 (22)	178.9 (14)	179.35 (12)	173.94 (19)	180.73 (11)
3	182.79 (14)	188.7 (3)	184.3 (11)	184.83 (10)	178.9 (20)	189.07 (2)	187.33 (5)	...	185.1 (8)	188.49 (4)	183.2 (12)	176.94 (21)	176.9 (22)	176.61 (23)	180 (18)
4	188.2 (5)	188.31 (4)	187.59 (11)	182.64 (23)	186.85 (13)	184.99 (16)	185.48 (15)	...	183.97 (21)	186.97 (12)	180.65 (24)	188.09 (6)	189.65 (2)	186.25 (14)	187.59 (10)
5	184.09 (4)	166.69 (24)	184.17 (3)	180.4 (10)	172.25 (22)	173.34 (19)	188.94 (2)	...	182.2 (6)	168.6 (23)	178.63 (15)	177.83 (17)	172.33 (21)	179.07 (14)	179.37 (12)
6	170.05 (20)	176.37 (14)	184.5 (4)	173.32 (17)	176.82 (13)	186.25 (2)	183.45 (6)	...	169.35 (21)	168.88 (22)	153.29 (24)	178.49 (12)	171.64 (19)	181.48 (8)	183.6 (5)
7	187.12 (3)	175.97 (24)	183.32 (11)	184.58 (9)	182.87 (14)	183.87 (10)	179.89 (20)	...	176.91 (23)	181.14 (15)	178.82 (21)	185.33 (6)	187.12 (4)	183.03 (12)	188.91 (2)
8	180.54 (19)	180.72 (18)	178.14 (22)	179.32 (21)	184.59 (11)	185.58 (8)	177.79 (23)		184.76 (10)	183.14 (15)	177.46 (24)	187.84 (2)	182.38 (16)	188.77 (1)	183.37 (14)
9	174.35 (18)	162.99 (24)	187.28 (2)	164.97 (23)	182.21 (7)	179.22 (12)	166.65 (21)	...	181.45 (8)	182.83 (5)	176.57 (16)	176.01 (17)	179.73 (11)	182.44 (6)	186.35 (3)
10	189.49 (7)	185.67 (15)	177.2 (24)	183.97 (19)	181.89 (21)	187.2 (12)	179.9 (22)	...	184.79 (16)	183.97 (18)	195.28 (1)	192.89 (2)	191.39 (4)	186.23 (13)	183.41 (20)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	171.6 (19)	179.42 (10)	185.52 (4)	169.9 (20)	189.76 (1)	184.55 (6)	187.44 (2)	...	173.97 (16)	176.26 (14)	184.45 (7)	171.85 (18)	173.97 (17)	177.28 (13)	169.71 (21)
100	182.62 (9)	179.04 (18)	182.57 (10)	180.54 (14)	181.02 (12)	180.97 (13)	190.64 (2)	...	176.73 (20)	175.96 (22)	180.35 (15)	190.09 (3)	167.84 (24)	183.8 (8)	176.06 (21)
$\sum R_{Ci}$	1 365	1 313	1 129	1 411	1 235	1 236	1 133	...	1 312	1 400	1 229	1 211	1 386	1 337	1 117
$\sum R_{Ci}^2$	1 863 225	1 723 969	1 274 641	1 990 921	1 525 225	1 527 696	1 283 689	...	1 721 344	1 960 000	1 510 441	1 466 521	1 920 996	1 787 569	1 247 689

Table B.39: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A2.5.1–A2.5.24. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A2.5.1	A2.5.2	A2.5.3	A2.5.4	A2.5.5	A2.5.6	A2.5.7	A2.5.8	...	A2.5.18	A2.5.19	A2.5.20	A2.5.21	A2.5.22	A2.5.23	A2.5.24
1	34 (22.5)	41 (16.5)	51 (2)	38 (19)	43 (14)	46 (6)	47 (4.5)	41 (16.5)	...	44 (10.5)	49 (3)	45 (7.5)	52 (1)	33 (24)	47 (4.5)	43 (14)
2	37 (14.5)	42 (9.5)	45 (5)	29 (24)	32 (21.5)	39 (12.5)	34 (20)	48 (2)	...	43 (8)	36 (17)	30 (23)	42 (9.5)	32 (21.5)	37 (14.5)	44 (6.5)
3	48 (3.5)	49 (2)	37 (22.5)	39 (21)	43 (13.5)	40 (18.5)	40 (18.5)	42 (15)	...	44 (12)	41 (16)	45 (9)	45 (9)	40 (18.5)	45 (9)	45 (9)
4	50 (6)	46 (12)	38 (22)	44 (15)	50 (6)	40 (19)	36 (24)	47 (10)	...	52 (2)	39 (20.5)	43 (17)	50 (6)	52 (2)	46 (12)	50 (6)
5	50 (2.5)	36 (19.5)	36 (19.5)	40 (13)	42 (10)	42 (10)	42 (10)	38 (16.5)	...	31 (24)	35 (21)	51 (1)	38 (16.5)	34 (22)	40 (13)	50 (2.5)
6	32 (24)	46 (6)	37 (18.5)	39 (14)	51 (1.5)	51 (1.5)	47 (5)	34 (23)	...	40 (11)	38 (16)	37 (18.5)	37 (18.5)	35 (21.5)	39 (14)	44 (7)
7	33 (24)	36 (22)	41 (13.5)	39 (18)	44 (8.5)	40 (16)	45 (7)	42 (12)	...	35 (23)	40 (16)	44 (8.5)	41 (13.5)	47 (3.5)	43 (10.5)	46 (5.5)
8	33 (23.5)	54 (1)	43 (16)	38 (21)	47 (10)	49 (6.5)	47 (10)	43 (16)	...	50 (5)	43 (16)	47 (10)	40 (18.5)	44 (13.5)	52 (3)	47 (10)
9	38 (12.5)	36 (15)	38 (12.5)	32 (20.5)	44 (5)	43 (6.5)	39 (11)	42 (8)	...	43 (6.5)	41 (9.5)	31 (22)	46 (4)	36 (15)	41 (9.5)	47 (3)
10	39 (21)	48 (8)	40 (20)	48 (8)	45 (12.5)	43 (17)	50 (4.5)	49 (6)	...	44 (15.5)	48 (8)	58 (1)	38 (22.5)	46 (10)	44 (15.5)	51 (3)
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
99	36 (17.5)	43 (7.5)	48 (2)	37 (16)	52 (1)	42 (11)	45 (4)	42 (11)	...	32 (20.5)	31 (22.5)	45 (4)	35 (19)	43 (7.5)	32 (20.5)	31 (22.5)
100	29 (23.5)	41 (11.5)	40 (13.5)	33 (21.5)	41 (11.5)	48 (2)	47 (3.5)	40 (13.5)	...	29 (23.5)	38 (17)	44 (7)	49 (1)	33 (21.5)	42 (10)	37 (18.5)
$\sum R_{Ci}$	1 344	1 304	1 306	1 351	1 211	1 157	1 056	1 073	...	1 456	1 422	1 060	1 243	1 431	1 225	1 165
$\sum R_{Ci}^2$	1 804 992.25	1 700 416	1 704 330.25	1 823 850.25	1 466 521	1 338 649	1 114 080.25	1 150 256.25	...	2 118 480.25	2 020 662.25	1 123 600	1 543 806.25	2 046 330.25	1 500 625	1 357 225

Table B.40: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the NSGA-II) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A2.5.1–A2.5.24.*

	Worst		Best	
	HA	NDS	HA	NDS
A2.5.1	Run 13	Run 13	Run 84	Run 63
A2.5.2	Run 9	Run 34	Run 73	Run 25
A2.5.3	Run 44	Run 79	Run 47	Run 38
A2.5.4	Run 75	Run 57	Run 81	Run 73
A2.5.5	Run 28	Run 29	Run 18	Run 99
A2.5.6	Run 75	Run 16	Run 39	Run 35
A2.5.7	Run 9	Run 57	Run 16	Run 47
A2.5.8	Run 6	Run 74	Run 47	Run 18
A2.5.9	Run 68	Run 32	Run 42	Run 78
A2.5.10	Run 27	Run 99	Run 84	Run 74
A2.5.11	Run 91	Run 98	Run 93	Run 16
A2.5.12	Run 34	Run 87	Run 10	Run 82
A2.5.13	Run 75	Run 98	Run 20	Run 52
A2.5.14	Run 75	Run 9	Run 73	Run 93
A2.5.15	Run 27	Run 92	Run 39	Run 96
A2.5.16	Run 75	Run 28	Run 73	Run 19
A2.5.17	Run 99	Run 60	Run 56	Run 86
A2.5.18	Run 27	Run 75	Run 47	Run 92
A2.5.19	Run 75	Run 72	Run 86	Run 36
A2.5.20	Run 6	Run 2	Run 10	Run 48
A2.5.21	Run 75	Run 46	Run 39	Run 81
A2.5.22	Run 32	Run 2	Run 36	Run 96
A2.5.23	Run 75	Run 97	Run 39	Run 92
A2.5.24	Run 75	Run 26	Run 62	Run 33

Table B.41: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the hyperareas of the approximation fronts for the hyperparameter combinations A2.5.1–A2.5.24.*

	A2.5.2	A2.5.3	A2.5.4	A2.5.5	A2.5.6	A2.5.7	A2.5.8	A2.5.9	A2.5.10	A2.5.11	A2.5.12	A2.5.13	A2.5.14	A2.5.15	A2.5.16	A2.5.17	A2.5.18	A2.5.19	A2.5.20	A2.5.21	A2.5.22	A2.5.23	A2.5.24
A2.5.1	1	1	1	1	1	1	1	0	1	1	0.02	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.2		1	1	1	1	1	1	0	1	1	0.17	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.3			1	1	1	1	1	0.09	1	1	1	1	1	1	0.12	1	1	1	1	1	1	1	1
A2.5.4				1	1	1	1	0	1	0.72	0	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.5					1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.6						1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.7							1	0.08	1	1	1	1	1	1	0.14	1	1	1	1	1	1	1	1
A2.5.8								0	1	1	1	1	1	1	1	1	1	1		1	1	1	1
A2.5.9									0.002	0.18	1	0	0	0	0	0	0	0	0	0	0	0	0.14
A2.5.10										1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.11											1	1	1	1	0.06	1	1	1	1	1	1	1	1
A2.5.12												0.02	0.01	0.44	0	0.02	0.18	0	1	1	0.01	0.07	1
A2.5.13													1	1	1	1	1	1	1	1	1	1	1
A2.5.14														1	1	1	1	1	1	1		1	1
A2.5.15															1	1	1	1	1	1	1	1	1
A2.5.16																1	1	1	1	1	1	1	0.08
A2.5.17																	1	1	1	1	1	1	1
A2.5.18																		1	1	1	1	1	1
A2.5.19																			1	1	1	1	1
A2.5.20																				1	1	1	1
A2.5.21																					1	1	1
A2.5.22																						1	1
A2.5.23																							1

Table B.42: *The adjusted p-values obtained by the Nemenyi post hoc test for the multiple comparisons based on the number of non-dominated solutions found for the approximation fronts for the hyperparameter combinations A2.5.1–A2.5.24.*

	A2.5.2	A2.5.3	A2.5.4	A2.5.5	A2.5.6	A2.5.7	A2.5.8	A2.5.9	A2.5.10	A2.5.11	A2.5.12	A2.5.13	A2.5.14	A2.5.15	A2.5.16	A2.5.17	A2.5.18	A2.5.19	A2.5.20	A2.5.21	A2.5.22	A2.5.23	A2.5.24
A2.5.1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.2		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.3			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.4				1	1	0.88	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.5					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.6						1	1	1	1	1	1	1	1	1	0.63	1	0.78	1	1	1	1	1	1
A2.5.7							1	1	1	1	1	1	1	1	0.01	0.37	0.02	0.07	1	1	0.049	1	1
A2.5.8								1	1	1	1	1	1	1	0.03	0.65	0.04	0.13	1	1	0.09	1	1
A2.5.9									1	1	1	1	1	1	0.04	0.81	0.046	0.17	1	1	0.12	1	1
A2.5.10										1	1	1	1	1	1	1	1	1	1	1	1	1	1
A2.5.11											1	1	1	1	0.23	1	0.29	0.92	1	1	0.69	1	1
A2.5.12												1	1	1	1	1	1	1	1	1	1	1	1
A2.5.13													1	1	1	1	1	1	1	1	1	1	1
A2.5.14														1	1	1	1	1	1	1	1	1	1
A2.5.15															1	1	1	1	1	1	1	1	1
A2.5.16																1	1	1	0.02	1	1	1	0.82
A2.5.17																	1	1	0.43	1	1	1	1
A2.5.18																		1	0.02	1	1	1	1
A2.5.19																			0.08	1	1	1	1
A2.5.20																				1	0.06	1	1
A2.5.21																					1	1	1
A2.5.22																						1	1
A2.5.23																							1

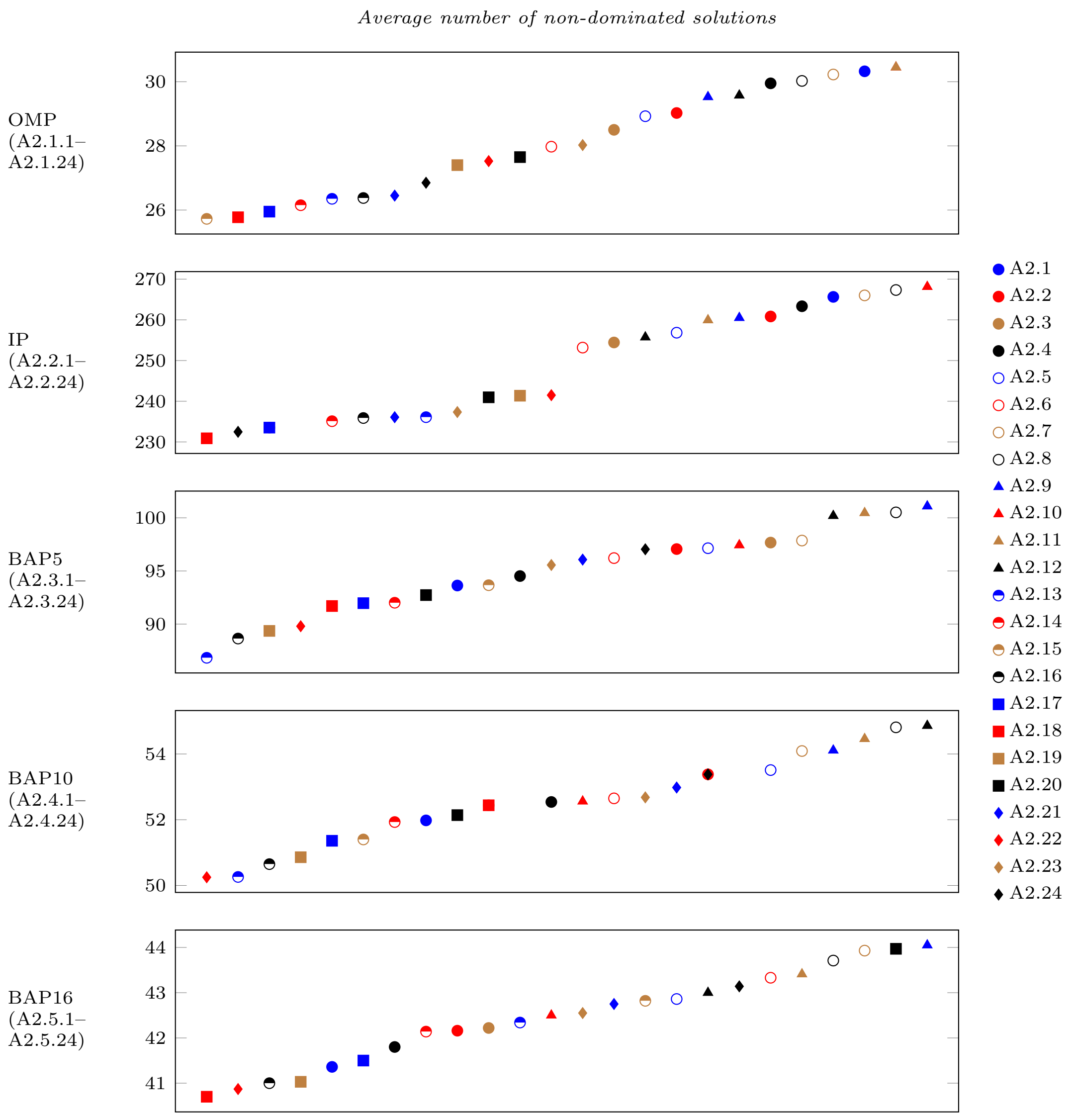


Figure B.35: *The average number of non-dominated solutions obtained for hyperparameter combinations A2.1–A2.24 for the respective simulation problems.*

B.3 DBMOSA

This section presents the results for the hyperparameter study conducted for the DBMOSA.

B.3.1 Open mine problem

Tables B.43 and B.44 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–40 for hyperparameter combinations A3.1.1–A3.1.4. In Figure B.36 the worst and best approximation fronts are plotted for the respective hyperparameter combinations (or move operators) A3.1.1–A3.1.4, for the respective simulation problems as summarised in Table B.45.

B.3.2 (s, S) Inventory problem

Tables B.46 and B.47 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A3.2.1–A3.2.4. In Figure B.37 the worst and best approximation fronts are plotted for the respective hyperparameter combinations (or move operators) A3.2.1–A3.2.4, for the respective simulation problems as summarised in Table B.48.

B.3.3 Buffer allocation problem: five machines

Tables B.49 and B.50 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A3.3.1–A3.3.4. In Figure B.38 the worst and best approximation fronts are plotted for the respective hyperparameter combinations (or move operators) A3.3.1–A3.3.4, for the respective simulation problems as summarised in Table B.51.

B.3.4 Buffer allocation problem: 10 machines

Tables B.52 and B.53 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A3.4.1–A3.4.4. In Figure B.39 the worst and best approximation fronts are plotted for the respective hyperparameter combinations (or move operators) A3.4.1–A3.4.4, for the respective simulation problems as summarised in Table B.54.

B.3.5 Non-linear buffer allocation problem: 16 machines

Tables B.55 and B.56 present a preview of the hyperareas and number on non-dominated solutions and their corresponding ranks for run 1–100 for hyperparameter combinations A3.5.1–A3.5.4. In Figure B.40 the worst and best approximation fronts are plotted for the respective hyperparameter combinations (or move operators) A3.5.1–A3.5.4, for the respective simulation problems as summarised in Table B.57.

In Figure B.41 the average number of non-dominated solutions obtained for hyperparameter combinations A3.1–A3.4 is illustrated for the respective simulation problems.

Table B.43: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–40 for hyperparameter combinations A3.1.1–A3.1.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.1.1	A3.1.2	A3.1.3	A3.1.4
1	9 199.23 (2)	5 384.99 (4)	8 898.58 (3)	9 678.68 (1)
2	9 117.74 (1)	6 114.05 (3)	7 637.54 (2)	5 675.39 (4)
3	10 800.56 (1)	7 399.36 (3)	7 021.79 (4)	9 473.27 (2)
4	7 334.52 (3)	8 640.86 (1)	7 532.07 (2)	6 736.11 (4)
5	7 811.52 (3)	8 018.12 (1)	7 969.69 (2)	6 519.96 (4)
6	6 701.76 (3)	4 420.00 (4)	8 274.31 (1)	7 573.69 (2)
7	8 122.81 (2)	5 582.25 (4)	7 408.69 (3)	9 303.05 (1)
8	6 454.90 (2)	5 577.79 (3)	5 243.40 (4)	7 952.99 (1)
9	6 447.06 (3)	5 103.89 (4)	8 112.39 (1)	7 099.57 (2)
10	12 070.97 (1)	4 620.59 (4)	10 656.92 (2)	8 936.71 (3)
⋮	⋮	⋮	⋮	⋮
39	6 518.99 (3)	7 925.88 (1)	7 124.13 (2)	6 054.66 (4)
40	6 500.90 (4)	6 909.59 (3)	10 594.92 (1)	8 848.14 (2)
$\sum R_{Ci}$	90	109	88	113
$\sum R_{Ci}^2$	8 100	11 881	7 744	12 769

Table B.44: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–40 for hyperparameter combinations A3.1.1–A3.1.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.1.1	A3.1.2	A3.1.3	A3.1.4
1	24 (2)	14 (4)	23 (3)	25 (1)
2	24 (1)	16 (3)	20 (2)	15 (4)
3	27 (1)	20 (3)	19 (4)	24 (2)
4	19 (3)	22 (1)	21 (2)	17 (4)
5	20 (2.5)	21 (1)	20 (2.5)	18 (4)
6	18 (3)	12 (4)	21 (1)	20 (2)
7	21 (2)	16 (4)	19 (3)	25 (1)
8	18 (2)	15 (3)	14 (4)	20 (1)
9	17 (3)	14 (4)	21 (1)	19 (2)
10	31 (1)	14 (4)	27 (2)	24 (3)
⋮	⋮	⋮	⋮	⋮
39	17 (3)	20 (1)	19 (2)	16 (4)
40	18 (3.5)	18 (3.5)	27 (1)	22 (2)
$\sum R_{Ci}$	93	109	88.5	109.5
$\sum R_{Ci}^2$	8 649	11 881	7 832.25	11 990.25

Table B.45: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the DBMOSA) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A3.1.1–A3.1.4.*

	Worst		Best	
	HA	NDS	HA	NDS
A3.1.1	Run 8	Run 28	Run 33	Run 33
A3.1.2	Run 33	Run 22	Run 15	Run 15
A3.1.3	Run 35	Run 8	Run 38	Run 38
A3.1.4	Run 32	Run 21	Run 30	Run 30

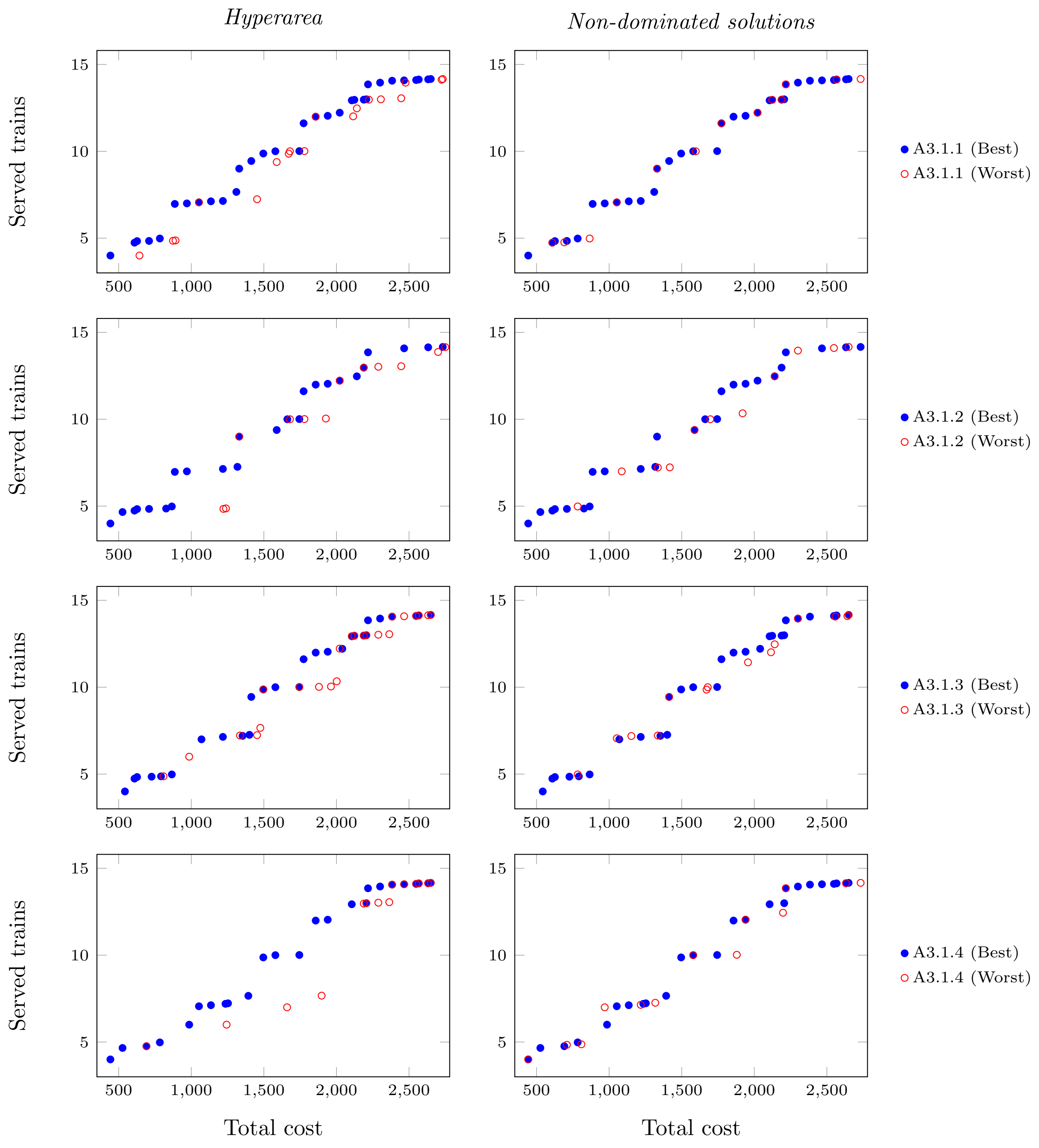


Figure B.36: *The best and worst best approximation fronts for hyperparameter combinations (or move operators) A3.1.1–A3.1.4.*

Table B.46: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A3.2.1–A3.2.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.2.1	A3.2.2	A3.2.3	A3.2.4
1	112 804.07 (1)	111 381.09 (3)	108 706.16 (4)	111 633.64 (2)
2	112 712.34 (1)	111 546.09 (3)	111 315.14 (4)	112 078.12 (2)
3	113 925.8 (1)	113 110.8 (2)	105 669.79 (4)	110 381.22 (3)
4	112 902.02 (2)	113 541.5 (1)	111 165.58 (4)	112 155.7 (3)
5	112 666.71 (1)	111 847.99 (3)	108 289.54 (4)	112 666.35 (2)
6	113 200.88 (1)	111 054.7 (3)	110 461.12 (4)	112 589.25 (2)
7	113 016.04 (1)	111 289.27 (2)	100 622.78 (4)	111 211.16 (3)
8	113 544.92 (2)	113 597.58 (1)	111 551.79 (3)	111 162.96 (4)
9	112 723.49 (1)	110 520.39 (3)	109 914.84 (4)	111 870.97 (2)
10	113 462.11 (1)	111 493.86 (4)	112 956.93 (3)	113 026.35 (2)
⋮	⋮	⋮	⋮	⋮
99	112 944.81 (2)	113 485.79 (1)	112 324.02 (3)	112 230.69 (4)
100	113 225.38 (1)	111 076.44 (3)	109 549.16 (4)	111 697.29 (2)
$\sum R_{Ci}$	138	235	331	296
$\sum R_{Ci}^2$	19 044	55 225	109 561	87 616

Table B.47: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A3.2.1–A3.2.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.2.1	A3.2.2	A3.2.3	A3.2.4
1	198 (2)	313 (1)	23 (4)	145 (3)
2	183 (2)	259 (1)	44 (4)	118 (3)
3	156 (3)	135 (4)	208 (2)	219 (1)
4	169 (1)	149 (2)	118 (4)	129 (3)
5	214 (1)	112 (3)	102 (4)	177 (2)
6	167 (4)	321 (1)	184 (3)	204 (2)
7	162 (3)	414 (1)	80 (4)	165 (2)
8	156 (3)	163 (2)	108 (4)	177 (1)
9	186 (2)	318 (1)	147 (3)	124 (4)
10	137 (3)	198 (1)	42 (4)	189 (2)
⋮	⋮	⋮	⋮	⋮
99	160 (3)	178 (2)	58 (4)	214 (1)
100	188 (1)	163 (2)	66 (4)	117 (3)
$\sum R_{Ci}$	240.5	223.5	309	227
$\sum R_{Ci}^2$	57 840.25	49 952.25	95 481	51 529

Table B.48: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the DBMOSA) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A3.2.1–A3.2.4.*

	Worst		Best	
	HA	NDS	HA	NDS
A3.2.1	Run 47	Run 38	Run 33	Run 47
A3.2.2	Run 79	Run 98	Run 20	Run 7
A3.2.3	Run 36	Run 1	Run 59	Run 41
A3.2.4	Run 54	Run 40	Run 37	Run 76

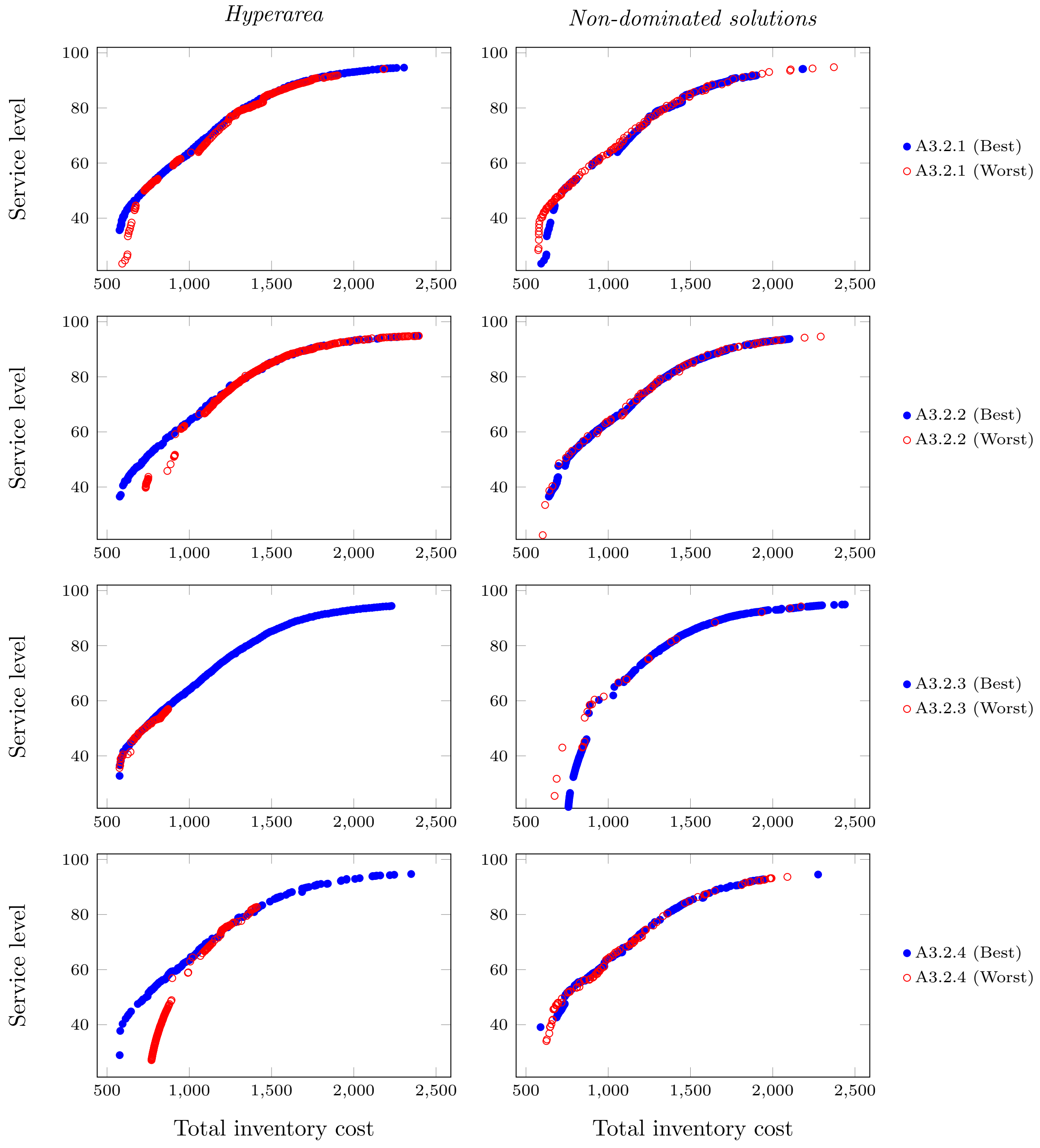


Figure B.37: *The best and worst best approximation fronts for hyperparameter combinations (or move operators) A3.2.1–A3.2.4.*

Table B.49: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A3.3.1–A3.3.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.3.1	A3.3.2	A3.3.3	A3.3.4
1	33.67 (2)	33.65 (3)	33.72 (1)	33.54 (4)
2	33.70 (1)	33.42 (3)	33.61 (2)	33.20 (4)
3	33.73 (1)	33.69 (2)	33.49 (3)	33.38 (4)
4	33.67 (1)	33.53 (4)	33.63 (3)	33.65 (2)
5	33.63 (2)	33.46 (3)	33.65 (1)	33.42 (4)
6	33.64 (3)	33.71 (1)	33.70 (2)	33.22 (4)
7	33.68 (2)	33.54 (4)	33.72 (1)	33.65 (3)
8	33.65 (1)	33.19 (4)	33.46 (2)	33.44 (3)
9	33.70 (1)	33.45 (3)	33.45 (2)	33.19 (4)
10	33.75 (1)	33.43 (4)	33.59 (3)	33.69 (2)
⋮	⋮	⋮	⋮	⋮
99	33.52 (3)	33.59 (2)	33.07 (4)	33.67 (1)
100	32.20 (4)	33.24 (3)	33.48 (2)	33.50 (1)
$\sum R_{Ci}$	164	315	242	279
$\sum R_{Ci}^2$	26 896	99 225	58 564	77 841

Table B.50: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A3.3.1–A3.3.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.3.1	A3.3.2	A3.3.3	A3.3.4
1	62 (1)	55 (2)	54 (3)	46 (4)
2	60 (1)	49 (4)	52 (2.5)	52 (2.5)
3	76 (1)	39 (4)	48 (2)	41 (3)
4	55 (1)	43 (4)	54 (2)	46 (3)
5	60 (1)	31 (4)	55 (2)	40 (3)
6	64 (2)	50 (4)	65 (1)	53 (3)
7	52 (2)	47 (4)	63 (1)	49 (3)
8	56 (1)	46 (3)	39 (4)	52 (2)
9	63 (1)	46 (2)	45 (3)	43 (4)
10	77 (1)	39 (4)	56 (2)	43 (3)
⋮	⋮	⋮	⋮	⋮
99	67 (1)	56 (3)	62 (2)	44 (4)
100	41 (2)	38 (3)	59 (1)	37 (4)
$\sum R_{Ci}$	139.5	338	214	308.5
$\sum R_{Ci}^2$	19 460.25	114 244	45 796	95 172.25

Table B.51: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the DBMOSA) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A3.3.1–A3.3.4.*

	Worst		Best	
	HA	NDS	HA	NDS
A3.3.1	Run 100	Run 100	Run 63	Run 84
A3.3.2	Run 88	Run 17	Run 6	Run 36
A3.3.3	Run 97	Run 68	Run 18	Run 98
A3.3.4	Run 57	Run 32	Run 10	Run 21

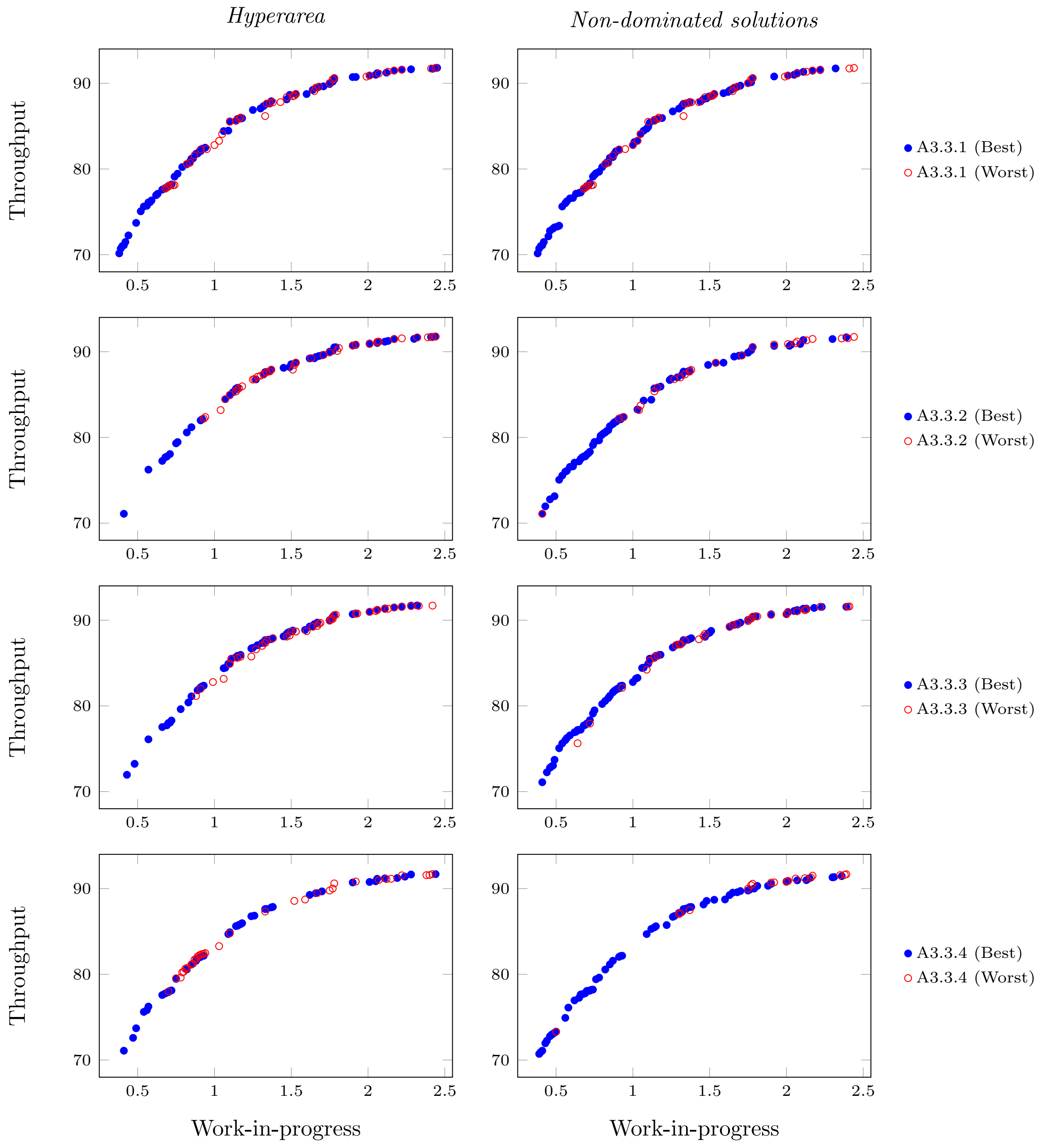


Figure B.38: *The best and worst best approximation fronts for hyperparameter combinations (or move operators) A3.3.1–A3.3.4.*

Table B.52: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A3.4.1–A3.4.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.4.1	A3.4.2	A3.4.3	A3.4.4
1	22.1 (1)	21.86 (4)	21.98 (2)	21.97 (3)
2	21.36 (1)	20.87 (3)	21.16 (2)	20.78 (4)
3	22.02 (1)	21.29 (2)	20.98 (3)	19.57 (4)
4	21.76 (1)	21.37 (3)	20.78 (4)	21.59 (2)
5	21.8 (1)	21.32 (2)	21.18 (3)	21.07 (4)
6	21.05 (2)	21.11 (1)	19.81 (3)	18.81 (4)
7	21.45 (2)	20.18 (4)	20.94 (3)	21.45 (1)
8	21.65 (1)	20.53 (4)	20.64 (3)	21 (2)
9	21.36 (1)	20.25 (3)	20.46 (2)	19.95 (4)
10	21.97 (1)	19.37 (4)	20.32 (3)	20.7 (2)
⋮	⋮	⋮	⋮	⋮
99	21.23 (2)	19.89 (4)	21.04 (3)	21.64 (1)
100	21.12 (3)	19.56 (4)	21.14 (2)	21.25 (1)
$\sum R_{Ci}$	145	327	274	254
$\sum R_{Ci}^2$	21 025	106 929	75 076	64 516

Table B.53: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A3.4.1–A3.4.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.4.1	A3.4.2	A3.4.3	A3.4.4
1	35 (3)	31 (4)	39 (1)	38 (2)
2	40 (1)	27 (4)	29 (3)	33 (2)
3	46 (1)	28 (3)	33 (2)	26 (4)
4	39 (2)	31 (3.5)	31 (3.5)	41 (1)
5	40 (1)	26 (4)	33 (3)	35 (2)
6	34 (1)	29 (2)	27 (3)	19 (4)
7	36 (2)	29 (4)	35 (3)	42 (1)
8	40 (1)	30 (3.5)	30 (3.5)	36 (2)
9	33 (1.5)	24 (4)	33 (1.5)	28 (3)
10	47 (1)	29 (3.5)	34 (2)	29 (3.5)
⋮	⋮	⋮	⋮	⋮
99	43 (1)	28 (4)	34 (2)	32 (3)
100	32 (2)	29 (4)	41 (1)	31 (3)
$\sum R_{Ci}$	148.5	328.5	250	273
$\sum R_{Ci}^2$	22 052.25	107 912.25	62 500	74 529

Table B.54: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the DBMOSA) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A3.4.1–A3.4.4.*

	Worst		Best	
	HA	NDS	HA	NDS
A3.4.1	Run 40	Run 11	Run 76	Run 92
A3.4.2	Run 55	Run 54	Run 1	Run 82
A3.4.3	Run 25	Run 25	Run 1	Run 31
A3.4.4	Run 32	Run 32	Run 1	Run 93

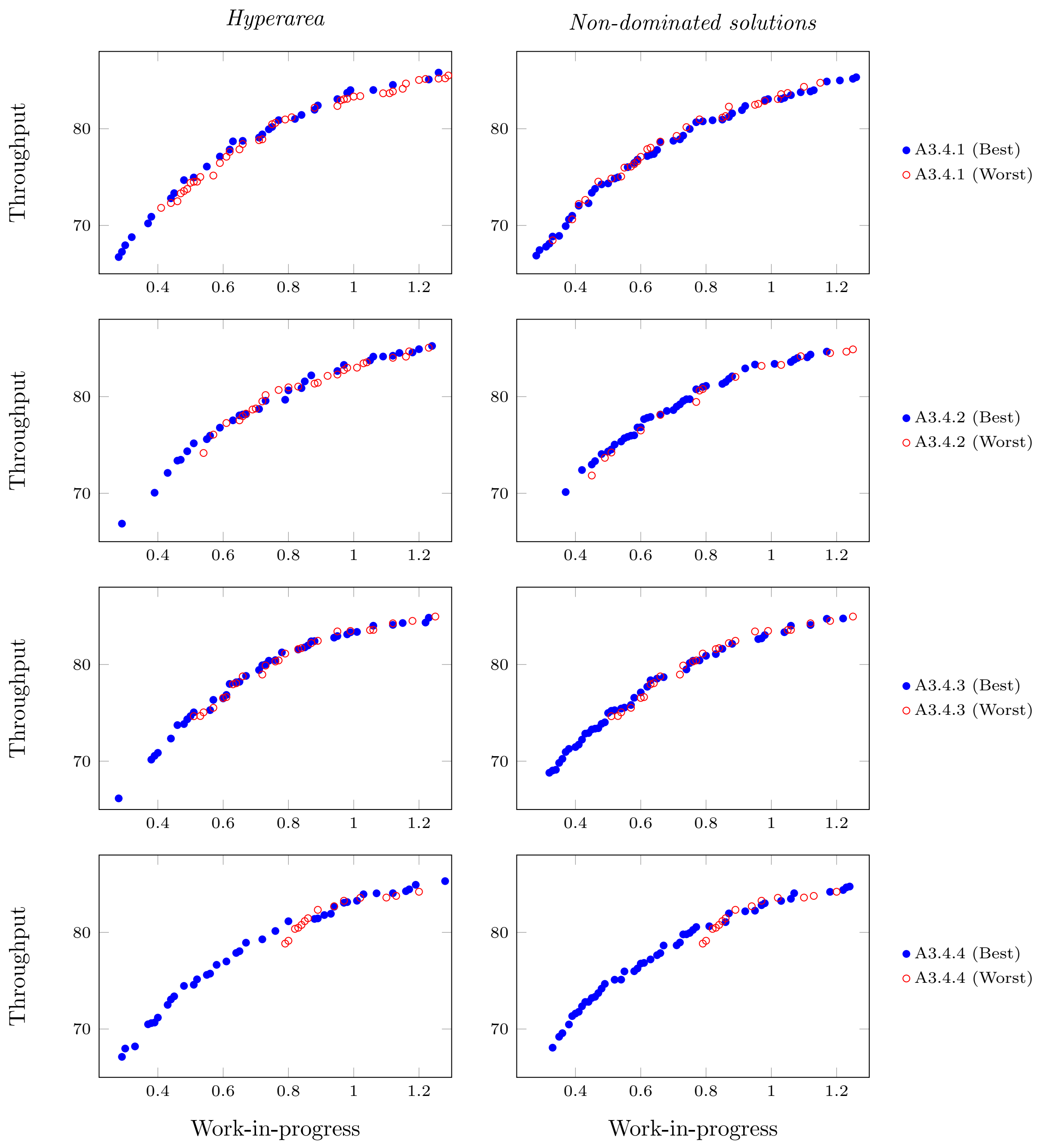


Figure B.39: *The best and worst best approximation fronts for hyperparameter combinations (or move operators) A3.4.1–A3.4.4.*

Table B.55: *A preview of the hyperareas (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A3.5.1–A3.5.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.5.1	A3.5.2	A3.5.3	A3.5.4
1	205.74 (2)	178.94 (4)	203.61 (3)	205.85 (1)
2	183.55 (3)	182.99 (4)	191.91 (2)	200.81 (1)
3	205.97 (1)	184.54 (2)	179.41 (3)	178.97 (4)
4	180.93 (1)	174.15 (4)	174.74 (3)	179.45 (2)
5	183.24 (1)	172.86 (3)	164.59 (4)	177.03 (2)
6	174.51 (3)	179.59 (2)	170.54 (4)	187.56 (1)
7	191.64 (1)	173.07 (4)	179.66 (3)	182.57 (2)
8	188.19 (1)	173.72 (4)	179.59 (2)	179.59 (3)
9	188.33 (1)	164.58 (4)	181.09 (2)	171.32 (3)
10	202.83 (1)	179.86 (2)	173.66 (4)	177.67 (3)
⋮	⋮	⋮	⋮	⋮
99	178.91 (3)	184.86 (2)	188.48 (1)	166.98 (4)
100	182.53 (2)	177.85 (3)	172 (4)	182.68 (1)
$\sum R_{Ci}$	153	318	269	260
$\sum R_{Ci}^2$	23 409	101 124	72 361	67 600

Table B.56: *A preview of the number of non-dominated solutions (and corresponding ranks) obtained for run 1–100 for hyperparameter combinations A3.5.1–A3.5.4. The ranks are given in the parentheses and are used in the computation of the Friedman and Iman-Davenport hypothesis tests.*

Run	A3.5.1	A3.5.2	A3.5.3	A3.5.4
1	42 (1)	31 (4)	37 (2)	33 (3)
2	31 (4)	33 (2)	37 (1)	32 (3)
3	53 (1)	31 (3)	27 (4)	36 (2)
4	32 (2.5)	32 (2.5)	35 (1)	30 (4)
5	35 (1)	29 (3)	25 (4)	33 (2)
6	33 (2)	28 (4)	34 (1)	30 (3)
7	31 (3)	26 (4)	35 (1.5)	35 (1.5)
8	32 (2)	31 (3)	36 (1)	28 (4)
9	33 (1)	26 (3)	32 (2)	25 (4)
10	38 (1)	28 (2.5)	25 (4)	28 (2.5)
⋮	⋮	⋮	⋮	⋮
99	34 (3)	40 (2)	49 (1)	28 (4)
100	32 (1)	26 (4)	27 (3)	31 (2)
$\sum R_{Ci}$	163.5	306	266.5	264
$\sum R_{Ci}^2$	26 732.25	93 636	71 022.25	69 696

Table B.57: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the DBMOSA) for the hyperarea performance indicator and the number of non-dominated solutions found for hyperparameter combinations A3.5.1–A3.5.4.*

	Worst		Best	
	HA	NDS	HA	NDS
A3.5.1	Run 82	Run 80	Run 30	Run 3
A3.5.2	Run 64	Run 42	Run 36	Run 99
A3.5.3	Run 71	Run 12	Run 1	Run 99
A3.5.4	Run 27	Run 27	Run 1	Run 97

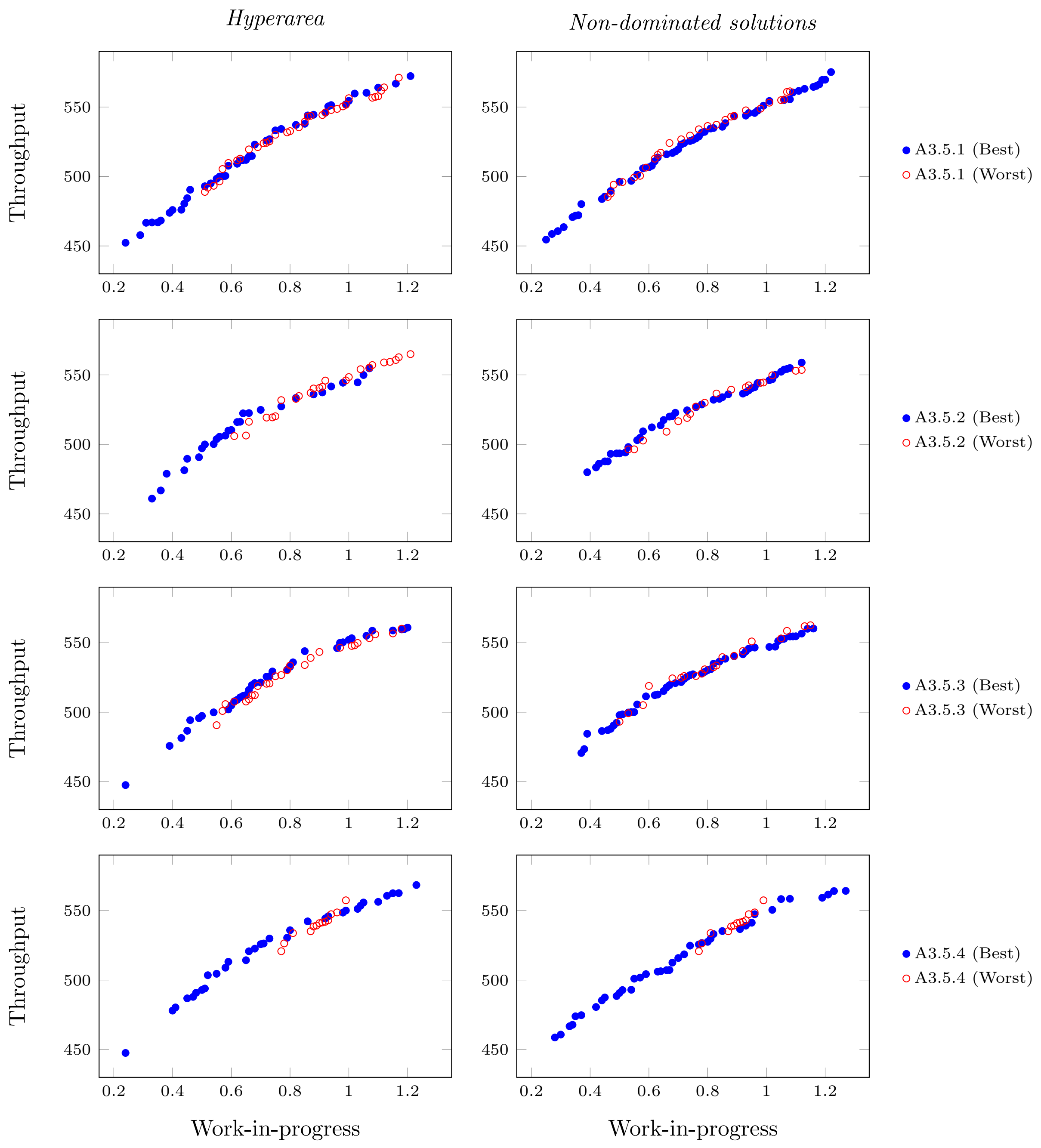


Figure B.40: *The best and worst best approximation fronts for hyperparameter combinations (or move operators) A3.5.1–A3.5.4.*

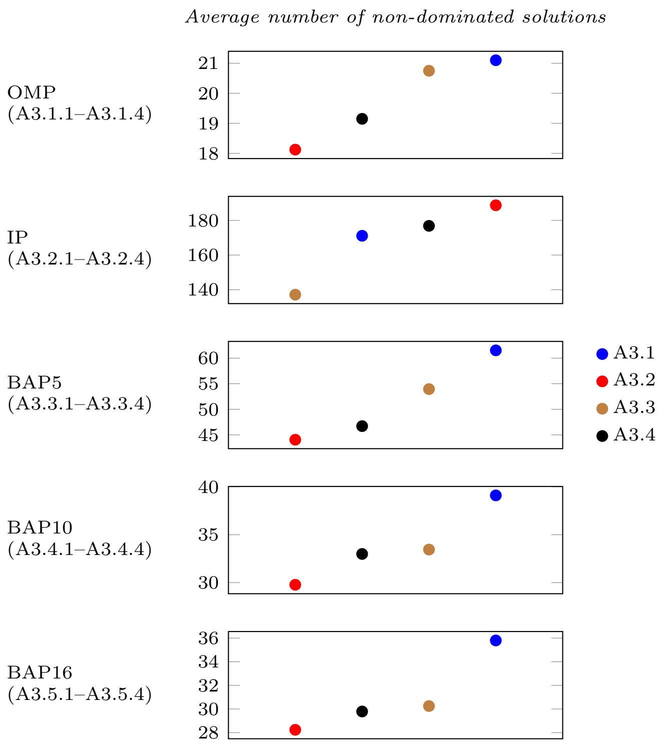


Figure B.41: *The average number of non-dominated solutions obtained for hyperparameter combinations A3.1–A3.4 for the respective simulation problems.*

APPENDIX C

The Backpropagation Training Algorithm

This appendix contains details concerning the backpropagation training algorithm for ANNs training using gradient methods, namely, gradient descent and stochastic gradient descent. The derivation of the backpropagation of errors is presented in the context of a single layered perceptron. This derivation is a modified reproduction of the derivatives presented in [30, 158, 177, 194, 159].

Furthermore, the gradient-based optimisation technique employed in the training algorithm is also elaborated upon. This appendix should be read in conjunction with Chapter 2, as the terminologies and notations employed in this appendix are introduced there.

The purpose of the backpropagation algorithm is to find suitable values for the network weights between the input and hidden layer as well as between the hidden and the output layer. This is achieved by minimising the error function, denoted by E. The individual errors $\{E_1, \ldots, E_k, \ldots, E_m\}$, where m is the number of outputs, may be defined seperately, where E is the summation of the individual errors, *i.e.* $\sum_{k=1}^{m} E_i$. Assume that the individual errors can be expressed as a differentiable function of the network outputs.

The backpropagation procedure obtains the derivatives of the error function, where the error function is influenced by the network weights, biases and activations. Note that only the weights and biases may be changed to minimise E. The weights and biasses are updated in proportion to their contribution to E.

C.1 Backpropagation

Figure C.1 is used to help elucidate the process followed to perform backpropagation.

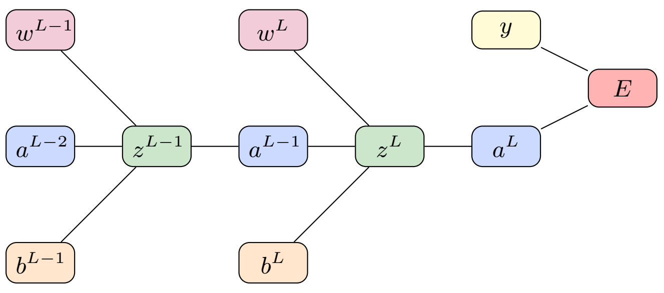


FIGURE C.1: *Backpropagation visual representation of the chain rule.*

Assume that input training vector $\boldsymbol{x}$ has been propagated forward through the network, therefore the activations of the hidden and output neurons have been calculated using (2.9)–(2.12) and are rewritten in the context of Figure C.1 to simplify the explanation, such that

$$\begin{aligned} z^L &= w^L a^{L-1} + b^L, && (2.9), && \text{(C.1)} \\ a^L &= \sigma(z^L), && (2.10), && \text{(C.2)} \\ z^{L-1} &= w^{L-1} a^{L-2} + b^{L-1}, && (2.11), && \text{(C.3)} \\ a^{L-1} &= \sigma(z^{L-1}), && (2.12). && \text{(C.4)} \end{aligned}$$

<u>The output layer (L):</u>

The partial derivative of the error function given the weight in layer L is,

$$\frac{\partial E}{\partial w^L} = \frac{\partial E}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial w^L}, \tag{C.5}$$

where the partial derivatives are defined as follows. Given $E = \frac{1}{2}(a^L - y)^2$, the derivative is

$$\frac{\partial E}{\partial a^L} = (a^L - y). \tag{C.6}$$

Given $a^L = \sigma(z^L)$, the derivative is

$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L). \tag{C.7}$$

Given $z^L = w^L a^{L-1} + b^L$, the derivative is

$$\frac{\partial z^L}{\partial w^L} = a^{L-1}. \tag{C.8}$$

Substituting (C.6)–(C.8) into (C.5) yields,

$$\frac{\partial E}{\partial w^L} = (a^L - y)\ \sigma'(z^L)\ a^{L-1}. \tag{C.9}$$

<u>The hidden layer ($L-1$):</u>

The partial derivative of the error function given the weight in layer $L-1$ is,

$$\frac{\partial E}{\partial w^{L-1}} = \frac{\partial E}{\partial a^{L-1}} \frac{\partial a^{L-1}}{\partial z^{L-1}} \frac{\partial z^{L-1}}{\partial w^{L-1}} \tag{C.10}$$

where the partial derivatives are defined as follows

$$\begin{aligned} \frac{\partial E}{\partial a^{L-1}} &= \sum_{k=1}^{m} \frac{\partial E_k}{\partial a^{L-1}}, && \text{(C.11)} \\ &= \sum_{k=1}^{m} \frac{\partial E_k}{\partial z_k^L} \frac{\partial z_k^L}{\partial a_k^{L-1}}, && \text{(C.12)} \\ &= \sum_{k=1}^{m} \frac{\partial E_k}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_k^L} \frac{\partial z_k^L}{\partial a_k^{L-1}}, && \text{(C.13)} \end{aligned}$$

and since backpropagation is performed on a MFNN with one hidden layer (C.13), can simplify to

$$\begin{aligned}\frac{\partial E}{\partial a^L} &= \frac{\partial E}{\partial a^L}\,\frac{\partial a^L}{\partial z^L}\,\frac{\partial z^L}{\partial a^{L-1}}, && \text{(C.14)}\\ &= (a^L - y)\;\sigma'(z^L)\;w^L. && \text{(C.15)}\end{aligned}$$

The partial derivative of the activation of layer $L-1$ with respect to the weighted sum of layer $L-1$ is given by

$$\frac{\partial a^{L-1}}{\partial z^{L-1}} = \sigma'(z^{L-1}). \tag{C.16}$$

The partial derivative of the activation of layer $L-1$ with respect to the weighted sum of layer $L-1$ is given by

$$\frac{\partial z^{L-1}}{\partial w^{L-1}} = a^{L-2}. \tag{C.17}$$

Substituting (C.14)–(C.17) into (C.10) yields

$$\frac{\partial E}{\partial w^{L-1}} = (a^L - y)\;\sigma'(z^L)\;w^L\;\sigma'(z^{L-1})\;a^{L-2}. \tag{C.18}$$

The weights are updated as follows:

$$\begin{aligned}w^{L^+} &= w^L - \eta\,\frac{\partial E}{\partial w^L}, && \text{(C.19)}\\ w^{L-1^+} &= w^{L-1} - \eta\,\frac{\partial E}{\partial w^{L-1}}, && \text{(C.20)}\end{aligned}$$

where w^{L^+} represents the updated weight for layer L, w^{L-1^+} the updated weight in layer $L-1$ and η is the learning rate.

APPENDIX D

Discrete-event Simulation Optimisation Problem Definitions

This appendix defines each of the bi-objective simulation optimisation problems studied. First, the (s, S) inventory problem (IP) is presented. Next, a well-researched problem, the buffer-allocation problem (BAP) is studied as well as instances thereof and lastly, the Open mine problem (OMP) is presented. The experiments used to determine a sensible upper bound and the sufficient number of observations per solution are also defined, where applicable, for the respective problems.

D.1 The (s, S) inventory problem

[The content of this section is an extract from unpublished notes, from the University of Stellenbosch (US), and are included here to define the problem.]

The (s, S) IP used in this study is dynamic and stochastic in nature as it deals with uncertain demand, where s refers to the *re-order level* and S to the *re-order quantity*. For a detailed description of the problem, the reader is referred to [19].

A company sells a single commodity and has to determine the level of inventory necessary for the following n months in order to satisfy the demand. The inventory level can be calculated as stock on hand minus backorders minus demand and the inventory position as inventory level plus outstanding orders. Customer inter-arrival times follow an exponential distribution with a mean of 3 minutes. Assume that the demand follows a distribution of $\lfloor$Weibull(1, 8)$\rfloor$ for each customer i and order lead time is distributed *TRI(12,14,20)* hours. In order to know what quantity should be ordered, the company assesses the inventory level after each sale. Once the inventory level drops below the reorder point s after the sale, a quantity of S items are ordered. Due to the order lead time and the continuity of demand, there exists a possibility that the inventory level will reach zero. If this is the case, and replenishment stock has not yet arrived, a stock-out period is experienced and customer demand cannot be fulfilled, which leads to lost sales.

When a customer requests a certain number of units x, the inventory level I will drop by x:

$$I_{t+1} = I_t - x. \tag{D.1}$$

Once the inventory level drops below the re-order level s, a supplier is notified. The supplier will dispatch a quantity equal to the re-order quantity S. There is a delay (in hours), distributed *TRI(1,2,4)*, between the notification and the delivery of S. It is assumed that the supplier is reliable and will always deliver the number S units, but it is delayed as stated earlier. Ignoring possible capacity constraints for the moment, a delivery has the following effect on inventory:

$$I_{t+1} = I_t + S. \tag{D.2}$$

If the inventory level drops too low to be able to meet customer demand, customer satisfaction is affected negatively. *Assume that a customer whose exact demand cannot be met, will not be satisfied with an alternative amount and will leave without having drawn cash..* The seller therefore has to maintain an inventory level high enough to meet most customers' demand. Refer to the percentage of demand met as service level and it is defined in (D.3).

$$\text{Service level} = \frac{\text{Number of customers serviced}}{\text{Number of customers requiring service}}. \tag{D.3}$$

There is an inventory cost C_I associated with inventory. Inventory being stored has to be paid for, and each delivery also has a cost associated with it. Delivery cost is denoted by C_D. The break down of the total cost C_T incurred is shown in (D.4), (D.5) and (D.6).

$$C_T = C_D + C_I, \tag{D.4}$$

where

$$C_D = d \times c_d, \tag{D.5}$$

and

$$C_I = \sum_{i=1}^{n} 0.06 \times u_i, \tag{D.6}$$

where u_i is the number of units in store at the end of day i and 0.06 is the holding cost per unit per day. A single delivery d costs R120 (this is denoted by c_d) and n denotes the total number of days. The goal is to minimise *total inventory cost* while maximising *service level.* The typical inventory consumption and replenishment process is shown in Figure D.1.

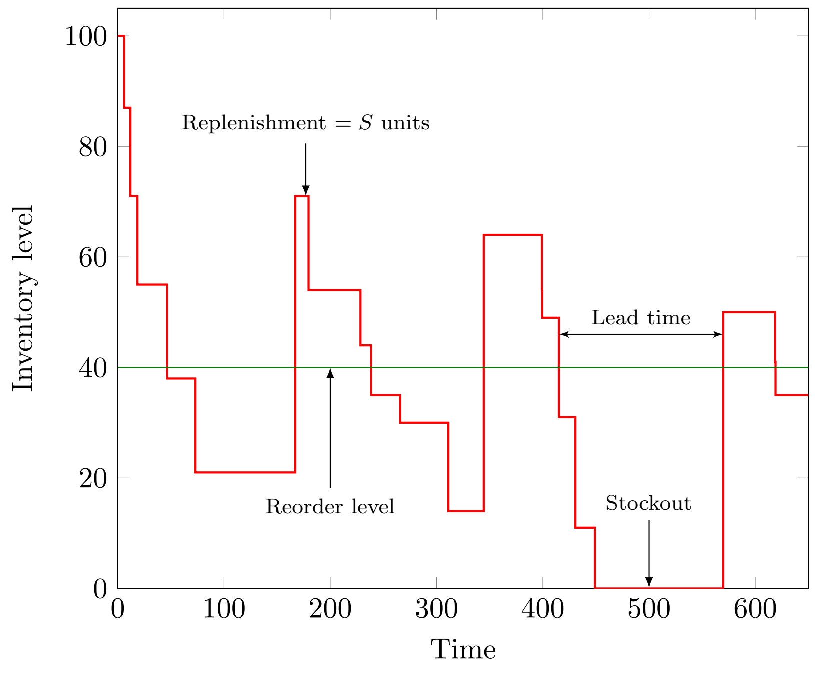
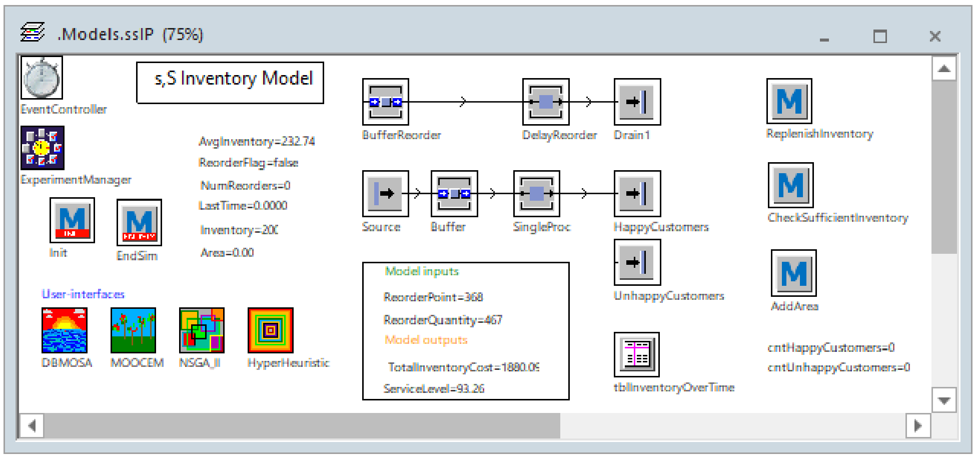


Figure D.1: *Some characteristics of the generalised (s, S) inventory problem*

Figure D.2 shows the simulation model as it is built in Tecnomatix and represents the IP.



AvgInventory=232.74
ReorderFlag=false
NumReorders=0
LastTime=0.0000
Inventory=20(
Area=0.00

Model inputs
ReorderPoint=368
ReorderQuantity=467
Model outputs
TotalInventoryCost=1880.0(
ServiceLevel=93.26

cntHappyCustomers=0
cntUnhappyCustomers=0

Figure D.2: *A screenshot of the IP model in Tecnomatix.*

Observations per solution

Table D.1 shows the experiments used to determine whether or not 100 observations per solution are sufficient for the IP and is summarised in Table D.2.

Table D.1: *Experiments used to determine a sufficient number of observations per solution for the IP*

			Total inventory cost			Service level		
	s	S	A	B	C	A	B	C
Exp 1	50	50	744.08	750.95	744.58	28.98	29.22	28.66
Exp 2	100	100	816.61	814.90	809.19	40.05	40.43	39.92
Exp 3	150	150	914.15	909.34	906.91	51.20	51.69	51.07
Exp 4	200	200	1 049.33	1 044.23	1 042.70	62.54	63.37	62.32
Exp 5	250	250	1 209.69	1 200.14	1 194.99	72.55	72.83	72.15
Exp 6	300	300	1 361.72	1 382.71	1 380.13	81.08	81.08	80.47
Exp 7	350	350	1 594.72	1 604.73	1 599.25	87.96	87.72	86.84
Exp 8	400	400	1 863.48	1 847.64	1 840.77	91.62	91.70	91.06
Exp 9	450	450	2 118.69	2 122.16	2 121.81	93.54	94.04	93.22
Exp 10	500	500	2 434.06	2 438.60	2 417.66	93.77	94.94	94.25

Table D.2: *Summary of the experiments used to determine the sufficient number of observations per solution for the IP.*

		Total inventory cost			Service level		
Groups	Count	Sum	Average	Variance	Sum	Average	Variance
Exp 1	3	2 239.61	746.54	14.66	86.86	28.95	0.08
Exp 2	3	2 440.69	813.56	15.07	120.40	40.13	0.07
Exp 3	3	2 730.41	910.14	13.60	153.96	51.32	0.11
Exp 4	3	3 136.26	1 045.42	12.04	188.23	62.74	0.31
Exp 5	3	3 604.82	1 201.61	55.64	217.54	72.51	0.12
Exp 6	3	4 124.56	1 374.85	131.02	242.63	80.88	0.12
Exp 7	3	4 798.70	1 599.57	25.13	262.52	87.51	0.34
Exp 8	3	5 551.89	1 850.63	135.72	274.38	91.46	0.12
Exp 9	3	6 362.66	2 120.89	3.64	280.80	93.60	0.17
Exp 10	3	7 290.32	2 430.11	121.29	282.96	94.32	0.34

D.2 The buffer-allocation problems

[The content of this section is an extract from [21] and is included here with permission from the author.]

Finite queuing networks are associated with many practical systems through which discrete or continuous flow occurs, such as manufacturing systems and telecommunication networks. These networks often exhibit flow variation or asynchronous part movement; hence the need for buffer space in the network. One of the network design priorities is to maximise the network throughput or throughput rate, which increases with more buffer space [53].

However, buffer space may be costly for several reasons in commercial projects and costs must be minimised. This gives rise to the BAP which is usually formulated as a stochastic, non-linear, integer mathematical programming problem and which is computationally hard to solve [54]. The problem has several variations, including problems with reliable or unreliable servers (for example, manufacturing machines), the type of distribution assigned to service time, repair times and time-to-failure, synchronous or asynchronous part movement, and the network topology.

Papadopoulos & Heavey [202] classified queuing network models for production and transfer lines, while [54] identified four traditional methodological approaches to the BAP: simulation methods, metaheuristics, dynamic programming and search methods. In some studies the processing times were assumed to be deterministic, while time-to-failure and repair times were exponentially distributed. Researchers also adopt different performance measures (objectives) and consider single or multi-objective problem variants, while decision variables include buffer sizes and server processing rates.

A production line consists of a series of m machines with $m - 1$ buffers (or niches). Discrete parts are processed by each of these machines in sequence, and each discrete part takes up one buffer space or occupies a machine. There are n such spaces available, while the spaces in front of the first and beyond the last machine are considered infinite.

Generally, the machines have exponentially distributed processing and repair times with mean rates μ_i and r_i, respectively. The machine failures of the models in this study are *operation dependent failures* (ODF), which are more realistic than time-based failures [259]. A machine thus fails after a number of operations has been completed, and these operation counts are dictated by Poisson distributions with rates β_i.

Note that the ODF approach results in longer times between failures when finite buffers are required. Suppose the time between failures for Machine 1 is exponentially distributed with rate β_1 and the repair rate is r_1. Then the expected number of failures for this machine for a simulation run length of T, when infinite buffers are used, is

$$\frac{T}{\frac{1}{\beta_1} + \frac{1}{r_1}}.$$

When the buffer sizes are limited, this number decreases and the time between failures increases. An upstream machine can become blocked when its successor has failed, while a downstream machine can eventually become starved if its predecessor has failed. The basic performance measures of such a system are the throughput rate and the *work-in-progress* (WIP).

The objective is to determine the best buffer allocations, given a certain size of n. For example if $n = 10$, there are 10 buffers slots that can each hold a part. All 10 of these slots can be assigned to one machine and zero to the other three machines. Two buffers slots can be assigned, each to three machines and four buffers slots to the remaining machine. The goal is to minimise *work-in-progress* while maximising the *throughput*.

The three instances of the BAP are discussed in the next sections.

D.2.1 The buffer-allocation problem: five machines

The linear buffer allocation problem with $m = 5$ (BAP5) has exponential processing and repair times, schematically presented in Figure D.3.

$$\begin{tikzcd} {} \arrow[r, "B_0 = \infty"] & \boxed{M_1} \arrow[r, "B_1"] & \boxed{M_2} \arrow[r, "B_2"] & \boxed{M_3} \arrow[r, "B_3"] & \boxed{M_4} \arrow[r, "B_4"] & \boxed{M_5} \arrow[r, "B_5 = \infty"] & {} \end{tikzcd}$$

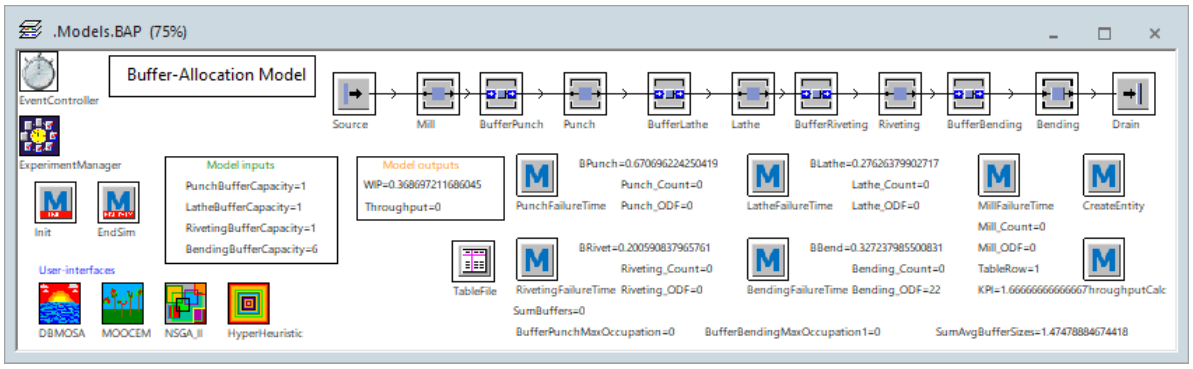
Figure D.3: *A series of five machines $M_1, \ldots, M_5$ with finite buffers $B_1, \ldots, B_{5-1}$ in a queuing network, representative of BAP5*

Table D.3 shows the processing times for machines $M_1 - M_5$, respectively, note that the processing times become progressively faster.

Table D.3: *Processing times for the machines in BAP5*

	M1	M2	M3	M4	M5
Processing time	1:00:00	55:00	50:00	46:00	43:00

Figure D.4 shows the simulation model as it is built in Tecnomatix and represents BAP5.



Model inputs
PunchBufferCapacity=1
LatheBufferCapacity=1
RivetingBufferCapacity=1
BendingBufferCapacity=6

Model outputs
WIP=0.368697211686045
Throughput=0

BPunch=0.670696224250419
Punch_Count=0
Punch_ODF=0

BLathe=0.27626379902717
Lathe_Count=0
Lathe_ODF=0

Mill_Count=0
Mill_ODF=0
TableRow=1
KPI=1.66666666666667

BRivet=0.200590837965761
Riveting_Count=0
Riveting_ODF=0

BBend=0.327237985500831
Bending_Count=0
Bending_ODF=22

SumBuffers=0
BufferPunchMaxOccupation=0
BufferBendingMaxOccupation1=0
SumAvgBufferSizes=1.47478884674418

Figure D.4: *A screenshot of the BAP5 model in Tecnomatix.*

Upper bounds

Table D.4 shows the experiments used to determine a sensible upper bound for BAP5.

Observations per solution

Table D.5 shows the experiments used to determine whether or not 100 observations per solution are sufficient for BAP5 and is summarised in Table D.6.

Table D.4: *The experiments used to determine a suiteable upper bound for BAP5*

	B1	B2	B3	B4	Work-in-progress	Throughput
Exp 1	10	10	10	10	2.13	91.83
Exp 2	12	12	12	12	2.28	92.14
Exp 3	14	14	14	14	2.38	92.23
Exp 4	16	16	16	16	2.43	92.31
Exp 5	17	17	17	17	2.46	92.28
Exp 6	18	18	18	18	2.48	92.30
Exp 7	19	19	19	19	2.49	92.34
Exp 8	50	50	50	50	2.54	92.34
Exp 9	100	100	100	100	2.54	92.34

Table D.5: *Experiments used to determine a sufficient number of observations per solution for BAP5*

					Work-in-progress			Throughput		
	B1	B2	B3	B4	A	B	C	A	B	C
Exp 1	1	1	1	1	0.37	0.38	0.39	69.30	70.15	71.34
Exp 2	2	2	2	2	0.73	0.71	0.72	77.10	77.45	78.37
Exp 3	3	3	3	3	0.99	1	1.02	80.40	82.22	82.94
Exp 4	4	4	4	4	1.26	1.21	1.25	83.60	85.54	86.05
Exp 5	5	5	5	5	1.45	1.43	1.46	87.30	87.73	88.02
Exp 6	6	6	6	6	1.73	1.62	1.65	88.30	88.95	89.52
Exp 7	7	7	7	7	1.92	1.76	1.80	89.80	89.87	90.50
Exp 8	8	8	8	8	2.07	1.89	1.93	88.50	90.22	91.08
Exp 9	9	9	9	9	2.05	2	2.04	90.10	90.52	91.59
Exp 10	10	10	10	10	2.14	2.06	2.13	89.20	90.80	91.83

Table D.6: *Summary of the experiments used to determine the sufficient number of observations per solution for BAP5.*

		Work-in-progress			Throughput		
Groups	Count	Sum	Average	Variance	Sum	Average	Variance
Exp 1	3	1.14	0.38	0	210.79	70.26	1.05
Exp 2	3	2.16	0.72	0	232.92	77.64	0.43
Exp 3	3	3.00	1.00	0	245.56	81.85	1.71
Exp 4	3	3.71	1.24	0	255.19	85.06	1.67
Exp 5	3	4.34	1.45	0	263.05	87.68	0.13
Exp 6	3	5.00	1.67	0	266.77	88.92	0.38
Exp 7	3	5.48	1.83	0.01	270.17	90.06	0.15
Exp 8	3	5.90	1.97	0.01	269.80	89.93	1.72
Exp 9	3	6.08	2.03	0	272.21	90.74	0.59
Exp 10	3	6.34	2.11	0	271.83	90.61	1.75

D.2.2 The buffer-allocation problem: 10 machines

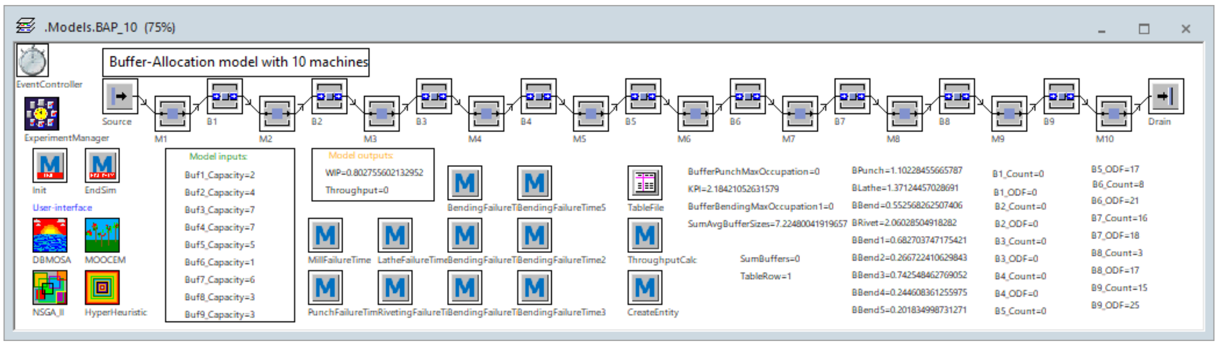
The linear buffer allocation problem with $m = 10$ (BAP10) is similar to BAP5, *i.e.* has exponential processing and repair times. BAP10 is schematically presented in Figure D.5. Table D.7 shows the processing times for machines $M_1 - M_{10}$, respectively. Again, note that the processing times become progressively faster. Figure D.6 shows the simulation model as it is built in Tecnomatix and represents BAP10.

$$\begin{tikzcd}
{} \arrow[r, "B_0 = \infty"] & \boxed{M_1} \arrow[r, "B_1"] & \boxed{M_2} \arrow[r, "B_2"] & \boxed{M_3} \arrow[r, "B_3"] & \boxed{M_4} \arrow[r, "B_4"] & \boxed{M_5} \arrow[r, "B_5"] & \boxed{M_6} \arrow[r, "B_6"] & \boxed{M_7} \arrow[r, "B_7"] & \boxed{M_8} \arrow[r, "B_8"] & \boxed{M_9} \arrow[r, "B_9"] & \boxed{M_{10}} \arrow[r, "B_{10} = \infty"] & {}
\end{tikzcd}$$

Figure D.5: *A series of ten machines $M_1, \ldots, M_10$ with finite buffers $B_1, \ldots, B_{10-1}$ in a queuing network, representative of BAP10*

Table D.7: *Processing times for the machines in BAP10*

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
Processing time	1:00:00	55:00	50:00	46:00	43:00	39:00	35:00	31:00	27:00	23:00



Buffer-Allocation model with 10 machines

Model inputs:
Buf1_Capacity=2
Buf2_Capacity=4
Buf3_Capacity=7
Buf4_Capacity=7
Buf5_Capacity=5
Buf6_Capacity=1
Buf7_Capacity=6
Buf8_Capacity=3
Buf9_Capacity=3

Model outputs:
WIP=0.802755602132952
Throughput=0

BufferPunchMaxOccupation=0
KPI=2.18421052631579
BufferBendingMaxOccupation1=0
SumAvgBufferSizes=7.22480041919657
SumBuffers=0
TableRow=1

BPunch=1.10228455665787
BLathe=1.37124457028691
BBend=0.552568262507406
BRivet=2.06028504918282
BBend1=0.682703747175421
BBend2=0.266722410629843
BBend3=0.742548462769052
BBend4=0.244608361255975
BBend5=0.201834998731271

B1_Count=0
B1_ODF=0
B2_Count=0
B2_ODF=0
B3_Count=0
B3_ODF=0
B4_Count=0
B4_ODF=0
B5_Count=0
B5_ODF=17
B6_Count=8
B6_ODF=21
B7_Count=16
B7_ODF=18
B8_Count=3
B8_ODF=17
B9_Count=15
B9_ODF=25

Figure D.6: *A screenshot of the BAP10 model in Tecnomatix.*

Observations per solution

Table D.8 shows the experiments used to determine whether or not 100 observations per solution are sufficient for BAP10 and is summarised in Table D.9.

Table D.8: *Experiments used to determine a sufficient number of observations per solution for BAP10*

										Work-in-progress			Throughput		
	B1	B2	B3	B4	B5	B6	B7	B8	B9	A	B	C	A	B	C
Exp 1	1	1	1	1	1	1	1	1	1	0.32	0.34	0.34	62	62.74	62.77
Exp 2	2	2	2	2	2	2	2	2	2	0.54	0.61	0.63	66.5	69.64	69.59
Exp 3	3	3	3	3	3	3	3	3	3	0.81	0.84	0.86	71.4	73.67	73.74
Exp 4	4	4	4	4	4	4	4	4	4	1.00	1.02	1.04	71.2	75.94	76.02
Exp 5	5	5	5	5	5	5	5	5	5	1.19	1.15	1.17	72.6	77.64	77.7
Exp 6	6	6	6	6	6	6	6	6	6	1.26	1.27	1.29	76.3	78.39	78.73
Exp 7	7	7	7	7	7	7	7	7	7	1.45	1.38	1.38	72.7	78.35	79.25
Exp 8	8	8	8	8	8	8	8	8	8	1.42	1.43	1.45	74.8	78.93	79.58
Exp 9	9	9	9	9	9	9	9	9	9	1.48	1.5	1.51	77.7	79.12	79.69
Exp 10	10	10	10	10	10	10	10	10	10	1.49	1.53	1.55	76.3	78.69	79.93

D.2.3 The non-linear buffer-allocation problem: 16 machines

The non-linear buffer allocation problem with $m = 16$ (BAP16), proposed by MacGregor Smith *et al.* [174], has exponential processing and repair times as well as probabilistic routings. BAP16 is schematically presented in Figure D.7.

Table D.10 shows the processing times for machines $M_1 - M_{16}$, respectively. However, note that

Table D.9: *Summary of the experiments used to determine the sufficient number of observations per solution for the BAP10.*

		Work-in-progress			Throughput		
Groups	Count	Sum	Average	Variance	Sum	Average	Variance
Exp 1	3	1.00	0.33	0	187.51	62.50	0.19
Exp 2	3	1.78	0.59	0	205.73	68.58	3.23
Exp 3	3	2.51	0.84	0	218.81	72.94	1.77
Exp 4	3	3.06	1.02	0	223.16	74.39	7.61
Exp 5	3	3.51	1.17	0	227.94	75.98	8.57
Exp 6	3	3.82	1.27	0	233.42	77.81	1.73
Exp 7	3	4.21	1.40	0	230.30	76.77	12.61
Exp 8	3	4.31	1.44	0	233.31	77.77	6.71
Exp 9	3	4.48	1.49	0	236.51	78.84	1.05
Exp 10	3	4.57	1.52	0	234.92	78.31	3.41

the processing times do not become progressively faster because of the non-linear topology.

Table D.10: *Processing times for the machines in BAP16*

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
Processing time	6:00	6:00	6:00	12:00	12:00	12:00	12:00	12:00	12:00	12:00	12:00	12:00	12:00	6:00	6:00	3:00

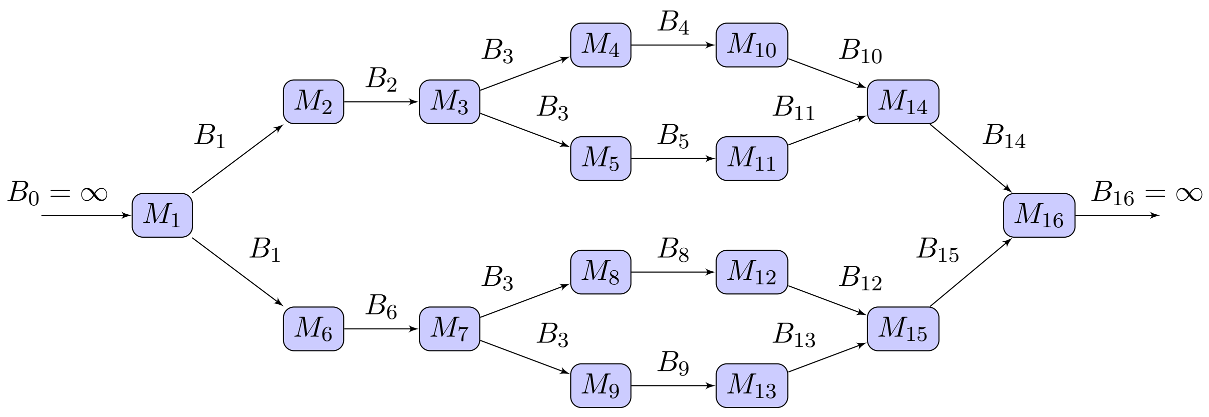
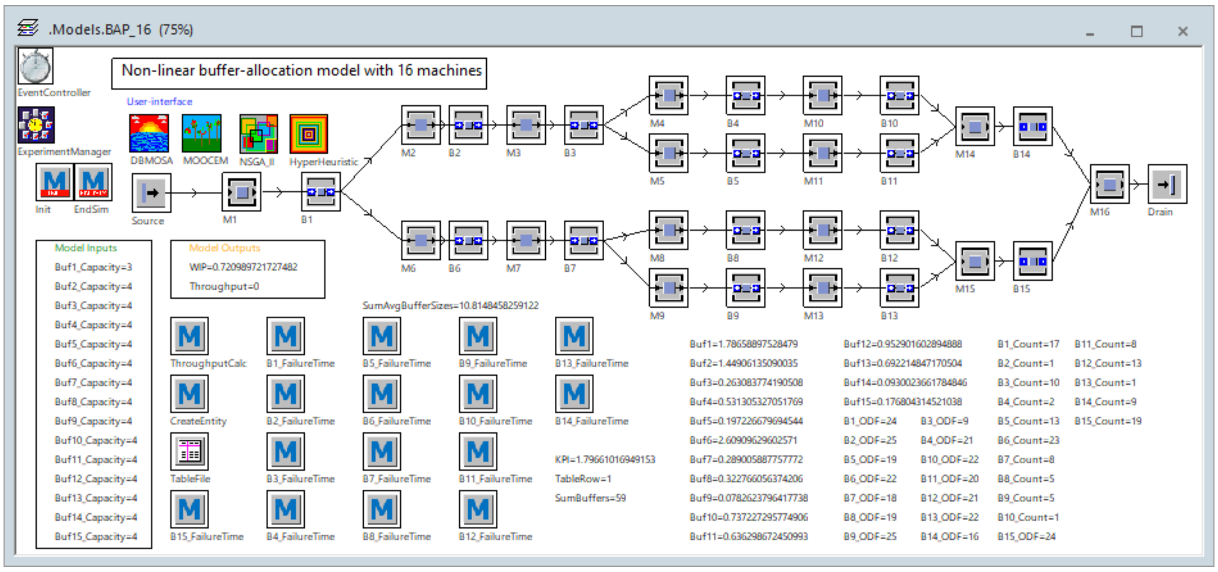


Figure D.7: *The non-linear BAP with sixteen machines $M_1, \ldots, M_{16}$ with finite buffers $B_1, \ldots, B_{16-1}$ in a queuing network*

Figure D.8 shows the simulation model as it is built in Tecnomatix and represents BAP16.

Observations per solution

Table D.11 shows the experiments used to determine whether or not 100 observations per solution are sufficient for BAP16 and is summarised in Table D.12.



Non-linear buffer-allocation model with 16 machines

Model Inputs

Buf1_Capacity=3
Buf2_Capacity=4
Buf3_Capacity=4
Buf4_Capacity=4
Buf5_Capacity=4
Buf6_Capacity=4
Buf7_Capacity=4
Buf8_Capacity=4
Buf9_Capacity=4
Buf10_Capacity=4
Buf11_Capacity=4
Buf12_Capacity=4
Buf13_Capacity=4
Buf14_Capacity=4
Buf15_Capacity=4

Model Outputs

WIP=0.720989721727482
Throughput=0

SumAvgBufferSizes=10.8148458259122

KPI=1.79661016949153
TableRow=1
SumBuffers=59

Buf1=1.78658897528479
Buf2=1.44906135090035
Buf3=0.263083774190508
Buf4=0.531305327051769
Buf5=0.197226679694544
Buf6=2.60909629602571
Buf7=0.289005887757772
Buf8=0.322766056374206
Buf9=0.0782623796417738
Buf10=0.737227295774906
Buf11=0.636298672450993
Buf12=0.952901602894888
Buf13=0.692214847170504
Buf14=0.0930023661784846
Buf15=0.176804314521038

B1_ODF=24
B2_ODF=25
B3_ODF=9
B4_ODF=21
B5_ODF=19
B6_ODF=22
B7_ODF=18
B8_ODF=19
B9_ODF=25
B10_ODF=22
B11_ODF=20
B12_ODF=21
B13_ODF=22
B14_ODF=16
B15_ODF=24

B1_Count=17
B2_Count=1
B3_Count=10
B4_Count=2
B5_Count=13
B6_Count=23
B7_Count=8
B8_Count=5
B9_Count=5
B10_Count=1
B11_Count=8
B12_Count=13
B13_Count=1
B14_Count=9
B15_Count=19

Figure D.8: *A screenshot of the BAP16 model in Tecnomatix.*

Table D.11: *Experiments used to determine a sufficient number of observations per solution for BAP16*

											Work-in-progress			Throughput		
	B1	B2	B3	B4	B5	···	B11	B12	B13	B14	B15	A	B	C	A	B	C
Exp 1	1	1	1	1	1	···	1	1	1	1	1	0.24	0.24	0.25	452.4	449.09	453.98
Exp 2	2	2	2	2	2	···	2	2	2	2	2	0.43	0.42	0.43	473.4	487.57	481.43
Exp 3	3	3	3	3	3	···	3	3	3	3	3	0.66	0.60	0.59	524.6	502.92	503.47
Exp 4	4	4	4	4	4	···	4	4	4	4	4	0.75	0.69	0.72	504.5	510.35	516.28
Exp 5	5	5	5	5	5	···	5	5	5	5	5	0.81	0.81	0.82	525.4	521.95	526.17
Exp 6	6	6	6	6	6	···	6	6	6	6	6	0.90	0.90	0.92	534.8	533.93	537.18
Exp 7	7	7	7	7	7	···	7	7	7	7	7	1.02	0.96	1.00	548.2	536.7	546.74
Exp 8	8	8	8	8	8	···	8	8	8	8	8	0.94	1.05	1.08	552.1	551.52	553.38
Exp 9	9	9	9	9	9	···	9	9	9	9	9	1.18	1.14	1.12	560.8	554.92	557.29
Exp 10	10	10	10	10	10	···	10	10	10	10	10	1.32	1.18	1.19	575.9	559.06	561.72

Table D.12: *Summary of the experiments used to determine the sufficient number of observations per solution for the BAP16.*

		Work-in-progress			Throughput		
Groups	Count	Sum	Average	Variance	Sum	Average	Variance
Exp 1	3	0.73	0.24	0	1 355.47	451.82	6.22
Exp 2	3	1.28	0.43	0	1 442.40	480.80	50.50
Exp 3	3	1.85	0.62	0	1 530.99	510.33	152.79
Exp 4	3	2.16	0.72	0	1 531.13	510.38	34.69
Exp 5	3	2.44	0.81	0	1 573.52	524.51	5.05
Exp 6	3	2.71	0.90	0	1 605.91	535.30	2.83
Exp 7	3	2.99	1.00	0	1 631.64	543.88	39.21
Exp 8	3	3.07	1.02	0.01	1 657.00	552.33	0.90
Exp 9	3	3.45	1.15	0	1 673.01	557.67	8.75
Exp 10	3	3.69	1.23	0.01	1 696.68	565.56	81.94

D.3 The Open mine problem

[The content of this section is an extract from unpublished notes, from the University of Stellenbosch (US), and are included here to define the problem.]

A rail operator has to transport crushed ore to a port for export. There are a number of trains in service which cycles between the open mine and the port. Each train consists of a set of locomotives and trucks. The trains arrive at a buffer yard where one of two loaders in the mine site is assigned to each train for loading.

There are maintenance requirements that need to be met and as a result the loaders must be cycled, *i.e.* when Loader 1 has just been assigned a train, the next train is assigned to Loader 2, regardless of whether Loader 2 is occupied or not. The cycle is thus Loader 1–Loader 2–Loader 1–Loader 2. There are two different physical waiting areas that the trains can be assigned to, one in front of each loader.

The mine operation uses a shovel (see Figure D.9a[1]) that loads a number of mine trucks (see Figure D.9b[2]). The shovel loads *TRI(140, 160, 200)*1000* kg per truck per loading time as given in Table D.13. The mine trucks then transport the ore to a crusher, which in turn forms a stockpile. The crushing processing time of a load is given in Table D.13.

Table D.13: *Loading and processing times for OMP*

	Distribution (minutes)
Shovel loading time	*TRI(18,12,28)*
Crusher processing time	*TRI(9,6,15)*

The distance between the shovel and crusher is 2 500m and the trucks travel at 15km/hr (loaded and unloaded). The delivery of a truck load of ore has the following effect on the stockpile (after crushing):

$$\text{StackSize} = \text{StackSize} + \text{Truck.LoadMass}. \tag{D.7}$$

The loaders retrieve the crushed product from the stockpile to fill the train trucks. The loaded mass is distributed *UNI(65,85)* metric tons and it takes *EXP(3)* minutes to load a train truck. A loader can only start to retrieve ore from the stockpile if the amount of ore is available to fill the alternate loader's train (*i.e.* the number of train trucks to be filled multiplied by 80 tons, plus 100 tons safety margin). The 80 ton value is just a guide the mine agreed upon.

If sufficient ore is not available in the stockpile, the loader must wait for the mine trucks to replenish the stockpile. If the stockpile gets depleted while a loader is loading, the loader will cease operation until some ore is available. The other loader must wait until the current loader has completely filled the train it was busy with. The two loaders thus do not compete for crushed ore when the stockpile gets low – one train is filled and dispatched rather than having two trains waiting.

So, suppose Loader 1 has loaded 36 of its train trucks and Loader 2 has loaded seven of its train trucks, when the stockpile becomes depleted. Now the loaders have to wait. When there are at least 200 metric tons of crushed ore available again, the loading process at Loader 1 resumes

[1]source: http://www.thompsoncreekmetals.com/i/photos/milligan/milligan9.jpg
[2]source: http://i.telegraph.co.uk/multimedia/archive/02256/Metal_to_Medal_0_2256835b.jpg

(a) *Mine shovel*

(b) *Mine truck*

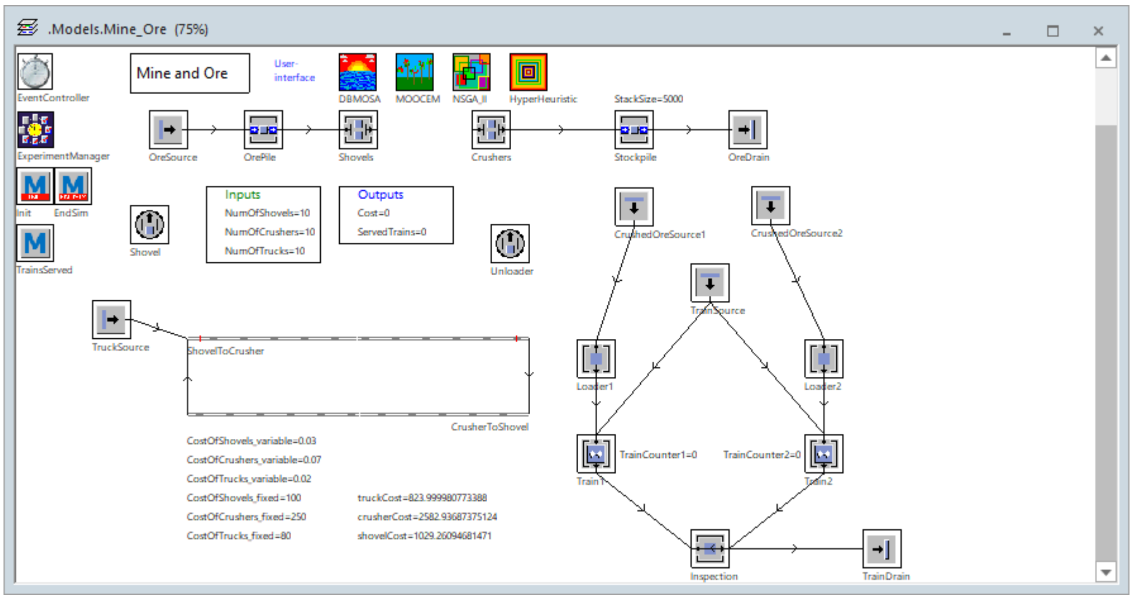
Figure D.9: *Examples of a (a) mine shovel and (b) mine truck in an open mine*

and Loader 2 does not load at all. Once Loader 1 has completed loading its 50 train trucks, Loader 2 may resume loading, provided the stockpile has sufficient (more than 200 metric tons) crushed ore. After a train is loaded by one of the loaders, the train is inspected. Inspection time is distributed *EXP(10)* minutes, after which it starts its 24 hour journey to the port.

Figure D.10 shows the simulation model as it is built in Tecnomatix and represents OMP.

Observations per solution

Table D.14 shows the experiments used to determine whether or not 100 observations per solution are sufficient for the OMP and is summarised in Table D.15.



StackSize=5000

Inputs
NumOfShovels=10
NumOfCrushers=10
NumOfTrucks=10

Outputs
Cost=0
ServedTrains=0

TrainCounter1=0
TrainCounter2=0

CostOfShovels_variable=0.03
CostOfCrushers_variable=0.07
CostOfTrucks_variable=0.02
CostOfShovels_fixed=100
CostOfCrushers_fixed=250
CostOfTrucks_fixed=80

truckCost=823.999980773388
crusherCost=2582.93687375124
shovelCost=1029.26094681471

Figure D.10: *A screenshot of the OMP model in Tecnomatix.*

Table D.14: *Experiments used to determine a sufficient number of observations per solution for the OMP*

				Total cost			Served trains		
	Shovels	Crushers	Trucks	A	B	C	A	B	C
Exp 1	4	4	4	1 773.93	1 773.82	1 773.81	11.8	11.61	11.55
Exp 2	5	5	5	2 217.44	2 217.41	2 217.37	13.8	13.85	13.84
Exp 3	6	6	6	2 660.96	2 660.88	2 660.89	14	14.16	14.18
Exp 4	7	7	7	3 104.66	3 104.45	3 104.41	14	14.16	14.18
Exp 5	8	8	8	3 548.03	3 547.93	3 547.92	14	14.16	14.18
Exp 6	9	9	9	3 991.66	3 991.43	3 991.38	14	14.16	14.18
Exp 7	10	10	10	4 434.92	4 434.78	4 434.83	14	14.16	14.18
Exp 8	11	11	11	4 878.51	4 878.30	4 878.31	14	14.16	14.18
Exp 9	12	12	12	5 321.86	5 321.78	5 321.77	14	14.16	14.18
Exp 10	13	13	13	5 765.36	5 765.14	5 765.20	14	14.16	14.18

Table D.15: *Summary of the experiments used to determine the sufficient number of observations per solution for the OMP.*

		Total cost			Served trains		
Groups	Count	Sum	Average	Variance	Sum	Average	Variance
Exp 1	3	5 321.57	1 773.86	0	34.96	11.65	0.02
Exp 2	3	6 652.23	2 217.41	0	41.49	13.83	0
Exp 3	3	7 982.74	2 660.91	0	42.34	14.11	0.01
Exp 4	3	9 313.51	3 104.50	0.02	42.34	14.11	0.01
Exp 5	3	10 643.88	3 547.96	0	42.34	14.11	0.01
Exp 6	3	11 974.47	3 991.49	0.02	42.34	14.11	0.01
Exp 7	3	13 304.53	4 434.84	0.01	42.34	14.11	0.01
Exp 8	3	14 635.12	4 878.37	0.01	42.34	14.11	0.01
Exp 9	3	15 965.41	5 321.80	0	42.34	14.11	0.01
Exp 10	3	17 295.70	5 765.23	0.01	42.34	14.11	0.01

APPENDIX E

Algorithmic Comparison Results

This appendix contains additional results obtained during the algorithmic comparisons for the hyperheuristics, as described in Chapter 7 and were omitted from the respective sections so as to enhance the understanding of the main text.

In Figures E.1 and E.2, the best and worst approximation fronts are plotted for the BOSAH and the BOCEGAH for hyperarea and number of non-dominated solutions, for the respective simulation problems as summarised in Tables E.2 and E.1.

Table E.1: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the BOCEAH) for the hyperarea performance indicator and the number of non-dominated solutions found for each simulation problem.*

	Worst		Best	
	HA	NDS	HA	NDS
OMP	Run 1	Run 1	Run 30	Run 5
IP	Run 94	Run 49	Run 19	Run 14
BAP5	Run 60	Run 34	Run 16	Run 89
BAP10	Run 57	Run 27	Run 35	Run 48
BAP16	Run 75	Run 75	Run 97	Run 89

Table E.2: *The simulation runs that correspond to the best and worst approximation fronts (obtained by the BOSAH) for the hyperarea performance indicator and the number of non-dominated solutions found for each simulation problem.*

	Worst		Best	
	HA	NDS	HA	NDS
OMP	Run 39	Run 39	Run 3	Run 3
IP	Run 97	Run 42	Run 3	Run 16
BAP5	Run 57	Run 83	Run 35	Run 25
BAP10	Run 57	Run 79	Run 76	Run 1
BAP16	Run 27	Run 19	Run 3	Run 24

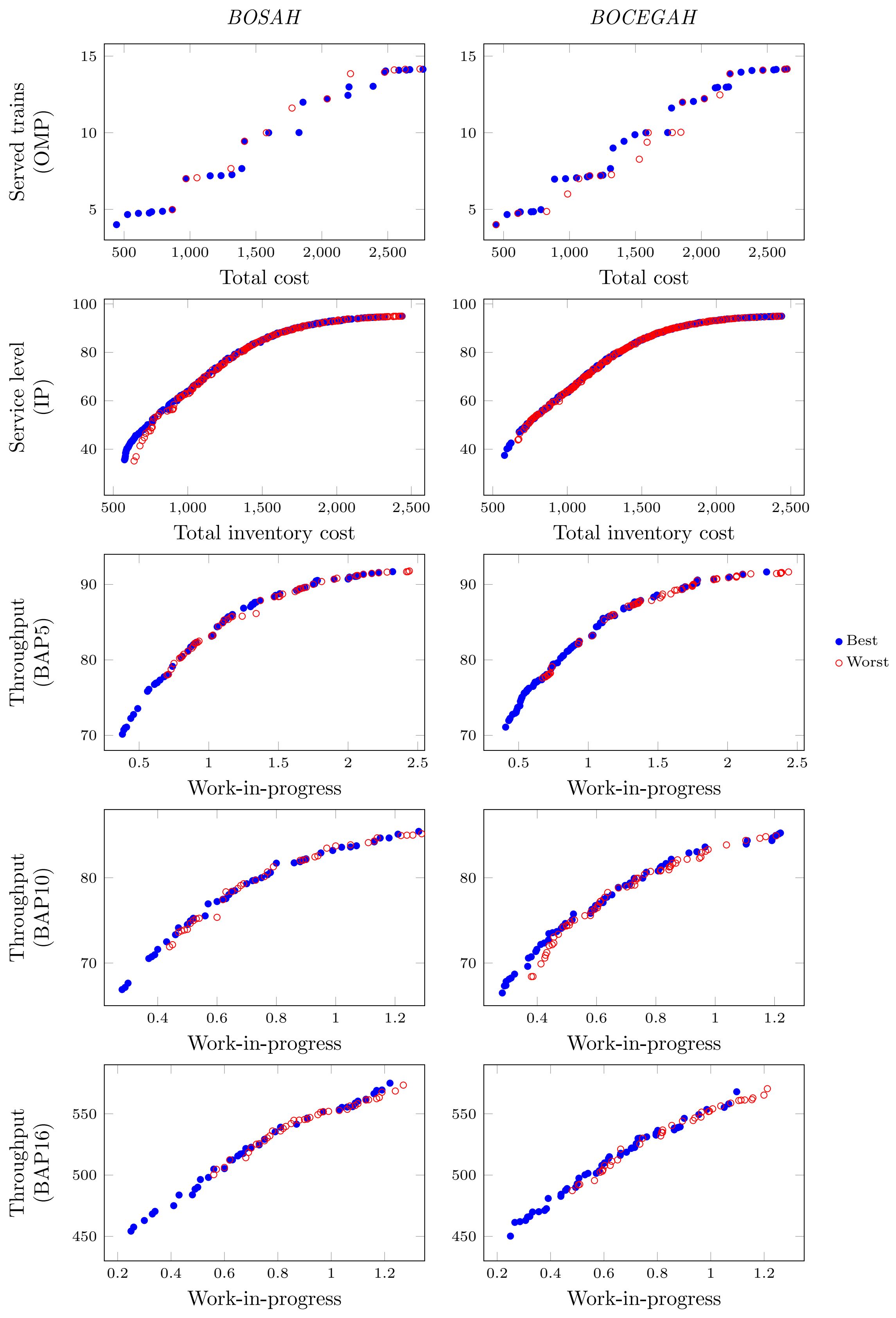


Figure E.1: *The best and worst best approximation fronts obtained for the BOSAH and the BOCEGAH in terms of hyperarea.*

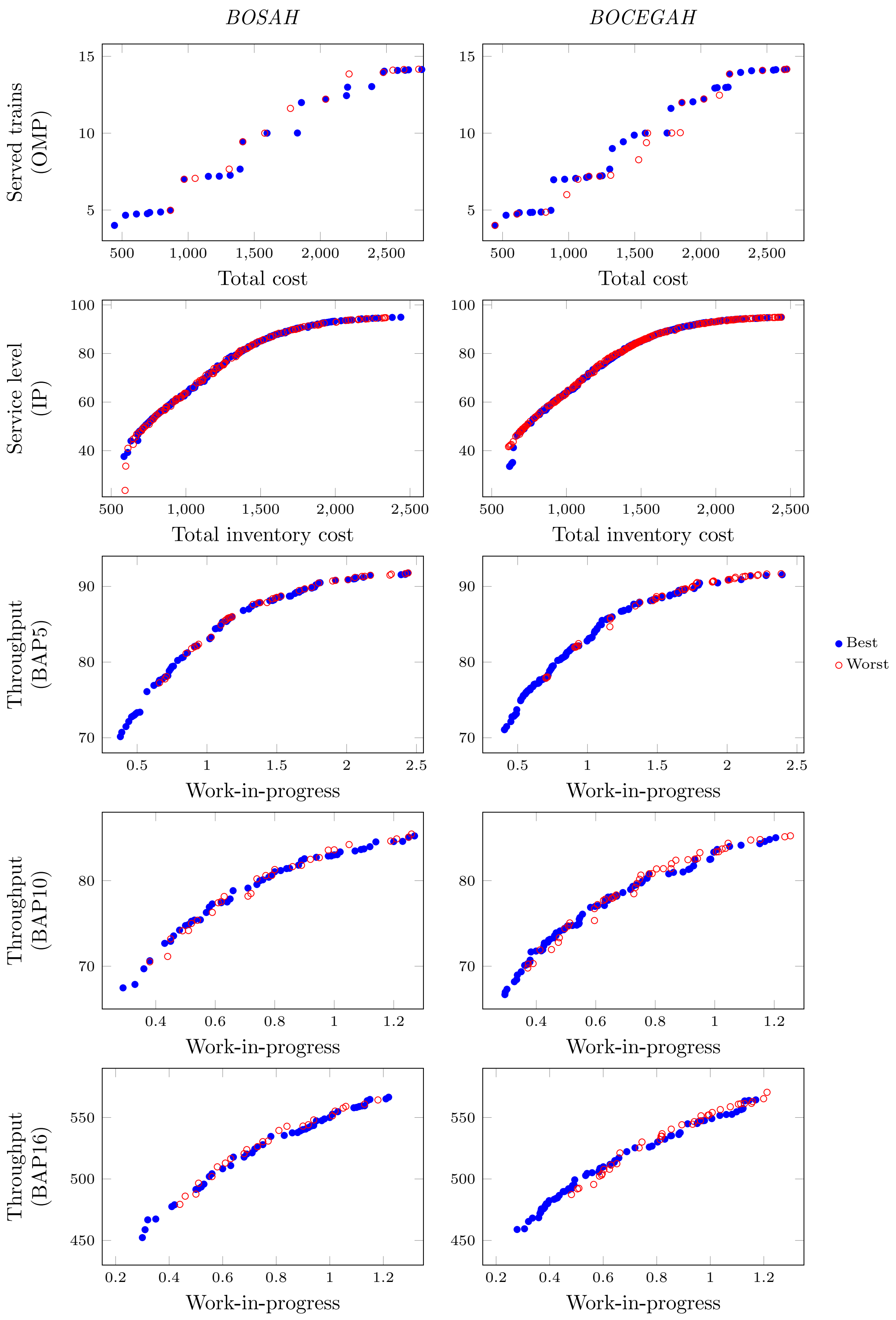


Figure E.2: *The best and worst best approximation fronts obtained for the BOSAH and the BOCEGAH in terms of the number of non-dominated solutions.*