

## University of Southampton Research Repository

Copyright © and Moral Rights for this thesis and, where applicable, any accompanying data are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis and the accompanying data cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content of the thesis and accompanying research data (where applicable) must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holder/s.

When referring to this thesis and any accompanying data, full bibliographic details must be given, e.g.

Thesis: Yiyun Cao (2024) "The Use of Multi-Fidelity Simulation Optimisation for Real-Time Management of a Manufacturing Line", University of Southampton, name of the University Faculty or School or Department, PhD Thesis, pagination.



UNIVERSITY OF SOUTHAMPTON

Faculty of Social Sciences  
Southampton Business School

**The Use of Multi-Fidelity Simulation  
Optimisation for Real-Time Management of  
a Manufacturing Line**

*by*

**Yiyun Cao** 

MSc

*A thesis for the degree of  
Doctor of Philosophy*

January 2024




University of Southampton

Abstract

Faculty of Social Sciences  
Southampton Business School

Doctor of Philosophy

**The Use of Multi-Fidelity Simulation Optimisation for Real-Time Management of a Manufacturing Line**

by Yiyun Cao 

Manufacturers often rely on simulation models to support decision-making for a wide range of tasks from system design to routine operations. There is scope for simulation modelling to extend further to deal with situations that look for fast responses, ideally in real time. Simulation models for manufacturing systems are usually complex and large, meaning they take a long time to run. They are not able to support real-time decision-making when working with conventional simulation optimisation procedures which require multiple replications for producing the recommendations. A multi-fidelity simulation optimisation framework is designed to address this problem. It includes a low-fidelity metamodel to guide the search using the high-fidelity simulation model, which aims to reduce the replications of the high-fidelity model needed in the optimisation. The proposed method has been tested on a well-known simulation model of an inventory system and a new model of a production line. The results from both the textbook example and the manufacturing system show that it satisfies the need for real-time optimisation and performs well. In the production line case, we aim to optimise the repair policy when multiple machines break down simultaneously. The order to repair the failed machines could affect how the system would recover and would lead to different short-term system throughput. By optimising the repair policy, the system throughput is optimised. In order to carry out the optimisation in real time, a “hot-start” simulation model is structured for the production line system. The state space of the metamodel for the production line is modified and reduced in order to fit the metamodel with fewer data. An adaptive sequential sampling method is developed to efficiently sample from the stochastic simulation model.



# Contents

<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>ix</b>
<b>List of Selected Notations</b>	<b>xi</b>
<b>Declaration of Authorship</b>	<b>xii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Description . . . . .	2
1.2 Project Overview . . . . .	3
1.3 Research Questions . . . . .	5
1.4 Structure of the Thesis . . . . .	6
<b>2 Literature Review</b>	<b>7</b>
2.1 Simulation in Manufacturing . . . . .	7
2.2 Real-time Optimisation with Digital Twin and Symbiotic Simulation . .	8
2.3 Discrete Optimisation via Simulation . . . . .	10
2.3.1 Indifference-Zone Procedure . . . . .	10
2.3.2 Efficient Simulation Budget Allocation Procedure . . . . .	11
2.3.3 Fast Simulation Optimisation for Real-time Decision Making . .	12
2.4 Multi-Fidelity Simulation Optimisation . . . . .	13
2.4.1 MO <sup>2</sup> TOS . . . . .	14
2.4.2 MFSKO & VGO . . . . .	15
2.4.3 Other Methods . . . . .	16
2.5 Simulation Metamodels . . . . .	18
2.5.1 Polynomial Regression . . . . .	18
2.5.2 Radial Basis Function Regression . . . . .	18
2.5.3 Gaussian Process Regression . . . . .	19
2.5.4 Random Forest . . . . .	19
2.5.5 Artificial Neural Network models . . . . .	20
2.5.5.1 Deep Neural Network . . . . .	21
2.5.6 Explainable Boosting Machine . . . . .	22
2.5.7 Comparison of modelling methods . . . . .	22
2.6 Experimental Design . . . . .	23
2.6.1 Space-filling Design . . . . .	23
2.6.2 Adaptive Sequential Sampling . . . . .	26
2.7 Multi-Objective Simulation Optimisation . . . . .	27

2.7.1	Multiple Attribute Utility Theory . . . . .	27
2.7.2	Pareto Front . . . . .	28
2.8	Conclusion . . . . .	30
<b>3</b>	<b>Multi-fidelity Simulation Optimisation</b>	<b>33</b>
3.1	Optimisation Algorithm . . . . .	34
3.2	Metamodel . . . . .	36
3.3	Ranking & Selection . . . . .	37
3.4	Numerical Experiments . . . . .	39
3.4.1	Inventory System Example . . . . .	40
3.4.2	Building the Metamodel . . . . .	41
3.4.3	Optimisation . . . . .	43
3.4.4	Results Analysis . . . . .	45
3.5	Conclusion . . . . .	49
<b>4</b>	<b>Modelling the Production Line</b>	<b>51</b>
4.1	System Description . . . . .	51
4.2	Repair Policies . . . . .	54
4.3	Simulation Model . . . . .	56
4.3.1	Simulation Model Structure . . . . .	57
4.3.2	Simulation Output . . . . .	60
4.4	Metamodel . . . . .	61
4.4.1	Input Variables Formation . . . . .	62
4.4.2	Experimental Design . . . . .	68
4.4.3	Test Set Preparation . . . . .	74
4.4.4	Fitting the Metamodel . . . . .	76
4.5	Summary . . . . .	78
<b>5</b>	<b>Real-Time Optimisation of the Production Line</b>	<b>81</b>
5.1	Multi-fidelity Framework . . . . .	82
5.2	Numeric Experiments . . . . .	84
5.3	Conclusions . . . . .	91
<b>6</b>	<b>Conclusions</b>	<b>93</b>
6.1	Contributions . . . . .	93
6.1.1	Methodological Contributions . . . . .	94
6.1.2	Empirical Contributions . . . . .	94
6.2	Implications for Practice . . . . .	95
6.3	Future Work . . . . .	97
6.3.1	Fusing multi-objective optimisation into the multi-fidelity simulation optimisation framework . . . . .	97
6.3.2	Working with different types of metamodels . . . . .	98
<b>Appendix A Difference Between the Optimal Repair Policy and Other Alternatives in 100-replication Gold Standard (<math>D_m</math>)</b>		<b>101</b>
<b>Appendix B Complete Experiment Results in Production Line Example</b>		<b>103</b>
<b>References</b>		<b>105</b>



# List of Figures

1.1	Overview of the Multi-Fidelity Approach . . . . .	4
2.1	Feedforward Neural Network . . . . .	21
2.2	Illustrations of the Maximin Distance Design (left) and the Minimax Distance Design (right) . . . . .	24
2.3	An illustration of the Maximin Latin Hypercube Sampling . . . . .	26
3.1	Flowchart of the proposed algorithm. Left: offline process, output fitted metamodel. Right: online optimisation process, output final solution. . .	34
3.2	Evaluation of solutions from metamodel . . . . .	43
3.3	Residual between metamodel and simulation model . . . . .	43
3.4	Grouping clusters on ranked solutions (a=1) . . . . .	44
3.5	Grouping clusters on ranked solutions (a=2) . . . . .	44
3.6	Grouping clusters on ranked solutions (a=3) . . . . .	44
3.7	Comparison of the proposed method with different aggression parameters and MO <sup>2</sup> TOS . . . . .	46
3.8	Comparison of the proposed method with different aggression parameters and MO <sup>2</sup> TOS for Less-accurate Metamodel . . . . .	47
4.1	Frequency of overlapping breakdowns . . . . .	52
4.2	Frequency of multiple simultaneous breakdowns . . . . .	53
4.3	Illustration of a production line . . . . .	57
4.4	Plot of hourly throughput for a 500-hour simulation with 'FIFO' repair policy . . . . .	60
4.5	Plot of jobs per hour for 500-hour simulations with 10 repair policies . .	60
4.6	95% CI of the differences between scenarios with even split of 3 sections	64
4.7	95% CI of the differences between scenarios with even split of 4 sections	66
4.8	Flowchart of the Experimental Design . . . . .	70
4.9	Production line section buffer level (V1) of 10 test sets . . . . .	75
4.10	Breakdowns (instances of V2 & V3) of 10 test sets . . . . .	75
5.1	Flowchart of the Multi-fidelity Simulation Optimisation Framework returning single solution . . . . .	82
5.2	Illustration of aggression parameter for 10 repair policies . . . . .	83
5.3	Histogram for the situation when the repair policy performs the best from the collected simulation data . . . . .	85
5.4	Aggregated frequency of optimisation results meet the optimal repair policies and the alternatives that are not significantly different from the optimum . . . . .	88

5.5	Aggregated difference to the optimum . . . . .	90
-----	------------------------------------------------	----

# List of Tables

3.1	Structure of the groups (a=1)	45
3.2	Structure of the groups (a=2)	45
3.3	Structure of the groups (a=3)	45
4.1	Input parameter form for DES model	58
4.2	Input status for DES model	59
4.3	The assignment of the buffers for three sets of configurations within each section in the pilot test (3-section design)	63
4.4	27 scenarios of the 3-section design	63
4.5	The assignment of the buffers for three sets of configurations within each section in the pilot test (4-section design)	64
4.6	81 scenarios of the 4-section design	65
4.7	Average throughput of the collected data for test	76
4.8	Metrics recorded for the metamodel	78
5.1	Structure of the groups	83
5.2	Optimisation result for the production line model on one trial (yellow cells capture the optimal repair policies and the light blue cells capture the repair policies that are not significantly different from the optimum)	87
5.3	Difference between selected solution and optimum on one trial	87
5.4	Aggregated difference between selected solution and optimum	90
Appendix A.1	Difference between the optimal repair policy and other alternatives ( $D_m$ ) Part 1	101
Appendix A.2	Difference between the optimal repair policy and other alternatives ( $D_m$ ) Part 2	102
Appendix B.1	Complete experiment results for 10 testsets in 10 trials	104



## List of Selected Notations

$a$	aggression parameter	MFSO
$b$	best group in the iteration	algorithm 1, OCBA
$d(\cdot)$	distance	
$D$	experimental designs	
$\mathbb{E}[\cdot]$	expectation	
$i^*$	group for more simulation	algorithm 1, OCBA
$j^*$	best solution	algorithm 1, OCBA
$k$	groups of solutions	algorithm 1, OCBA
$k^*$	best group	algorithm 1, OCBA
$f(\cdot)$	objective function	
$I$	inventory level	inventory system
$m$	trial	production line case
$n$	sample budget for each iteration	adaptive sequential experimental design
$n$	replication in 'gold standard'	production line case
$N$	total sample budget	adaptive sequential experimental design
$P$	number of clusters	MFSO
$Q$	number of groups	MFSO
$s$	re-order point	inventory system
$S$	maximum inventory level	inventory system
$t$	test set	production line case
$x$	decision variable	
$X$	decision set	
$Y(x, \zeta)$	output random variable	
$Z$	order size	inventory system
$\beta$	total budget	algorithm 1, OCBA
$\theta$	budget increment	algorithm 1, OCBA
$\mu$	mean	
$\sigma$	standard deviation	
$\sigma^2$	variance	
$\tau$	Kendall's rank correlation coefficient	

## Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as: Yiyun Cao, Christine Currie, Bhakti Stephan Onggo, and Michael Higgins. Simulation optimization for a digital twin using a multi-fidelity framework. In *Proceedings - Winter Simulation Conference*, pages 1–12, 2021.

Signed: .....Yiyun Cao.....

Date: .....19 December 2023.....

# Chapter 1

## Introduction

Digital twin technology has been applied in many different scenarios for next-generation manufacturing. However, there is still a research gap related to digital twin in using simulation to optimise the manufacturing system. A digital twin is a virtual replica of a real machine, system or process that has the same behaviour or function as the original real-world entity. Its applications can be found in many manufacturing phases, for example, design, operation and maintenance. In system operations, digital twin requires optimal solutions to be generated in real-time. Fast (real-time/near real-time) simulation optimisation thus should lie at the core of digital twin to obtain the best performance for the system. In the road map introduced by [Glaessgen and Stargel \(2012\)](#), the digital twin was defined as an integrated multiphysics, multiscale, probabilistic simulation of an as-built system.

This project focuses on proposing a method to utilise multi-fidelity simulation optimisation for fast and reasonable decision support. After testing the method on a textbook example (see Section 3.4), it is applied to a real-life problem. We build a discrete event simulation model for a complex production line (see Chapter 4) and use near real-time optimisation to select the best repair order when multiple machines on the production line break down and request repair simultaneously.

Simulation optimisation is often used to obtain the optimal settings for those problems containing some uncertainty that have no closed form of the objective function. As we are working with a stochastic system and using stochastic simulation to describe it, we aim to minimise the expected value of a random output variable. It can be represented as

$$\min_{\mathbf{x} \in \Theta} f(\mathbf{x}), \quad (1.1)$$

where  $f(\mathbf{x}) = \mathbb{E}[Y(\mathbf{x}, \xi)]$  ([Fu, 2014](#)). The objective function  $f(\mathbf{x})$  represents the expectation of a random variable  $Y(\mathbf{x}, \xi)$ , where  $\xi$  stands for the randomness (which is represented by the random numbers in a stochastic simulation model). Although the

expectation is the most used objective function, other statistics can be used; for example a quantile or a variance. As discussed in [Hong and Nelson \(2009\)](#), simulation optimisation algorithms aim to maximise the information that can be obtained from a fixed computational budget or reach certain accuracy criteria with a proper number of simulation replications. A large number of simulation replications are normally required for a conventional ranking and selection procedure, e.g. KN++ ([Kim and Nelson, 2006b](#)), Optimal Computing Budget Allocation (OCBA) ([Chen et al., 2000](#)). It is computationally expensive to simulate complex systems. Thus, the whole optimisation process can be inevitably time-consuming. Running the algorithm on more than one computing processor, such as parallel computing ([Mubarak et al., 2017](#)) and cloud computing ([Calheiros et al., 2009](#); [Taylor et al., 2020](#)), is a way to decrease the computing time with the cost on devices or services. Apart from advanced hardware, the multi-fidelity framework ([Xu et al., 2014](#)) is one of the approaches that could help. This is the focus of this thesis.

Multi-fidelity models for simulation optimisation were introduced to involve a less complex low-fidelity model in the procedure. It is used to identify promising solutions which can then be tested on the high-fidelity model. This reduces the computational burden at runtime and so improves the speed at which a response is returned. The low-fidelity model is a simplified model with more assumptions and/or neglecting some details. It can be a discrete event simulation model or a statistical metamodel. The metamodel aims to map the relationship between input and output variables, which describes the behaviour of the original simulation model. We choose to use a neural network model in this project as our low-fidelity model. This allows us to achieve good enough results when having a limited computing budget or limited time for optimisation. A reduction in the number of replications means less computation is needed to obtain the optimal selection, which leads to a faster process for simulation optimisation.

In this project, a multi-fidelity model framework is used to find the best solution from a given search space in a comparably fast sense for a production line. We test the proposed method initially on a small example before implementing it on a simulation of a manufacturing system.

## 1.1 Problem Description

We consider a complex production line made up of more than twenty workstations consisting of either single or parallel machines. Having used simulation frequently in the past for planning tasks associated with the production line, the manufacturer now wishes to use simulation results in real-time for system optimisation, making use of sensors to indicate the current state of the line. Results are required in near real-time if



they are to be useful in facilitating decision-making. Incorporating a low-fidelity model, such as a metamodel, into the optimisation procedure may enable it to be practical for routine management. Building a statistical model from the real data directly seems attractive and could bypass the time-consuming high-fidelity simulation. However, very limited scenarios would have a chance to appear in operation and be documented. The small amount of data from the shop floor makes it hard to fit models from collected data. Building a metamodel from a simulation model allows users to have more control over the process of data collection and thus be more confident about the results.

We consider the problem of scheduling repairs on the production line when the maintenance team is limited. Repairing all broken machines at the same time may not be possible due to the limited capacity of the maintenance team. Priority should be given to the machine whose recovery would have the most positive effect. The optimisation is carried out by selecting a strategy for ordering repairs that would have the highest expected simulated throughput in a three-hour time window.

In order to achieve the aim of the research, we work on the following tasks: designing the overall multi-fidelity approach, testing the multi-fidelity approach on a textbook example, building a simulation model and enabling “hot starts”, deciding on the state space, designing experiments and collecting data, fitting the metamodel and testing the multi-fidelity approach on the manufacturing system.

## 1.2 Project Overview

We overview the research of the multi-fidelity approach in Figure 1.1. It briefly describes how major components of the project interact with each other and indicates some of the contributions.

The state of the system is collected from the physical entity, which is input into the simulation model and the metamodel during the optimisation. The metamodel can be used to predict the performance of the system. The core part of the multi-fidelity optimisation framework is to use estimates from the low-fidelity model to guide the search using the high-fidelity model. We propose a method for doing this in Chapter 3.

The main components that are involved in forming the optimisation process are connected in Figure 1.1. Two interlinked models lie in the centre. The whole process is triggered when an optimisation request is raised. After that, the metamodel is run first and its results are fed into the simulation optimisation procedure using the high-fidelity model. The recommendation is sent back to support decision-making.

The metamodel is fitted to the output of the simulation model through the design of experiments. For complex systems with a limited computing budget for data

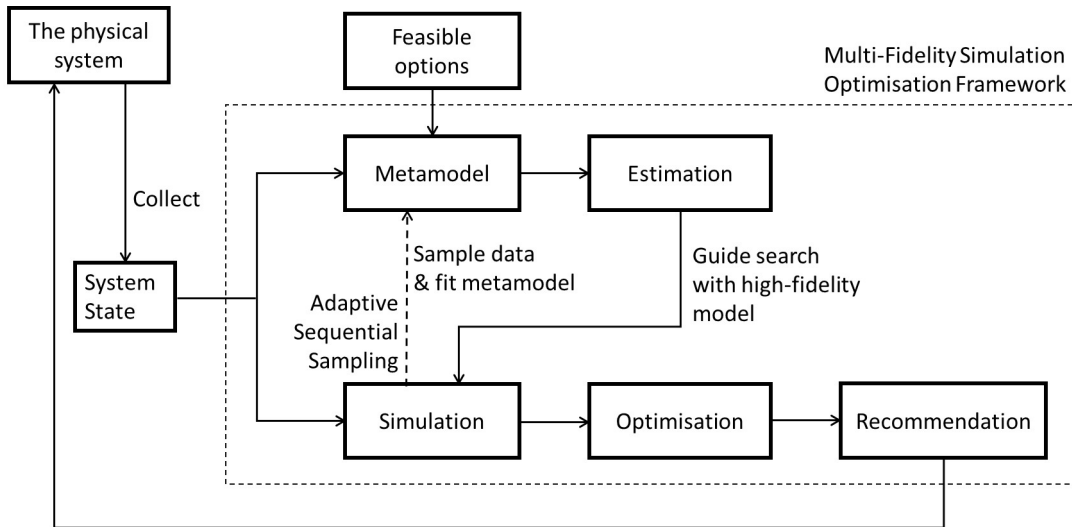


FIGURE 1.1: Overview of the Multi-Fidelity Approach

collection, a sampling method that sequentially collects data is developed to sample efficiently from the simulation model.

The work presented in this thesis contributes to both academia and industry. It contains both methodological contributions and empirical contributions. The multi-fidelity simulation optimisation methodology developed in Chapter 3 optimises the complex system in a fast sense while still giving reasonable recommendations. Our framework uses metamodel as the low-fidelity model other than the simplified simulation model in the literature and the estimations from the low-fidelity model are used differently. Alternative solutions are sorted into groups with various sizes based on the metamodel estimations so that we can put emphasis on the alternatives that are predicted to have better performances. An experimental design method is proposed in Section 4.4.2 to sequentially sample the complex system and make the most out of the limited computing budget. The sampling method explores new areas and exploits the sampled areas when necessary in each iteration. A case study that applies the proposed method to a production line model is demonstrated in Chapter 4 and 5. The relevant contents, for example, modelling the production line system and reducing the dimension of the input variables contribute empirically.

The case study is based on the operations of a major company in the automobile industry that runs a similar production plan and uses simulation models to support its decision-making. The production system under study is modified from a small part of the original line. We are not able to provide every detail for reasons of confidentiality.

## 1.3 Research Questions

The thesis aims to address the following research questions, which draw on the production line example and intend to fill gaps identified in the literature, as will be discussed in Chapter 2.

1. How can existing ranking & selection procedures be improved to provide results close to real-time?

The ranking and selection (R&S) procedures simulate all the candidate solutions a number of times and select the one that performs the best. The procedure would thus take a lot of time to run when the simulation model is large and complex and/or there are a large number of candidate solutions. Both of these may be true for many manufacturing systems. This means that conventional R&S may not be able to meet the requirement of optimising in real-time for manufacturing systems.

2. How to implement a multi-fidelity framework for fast (real-time) optimisation of a production line?

The multi-fidelity approach utilises multiple models at various fidelities for optimisation: typically a high-fidelity model and a low-fidelity model. The estimation from the low-fidelity model is expected to guide the search using the high-fidelity simulation model. Different algorithms in the literature cope with the problem differently. We introduce a multi-fidelity framework that can speed up the optimisation of a stochastic system.

3. What is the most efficient design for a metamodel of a production line simulation?

There are a large number of variables for the system state in the “hot-start” simulation model. A large amount of data is required for fitting the metamodel and it could be time-consuming for collecting data. If the number of input variables for the metamodel could be reduced, less data need to be collected for fitting the metamodel thus the whole process could become faster. We describe a state-space reduction that maintains the functionality of the metamodel.

4. How to quantify the performance of the metamodel on the quality of the optimisation results?

The metamodel that works as the low-fidelity model in the multi-fidelity framework guides the search for the optimal solution with the high-fidelity model. Estimation accuracy is usually used to measure the performance of a model/metamodel. However, there might be other metrics to efficiently quantify the performance of the metamodel in the setting of a multi-fidelity approach.

#### 5. How to simulate a production line for optimising the repair order in real time?

For an asset like a production line, a normal way to build the simulation model is to assume that the system would start from time zero when the system is empty. However, this introduces significant bias to the evaluation of outputs for real-time simulation optimisation. As a result, we describe the use of hot-start simulation to ensure the initial state of the simulation closely matches that of the real system. The repair policy is used as the decision variable in the hot-start simulation which determines the repair orders in the simulation. It avoids the option for repair orders to increase exponentially when the number of breakdowns increases.

## 1.4 Structure of the Thesis

There are another five chapters in the rest of the thesis. We review the literature on the topics related to the research in Chapter 2, which includes simulation in manufacturing (Section 2.1), digital twin (Section 2.2), discrete simulation optimisations (Section 2.3), multi-fidelity model for simulation optimisations (Section 2.4), simulation metamodels (Section 2.5) and experimental designs (Section 2.6). A multi-fidelity simulation optimisation framework that utilises a metamodel along with the simulation for optimisation is proposed in Chapter 3. We test the methodology with a textbook example - an inventory system. The result of the experiments and analyses are elaborated in Section 3.4. In Chapter 4, we illustrate the process of structuring the simulation model and the metamodel for a production line. An experimental design is proposed for efficiently collecting data and fitting the metamodel in Section 4.4.2. In Chapter 5, we apply the proposed multi-fidelity simulation optimisation framework to the production line models. The conclusion and some future plans are given in Chapter 6.

## Chapter 2

# Literature Review

This chapter is divided into six major sections and reviews the literature for the key topics of the thesis. The contents move step by step from the big concepts down to the specific approaches. The literature under review is the research published before 2022. It begins by discussing research that concerns the utilisation of simulation in manufacturing. Since the case study is based on an example in the automotive industry, we review the application of simulation in the automotive industry in Section 2.1. In the next section, a discussion of the literature associated with real-time optimisation with digital twins and symbiotic simulation is given. We continue to walk through the methodology of simulation optimisation and then the more focused topic of multi-fidelity simulation optimisation. After that, methods for structuring metamodel are reviewed, and next, the topic is extended to include multi-objective simulation optimisation. The multi-objective optimisation is reviewed but not covered in our proposed real-time simulation optimisation methodology due to the project time limit. However, it is still valuable and would be part of the future work. Lastly, a conclusion is given before we finish this chapter and move on to the methodology.

### 2.1 Simulation in Manufacturing

Simulation as a tool to model the behaviour of the system has been widely used in the manufacturing industry. [Smith \(2003\)](#) reviewed the application of simulation in manufacturing and classified the relevant research into three major groups, which are manufacturing system design, manufacturing system operation and simulation language. The review claims that system design concerns long-term plans (e.g. facility layout design and system configuration) and system operation concerns short-term decision-making. Later reviews (e.g. [Negahban and Smith \(2014\)](#); [Mourtzis \(2020\)](#)) follow the classification for the application of simulation and include more state-of-the-art works.

Negahban and Smith (2014) argue discrete event simulation has been widely used to assist the planning and scheduling of manufacturing operations. Its applications could be found in the research studying flow shop (Huang et al., 2013), job shop (Thürer et al., 2012) and flexible manufacturing systems (Joseph and Sridharan, 2012). Mourtzis (2020) categorised the research into finer phases in the product and production lifecycle from system design to process planning to supply chain planning. The author includes the 'digitalisation of simulation' in their survey, which refers to the simulation research towards new technology such as digital twin, augmented reality and virtual reality. Alrabghi and Tiwari (2015) reported that discrete event simulation is the most widely used simulation technique (which has been applied to sixty per cent of their reviewed papers) in their survey of simulation-based optimisation for maintenance.

Automotive, as a major manufacturing industry, also attracts a lot of research interest in applying simulation to model the manufacturing system. For example, Xu et al. (2012) developed a simulation model for multi-criteria decision-making in an automotive manufacturing system. Ordaz et al. (2015) described a simulation-based virtual training system for educating and training the operators of an automotive production line. El-Khalil (2015) implemented discrete-event simulation to identify the bottleneck of the assembly line and optimise the production plan for an automotive company.

## 2.2 Real-time Optimisation with Digital Twin and Symbiotic Simulation

A digital twin can be thought of as a virtual replica of the real-world system that runs alongside the physical system. When the system updates or input changes, data are synchronised to the digital twin. The system state of the model is then modified and the model parameters could also be reconfigured. The concept has been prevalent in many different industries since the idea was mooted by NASA during the Apollo project in 1969 (Glaessgen and Stargel, 2012). It also goes under different names, digital twins (Glaessgen and Stargel, 2012), symbiotic simulation (Onggo et al., 2018) and real-time simulation (Pedrielli et al., 2019) are the three most common ones. The aim of this technology is to enable real-time decision-making that accounts for the current state of the system.

Zhang and Zhu (2019) claimed that digital twins have three key parts: object, model and data. While their focus in the paper is on its use at the machine level, digital twin technology is applicable from the machine level to the production line level (manufacturing system) to even the factory level (system of systems). Most research on digital twins lies in optimising production processes, fault diagnosis and

prognostics. They display an example of producing turbine blades under the framework of a digital twin at the machine level by mounting extra sensors and a vision system onto a computer numerical control (CNC) machine to track the movement of the physical system. After this, both machine time and the precision of the product are improved.

Digital twins rely on data from the systems being modelled and these data may come from sensors or, as [Ivanov et al. \(2019\)](#) described, they can also make use of radio-frequency identification (RFID) or geographic information systems (GIS) to update the system state. The choice of input sources depends on the nature of the system being modelled but is also constrained by the availability of data.

The framework proposed in [Ivanov et al. \(2019\)](#) makes use of blockchain, RFID and GIS to communicate with the supply chain model in simulation and optimisation. Their method aims to mitigate the risk and ripple effect by predicting disruptions and taking recovery policies.

While in [Rosen et al. \(2018\)](#), the digital twin is regarded as a collection of data from design, engineering and operation. The digital twin of a large-scale system could consist of a set of smaller-scale digital twins. In addition, digital twin could be circulated between suppliers and integrators to improve the value chain. It could even become a part of the product. [Rosen et al. \(2018\)](#) showed a case study of a point machine in the railway system for operating the turnouts involving a digital twin from design to operation.

[Onggo et al. \(2018\)](#) claimed that a symbiotic system is comprised of the digital twin, the optimisation methods, data collection and a control system, together with the corresponding physical entity. It is designed to work with analytic approaches so that the optimisation can interact with the physical system. Heuristic algorithms and machine learning are considered two important groups of approaches for system optimisation. [Kück et al. \(2016\)](#) consider symbiotic simulation promising for flexible manufacturing systems, which are able to deal with changes in demands and product customisations.

[Zhang et al. \(2022\)](#) structured a digital twin to schedule a shop floor. The authors extend the multi-fidelity simulation optimisation framework proposed in [Xu et al. \(2014\)](#) to work with heuristic algorithms. The algorithm developed in [Xu et al. \(2014\)](#) is a multi-fidelity ordinal transformation optimal sampling framework which involves a low-fidelity model to guide the search using a high-fidelity model to save the computing budget for simulation optimisation. [Goodwin et al. \(2022\)](#) propose the sequential allocation using machine-learning predictions as lightweight estimates (SAMPLE) framework, which uses multiple simulation-based machine-learning models to support real-time simulation optimisation for the digital twin.

## 2.3 Discrete Optimisation via Simulation

Simulation optimisation or optimisation via simulation (OvS) is the technique to minimise or maximise the expected value of objective(s) for a system. These systems usually have stochastic processes, for which objective functions are either expensive or difficult to build.

These simulation optimisation methods use the simulation as a proxy for the real system and aim to maximize/minimize a given objective function. As the simulation models we work with (typically Discrete Event Simulation (DES) models) have stochastic or random outputs, the objective is typically the expected value of the model output of interest. Using mathematical notation, we wish to minimise an output  $f(\mathbf{x})$ , where  $\mathbf{x}$  is a vector of decision variables and  $f(\mathbf{x})$  is the expected value of the random output  $Y(\mathbf{x})$ ,

$$f(\mathbf{x}) = \mathbb{E}[Y(\mathbf{x})]. \quad (2.1)$$

Most researchers assume the problems have a single objective and consequently the output  $f(\mathbf{x})$  is a single value. In this project, we focus on the problems where the feasible decision region  $x$  has a limited number of discrete solutions. Problems of this nature are described as ranking and selection (R&S) problems (Fu, 2014). R&S procedures aim to select the best from the alternatives. Two main groups of R&S methods are indifference-zone procedures and efficient simulation budget allocations (Fu, 2014). Branke et al. (2007) discussed the R&S methods, indifference-zone procedures guarantee frequentist correct selection and efficient simulation budget allocation procedures quantify posterior evidence for correct selection.

### 2.3.1 Indifference-Zone Procedure

A practical approach to measuring confidence in the solution is to analyse the probability of correct selection (PCS). For the indifference-zone procedures, the selected best solution  $k$  is at least  $\delta$  (the minimal distance, e.g. euclidean distance) away from others, or they are regarded as having no difference from each other. This ensures  $\text{PCS} \geq \alpha$ . The minimal distance  $\delta$  is the smallest difference the decision maker cares about and the  $\alpha$  is the preset confidence level for the results (Law, 2015). Bechhofer (1954) formulated the problem as  $\Pr\{\text{select } k \mid \mu_k - \mu_{k-1} \geq \delta\} \geq 1 - \alpha$  where  $\mu_k$  is the performance of solution  $k$  (Kim and Nelson, 2006a).

Bechhofer's Procedure (Bechhofer, 1954) is one of the earliest procedures and was developed in the 1950s for choosing the most suitable medical treatment from a small number of alternatives with relatively equal variances. It does not make any



computation from the samples until the end of the procedure. Under the same assumptions as the previous method, the sequential approach Paulson's Procedure (Paulson, 1967) was developed. It evaluates the selection once a new round of replications is sampled. Solutions that are not competitive are removed for further exploration, which makes the search more efficient. KN Procedure (Kim and Nelson, 2001) follows the concept of Paulson's Procedure while reducing the limitation of their assumption so that alternative solutions could have unknown and unequal variances. In addition, it supports the application of common random numbers, a variance reduction technique, which can help to reduce the number of replications needed to obtain a statistically significant result. KN++ Procedure (Kim and Nelson, 2006b) is one of the major developments that extend the validity of the procedure to non-normal and dependent alternatives with faster convergence.

### 2.3.2 Efficient Simulation Budget Allocation Procedure

The efficient budget allocation methods are designed to intelligently allocate the sampling budget to alternatives while they could still assure a high probability of correctly selecting the optimal alternative. This would be achieved by assigning more sampling budgets to the alternatives with high estimated mean performance and high uncertainty. While those alternatives with low estimated mean performance or with a low degree of uncertainty would be sampled less often.

The algorithms of efficient simulation budget allocation methods draw on the concept from Bayesian statistics to describe the uncertainty of the estimated mean with a posterior. Replications of simulation are assumed independent and distributed normally. Optimal Computing Budget Allocation (OCBA) (Chen et al., 2000) algorithm is an important class in this category.

A certain amount of replications are simulated in one iteration and then the variance ( $\sigma^2$ ) of each alternative and their distances ( $d$ ) to the best-performing solution in that round are computed to derive the ratio of budget allocation for the next iteration.

$$\frac{N_{i,t+1}}{N_{j,t+1}} = \left( \frac{\hat{\sigma}_i d_{b,j}}{\hat{\sigma}_j d_{b,i}} \right)^2, \quad i \neq j \neq b \quad (2.2)$$

$$N_{b,t+1} = \hat{\sigma}_b \sqrt{\sum_{i:i \neq b} \frac{N_{i,t+1}^2}{\hat{\sigma}_i^2}} \quad (2.3)$$

where  $i$  and  $j$  are two different solutions other than the best one ( $b$ ) in the previous iteration,  $N$  is the budget allocated to one solution for the next iteration and  $\hat{\sigma}$  is the sample standard deviation (Fu, 2014). The procedure stops when the budgets are exhausted or the preset PCS is satisfied.

Minimising expected opportunity cost (EOC) could be applied in place of maximising PCS to measure the selection. EOC is the expected reward with additional information subtract the reward without that information. Estimated EOC (EEOC) is defined as the upper bound of EOC and is used in algorithms,  $EEOC(i) = \sum_{j:j \neq k} \int_0^{+\infty} x f_{k,j,i}^*(x) dx$  where  $f_{k,j,i}^*(x)$  is the probability density function of the distance ( $x$ ) between solution  $k$  and  $j$  when solution  $i$  receives allocation (Fu, 2014). EEOC is calculated for every alternative and the alternatives having the smallest EEOC get allocation for the next iteration. It is worth noting that EOC is much more complicated to calculate compared to PCS. In addition, EOC penalises bad choices more than slightly incorrect selections, which makes it preferred over PCS when the performance is measured for economic value.

Derivatives of OCBA have been developed to work with multi-objective problems (Lee et al., 2010b). These methods aim to give out an optimal subset (Zhang et al., 2016) and have been used to assist heuristic optimisations (e.g. Particle Swarm optimisation (Zhang et al., 2017)).

The Expected Value of Information (EVI) method has its root in Bayesian decision theory (Fu, 2014). Both PCS and EOC mentioned in the previous paragraphs are possible loss functions for EVI. The algorithm allocates additional samples with the objective of minimising the expected loss after sampling. Linear Loss procedure (LL) (Chick and Inoue, 2001) is a typical EVI procedure minimising EOC. It has an important extension LL1 procedure (Chick et al., 2010) that simplifies the process of allocating all budgets to a single solution and maximises EVI in one iteration.

There is another group of approaches called single-run simulation optimisation, where all the solutions are concurrently simulated in one simulation run. One of the latest research is Time Dilation - Optimal Computing Budget Allocation (TD-OCBA) (Zhu et al., 2019). Time Dilation (TD) manages the time scale of the solutions in one simulation run following the allocation recommendation obtained by OCBA (Hyden and Schruben, 2000). TD-OCBA is more robust in terms of the parameter setting compared to the original OCBA when the performance is claimed to remain similar. The performance (PCS) in TD-OCBA increases quicker than applying TD alone when having a low computing budget. Apart from the comments on the accuracy, TD-OCBA needs only one warm-up period while conventional multiple-replication-based simulation optimisation approaches require one warm-up period for each experiment. For a system that has a long warm-up period, TD-OCBA could be an option that saves computing budget.

### 2.3.3 Fast Simulation Optimisation for Real-time Decision Making

It is obvious that the ranking and selection procedures reviewed in this section rely heavily on having enough computation to generate simulation replications. This leads

to the fact that the time needed to derive the optimal solution can be significant. They then become impractical when we are looking to obtain the optimal solution in a real-time or near-real-time setting. More powerful computing ability is an instinctive solution that may decrease the processing time. Parallel computing simultaneously runs replications on multiple processors to reduce computing time. Simulation with parallel computing often relies heavily on expensive hardware infrastructure. Parallel and distributed simulation for discrete event simulation were surveyed in [D'Angelo and Marzolla \(2014\)](#). The authors argued more work is necessary to enable available hardware architectures to work well with the current algorithm to offer usable and well-performed service.

Parallel OCBA ([Luo et al., 2000](#)) is one of the earliest works of parallel ranking and selection and it enables an existing method to work in a distributed computing environment with multiple processors. Another research parallelise OCBA could be found in [Yoo et al. \(2009\)](#). Good Selection Procedure ([Ni et al., 2017](#)) guarantees the probability of good selection which relaxes PCS and follows a standard output assumption. The approach extends the Divide-and-Conquer procedure ([Chen, 2005](#)) which divides the whole decision space into several parts and transfers the problem into multiple R&S problems, the obtained in-group optimums then form a final R&S problem.

[Pei et al. \(2018\)](#) proposed an expected false elimination rate, which is a novel objective for parallel ranking and selection. The authors claim this new guarantee is able to deal with a very large number of alternatives. Based on this, Parallel Survivor Selection (PSS) ([Pei et al., 2018](#)) and Parallel Adaptive Survivor Selection (PASS) ([Pei et al., 2018, 2022](#)) are developed.

Another growing area of research aims to reduce the number of high-fidelity replications for optimisation. Multi-fidelity framework (e.g. [Xu et al. \(2014\)](#); [Huang et al. \(2006a\)](#); [Pedrielli et al. \(2019\)](#)) is a group of methodologies that are built towards decision-making for manufacturing practice. Multi-fidelity framework is discussed in the following section.

## 2.4 Multi-Fidelity Simulation Optimisation

The concept of multi-fidelity comes from reducing the computing cost without much compromise on the accuracy of the result. A model with lower fidelity is involved in working with the high-fidelity model in order to reduce the utilisation of the high-fidelity model. While the computing budget is aimed to be minimised, the quality of the optimisation remains significant. Thus, balancing the cost of computing and the optimisation quality is crucial. The multi-fidelity framework has its origin in computer science and is then adopted and widely applied in engineering. The

multi-fidelity frameworks or algorithms discussed in this chapter focus on those in the field of simulation optimisation.

### 2.4.1 MO<sup>2</sup>TOS

Xu et al. (2014) proposed Multi-fidelity Optimisation with Ordinal Transformation & Optimal Sampling (MO<sup>2</sup>TOS) to leverage a low-fidelity model for narrowing down the design space. All the alternative designs are first tested on a low-fidelity model and then grouped based on the rankings of the estimates. The number of groups needs to be decided beforehand. The optimal sampling part of MO<sup>2</sup>TOS borrows ideas from OCBA to assign computational budgets to the groups obtained from the low-fidelity model. For example, initial replications for different groups are sampled and simulated via the high-fidelity model, which generates the initial means and variances for each group.

Unlike the greedy sampling strategy which spends all computing budget on the high-rank group from the low-fidelity simulation results, the optimal sampling method samples from all groups in case the low-fidelity model has a large bias. The procedure keeps running until the computing budgets are exhausted. The sampled design with the best average performance is selected as the optimal solution.

The framework builds on the following assumptions. The high-fidelity model ( $f(x)$ ) is a realisation of a random variable with finite variance. The low-fidelity model ( $g(x)$ ) can be approximated by a uniformly distributed random variable when randomly sampling  $x$  from the solution space, and  $g(x)$  is independent of the error  $\delta(x) = f(x) - g(x)$  for each solution. It should be pointed out that the optimisation output could only guarantee a comparably good solution since the random sampling process within each group in the optimal sampling stage only samples and simulates part of the alternatives.

In Xu et al. (2016b), MO<sup>2</sup>TOS partners with three low-fidelity models with different accuracy to check their effectiveness. The results show the surrogates have similar performance in their tests. The same framework (MO<sup>2</sup>TOS) in Xu et al. (2016a) includes a detailed application of assisting decision-making for operating a semiconductor fabrication plant. In addition, the paper combined archived high-fidelity simulation replications with low-fidelity simulation results to build the low-fidelity approximation. The comparison shows the performance of MO<sup>2</sup>TOS increases significantly compared to the test in Xu et al. (2014) when having a larger computing budget.

Song et al. (2019) introduced the Expected Optimality Gap (EOG) to measure the performance of the optimal sampling. The MO<sup>2</sup>TOS framework is referred to as the Multi-Fidelity Optimal Sampling (MFOS) in this paper. When there are more than 3

groups, MFOS has a larger upper bound of EOG than equal allocation policy and OCBA. The variance of members in the group reduces with more groups, at the same time, the sampling budgets allocated to each group decline as well. It would thus have a lower chance of correctly identifying the best alternative among the groups. The suggested proper number of groups is between 5 and 20, in which range MFOS performs comparatively stable.

The performance of the MO<sup>2</sup>TOS has been tested on a semiconductor manufacturing product line with 2 products, 5 workstations and a total of 37 machines (Xu et al., 2014). Optimal sampling with ordinal transformation outperforms the OCBA and equal allocation policy. Although the semiconductor production line in the paper is a system with a reduced scale, it shares many common challenges and features with large-scale systems, for example, the re-entry mechanism, priority schedule and batch processing. More details and analysis of a similar scenario could be found in Song et al. (2019). A Jackson network is applied as the surrogate or low-fidelity model while discrete event simulation works as the high-fidelity model. MFOS saves two-thirds of high-fidelity replications to attain the same EOG as OCBA does. It has been reported that MFOS still achieves fairly good performance even when the correlation between low-fidelity and high-fidelity models is as low as 0.6. The number of groups is suggested to be set at around 10. The authors constrain the decision-making time so that only 50 out of 128 plans could be simulated via the high-fidelity model for a robust test. A five per cent loss in profit is caused by the schedule worked out by MFOS while other competitors have at least a twelve per cent loss in profit.

Multi-Fidelity Budget Allocation (MFBA) (Peng et al., 2019) follows the concept of MO<sup>2</sup>TOS. Estimates from the low-fidelity model form the information to cluster the solutions in accordance with the Gaussian mixture model-based Bayesian framework instead of evenly separated in MO<sup>2</sup>TOS.

MO-MO<sup>2</sup>TOS (Li et al., 2016) extended MO<sup>2</sup>TOS to work with multi-objective problems.

### 2.4.2 MFSKO & VGO

MFSKO (Multi-Fidelity Sequential Kriging Optimisation) is an extension of a stochastic optimisation method, Sequential Kriging Optimisation (SKO) (Huang et al., 2006b), which looks for global optimum with cheaper cost of data (Huang et al., 2006a). Models at different levels are combined together to form a kriging metamodel (Rasmussen and Williams, 2005). The algorithm selects the fidelity level and data points for the simulation in the next step using augmented expected improvement. The algorithm holds the assumption that the solution space is connected and compact.

An application of decision-making in the metal-forming process has been demonstrated in the paper.

Value-based Global Optimisation (VGO) (Moore et al., 2014) has a similar structure to MFSKO, but it uses a utility function to extract information of value (IoV) rather than expected improvement to select the next point to refine the metamodel. The procedure stops when IoV turns from positive to negative. The authors give an example of designing a hydraulic hybrid vehicle.

### 2.4.3 Other Methods

Horng and Lin (2009) proposed a two-stage procedure to solve G/G/1/K polling system problems with  $k$  limited. The optimisation aims to select the optimal service limit ( $k$ ). In the first stage, an Artificial Neural Network (ANN) assisted Genetic Algorithm (GA) narrows down the candidate solutions by eliminating the low-rank solutions in the design space. ANN model which works as the fitness function of GA is a metamodel of the simulation. In the second stage, the algorithm iterates to continue reducing the promising solutions gradually. The surrogates used in stage two are refined with extra simulations that have various run lengths. The high-fidelity model is used lastly to evaluate the rest of the solutions. Horng and Lin (2013) applied an OCBA algorithm to select the best in the second stage after GA produced a good enough subset. Horng and Lin (2013) and Horng and Lin (2009) reduced the computing cost spent on the high-fidelity model in a way similar to MO<sup>2</sup>TOS. However, these two methods simulate all candidates in the subset obtained from GA. In this case, the surrogate would be expected to have limited bias. Among a list of tested surrogate models, ANN with the Levenberg-Marguardt algorithm has a better performance in a hotel booking limit problem.

Horng et al. (2013) proposed a memetic algorithm which adopts three phases with different fidelity. The structure looks like a mixture of the two methods mentioned above. In phase one, a GA optimisation is performed with an offline surrogate model to shrink the solution number. In phase two, local search works with the refined online surrogate to make the approximation for results from phase one and their neighbours. In phase three, OCBA is used with the original model for the final result.

March and Willcox (2012) presented an approach avoiding computing high-fidelity functions directly in the optimisation process. The errors of different fidelity functions are modelled together with low-fidelity functions to approximate the behaviour of the high-fidelity model. The result is proved convergent. The method has not yet been applied to discrete event simulation, thus to verify how it would work needs further research. The error between low and high-fidelity models is added to the low-fidelity model in Pellegrini et al. (2016) to estimate the simulation (high-fidelity model)

output. It is proposed to achieve accuracy close to the high-fidelity solver while maintaining low computational costs. Numerical tests reveal that there is at least a 20% reduction in the computational cost compared to building the high-fidelity-trained metamodel.

In Offline Simulation Online Application (OSOA) (Hong and Jiang, 2019), data generated by the simulation model are analysed to build a predictive model. Simple models or simple rules are normally used for the need for quick calculation. It is fast but may not always lead to the best solution. The covariates that are built via simulation offline are used to fit the closed-form predictive model. In the online application, evaluation via the predictive model will be carried out to support decision-making when the system observations are available.

Multi-fidelity model is applied to solve the aircraft recovery problem in Rhodes-Leader et al. (2018) to achieve real-time simulation optimisation where integer programming is used for the low-fidelity model. The stochastic process is eliminated for simplifying the model. The simulation model is adopted as the high-fidelity model. It searches the result area from integer programming for the optimal solution. The paper shows the proposed algorithm could effectively provide the solution to the aircraft recovery problem. The computational cost depends on the number of delay events to alter which affects the simulation runs.

Generalised Ordinal Learning Framework (GOLF) (Pedrielli et al., 2019) was proposed to support real-time decision-making for cyber-physical systems. The parameter of the system is optimised once the scenario and the system states are updated. The system responses vary for different scenarios and knowing all potential scenarios (e.g. the realisation of stochastic demands) is not possible. The proposed framework suggests sampling data from one or a small number of scenarios. Additional sampling could be carried out on the actual scenario in use for model refinement. The authors argue such expensive online sampling may not always be necessary to make a near-optimal decision. The whole design space is partitioned into several regions for efficient model building. An ordinal learning approach is applied to efficiently select the optimum and is a key part of the framework. The objective function for the current scenario consists of a weighted linear combination of a number of objective functions fitted from data of the history scenario and their distances to the current one. One of the constraints is that it is designed for single-objective optimisation. Considering the expensive cost of simulating complex systems, the authors argue that computer simulation could be extremely time-consuming even in an offline stage. They claim that the structure of GOLF is able to cover most challenges in real-time optimisation.

## 2.5 Simulation Metamodels

A metamodel is a model projecting the behaviour of another model that is usually complex in structure and slow in running (for example, a simulation model in our case) from input to output. There are not only conventional approaches like linear regression, polynomial regression or radial basis function regression but also a surge in machine learning methods like Gaussian process regression, tree-based algorithms and artificial neural networks. Each method is described in turn. Besides, the interpretability of the models and their suitability for adapting to streaming data (new data are continuously generated from the system) are also discussed.

### 2.5.1 Polynomial Regression

The linear regression model for simulation is formulated as

$$y(x) = \beta_0 + \beta_1 g_1(x_1, x_2, \dots, x_d) + \dots + \beta_p g_p(x_1, x_2, \dots, x_d) + \varepsilon \quad (2.4)$$

according to Barton (2020), where  $\varepsilon$  is independent normal random quantities. It is in the form of linearity between  $g_i(x)$  ( $i = 1, 2, \dots, p$ ) and  $y(x)$  while each  $g(x)$  could be either linear or non-linear. So that this generalised form of linear regression is able to fit non-linear functions. When  $g(x_1, x_2, \dots, x_d)$  is a linear function, the estimation of coefficient could be solved via ordinary least square  $[\hat{\beta}_0, \hat{\beta}_1 \dots \hat{\beta}_p]^T = \hat{\beta} = (X^T X)^{-1} X^T y$ . The linear regression model offers a straightforward relationship between variables which implies an insight into the system. However, such a linear framework with polynomial elements has limited flexibility and may not be proper for complex systems.

### 2.5.2 Radial Basis Function Regression

Radial Basis Function (RBF) regression is one of the most commonly used basis function regressions. It is formulated as

$$y(x) = \sum_{i=1}^N \omega_i \varphi(\|x - x_i\|), \quad (2.5)$$

where  $\varphi(r)$  is the radial basis function and is weighted by the coefficient  $\omega_i$ . One popular choice based on Gaussian function is  $\varphi(r) = e^{-(er)^2}$ . According to Barton and Meckesheimer (2006), RBF regression has a simple form consisting of a series of radially symmetric functions with different centres and widths. It is capable of carrying out an approximation of non-linear processes. However, the model could easily get over-fitted. Data need to be carefully processed (e.g. rescaling) before being



sent to fit the model. These days, RBF is more often being used as a kernel in neural networks to form RBF neural networks (RBFNN) or sometimes called RBF networks. RBFNN has been used as a metamodel for a stochastic simulation shortly after it was introduced (Meghabghab and Nasr, 1999).

### 2.5.3 Gaussian Process Regression

Gaussian Process Regression (GPR) is constituted of a mean function and a covariance (also known as the kernel) function (Rasmussen and Williams, 2005). A GPR can be noted as

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')), \quad (2.6)$$

where function  $f(x)$  is distributed as Gaussian process with a mean function  $m(x)$  and a covariance function  $k(x, x')$ . It is able to flexibly work for different non-linear approximations with good accuracy. The training process treats the test data set as a prior for Bayesian inference. Any non-negative definite function could be used as a covariance function. It is not difficult to train the hyperparameter, which is a set of coefficients from the mean and covariance functions since it assumes the data is normally distributed. Despite its many advantages, GPR has a computational complexity of  $\mathcal{O}(n^3)$  and a memory complexity of  $\mathcal{O}(n^2)$ , which means the complexities increase quadratically and cubically with the input size.

Sparse Gaussian Process (SGP) (Quiñonero-Candela and Rasmussen, 2005) and a Mixture of Experts (ME) (Nguyen-Tuong et al., 2009) are two major groups to deal with the problem of complexity. SGP trains the model through inducing inputs which are selected from the training set (Quiñonero-Candela and Rasmussen, 2005). ME partitions the training set into several subsets which are trained to be local models. Local Gaussian Process (LGP) in Nguyen-Tuong et al. (2009) combines the idea of Locally Weighted Projection Regression (LWPR) and GPR. LWPR makes approximations through weighted averages over nearby local models which are trained via clustered data. The application in approximating inverse robot dynamics shows that LGP is capable of fast updates and could be applied to situations that look for real-time responses.

### 2.5.4 Random Forest

Random Forest (RF) (Ho, 1995) integrates the output of a number of random decision trees (Biau and Scornet, 2016). A decision tree (Quinlan, 1980) is a classification or regression tree where each internal node divides a feature, each branch shows the outcome of the division and end nodes give out results. The computing cost increases linearly with larger-scale data sets. The idea is simple but empirically effective.

However, we know little about the theory behind the method. Although the structure of the decision tree is simple, it is sensitive to small differences in data. Stable and Interpretable RULe Set (SIRUS) regression (Bénard et al., 2021) is developed to respond to the demand for stability. It is a random forest regression with restrictions on node splits. The restrictions have a limited impact on prediction accuracy while increasing stability. Each decision tree in the model is then broken down into rules. Redundant rules above the given threshold are selected to form an aggregated rule. The authors claim that there are three minimum requirements for the interpretability of random forests. They are simplicity, stability and predictivity.

The random forest has a lot of extensions to expand the usage scenarios, such as adaptive random forest classifier (Gomes et al., 2017) for streaming data. It uses drift monitoring for every decision tree to track warnings and drifts. A new tree will be trained in the background when a warning is shown and replace the active one when the warning turns into drift (Zhukov et al., 2017). The drift refers to the appearance of new classes and the warning refers to a buffer before drift is triggered. Gomes et al. (2018) proposed a regression algorithm to further implement the idea. It was applied to three real-world datasets against several other tree-based ensemble learning algorithms and has the lowest error rate among them.

RF model could be found being used as the metamodel in the literature. For example, Shen et al. (2022) applied RF metamodel for optimising the design of wind-turbine foundation, Bjerre et al. (2022) use the RF model as the metamodel for predicting drainage fraction.

### 2.5.5 Artificial Neural Network models

Artificial Neural Network (ANN) is a modelling approach that became widely known many years after the concept was developed in the 1940s (McCulloch and Pitts, 1943). An ANN consists of a number of connected neurons (Ojha et al., 2017). An artificial neuron that receives inputs and sums them to produce an activation is a basic unit of the neural network. The sum is operated through a non-linear function which is known as an activation function or transfer function. Different activation functions and weights of neurons together with the way they are linked enable the neural network to implement desired behaviour. Thus several non-linear stages may be required for the realisation of certain functions where each stage is generally understood as one layer. It will be called a deep neural network when there are multiple hidden layers. Neural networks are suitable to deal with very large data sets or we can say that more data points normally bring higher accuracy. It has successful applications in numerous real-world problems. Nevertheless, it is a black-box system with little interpretability. Feedforward Neural Network (FNN) is a kind of ANN that could be used to perceive a model or a function, in which connections do not form a

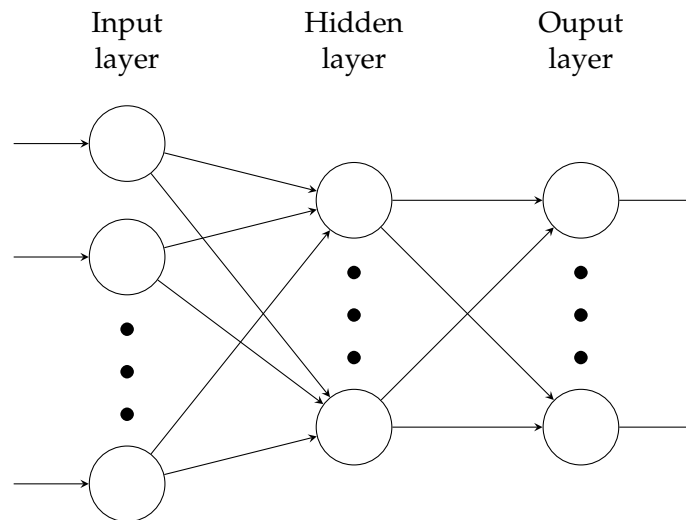


FIGURE 2.1: Feedforward Neural Network

loop (Ojha et al., 2017). The structure of FNN is shown in Figure 2.1. Barton and Meckesheimer (2006) stated that a neural network could also be thought of as a technique for obtaining coefficients of the metamodel.

According to the review in Negahban and Smith (2014), ANN models are often used as metamodels in literature. The application could be found as early as 2003 in Fonseca et al. (2003), where an ANN metamodel was used to estimate the lead time in a job shop.

### 2.5.5.1 Deep Neural Network

A Deep Neural Network (DNN) is an ANN that has multiple hidden layers between input and output layers. More hidden layers provide the neural network with a greater capability to extract information from the data while too many hidden layers could cause overfitting and reduce the generalisation of the model.

Other than the basic feedforward structure, there exist different forms of networks. Recurrent Neural Network (RNN) that contains a loopback is proven to be good at dealing with time-series data such as modelling language (Abiodun et al., 2018). Long Short Term-Memory (LSTM) was developed to extend RNN to handle large historical memory better, which has been successfully widely used in stock market forecasting (Hochreiter and Schmidhuber, 1997). Convolutional Neural Network (CNN) with convolution and pooling layers is another important variant of DNN that could process visual analysis (Abiodun et al., 2018). DNN is also a major research target for transfer learning. Transfer learning reuses the information from previous tasks and transfers it to a new task. It may enable the metamodel to work with the scenarios that are not covered by the metamodel but have appeared in the simulation model.

Selecting DNN for metamodel places a potential to leverage rich relevant research in machine learning.

### 2.5.6 Explainable Boosting Machine

Some algorithms mentioned above have extensions to cope with the problem of interpretability or explainability. Some other methods (e.g. Local Interpretable Model-agnostic Explanations (LIME), Partially Dependence) are trying to offer an explanation for black-box models externally. At the same time, it is also concerned from the very start when some new machine learning algorithms are developed. Among which the ones that received great attention is the Explainable Boosting Machine (EBM) (Nori et al., 2019). It builds on the generalised additive model with interaction terms.

### 2.5.7 Comparison of modelling methods

Metamodel is a model of a model. The modelling technique used for metamodelling is not different to modelling from the data. Regression analysis is a set of major approaches in statistical modelling. Polynomial regression, radial basis function regression, gaussian process regression, random forest and neural network are reviewed in this chapter. Although they all have different advantages and disadvantages, some papers carried out comparisons of the methods. We will have a look at some of them in the following paragraphs. They may shed some light on model selection.

Hultquist et al. (2014) analysed burn severity in the diseased forest via three machine learning methods (Random Forest (RF), Support Vector Machine (SVM), Gaussian Process Regression (GPR)) and multiple linear regression. All four types of models are sensitive to the number of independent variables when it is under a certain level. The prediction error slumps with the addition of variables before it reaches five. Different variable combinations bring higher variance to RF & SVM than the other two methods. All three machine learning algorithms outperform the multiple linear regression. Two kernel-based approaches SVM and GPR have similar behaviour when RF has better performance in the assessment.

Elsayed and Lacor (2014) compared the performance of the polynomial regression models, Kriging, RBF and RBFNN for printing process problems. The predictions from Kriging, RBF and RBFNN are close. The accuracies of these three approaches are superior to polynomial regression, and they do not have a significant difference between each other according to the author.

## 2.6 Experimental Design

Design of experiments is a group of methodologies that compose the plan for experiments to unveil the relation between the factors (variables) of interest. The data collected from the computer simulations which are conducted following the experimental design would be used to build the metamodel of the simulation in our case.

The experimental design aims to efficiently reflect the variation of the design space and describe the information in it (Durakovic, 2017). Full Factorial Design  $2^k$  and Fractional Factorial Design  $2^{k-p}$  can work with multiple factors with two levels (low and high) where  $k$  is the number of factors and  $p$  stands for the fraction of involved full factorial (Law, 2015). The main effect of factors and the interaction of multiple factors are considered in these methods. Confounding of the main factors and interactions are assumed in the Fractional Factorial Design, and the experiments are reduced to  $1/2^p$  of all  $2^k$  combinations. Factorial design is hard to deal with variables with more than two levels, other approaches are needed if we are looking for information in this situation. Central Composite Designs (CCD) extends the factorial design for full second-order models (Sanchez et al., 2021). It collects nine points for two variables with two levels. Four points of which are via a  $2^2$  factorial design, one from the centre of the design space, and another four are on the axis. The design has eight points lying on a circle and one at its centre for two factors (Law, 2015).

We are going to review the space-filling sampling methods and the adaptive samplings. The space-filling methods are initially designed for deterministic computer models where the factors need to be either continuous or discrete with a large number of levels. Adaptive samplings are the sampling methods that are carried out iteratively.

### 2.6.1 Space-filling Design

A space-filling design has sample points filling the design region with the smallest gaps. It is intuitive to place the samples covering the design space so that the samples can represent different areas of the design space. For  $p$  factors having feasible area  $\mathcal{X}$ ,  $\mathcal{X} = [0, 1]^p$  after normalisation (also called min-max scaling). Experimental design  $D = \{s_1, \dots, s_n\}$  is a collection of design sample points  $s_i \in \mathcal{X}$ .

The distance between one point in the design space and the experimental design is measured by its distance to the closest design point,  $d(s, D) = \min_{s_i \in D} d(s, s_i)$ , where  $s$  stands for any point in the feasible area and  $s_i$  is a sample in the design. Euclidean distance is a common choice for  $d(\cdot)$ . A smaller  $d(s, D)$  is desired as it implies the design is close to the points in the design space therefore the design has good

coverage over the design space. The data point that has the largest  $d(s, D)$  is the least ideal location in the whole feasible area. The design quality could be controlled when the least ideal point ( $\max_{s \in \mathcal{X}} d(s, D)$ ) is constrained.

$$\min_D \max_{s \in \mathcal{X}} d(s, D) \quad (2.7)$$

The above equation elaborates the concept of the Minimax distance design (Johnson et al., 1990).

A different approach sharing the same root was proposed to solve the design problem. Firstly, find the pair of design points that has the smallest distance,  $\min_{i,j} d(s_i, s_j)$ , where  $s_i, s_j \in D$ . By maximising this smallest distance, we can also obtain an experimental design that fills the design space, this is Maximin distance design

$$\max_D \min_{i,j} d(s_i, s_j). \quad (2.8)$$

The Maximin distance design (Johnson et al., 1990) costs less computation than the Minimax distance design because it calculates the pair-wise distance of the points in the design rather than all the locations in the feasible area. Although the Minimax and the Maximin distance design are straightforward to understand and easy to conduct they have no guarantee on the coverage or sparsity of each individual factor or low dimensional projection (Joseph, 2016).

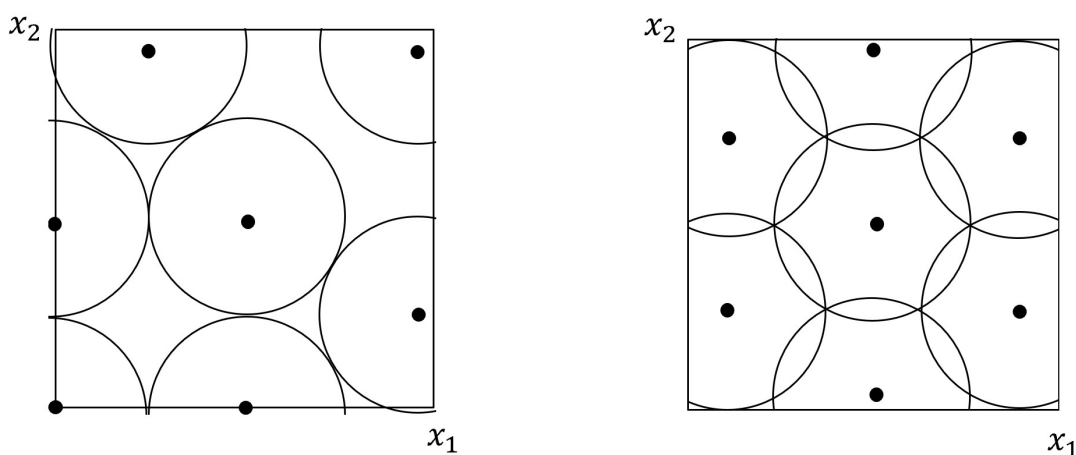


FIGURE 2.2: Illustrations of the Maximin Distance Design (left) and the Minimax Distance Design (right)

Figure 2.2 illustrates the Maximin distance design and the Minimax distance design in the same situation using Euclidean distance. The design space has two factors  $x_1$  and  $x_2$ . Two experimental designs with seven points are shown in the illustration. The

factor  $x_1$  has four samples in the Maximin distance design and three samples in the Minimax distance design when  $x_2$  has four and five samples respectively. Although it could tell from the chart that the designs are space-filling, the projections on both  $x_1$  and  $x_2$  are not ideal. In addition, [Pronzato \(2017\)](#) points out the Maximin distance design pushes the design points closer to the boundary of the feasible space.

[McKay et al. \(1979\)](#) introduced Latin Hypercube Sampling (LHS) (also known as Latin Hypercube Design (LHD)). The concept is expanded from the Latin square which is a  $n$ -by- $n$  array with  $n$  symbols and the elements in each row and column are not repeated. In the illustration given in Figure 2.3, we can see there is only one sample in each row or column. However this is not the only possible design following the LHS, it could also be a collection of points sitting on the diagonal which satisfies the criteria but fails to fill the space.

In order to select the optimal design among all candidate designs, variants of the LHS are developed. [Morris and Mitchell \(1995\)](#) proposed the Maximin Latin Hypercube Sampling (MmLHS). It maintains good projection on every dimension ([Joseph, 2016](#)) and non-collapsing ([Liu et al., 2018](#)) which means even when some points are not working effectively, the rest part of the design still makes up an LHS design.

The distinct pair-wise distances  $d_k$  (previously marked as  $d(x_i, x_j)$ ) are sorted into a list  $\{d_1, \dots, d_m\}$  in ascending order, the number ( $J_k$ ) of pairs of points with distance  $d_k$  makes up an index list  $\{J_1, \dots, J_m\}$ . The design is called Maximin while the  $d_k$  is maximised and  $J_k$  is minimised ([Morris and Mitchell, 1995](#)). The authors also proposed a scalar-valued design criterion to rank the designs. The design that has the lowest  $\phi_p(D)$  is the optimal design that meets the Maximin criterion,

$$\phi_p(D) = \left( \sum_{k=1}^m \frac{J_k}{d_k^p} \right)^{1/p}, \quad (2.9)$$

where  $p$  is a positive integer. The choice of  $p$  depends on the size of the design. The design has higher dimensions and more points is a larger design. The parameter  $p$  is suggested to pick from 5 for a small design to a value between 20 and 50 for a large design by [Morris and Mitchell \(1995\)](#).

In Figure 2.3, we demonstrate a Maximin Latin Hypercube Sampling design on two factors with seven points. The design fills both the space and the sub-spaces. Besides the MmLHS, other variants use various criteria (e.g. entropy criterion ([Park, 1994](#)), centred L2-discrepancy criterion ([Fang et al., 2002](#))) to select the optimal experimental design.

The experimental design methods discussed so far generate the design in one run, for which the number of points in the design should be decided in advance. Sequential

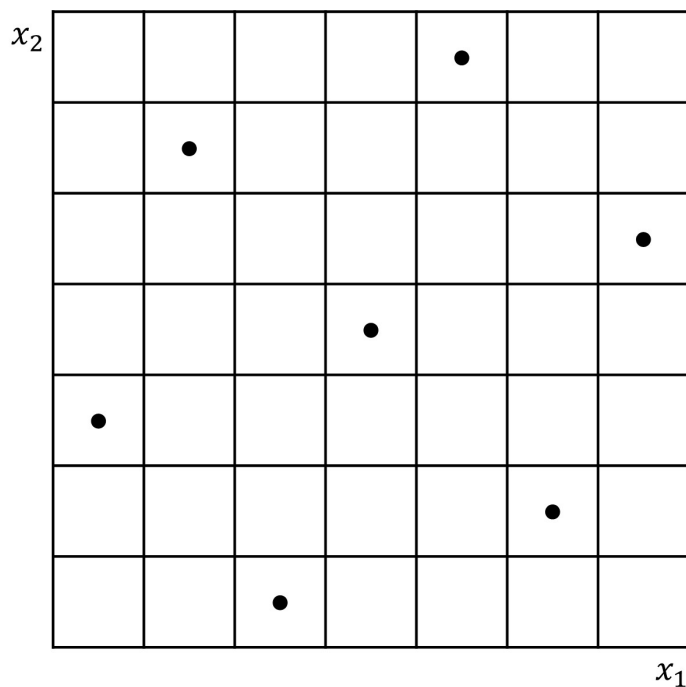


FIGURE 2.3: An illustration of the Maximin Latin Hypercube Sampling

sampling methods improve the efficiency of utilising computing budget by adding new design points iteratively.

### 2.6.2 Adaptive Sequential Sampling

Adaptive sequential sampling begins with a small number of samples and adds more data points iteratively until the budget (which could either be the number of observations or the sampling time) is exhausted or the performance of the model/surrogate reaches the preset measure of accuracy. Adaptive sequential sampling maximises the effectiveness of the limited sampling budget. It reduces the computational resource and time required for data collection when the sampling process satisfies early stop criteria.

Some one-shot experimental design methods could be transferred and adapted to fit into a sequential scheme. For example, methods like Incremental Latin Hypercube Sampling (Nuchitprasittichai and Cremaschi, 2013) and Sequential Latin Hypercube (Wang, 2003) are extended from the Latin Hypercube Sampling. van Dam et al. (2007) deal with the problem differently by treating the sampling as an optimisation problem and regarding space-filling as an objective.

Eason and Cremaschi (2014) proposed a sampling method for modelling deterministic chemical processes with neural networks. Their mixed adaptive sampling combines space-filling methods and adaptive approaches. It reduces the sample size required



for the CO<sub>2</sub> capture case study. Sequential sampling is also investigated in some research regarding stochastic kriging (Ankenman et al., 2010) where the adaptive sequential sampling is tailored for the kriging model. Binois et al. (2019) split the sampling process into exploring and exploiting to collect data for stochastic kriging. It balances different experiment areas by assigning more replications to the areas that do not have smooth response surfaces. Liu et al. (2018) argued that exploiting and exploring are two conflicting parts yet have equal importance. Exploiting finds the region that has been sampled and is still of interest. These regions could have large cross-validation errors, large prediction variance or other criteria that could represent the prediction error. Exploring finds the region that has not been sampled. Distance-based criteria could be used to identify the area.

## 2.7 Multi-Objective Simulation Optimisation

Real-life problems are normally complex and need to balance several different objectives, sometimes even contradictory ones. Therefore it is more practical to take it into consideration although it would increase the complexity of the problem and the single-objective optimisation problem has a lower cost. Hunter et al. (2019) surveyed nine industries that have multiple applications that have more than one objective.

### 2.7.1 Multiple Attribute Utility Theory

One way to address problems relevant to systems that have multiple performance measurements is multiple attribute utility theory (MAU) (Butler et al., 2001). It transfers multiple measurements into a single scalar in a way like calculating costs in the business. It could also be applied in place of cost when accurate cost data do not exist or are not suitable. The measurements could not only be scalars but also ranges or probabilistic distributions. One form of a single-attribute utility function that has been widely used is  $u_i(x_i) = A_i - B_i e^{x_i RT_i}$ , where  $u_i(x_i)$  represents the utility value of attribute  $x_i$ ,  $A_i$  and  $B_i$  are scaling constants of measurement  $i$ ,  $RT_i$  is risk tolerance to measure  $i$  defined by the user. A general expression of MAU is

$$\begin{aligned}
 u(X) = & \sum_{i=1}^n w_i u_i(X_i) + \sum_{i=1}^n \sum_{j>i} w_{ij} u_i(X_i) u_j(X_j) \\
 & + \sum_{i=1}^n \sum_{j>i} \sum_{m>j>i} w_{ijm} u_i(X_i) u_j(X_j) u_m(X_m) + \dots \\
 & + w_{12\dots n} u_1(X_1) u_2(X_2) \dots u_n(X_n),
 \end{aligned} \tag{2.10}$$

where different  $w$  show the weights of either single attributes or the interaction of multiple attributes which need to be assigned by the user. In Butler et al. (2001), MAU

was presented with an indifference zone approach to form an R&S procedure. The planning for a land seismic survey operation which contains four configurations was optimised using MAU and R&S in the paper.

### 2.7.2 Pareto Front

Another widely accepted concept is that there is no single best option exists for multi-objective optimisation but a set of non-dominated solutions. The result set is a Pareto set, Pareto front, Pareto frontier or non-dominated set (Lee et al., 2004). In order to obtain the non-dominated set, we begin by defining what it means for one solution ( $j$ ) to dominate another solution ( $i$ ). The solution  $j$  should at least have one objective that outperforms solution  $i$  while the other objectives are no worse than solution  $i$ . The probability of solution  $j$  dominates solution  $i$  in a minimisation problem is defined as  $P(\mu_j \prec \mu_i) = P(\mu_{jk} \leq \mu_{ik} \text{ for } k = 1, 2, \dots, m)$ , where  $\mu_{jk}$  is the evaluation of object  $k$  of solution  $j$ , in addition, there has to be at least one strict inequality. When the measurements (objectives) are assumed independent, they could then be converted into a product form for the overall probability of dominance between two solutions  $\prod_{k=1}^m P(\mu_{jk} \leq \mu_{ik})$ . Next, a performance index for each candidate solution dominated by other solutions is calculated by summing up all relevant pairwise probability of dominance,  $\Psi_i = \sum_{j=1, j \neq i}^n P(\mu_j < \mu_i)$ . Finally, those alternatives that rank high (having a small index  $\Psi$ ) constitute the Pareto set. The number of elements in the Pareto set could be controlled by a preset parameter (Lee et al., 2004).

Lee et al. (2004) proposed Multi-objective Optimal Computing Budget Allocation (MOCBA) that combines classic OCBA procedure with the Pareto front, which extends the capability of OCBA to multi-objective ranking & selection problems. After initial replications are performed for each candidate solution, a Pareto set is formed based on the simulation results. If the largest probability of dominance in the Pareto set is smaller than the preset threshold  $\Psi^*$ , the Pareto set is confirmed to be the optimal solution set. If the criteria are not satisfied, a certain amount of computing budget will be allocated. Candidate configurations that would maximise the change of probability of dominance for Pareto set members are selected. An OCBA-like process is then applied to the several best-performing policies from the last step. The Pareto set is then updated. Such a procedure iterates until the threshold is met or the computing budget runs out. The paper contains a comparison between MOCBA, theoretical optimal allocation (TOA) and theoretical uniform allocation (TUA) assuming the true mean and variance of designs are known. The total number of replications required to satisfy preset performance index  $\Psi^*$  for three approaches are compared. MOCBA requires 297 replications which is less than TOA (400 replications) and TUA (550 replications). It shows MOCBA could save computing budget when having a requirement on performance. MOCBA was compared with uniform computing

budget allocation (UCBA) in a more realistic scenario. MOCBA also outperformed UCBA by allocating nearly half the number of replications as UCBA does.

Lee et al. (2010a) modified the performance index  $\Psi_i$  in MOCBA to calculate the extent of non-dominance.

$$\Psi_i \equiv P\left(\bigcap_{j \in S, j \neq i} (j \hat{\succeq} i)\right) \quad (2.11)$$

MOCBA had a much higher probability of correct selection (PCS) than UCBA given the same computing budget in one of their tests. The highest PCS that MOCBA reaches in the test was very close to 100% while UCBA was only 75%. They borrowed the idea of Type I and Type II errors from statistics and added it into the restricted condition when minimising the simulation runs. For Type I errors, MOCBA and UCBA do not have much difference and are both at low levels even with low replications, while for Type II errors, MOCBA is significantly lower than UCBA. Correlated objections were taken into consideration in one of the tests. UCBA consumes around three times replications to attain the PCS that MOCBA gets with the same correlation condition (positive, independent or negative). For both methods, PCS increases in the order of positively correlated, independent and negatively correlated for the same computing budget. A flight schedule problem with three objectives and ten candidate policies displayed in Lee et al. (2010a) shows that MOCBA allocated roughly half the simulation computing budget (2880 replications) to obtain the same Pareto set as UCBA did (5649 replications).

Hunter et al. (2019) surveyed the multi-objective simulation optimisation methods, and two approaches other than MOCBA are described. They are Sampling Criteria for optimisation using Rate Estimators (SCORE) and Myopic Multi-Objective Budget Allocation (M-MOBA).

SCORE is designed for problems with two objectives (Feldman and Hunter, 2018). The probability of misclassification of non-dominated solutions converges to zero when the sampling budget is infinite. Misclassification of inclusive (Type-I error) and exclusive (Type-II error) are considered. Allocations are given based on the score of non-Pareto systems. A score is formed to measure the minimum distance between a non-Pareto system and phantom Pareto systems, which are artificially formed in the objective function space. The method has been generalised to cope with more than objectives in its extension MO-SCORE (Multi-Objective-SCORE) (Applegate et al., 2020).

M-MOBA (Branke and Zhang, 2015; Hunter et al., 2019) is a bi-objective optimisation concerning the expected value of information. The system that has the largest probability to change the Pareto set after the next allocation will be regarded as the best system and receive replications allocation. Its extension Myopic Multi-Objective Budget Allocation based on HyperVolume (M-MOBA-HV) (Branke et al., 2016)

measures the change of Pareto set over hypervolume, which demonstrates how close solutions are to the true Pareto set. Hypervolume was claimed to be more relevant to decision-makers than other measures by the authors.

Multi-Objective Multi-fidelity Optimisation with Ordinal Transformation Optimal Sampling (MO-MO<sup>2</sup>TOS) (Li et al., 2016) extends the ability of MO<sup>2</sup>TOS. Non-dominated sorting is performed first to obtain Pareto layers. Solutions are grouped by the performance index. Each group forms a layer (which is the shape on the plot for the cases having two objectives). The dominating layer ranks higher than the ones that are dominated. The layers are selected via an OCBA-like procedure for computing budget allocation. The crowding distance which shows the density around the solution in the Pareto layer is calculated, and the solution with the smaller value gets more attention. A lower value, which shows fewer similar alternatives around the solution, draws a higher computing budget. MO-MO<sup>2</sup>TOS performs well and better than random search after setting parameters carefully. It has been deployed to optimise the capacity plan of a large-scale container terminal (Li et al., 2017).

## 2.8 Conclusion

In this chapter, we have reviewed papers on seven key topics for the project. In accordance with the literature, some gaps in research are identified as we discuss below. These gaps were used, alongside the practical problem, to develop the research questions stated in Section 1.3.

Conventional simulation optimisation methods require multiple replications of the simulation for each alternative solution. It is not able to meet the need to obtain the optimisation result in real-time or in a short period of time, especially when there are a large number of alternatives and the simulation model is complex and time-consuming. Previous research shows that developing multi-fidelity models for simulation optimisation helps to reduce the computational burden but they are still not fast enough for optimising in real-time. This is a research gap that has been looked into in this project. For which, RQ (Research Question) 2 is raised.

Following MO<sup>2</sup>TOS framework Xu et al. (2014) which utilises OCBA to deal with evaluations from the metamodel, RQ1 is asked for further discussion on the gap of how the existing simulation optimisation procedures (e.g. OCBA) could be used to achieve real-time optimisation.

The low-fidelity models used in most research of optimisation via multi-fidelity models do not aim to offer accurate estimations. There is a research gap here to take the accuracy of the metamodel estimation into consideration (RQ4). The application of the multi-fidelity method in an industrial environment is also a research gap that is

explored in the project. Therefore, RQ3 and RQ5 (related to model building) are raised. The answers to these research questions are expected to contribute methodologically and practically.



## Chapter 3

# Multi-fidelity Simulation Optimisation

A multi-fidelity simulation optimisation algorithm is proposed in this section. It extends  $MO^2TOS$  to further improve the performance. Multi-fidelity simulation optimisation uses a low-fidelity metamodel to guide the optimisation using the high-fidelity model. There are three aspects that affect the quality of optimisation: choice of the metamodel, guidance provided to metamodel estimates for high-fidelity model and the utilisation of R&S procedure. A flowchart of the proposed algorithm is given in Figure 3.1.

We use the multi-fidelity framework with a discrete event simulation (DES) model as the high-fidelity model and a multi-layer feedforward neural network metamodel (Ojha et al., 2017) as the low-fidelity model. The left-hand side of the flowchart (Figure 3.1) describes the offline process used to build the metamodel (more details in Section 3.2), while the right-hand side describes the online process followed when carrying out the optimisation.

The optimisation procedure extends  $MO^2TOS$  (Xu et al., 2014) to have a finer operation on the output of the metamodel, which could be very useful when the metamodels have high accuracy. Unlike  $MO^2TOS$  which evenly separates the candidate solutions, we cluster and then group solutions based on the metamodel's estimates of their output values so that we could have more control over the number of solutions in each group. More details of the algorithm are given in Section 3.1. The metamodel and how we measure its performance can be found in Section 3.2. We explain the utilisation of the ranking and selection procedure in Section 3.3.

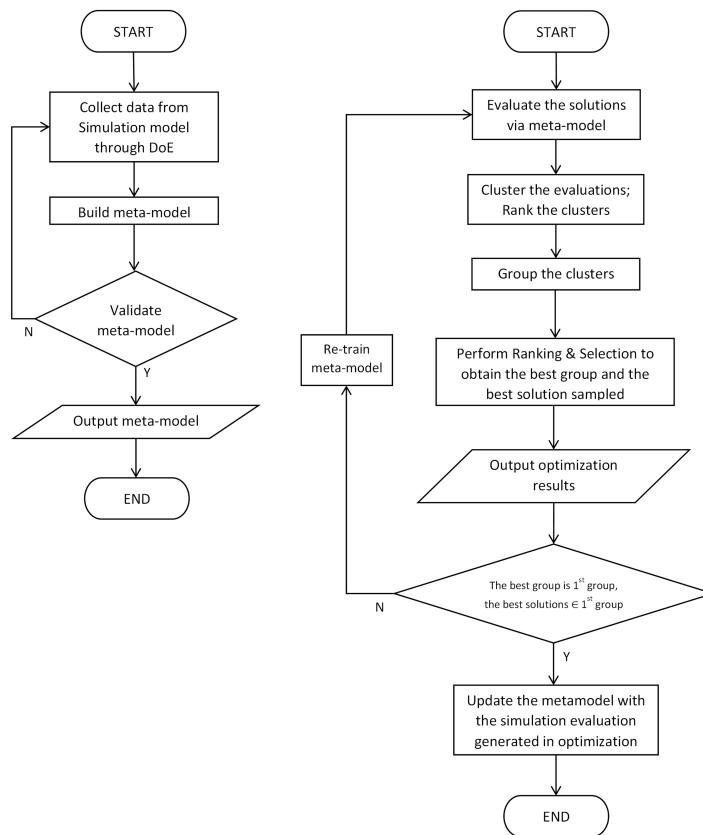


FIGURE 3.1: Flowchart of the proposed algorithm. Left: offline process, output fitted metamodel. Right: online optimisation process, output final solution.

### 3.1 Optimisation Algorithm

The right side of the flowchart (Figure 3.1) shows how our multi-fidelity optimisation algorithm works. First, we use the validated metamodel to evaluate the solutions. Next, we use the hierarchical agglomerative clustering method (Day and Edelsbrunner, 1984) to partition the solutions into clusters based on their performance. Then, the clusters are indexed in the order of their performance, where cluster 1 contains the most-promising solutions and cluster  $P$  contains the least-promising solutions ( $P$  is the predetermined number of clusters). We put clusters into groups for the next stage of the process as this gives us more control over the number of solutions in each group. The number of groups  $Q$  is predetermined. The grouping is designed so that the high-rank groups contain fewer but more promising clusters and low-rank groups contain more but less-promising clusters. A grouping strategy with 5 groups is introduced for this purpose.

Clusters are grouped into a handful of groups following a preset proportion of  $1 : 2^a : 3^a : 4^a : 5^a : \dots$ , where  $a$  is the aggression parameter. It controls the level of aggressiveness of the grouping strategies. A larger aggression parameter leads to more clusters and smaller average cluster size. The number of required clusters  $P$  is



calculated from the predetermined number of groups  $Q$  and the selected aggression parameter  $a$ ,

$$P = \sum_{q=1}^Q q^a. \quad (3.1)$$

The number of clusters comes from the summation of the grouping ratio rules where the aggression parameter is the exponent in the ratio. When  $a$  increases, the summation of the ratio sequence increases and more clusters are needed to hold these alternatives. Since the quantity of the alternatives is fixed, more clusters would lead to fewer solutions in each cluster on average. While the exact numbers of the in-cluster members depend on the data structure (whether the response surface is smooth or steep) and the clustering method. For example, in the case study in Section 3.4 we tested 3 aggression parameters with 5 groups, where  $a = 1, 2$  and  $3$  respectively. When  $a = 1$ , the grouping strategy is  $1 : 2 : 3 : 4 : 5$ , where there need to be 15 clusters in all. When  $a = 2$ , there will be 55 clusters for grouping strategy:  $1 : 2^2 : 3^2 : 4^2 : 5^2$ . When  $a = 3$ , grouping strategy:  $1 : 2^3 : 3^3 : 4^3 : 5^3$  needs 225 clusters in all (Equation 3.1).

In the final stage of the optimisation, we use an R&S procedure, such as OCBA, to sample from the groups and report the best solutions, where each group is treated as a single solution. When a simulation replication is assigned to one group by the R&S procedure, we randomly sample from the solutions in that group for simulation.

In our approach, a solution in a group with fewer clusters will have a higher chance of being sampled. In other words, we sample more solutions from the promising group. Our approach does not eliminate the chances of any solution being sampled.

Finally, we apply the following two questions to the best solutions obtained by the R&S procedure:

1. Is the best group selected from the R&S the same as the first group identified by the metamodel?
2. Do the best solutions belong to the first group?

If both questions get positive answers, we then run the high-fidelity model for the best solutions to get more accurate results. Otherwise, the metamodel needs to be refitted with additional simulation replications and a new experimental design to improve its accuracy. Then the multi-fidelity framework is repeated until the two questions return positive answers. This ensures the quality of the metamodel meets the requirements of the proposed framework.

## 3.2 Metamodel

The metamodel is a surrogate to the high-fidelity DES model, it no doubt plays a big role in any multi-fidelity method. In our framework, fitting the metamodel is carried out offline and the metamodel should be ready before running the online process. Most papers in the field focus on the simplicity of the low-fidelity model. For example, [Xu et al. \(2014\)](#) used M/M/C as a low-fidelity model to simplify a complex semiconductor manufacturing system. Even so, seeking efficiency is not necessary to compromise the estimation accuracy of the metamodel. The metamodel used in this algorithm is expected to be capable of non-linear regression with sufficient accuracy. The feedforward neural network is chosen as the metamodel in our multi-fidelity framework. There are several reasons for this choice, 1, it is good at regression; 2, its layer structure enables the opportunity to update the model by retraining some layers while fixing the rest; 3, fitting a feedforward neural network for regression is quick.

The left side of the flowchart in Figure 3.1 shows how we develop the metamodel. First, datasets for training and validating the metamodel are generated through the high-fidelity simulation model following experimental design. Next, a metamodel is trained with the selected modelling method. After that, the metamodel is validated by comparing its estimates with the validation dataset. If the metamodel is a poor fit, we then collect more simulated data and refit the metamodel.

In addition to the accuracy criteria (e.g. mean absolute error, mean square error) applied during the fitting, the ordinal association between the estimates by the simulation and the metamodel is also taken into consideration for validation. This is because we want to have an idea of how the metamodel would perform in the framework before running the online optimisation. The rank correlation of the models could be a useful indicator since the metamodel estimations would be used to rank the alternatives. Kendall's  $\tau$  coefficient ([Kendall, 1938](#)) was chosen to calculate this rank correlation. It explicitly indicates the quality of the ranking by the metamodel. Two sequences are highly positively correlated when  $\tau$  is close to 1, and not correlated when  $\tau$  is close to 0. The equation to calculate Kendall's  $\tau$  is given in Equation 3.2 ([Kendall, 1938](#)).

$$\tau = \frac{n_c - n_d}{\frac{n(n-1)}{2}} \quad (3.2)$$

where  $n_c$ ,  $n_d$  and  $n$  are the number of concordant pairs, discordant pairs and elements in the sequence respectively. Let  $(x_i, y_i)$  and  $(x_j, y_j)$  be two observations from joint random variables  $X$  and  $Y$ , where  $i < j$ . If  $x_i > x_j$  and  $y_i > y_j$  are both true or  $x_i < x_j$  and  $y_i < y_j$  are both true, this pair of observations are concordant ([Kendall, 1938](#)). If the above conditions are not met, they are discordant.

An extra set of data is sampled and evaluated through both the metamodel and the simulation for testing Kendall's  $\tau$  between the low-fidelity model and the high-fidelity model. A limited number of experiments are carried out to calculate Kendall's  $\tau$  since we want to keep the simulation running efficiently for preparing the test. The number of experiments should also take the shape of the feasible area (e.g. half of a square in the inventory system example) and the need for the experimental design into account. Our test data set for Kendall's  $\tau$  in Section 3.4.2 has a size larger than 1/1000 of the total number of alternatives.

If the simulation is costly in time, the test set for fitting the metamodel could be reused to compute the rank correlation. Kendall's  $\tau$  offers a different angle to look at the quality of metamodel. When the value is far away from 1, we need to check the metamodel and do more sampling or fitting the metamodel if necessary. The validated low-fidelity model is then used in the first step on the right side of the flowchart where the solutions are evaluated via metamodel).

### 3.3 Ranking & Selection

Ranking and selection procedures are originally designed to compare a handful of designs for a system and output the optimal selection. Multiple replications are normally required for each design to give out the correct solution in a stochastic system. These procedures become very slow and impractical when we have a large number of candidates.

In Xu et al. (2014), they proposed to sample from groups of solutions instead of running replications for each single solution for the OCBA procedure. We followed the idea and used it in our approach. The modified OCBA algorithm (Fu, 2014) with amended notation is given below:

---

**Algorithm 1** Optimal Computing Budget Allocation (OCBA) procedure for grouped solutions
 

---

```

1: procedure OCBA ( $k, \beta, \theta, n_0$ )
2:   INPUT
3:    $k$  – Groups of solutions
4:    $\beta$  – Total budget
5:    $\theta$  – Increment of budget in one iteration ( $\theta = 1$  in the experiments)
6:    $n_0$  – Initial samples number for each group
7:   INITIALISE
8:    $t \leftarrow 0$  ▷ iteration count
9:   Sample  $n_0$  solutions from all groups,  $n_{1,t} = n_{2,t} = \dots = n_{k,t} \leftarrow n_0$ 
10:  while  $\sum_{i=1}^k n_{i,t} < \beta$  do
11:    UPDATE
12:    Means of sampled solutions in each group,  $\bar{x}_i = \sum_{j=1}^{n_{i,t}} \frac{x_{ij}}{n_{i,t}}$ 
13:    Standard deviation of sampled solutions in each group,  $\hat{\sigma}_i = \sqrt{\sum \frac{(x_{ij} - \bar{x}_i)^2}{n_{i,t} - 1}}$ 
14:    Where  $x_{ij}$  is the  $j$ th solution sampled in the  $i$ th group,  $i = 1, 2, \dots, k$ ,  $j = 1, 2, \dots, n_{i,t}$ 
15:     $b = \arg \max_{i \in [1, k]} \bar{x}_i$  ▷ update the best group
16:    ALLOCATE
17:    Calculate
18:     $\frac{n_{(i),t+1}}{n_{(j),t+1}} = \left( \frac{\hat{\sigma}_{(i)} d_{b,j}}{\hat{\sigma}_{(j)} d_{b,i}} \right)^2$ ,  $(i) \neq (j) \neq b$ 
19:     $n_{b,t+1} = \hat{\sigma}_b \sqrt{\sum_{(i):(i) \neq b} \frac{n_{(i),t+1}^2}{\hat{\sigma}_{(i)}^2}}$ 
20:     $i^* = \arg \max_{(i) \in [1, k]} (n_{(i),t+1} - n_{(i),t})$  ▷ find the group for more simulation
21:    SIMULATE
22:    Sample  $\theta$  solutions from  $i^*$  group
23:     $n_{i^*,t+1} \leftarrow n_{i^*,t} + 1$ 
24:     $t += 1$ 
25:  end while
26:   $k^* = \arg \max_{i \in [1, k]} \bar{x}_i$  ▷ find the best group
27:   $j^* = \arg \max_{j \in [1, n_{i^*,t}]} \bar{x}_{i^*j}$  ▷ find the best solution
28: end procedure

```

---

Each group of solutions is regarded as equal to one alternative in the original OCBA procedure (Chen et al., 2000). Each time we need to perform additional replications, we sample from the group randomly to pick a solution for the simulation. The procedure iterates until the budget  $\beta$  runs out. Group mean and standard deviation are updated in every iteration. From this, the best group  $b$  in the iterations is obtained. Replications are allocated based on the standard deviation of the groups and the

distance between the best group  $b$  and other groups. The procedure produces the final best group  $k^*$  calculated in the last iteration and the best solutions  $j^*$ . There is a possibility that the best solution  $j^*$  is not in the best group  $k^*$ . The solutions in each group are not exhaustively sampled and the  $j^*$ s are calculated among the sampled solutions. The grouping strategy described in Section 3.1 is designed to ease this drawback because the most promising group has fewer solutions and each one will have a higher chance to be sampled.

### 3.4 Numerical Experiments

The proposed multi-fidelity simulation optimisation framework is tested on an inventory problem from Law (2015) as a proof of concept. We optimise the ordering policy so that the demands from the customer and the storage cost are balanced in the inventory problem. More detailed descriptions can be found in Section 3.4.1. After that, the metamodel for the simulation model is described in Section 3.4.2. The application of the proposed optimisation procedure is then illustrated in Section 3.4.3. Finally, we give the results for the experiments on the inventory system in Section 3.4.4.

To provide a baseline for comparison, we run five replications of the inventory system DES model at each feasible solution following the examples in Law (2015) and record the results. The simulation outputs of these five replications are averaged to form the ‘standard’ which is used as the true value of that alternative solution, from which, we obtain the solution with the lowest cost as the optimum. We formalise the hypotheses as the following. The null hypothesis  $H_0$  is that the optimisation result obtained by the proposed multi-fidelity simulation optimisation framework and the optimum are not different, the true mean difference between them is zero:  $\mu_d = 0$ . The alternative hypothesis  $H_1$  is that the optimisation result obtained by the proposed framework and the optimum are different, the true mean difference between them is not equal to zero:  $\mu_d \neq 0$ . We measure the quality of the solutions obtained from our method with different values of the aggression parameter and MO<sup>2</sup>TOS by comparing them against the ‘standard’. Each optimisation outputs the top five solutions with the lowest estimates for the objective function based on the final simulation optimisation results, which will be referred to as the ‘top-5’ in the following content. The performances of alternative solutions are compared with the optimal solution which is the best-performing solution according to the ‘standard’. We count how many of the results among the top-5 are not significantly different from the optimum. The 90% paired-t confidence intervals are calculated for the residual between the selected solution and the optimum using data collected for the ‘standard’. Two solutions are not statistically different when the confidence interval of the residual contains zero. The best case is that all top-5 solutions are not significantly different from the

optimum. A single run of the optimisation may not be able to provide a conclusive result because both the optimisation algorithm and the discrete-event simulation model involve multiple stochastic processes. Therefore, a large enough number of experiments were carried out for each optimisation algorithm. We run the optimisation one thousand times for each of the algorithms in comparison. The result could be shown in a frequency distribution plot like Figure 3.7, in the best case, the distribution would look extremely skewed to the left and having most experiments come with all top-5 that are not significantly different from the optimum.

### 3.4.1 Inventory System Example

The inventory system model could be used for tracking and predicting inventory levels, orders and their expected arrival time, and sales. In the manufacturing industry, such a model could help to schedule the work and make a production plan to ease the pressure of raw materials being overstocked or out of stock. We use an inventory system model from Law (2015) to demonstrate our algorithm. This is a well-known example that uses a DES model to describe the system behaviour and the effect of ordering policies on costs.

The simplified inventory system considers the situation of ordering and selling one product. It has four types of events: 1. the arrival of an order from the supplier, 2. demand for the product, 3. inventory evaluation and order placement following the ordering policy and 4. termination of the simulation. The demands are IID (Independent and Identically Distributed) random variables with preset probabilities. The intervals between two demands are IID exponential random variables. Inventory management includes some variable costs, for example, holding cost, order cost and shortage cost. Holding cost is proportional to the inventory level which includes but is not limited to the warehouse rent and overhead expenses. It is assumed that the holding cost would be zero when having no product in stock although it would still incur holding cost in the real world (e.g. rent). Making orders incurs extra expenditures like commission fees or setup costs, which are fixed for each order in addition to the product cost. Together they consist of the order cost. A penalty (shortage cost) is applied when the product is out of stock (in other words, the demands exceed the inventory level) to maintain backorders. The total cost is aimed to be minimised which sums the holding cost, the order cost and the shortage cost. These costs are directly affected by the ordering policy.

Ordering policies are described by two decision variables,  $(s, S)$ , where  $s$  is the reorder point of the inventory system and  $S$  is the maximum inventory. The ordering policy

suggests an order size of  $Z$ :

$$Z = \begin{cases} S - I & \text{if } I < s \\ 0 & \text{if } I \geq s \end{cases}$$

where  $I$  is the inventory level. The simulation model is carried out at the beginning of each month to generate a suggestion for the optimal ordering. We constrain the range of decision variables such that  $s \in [1, 150]$ ,  $S \in [2, 151]$  and  $s < S$  which results in a feasible area that includes 11,325 solutions regardless of the structure of the problem. The structure of the problem refers to some of the  $(s, S)$  ordering policy combinations that obviously would not work well, for example,  $[1, 2]$ ,  $[2, 3]$ .

### 3.4.2 Building the Metamodel

This section demonstrates the metamodel development following the description in Section 3.2 and the left part of the flowchart in Figure 3.1. The metamodel is fitted offline and consequently, there is less of a time constraint on this procedure but we still assume that the computational time should not be excessive. As a result, we should not run the simulation model at every single design point which is not possible when dealing with a complex system having a large feasible area. This procedure would become more time-critical if we would like to update the simulation model and the metamodel in real time.

The feedforward neural network metamodel was structured with Keras (Chollet, 2015) package in Python, where we use the mean absolute error (MAE) for the loss function, Adaptive Moment Estimation (ADAM) (Kingma and Ba, 2015) for the optimiser and Gaussian Error Linear Units (GELU) (Hendrycks and Gimpel, 2016) for activation functions. For simplicity, the neural network is designed to have the same number of neurons in its hidden layers. The numbers of neurons and hidden layers were paired and tested to find the combination that leads to the lowest prediction loss. After tuning, we result in a model with 2 hidden layers and there are 10 neurons in each of the hidden layers.

Data prepared for training and testing the metamodel are collected from sampling the simulation following the design of experiments. The experiments are designed with Latin Hypercube Sampling (LHS) (Morris and Mitchell, 1995), a space-filling method. Through LHS, the sampled data could evenly cover the feasible region. For stochastic systems, multiple replications are normally required to have a stable performance expectation. However, that reduces the efficiency of utilising the computing budget. We would like to know whether that is always worth the extra effort to fit a metamodel. We assume a fixed computational budget for the experimentation and consider two options for the sampling: 1. a single replication at each design point and

2. multiple replications at fewer design points. Specifically, we set the budget to 100 simulation replications and consider running the simulation with a single replication at each of 100 design points ( $100 \times 1$ ) and 5 replications at each of 20 design points ( $20 \times 5$ ). Each single design point is a distinctive combination of the reorder point  $s$  and the maximum inventory  $S$  in the feasible region. Both 100 design points and 20 design points are sampled via LHS. The quality of the metamodel is determined by comparing its estimations with the expected outputs from the simulation model using MAE where both designs have twenty per cent of the collected data left out for the test. The measurement for the multiple-replication strategy ( $20 \times 5$ ) strategy is 3.929, while the MAE for the single-replication strategy ( $100 \times 1$ ) is lower at 1.8. (Nine ordering policies tested in Law (2015) have their average total cost between 120 and 150.) The single-replication strategy outperforms the other strategy with multiple replications in this case. The nature of our inventory system model that its stochasticity is not at a high level may lead to the finding. It needs more work before we can generalise the suggestion to a wider range of systems. Thus the metamodel built with the single-replication strategy is used for the following experiments in the multi-fidelity simulation optimisation framework. For the rank correlation, Kendall's  $\tau$  of the metamodel based on 15 sampled points is 0.79. (When two sequences are identical, Kendall's  $\tau = 1$ ). It is reasonably good and indicates that the metamodel has the capability to rank candidate solutions properly in accordance with their estimations. So far, the metrics we use for measuring the quality of the metamodel show the built neural network metamodel satisfies the need to work as the low-fidelity model in the multi-fidelity simulation optimisation framework.

Next, we move one step further to understand how the metamodel works through the response surface. This is to complete the comparison although it is not necessary and may not be always possible in reality. A three-dimensional surface plot showing the fitted metamodel which reflects reorder level  $s$ , the maximum level of inventory  $S$  and the objective total cost is presented in Figure 3.2. It is clear that the estimated total costs are high when  $s$  and  $S$  are both at their extremes. There is an obvious funnel shape in the chart that shows the area may produce the desired low cost.

Figure 3.3 gives out the residual between the built metamodel and the simulation model (which is the 'standard' described in the preamble of Section 3.4). The plot for most areas is flat and close to zero residual. The only discrepancy between the two models is in the area where both  $s$  and  $S$  are close to zero, the residual drops rapidly to minus one hundred. We can see that the same area in the metamodel evaluation (Figure 3.2) also has a decline but is still larger than the bottom of the funnel shape. Similar to the previous paragraph, from the findings in these two charts, we can also draw the conclusion that the metamodel works well for most solutions in the feasible area.



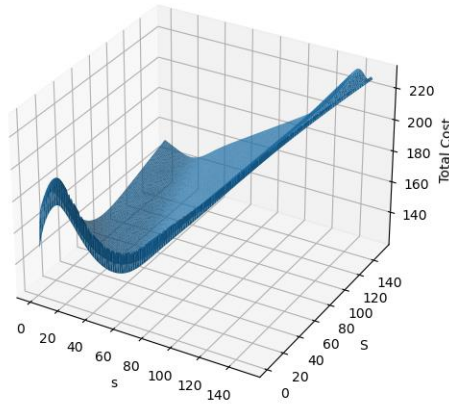


FIGURE 3.2: Evaluation of solutions from metamodel

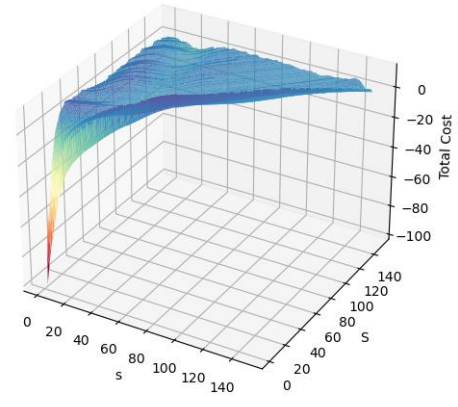


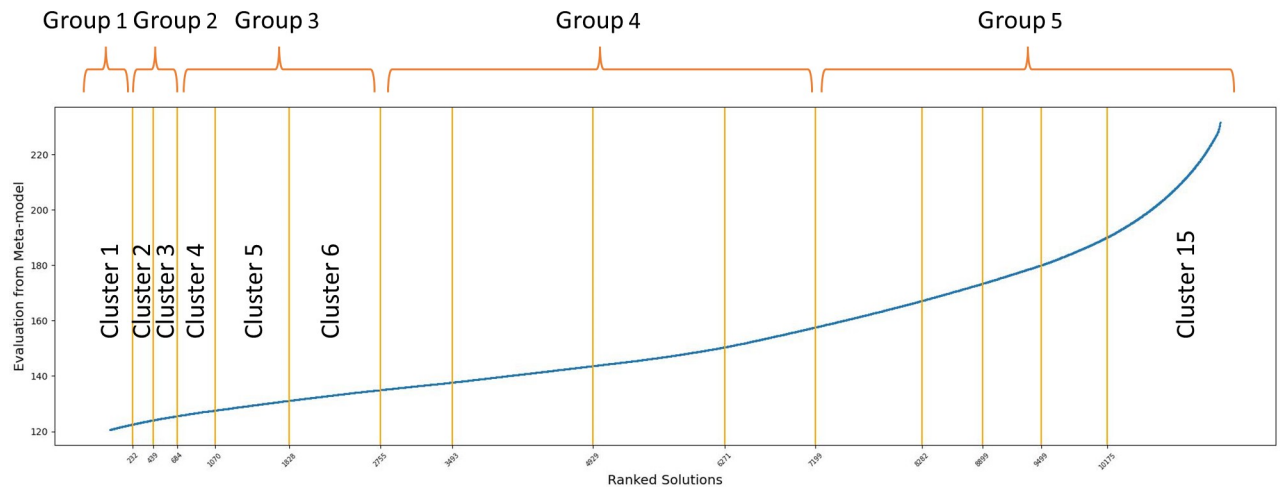
FIGURE 3.3: Residual between metamodel and simulation model

### 3.4.3 Optimisation

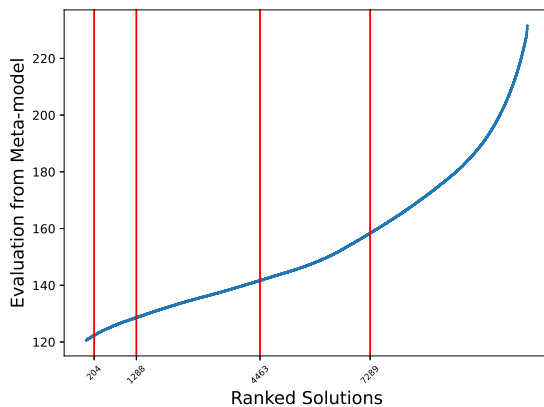
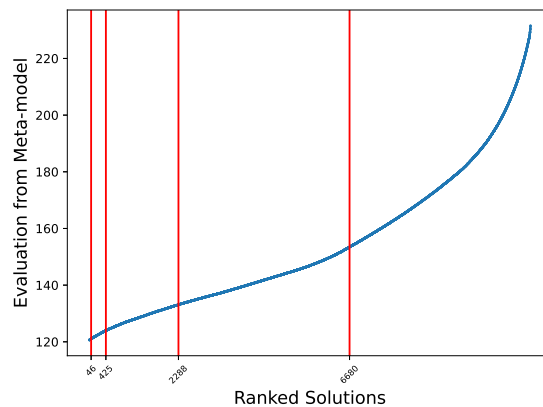
We are going to elaborate on the online process (right side of the flowchart in Figure 3.1) of the proposed multi-fidelity framework via the given inventory system experiment. Feasible solutions are first evaluated through the metamodel which is built in the previous section. We plot the total costs estimated by the metamodel for all feasible solutions  $(s, S)$  in Figures 3.4, 3.5 and 3.6 where they are separated by three different sets of aggression parameter  $a$  respectively. Solutions have been placed in a non-decreasing order for the metamodel estimation, as a result, the blue line is monotonic increasing.

We then apply hierarchical clustering to the estimates. Solutions are superimposed by orange vertical lines in Figure 3.4 when aggression parameter  $a = 1$ . Each separated segment stands for one cluster. The number of clusters needs to be calculated beforehand (Equation 3.1). They are dictated to be 15, 55 and 255 in order to align with the selected aggression parameters  $a = 1, 2$  and 3 respectively so that we could be ensured to meet our designed 5 groups at the end of the process. In Figure 3.4, the x-axis has the ranked candidate policies and the y-axis is the average total cost calculated from the metamodel. The solutions are ranked for display; it is not a necessary step in the optimisation since the hierarchical clustering contains ranking automatically. We can see from the chart that the curve has a lower gradient and is flatter in the higher ranks area (left side) while becomes steeper in the lower ranks area (right side).

Next, clusters are sorted into fewer groups following the proposed rule with aggression parameter  $a$ . The grouping strategy merges the clusters following the preset proportions  $(1 : 2^a : 3^a \dots)$  as detailed in Section 3.1. The number of solutions contained in each group is given in Table 3.1, 3.2 and 3.3 for three levels of aggression parameter respectively. The groupings are also visually shown in Figures 3.4 by

FIGURE 3.4: Grouping clusters on ranked solutions ( $a=1$ )

remarks above the plot for  $a = 1$  and in Figure 3.5 & 3.6 by red vertical lines for  $a = 2$  & 3. Remarks in Figure 3.4 attach group labels to the clusters to help explaining how the grouping strategy works. Since the grouping is based on the clustering which is resulted from the metamodel estimations, groups having more clusters would contain a larger metamodel-estimation range while the numbers of in-group solutions are not guaranteed to follow the same trend. For example, in Figure 3.5, the fourth group has a larger metamodel-estimation range than the third group while the third group has slightly more in-group solutions.

FIGURE 3.5: Grouping clusters on ranked solutions ( $a=2$ )FIGURE 3.6: Grouping clusters on ranked solutions ( $a=3$ )

Lastly, we perform the simulation optimisation based on the guidance provided by the groups. Five groups described in Tables 3.1, 3.2 and 3.3 are treated as five candidate solutions by the modified OCBA (Algorithm 1), which has its computing budget for simulation replications set to 100. This phase equates to online sampling and consequently is time-critical, hence the comparably small number of replications for OCBA. As detailed in Section 3.3, when OCBA selects a particular group for sampling,

TABLE 3.1: Structure of the groups (a=1)

Group	No. of solutions	No. of clusters
1	232	1
2	452	2
3	2071	3
4	4444	4
5	4126	5

TABLE 3.2: Structure of the groups (a=2)

Group	No. of solutions	No. of clusters
1	204	1
2	1084	4
3	3175	9
4	2826	16
5	4036	25

TABLE 3.3: Structure of the groups (a=3)

Group	No. of solutions	No. of clusters
1	46	1
2	379	8
3	1863	27
4	4392	64
5	4645	125

one solution in that group is randomly chosen and gets simulated through the discrete event simulation model. The sampled solution in the best group that performs the best is the result of the proposed multi-fidelity simulation optimisation framework.

### 3.4.4 Results Analysis

The optimal ordering policy  $(s, S)$  for the inventory system is  $(23, 69)$ , which is identified by the 'standard' (with five replications for each alternative). The average total cost with a 90% confidence interval of the optimal solution is  $118.953 \pm 1.674$ . The top-5 solutions obtained from the proposed method and MO<sup>2</sup>TOS are compared with this optimal solution.

The optimisation algorithms are repeated 1000 times for both the proposed multi-fidelity simulation optimisation (MFSO) with three aggression parameter settings and MO<sup>2</sup>TOS. For each run, the number of solutions in the top-5 results that are not significantly different from the optimum is recorded. The gathered data of four competing algorithms with 1000 experiments are plotted as histograms in four different colours side by side in Figure 3.7. The best situation is that all five top-5 solutions obtained from the optimisation algorithms are not significantly different from the optimal solution. For this frequency distribution plot, more solutions in the categories that are farther to the right stand for better performance. For example, MO<sup>2</sup>TOS has 69 trials out of 1000 trials that identify all top-5 solutions as the red bar in the rightmost group and has 23 trials that failed to identify any top-5 solutions in Figure 3.7. The proposed MFSO ( $a=3$ ) has the highest bar (in green) in the rightmost group. There are 61.2% experiments that have all 5 solutions in top-5 that are not

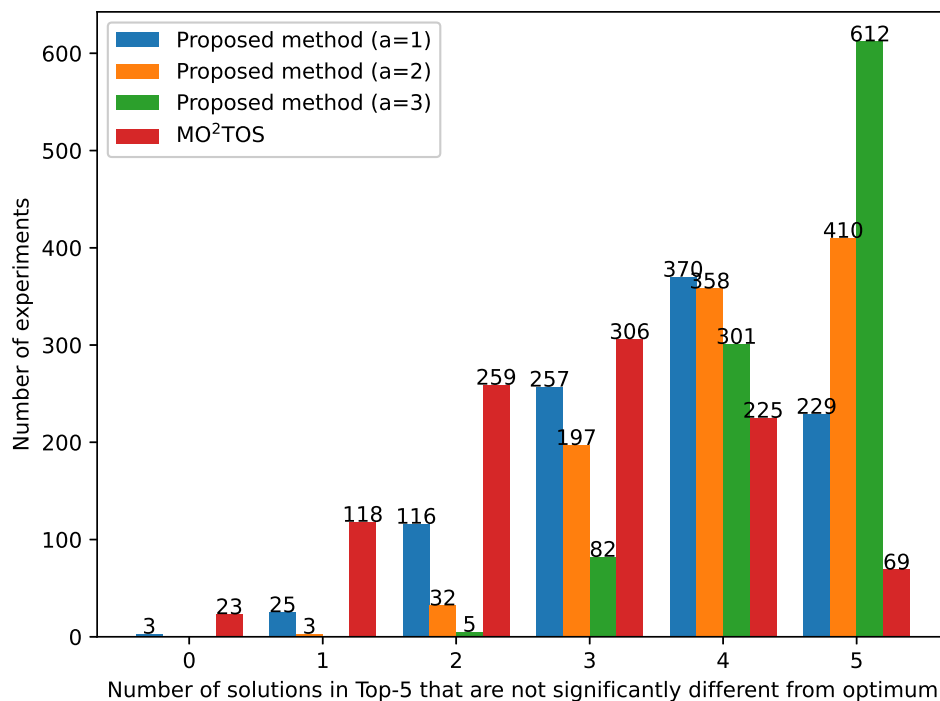


FIGURE 3.7: Comparison of the proposed method with different aggression parameters and MO<sup>2</sup>TOS

significantly different from the optimum. The performance of other solutions in the rightmost category are MFSO ( $a=2$ ) (41%), MFSO ( $a=1$ ) (22.9%) and MO<sup>2</sup>TOS (6.9%) in descending order.

Overall, MFSO  $a = 2$  (orange bars) and  $a = 3$  (green bars) are both skewed extremely to the left and have the highest bar in category five. Most experiments of these two algorithms are concentrated in categories four and five. MFSO  $a = 2$  has 35.8% and 41% of experiments returning results fall in these 2 categories respectively, and the figures are 30.1% and 61.2% for MFSO  $a = 3$ . Furthermore, both aggression parameter settings have a very low number of experiments in the left tail.

It is easy to notice that, in Figure 3.7, the distribution of MFSO  $a = 1$  (in blue) and MO<sup>2</sup>TOS (in red) are different from MFSO  $a = 2$  and  $a = 3$ . The shape of the histogram plots for both the proposed MFSO ( $a=1$ ) and MO<sup>2</sup>TOS have similar symmetric distributions. For MFSO ( $a=1$ ), 37% and 22.9% experiments have four or five solutions in top-5 that are not significantly different from the optimum respectively, which is a lot lower than the same method with  $a = 2$  and 3. For the same two categories, MO<sup>2</sup>TOS has fewer experiments (22.5% and 6.9%). Furthermore, for categories zero and one on the other side of the axis where either no solution or only 1 solution in the top-5 is not significantly different from the optimum, MFSO ( $a = 1$ ) has fewer solutions (0.3% and 2.5%) than MO<sup>2</sup>TOS has (2.3% and 11.8%). In contrast,

MFSO ( $a = 3$ ) has no experiment fit in these two categories and MFSO ( $a = 2$ ) has 0.3% experiments in category one.

To sum up, the results from the experiments suggest that, in this inventory system example, the proposed multi-fidelity simulation optimisation framework with aggression parameter  $a = 3$  outperforms other competing methods. We cannot reject the null hypothesis that the true difference in means is equal to zero. Both MFSO ( $a = 2$ ) and ( $a = 3$ ) lead to significant improvements over MO<sup>2</sup>TOS while  $a = 1$  also has a higher chance than MO<sup>2</sup>TOS to produce a higher quality top-5 solution set. The proposed MFSO framework is able to efficiently deal with the ranking and selection problem with a large number of alternatives and could perform reasonably well when the aggression parameter is properly picked and the metamodel is carefully structured.

Apart from the comparison given above for the metamodel that is well fit, we doubted how the performance of the optimisation would look like if the metamodel is of lower quality. This could happen in reality when very little data is available, the modelling techniques are not properly used or the target system itself is difficult to be modelled. Another metamodel was built with the same training dataset and reduced epoch to achieve lower accuracy intentionally. This neural network has an MAE of 7.8 and Kendall's  $\tau$  of 0.676 (MAE and Kendall's  $\tau$  for the higher quality metamodel are 1.8 and 0.79 respectively).

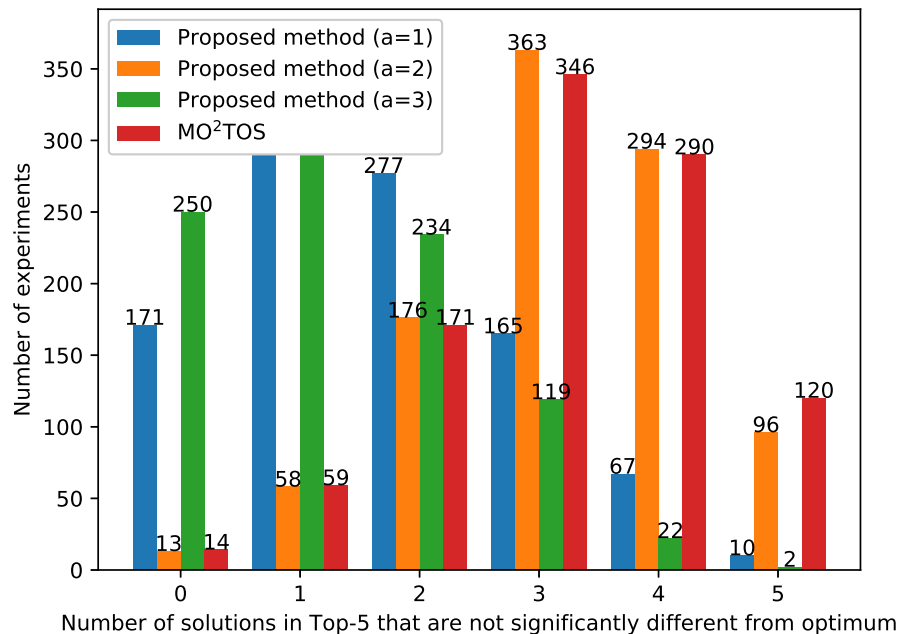


FIGURE 3.8: Comparison of the proposed method with different aggression parameters and MO<sup>2</sup>TOS for Less-accurate Metamodel

The comparison of algorithms for the lower-quality metamodel is illustrated in Figure 3.8. Similar to the previous experiments, we also have MFSO with three sets of aggression parameters and MO<sup>2</sup>TOS for this one. Frequency distribution for MFSO ( $a = 1&3$ ) (blue and green bars) are skewed to the right, where most experiments fall into the categories zero, one and two. On the other hand,  $a = 2$  is skewed left a little bit, which has a similar pattern to MO<sup>2</sup>TOS. In this case, the larger aggression parameter  $a$  does not achieve a better performance when the metamodel is of lower quality.

Overall, the results we got from the inventory system example suggest that our proposed method favours a well-fitted metamodel. When the metamodel is a good description of the underlying response surface, the performance of our method improves as the aggression level  $a$  increases. This suggests a general rule for implementation but more investigation is needed on other examples to make this rule more precise.

We move the focus back to the case with the well-fitted metamodel. In which, MFSO  $a = 1$  seems not as attractive as the other 2 aggression parameters. However, it is understandable if we have a look back at the problem itself. There are 11,325 alternatives in the feasible area of our inventory system ordering policy selection problem. When  $a = 1$ , the first group is constituted of 1430 solutions (Table 3.1) which is relatively close to the group size (2265) in MO<sup>2</sup>TOS. While for  $a = 2$  or 3, the first group contains 211 or 70 respectively. They are at least 85% smaller than the first group for  $a = 1$ .

The higher-quality metamodel gives a higher probability that the first group from the metamodel would contain the high-ranking solutions in the simulation model. Thus, the ranking and selection procedure would sample more from the first group. A smaller first group leads to the fact that the solutions in the most promising area would have a higher chance of being covered by the stochastic sampling process. It aligns well with our observations from these experiments.

Groups in the proposed algorithm are formed based on the clustering results, which depend on the shape of the response surface that is described by the metamodel. Among all the clusters, the first one is especially linked closely to the performance of the optimisation. The size of the clusters tends to be smaller when there are more clusters. A larger aggression parameter leads to more clusters and, on average, a smaller number of solutions per cluster. In this example with a well-fitted metamodel, the first clusters contain 1430, 211 and 70 solutions for MFSO  $a = 1, 2,$  and 3 respectively. With the increase of aggression parameter  $a$ , the decrease of the size of the first clusters slows down and the performance of the optimisation algorithm improves less significant at the same time. It may finally decelerate the improvement of the optimisation performance. In other words, if we keep increasing the aggression

parameter when it is already large enough, we will not have the expected improvement in the optimisation result. It needs to be noted that whether  $a$  is large enough depends on the response surface of the metamodel.

When the metamodel is not well-fitted, as shown in Figure 3.8,  $a = 2$  still produces an acceptable result in this case. In spite of that, more research is needed before it could be generalised.

### 3.5 Conclusion

In this chapter, a multi-fidelity simulation optimisation (MFSO) framework is proposed in order to perform real-time optimisation via simulation. It guides the high-fidelity simulation model to search the feasible area using a low-fidelity metamodel built on the data collected from the simulation model. The solutions are clustered and grouped with an aggression parameter  $a$  based rule in accordance with the metamodel estimations followed by a second step, where they will go through a ranking and selection procedure with the simulation model to derive the final result for optimisation. The method was applied to a textbook example of an inventory system model and the result shows the method performs well in comparison with a leading competitor algorithm MO<sup>2</sup>TOS. MFSO is able to produce a suggestion that is not significantly different to the true optimum solution with reduced replications required from the optimisation algorithm. In the next chapter, the application on a production line is explored where the system is more complicated and noisy.

So far, we have discussed a lot about the advantage of the proposed MFSO framework, as a coin always has two sides, there is also some limitations. The proposed MFSO is only useful for near real-time or relatively real-time decision-making since the simulation is designed to be involved in the online phase of the framework. The time required for the algorithm is still limited by the execution time of the simulation model even though the simulation runs have been reduced. There could be two ways to get out of the problem. The first one is to have a fast simulation model. One of the options out there is parallel simulation, simulation is coded in a way that multiple replications could be carried out at the same time. The other possible way is to bypass the simulation model in the online optimisation phase and only use the fitted metamodel. Since metamodel is deterministic, evaluation of the interest system via the metamodel is much faster and is able to make the optimisation closer to true real-time. However, more experiments on different problems and systems are needed before the metamodel could be confidently applied on its own.

It is also worth mentioning that the proposed method is able to deal with problems that do not have so many alternative solutions but some modification of the algorithm

is needed. An example of the application with ten alternatives could be found in Chapter 5 where all the details are included.

Performance changes are common for mechanical structures when they experience metal fatigue, parts change, maintenance actions, etc. The metamodel would be able to track these system changes if it could be updated with the data collected from the operation when on-site operation data is available. One of the reasons that we selected the neural network for metamodel is that it could be updated by re-training part of the parameters in the model while fixing the others. Nevertheless, some questions need to be answered in future research, for example, how much new data is needed? how many and which neurons should be re-trained? Another direction that could improve the proposed method in future is taking multiple objectives into consideration. Many operational problems have more than one major objective in reality and none of them is dominant. It meets the interest of the stakeholder to include multiple objectives in the optimisation algorithm.



## Chapter 4

# Modelling the Production Line

We have evaluated the proposed algorithm using the inventory system in Chapter 3. To evaluate the algorithm using a more complex system, we choose a manufacturing line that we will explain in this chapter. We first describe the manufacturing system under study in Section 4.1 and the repair policies are elaborated in Section 4.2. A simulation model for the production line is then described in Section 4.3, followed by structuring a metamodel for the same system in Section 4.4. These two models are applied together using a multi-fidelity simulation optimisation framework to obtain the optimal repair policy in Chapter 5.

The system we are working on is a production line that is comprised of a series of sequential workstations that have either a single machine or multiple parallel machines carrying out the same task. There are limited repair staff available who are responsible for the repair of the production line. The repair order of the broken machines affects how the system recovers from a failure, which we measure using the system throughput. We use the simulation model to estimate the throughput of the system under different repair policies.

In this example, we assume an extreme scenario where only one machine can be repaired at a time. The repair staff follow a repair policy which determines the repair order of the broken machines. It is expected that, with the help of the models and the algorithm, the selected repair policy would maximise the short-term throughput and bring the system back to its normal level in the least possible time.

### 4.1 System Description

The production line consists of a series of operations, where parts and materials are assembled or processed to produce final products. The studied production line contains both single and parallel workstations. A single station has one machine that

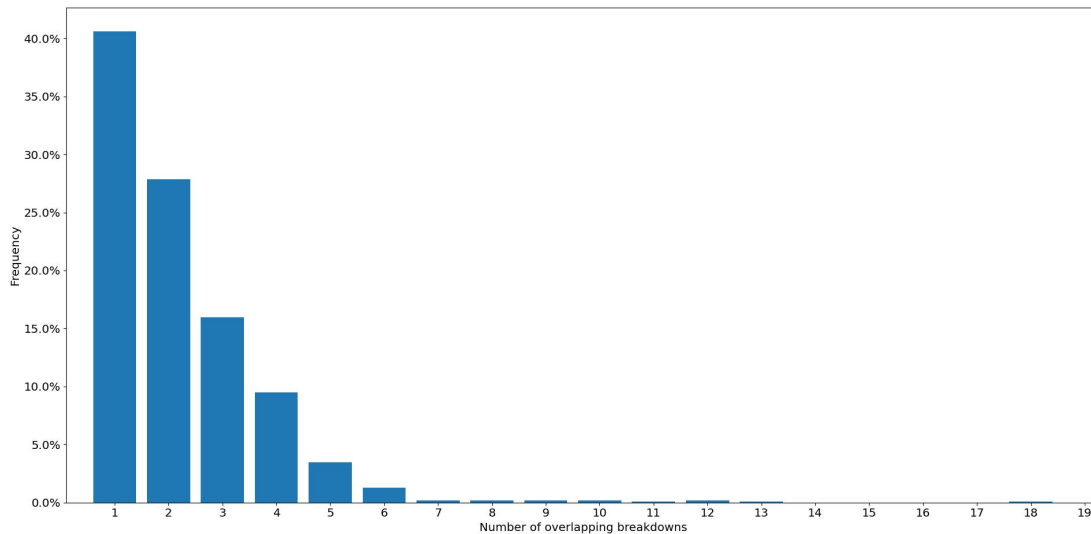


FIGURE 4.1: Frequency of overlapping breakdowns

carries out the designated task, while a parallel station has multiple similar machines each carrying out the same process. The products on the conveyor are moved to the next operation once the process in the current operation finishes. When the output conveyor is full, the machine is blocked or congested and stops taking products from its input. When a machine fails, a request for repair is raised. This can lead to starvation of workstations downstream of the broken machine, particularly if the buffer after the failed machine is empty or the repair takes a long time to complete. Both ‘congestion’ and ‘starving’ could affect the throughput severely. Especially if the ‘starving’ condition happens to the stations close to the exit of the system, there would be no throughput until the ‘starving’ is alleviated.

The production line we are working on has thirty-one machines in twenty-three stations where five stations have parallel processes using either two or three machines. Processing times, failure intervals and repair times are sampled from an empirical distribution derived from operations data. Machinery faults that take place on-site are documented, which includes details like machine identity, fault time, duration and fault code. From the recorded data, it is possible to observe that the situation where multiple machines fail together and wait for repair is common.

When a new failure happens, we count the number of broken-down machines on the line (including the machine that is failing) from the recorded data of a physical manufacturing system. In Figure 4.1, we display the percentage of all failures that see these numbers of broken-down machines. We include only the breakdowns that last over ten minutes since they are more likely to involve maintenance actions. Shorter breakdowns can often be fixed by line operators, e.g. by resetting the machine. Over half of the time ( $59\% = 1 - 41\%$ ) when we see a new failure, we would have more than one machine broken down and over one-third of the situations

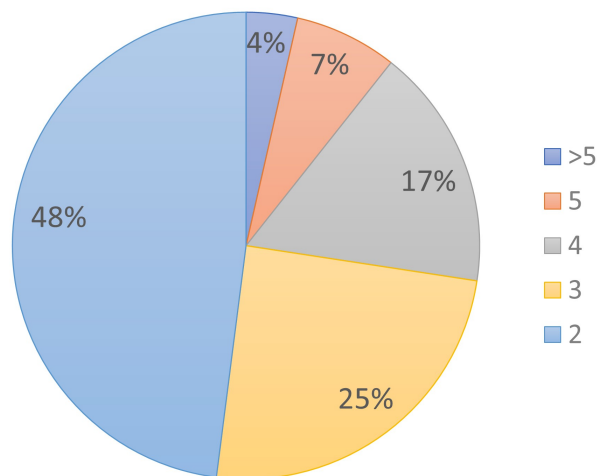


FIGURE 4.2: Frequency of multiple simultaneous breakdowns

(31% = 1 – 41% – 28%) have more than two breakdowns at the same time. These results suggest that there is a need to determine a method for optimising the repair strategy.

Figure 4.2 considers the scenarios in which there are two or more simultaneous breakdowns. This shows the areas of the slices decrease when the overlapping breakdowns increase. The situations with fewer simultaneous breakdowns take place more often. We thus constrain the repair order problem to have at most five failures simultaneously because this constitutes more than 90% of the cases.

Before describing our methodology for improving the decision-making process, we discuss how the on-site team on the shop floor reacts to multiple maintenance requests. Understanding the practical insights from the operations team could benefit the development of the algorithm and affect how it should be embedded in the system. Staff could make professional decisions on which machine to repair first based on multiple factors and different levels of experience in order to maximise the throughput. Factors that could be taken into account include the estimated repair time, the positions of the broken machines, the load in the buffers, whether the breakdown machines are adjacent to each other and whether it is in a single or parallel station. When major faults that are difficult to repair happen, the availability of tools and spare parts also affects the repair order. All these decisions are made manually in the current manufacturing system and we are looking to automate the decision-making process and offer optimised suggestions.

We measure the performance of a repair strategy using the throughput in the three hours following the final breakdown. This short-term performance score is considered the most relevant for the line manager, repair staff and operators in the shift.

## 4.2 Repair Policies

Once the production line is set up and starts operating, maintenance gets involved as a key part of the operation to keep the line working properly. In order to alleviate the effects of performance deterioration, malfunction or even failure, maintenance is needed.

The maintenance policy is the strategy designed for arranging the maintenance work for a complex system. It could be generally classified into two categories, preventive maintenance and corrective maintenance (Shafiee, 2015). More research attention has been given to preventive maintenance, which is necessary for industries like aviation, where in-operation failures might cause irreparable disasters (Kiyak, 2011). Preventive maintenance aims to avoid unplanned system outages while corrective maintenance reacts to breakdowns. Most research on corrective maintenance centres on fault diagnosis, in other words, identifying the root cause of the failure (Wang et al., 2014).

Although it is ideal if precautionary maintenance always works as planned, it is still likely that some unforeseen breakdown could happen and interrupt the service of the system. When there is more than one failed machine, the decision needs to be made on which one is repaired first. It is cost-effective to optimise the repair order since, by doing that, we could improve the throughput without modifying or updating the layout of the manufacturing system.

The permutations of possible repair orders rise exponentially with the increase in the number of simultaneous breakdowns. In order to reduce the number of options we consider in the optimisation and to incorporate new breakdowns that may occur during the three-hour time window, we use repair policies instead of all possible permutations. A repair policy is defined as a strategy for arranging multiple repair jobs that has its roots in dispatch rules. The repair strategies are derived from the scheduling rules (also known as dispatch rules or priority rules) that describe the priority of a list of activities breakdowns in the waiting list and decide which machine to repair next (Panwalkar and Iskander, 1977). The scheduling rules are commonly used in job shop operations. The problem of selecting the optimal repair policy based on real-time data on the current state of the production line has not been extensively researched.

The simulation is set to start from the current state of the system, where we only consider situations where there are between two and five breakdowns. Other machines have a chance to fail during the simulation and a function within the simulation will reorder the repairs following the rules of the repair policy after each new breakdown. Unlike the limit for the initial state, the number of concurrent broken machines is not constrained during the simulation.

Most of the ten repair policies we use below are modified from the scheduling rules in Panwalkar and Iskander (1977) except for the last two which are proposed to take the adjacency of the broken machines into consideration.

The repair policies included in this project are:

**FIFO - First In First Out** The earlier breakdowns get higher priorities for repair

**LIFO - Last In First Out** The later breakdowns get higher priorities for repair

**SRT - Shortest Repair Time first** The repair requiring the shortest time will be carried out first

**LRT - Longest Repair Time first** The repair requiring the longest time will be carried out first

**FOR - Fewer Operations Remaining first** The machine closest to the end of the line has the highest priority for repair

**MOR - More Operations Remaining first** The machine closest to the start of the line has the highest priority for repair

**NOSQ - Next Operation with Shortest Queue first** The broken machine that has the smallest number of items in the buffer immediately following it will have the highest priority.

**COLQ - Current Operation with Longest Queue first** The broken machine that has the highest number of items in the buffer immediately before it will have the highest priority.

**AP - Adjacent Processes first** Breakdowns on adjacent machines will have the highest priority for repair.

**NAP - Non-Adjacent Processes first** Breakdowns on non-adjacent machines have the highest priority for repair.

The first two rules, FIFO and LIFO, are time-based rules related to the fault time. SRT and LRT are the decisions made in accordance with the estimated repair time. The repair staff are assumed to have sufficient knowledge and experience to make reasonable estimates on the repair time with the help of fault messages. FOR and MOR are the strategies that consider the location of the broken machines, which is relative to the beginning or end of the production line. NOSQ and COLQ are the rules based on the size of the queue in the buffers located immediately after and before the broken machines, respectively. AP and NAP take the adjacency of breakdowns into account which is the relative locations of the fails. We limit the definition of the adjacent machines to those in the same station or in the stations strictly next to each

other. For those broken machines assigned the same priority, they further follow a first-in-first-out rule.

These ten repair policies are five pairs of opposite rules that prioritise the repair tasks based on five different criteria. In addition to these repair rules, we will always repair single machines before parallel machines according to practical insights. Through optimisation, we expect to find the most suitable repair policy that would lead to the largest expected throughput for the given situation.

### 4.3 Simulation Model

We use a discrete event simulation model (DES) to describe the production line. The simulation model has been accepted by industry-leading companies to assist decision-making in operation for some time (Higgins and Ladbrook, 2018). The simulation model is the foundation of the proposed multi-fidelity simulation optimisation framework. It is not only applied in online optimisation but also used for building the metamodel

We would see similar patterns in operational data repeatedly when the production line works normally. However, those situations having malfunctions, in which we have our interest, do not appear very often. With the simulation model, it is possible to estimate the performances for the given state. Therefore simulation model sits at the core of operation planning. Most manufacturers in the industry construct their models for the production line using commercial software.

However, there are problems with communicating with simulation models in commercial software with the coded optimisation algorithms. The commercial software normally enables access to model output data through application programming interfaces (API) and the output data could then be processed in the optimisation algorithm. In order to achieve real-time optimisation, the interaction between models and the optimisation algorithm should be efficient and convenient. Therefore, we choose to build the models and the optimisation algorithm in the same programming language. It is beneficial as we will have more control over the model-algorithm communication. Python is used in our experiments to build and run the simulation model, metamodel and optimisation algorithm for it has diversified packages that support our needs from model structuring to data processing. In addition, for a research project, building a simulation model from scratch instead of using the existing one makes it tailored for the research objective and could include all necessary details and desired functions (e.g. repair policies, hot-start).

### 4.3.1 Simulation Model Structure

A discrete event simulation (DES) model is structured for the production line using SimPy package (SimPy, 2020) in Python. SimPy offers basic components needed for a DES model, such as shared resources, waiting process, interruption, etc. Its event dispatcher is based on the Python generator object.

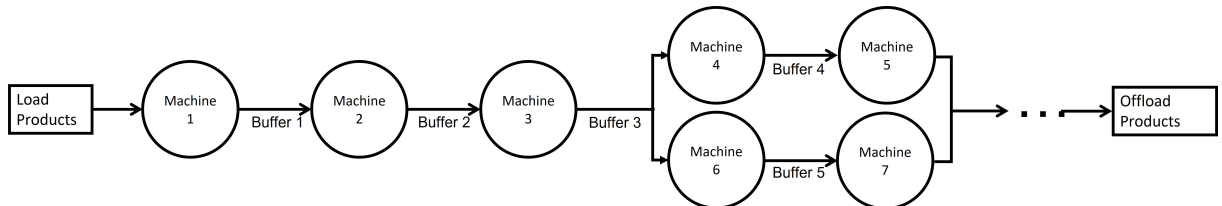


FIGURE 4.3: Illustration of a production line

The manufacturing system under research is a real-life production line. The products are loaded onto the production line from the entry point and they start their journeys on the line passing through the workstations that execute various operations. The time it takes to complete the specific operation is the cycle time (CT) of the machine, which we assume to be an independent and identically distributed (i.i.d.) random variable. When the workstation has parallel machines, each one has its own CT distribution. Machines could break down and two variables are used to describe the events. They are the time between two failures (time between failure (TBF)) and the time it requires for the repair (time to repair (TTR)), which we also assume to be i.i.d. After the process in the machine is complete, the product will go through a quality check, if it fails the check; the product is sent for rework. The rate of passing the quality check follows the 'first time through rate' (FTT, i.i.d.). The product will go through that machine again if it fails the check.

Once the product passes the quality check, it moves onto the conveyor towards the next operation. If the next operation is still working on the previous job or there are already products waiting in the queue, the product joins the queue and waits on the conveyor. The size of the buffer (conveyor) is set to ten products in the experiments. When the buffer is full, the machine before the buffer stops working until that buffer is able to accept products again. When a product completes all the processes and leaves the production line, the measurement of the processed units (throughput) is increased by one. The random variables (CT, TBF, TTR) we use for simulation are sampled from empirical distribution functions (edf) obtained from operation data collected from the plant. They could also be drawn from suitable parametric distributions (e.g. normal distribution, exponential distribution, gamma distribution). Both edf and parametric distribution are enabled in the simulation program, we use edf to align with what the company does in their simulation model.

Three Python classes (machine, buffers and product) are designed for reuse and generalisation so that the model can be easily adapted to a different production line. Figure 4.3 graphically illustrates the way these modules work together on a production line that has both single and parallel stations.

---

**Algorithm 2** Simulating the production line

---

```

1: procedure SIMULATION (production line parameter, initial state, simulation time,
   repair policy)
2:   Buffers = Buffer (production line parameter)   ▷ Buffer function reads in data
3:   Machines = Machine (production line parameter) ▷ Machine function reads in
   data
4:   if initial state exists then
5:     Load initial state
6:   else
7:     Start from empty line
8:   end if
9:   Throughput ← Run Simulation
10: end procedure

```

---

Machines and buffers are instantiated from the given data at the beginning and the products are instantiated one by one when they are loaded onto the line. If the model does not start from time zero, the initial states are loaded. The throughput is output at the end of the simulation. The procedure of simulation is briefly described in Algorithm 2.

The characteristics of the machines (CT, TTR, TBF, FTT) are recorded in the input form for parametric distribution. The empirical distribution function can be enabled by setting the corresponding parameter to zero and the model will read data from the relevant files. It also includes details of how the machines and conveyors are connected to each other. Table 4.1 demonstrates a sample, where machine a is a single process and machine b1 & b2 are the parallel processes.

TABLE 4.1: Input parameter form for DES model

machine name	CT		TBF	TTR	FTT	In	Out
	mean	std	mean	mean	rate		
machine a	1.5	0.5	100	9	1	-1	0
machine b1	2.3	0.6	100	10	0.9	0	1
machine b2	2.1	0.7	100	11	1	0	1

The two rightmost columns are the input and output conveyors of the machine, the first machine of the system will have '-1' for 'In' while the last one will have '-2' for 'Out'. Other conveyors are labelled from zero. Five parameters of the machines (six columns after the machine name) are used to form predetermined distributions. Mean



values and standard deviation are recorded in accordance with the needs of the preset distributions. For the cases without empirical distribution function, we assume CT follows normal distribution (Johnson, 2002), TBF has exponential distribution (Epstein, 1958) and TTR is Erlang distributed (Chen et al., 1993).

The simulation model is enabled to start from a given state instead of always from the empty line. The starting state includes the availability of the machines, the buffer loads, and whether the machine is occupied by any task. Table 4.2 is designed for initialisation. It makes the simulation more practical since the product line is rarely empty after the system comes into service.

TABLE 4.2: Input status for DES model

conveyor_id	location	status
1	1	3
2	2	3
3	3	5
4	4	5
5	4	5

machine_id	machine_loc	processed_time
machine 1	1	0.15
machine 2	2	0.15
machine 3	3	0.15
machine 4	4	0.15
machine 5	5	0.15
machine 6	4	0.15
machine 7	5	0.15

machine_broken
5
10

Table 4.2 consists of three parts: the status of the conveyors, the status of the machines and the broken-down machines. For the status of conveyors, 'conveyor\_id' is the identification number of the conveyor while the location could be different when the parallel machines exist. The 'conveyor\_id' is aligned with the code in the 'In' and 'Out' columns in Table 4.1. The 'status' is the load of the conveyor. For example, three and five in the table means there are three or five products on the corresponding conveyors. For the status of machines, it contains 'machine\_id', station number ('machine\_loc') and the time of the product that has been processed by the machine in 'processed\_time' column. Figure 4.3 illustrates both conveyors and machines in Table 4.2. How much work has been done in the machine ('processed\_time') is not easily accessible in most systems. They could be left at zero for simplification but at the cost of losing some accuracy. For the machine availability, the identification numbers of the broken-down machines are listed in the column 'machine\_broken'.

### 4.3.2 Simulation Output

The production line under study contains thirty-one machines in twenty-three stations and twenty-four buffers. We demonstrate the performance of the simulated production line using a 500-hour simulation with a 'FIFO' repair policy and cold start from time zero. The model works with the empirical distribution functions for initialisation. Although the optimisation would be applied with 3-hour system performance, we display a much longer simulation in this section in order to provide a more general view of the simulation model for the target manufacturing system.

The plots of the simulation output data can be found in Figure 4.4 and 4.5. The ticks on the y-axis are removed for confidential reasons. Two charts are for the jobs completed every hour with 'FIFO' and the rolling JPH (jobs per hour) with all ten repair policies under the same random stream respectively. The plot for the hourly throughput (Figure 4.4) fluctuates in a small range most of the time. But drops in throughput could easily be observed when the system experiences bigger faults. The manufacturing system simulation model is comprised of multiple stochastic processes, which could lead to situations where outputs have large variances. Multiple replications are made to obtain the expected system performance to fit the metamodel.

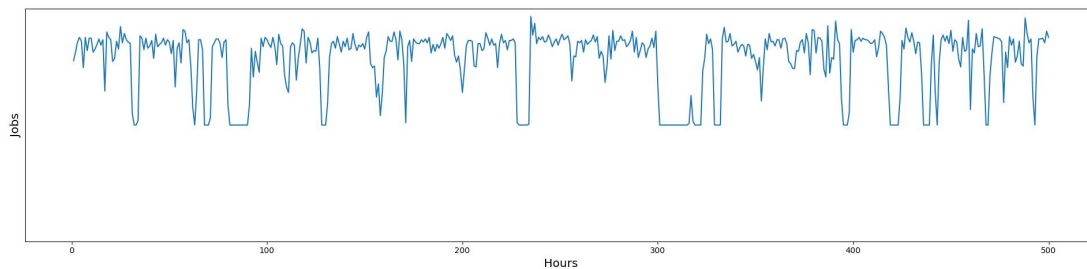


FIGURE 4.4: Plot of hourly throughput for a 500-hour simulation with 'FIFO' repair policy

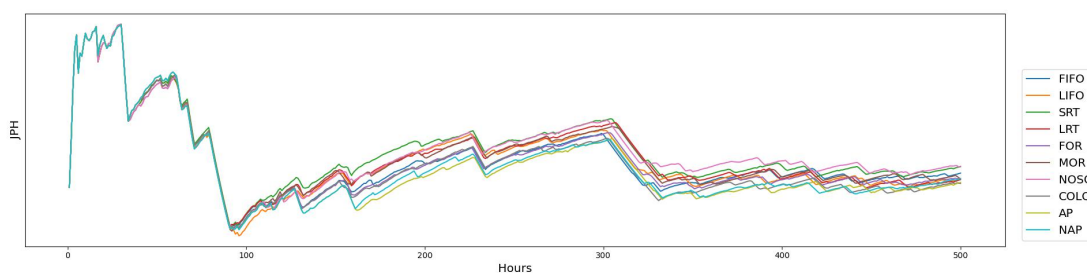


FIGURE 4.5: Plot of jobs per hour for 500-hour simulations with 10 repair policies

The other plot for the rolling JPH ( $JPH = \text{Throughput} / \text{Hours}$ ) in Figure 4.5 is made by having the same system work with all ten repair policies under the same random stream (Figure 4.5). Ten simulations are marked in different colours for ten repair policies. We can see from the chart that a similar trend is shared among different repair policies when they have the same starting state and the same random stream.

All ten line plots tend to be relatively smooth after a short time from starting the simulation when the system has experienced a few breakdowns.

The rolling JPH diverts from 100 hours onward. Some repair policies perform better than others. Also, different repair policies could outperform others in different time periods. For example, SRT (in green) produces higher throughput between 100 hours and 200 hours while NOSQ (in pink) is the best repair policy between 320 hours and 450 hours. This finding supports the reasonableness of performing optimisation for finding the best repair policy under various system states. Two plots are both made with one replication to show the viability and display the system behaviours, multiple replications are made for both data collection and optimisation.

## 4.4 Metamodel

A metamodel is a surrogate model or emulator of a simulation model. [Friedman and Pressman \(1988\)](#) define a metamodel to be “any analytic auxiliary model which is used to aid in the interpretation of a more detailed model”. It is built from noisy observations  $Y = [y_1, y_2, \dots, y_N]$  at design points  $X = [x_1^T, x_2^T, \dots, x_N^T]$  in a fixed feasible area  $R^d$ . The  $X$  is the system input and the  $Y$  is the expected throughput. As discussed in Section 3.2, we use a feedforward neural network for the low-fidelity metamodel in the multi-fidelity algorithm.

The simulation model has two groups of input variables, describing the system configuration and the state of the production line. If all of them were used as inputs to the metamodel, there would be a large number of parameters to fit even for a shallow (small-scale) neural network. The dimension of variables for the metamodel is reduced to ease the burden of data collection. In order to reduce the dimensionality of the problem, we use the structure and meaning of the input parameters to simplify the state space. This is described in Section 4.4.1. Reducing the number of input parameters for the neural network reduces the number of simulation replications needed to fit the metamodel.

An adaptive sequential sampling method is proposed in Section 4.4.2, which aims to design experiments for efficiently collecting data from a stochastic system and getting the most out of a limited budget. New areas are sampled iteratively. A small number of replications are applied at the designed points in the first place and more replications are allowed when necessary. The experimental design balances the need for exploring new areas and exploiting the sampled areas.

#### 4.4.1 Input Variables Formation

The output of the metamodel is the system throughput, which we define to be the expected throughput in three hours, given the starting state of the system. The input parameters/variables of the simulation model can be divided into two groups. The first group gives the configuration of the production line (Table 4.1). The configuration of the production line would become the implicit information of the metamodel after fitting. A different line can use the same simulation model with a modified input file but the metamodel needs to be rebuilt. The other group gives the initial state of the system which becomes the starting point of the simulation (Table 4.2). The initial state of the system will be the input to the metamodel, which contains the buffer states, machine availability and the repair policy.

The state space of the metamodel is aimed to be reduced so that fewer simulation runs are needed for the data collection which would be used for metamodel fitting. While fewer input variables are desired, we should be cautious to avoid oversimplification. The quality of the fit of the metamodel is taken into consideration since it directly affects the performance of the multi-fidelity approach. The balance between the number of input variables and the consequent estimation accuracy needs to be found.

Twenty-four conveyors in the production line could have the possible states as large as  $(10 + 1)^{24} = 9.850 \times 10^{24}$  for the buffer state alone given the buffer capacity is ten plus an empty state. If we evenly split the conveyors into four sections, each section has six conveyors and there are sixty-one possible states for every section. The possible states for the system are reduced to  $(6 \times 10 + 1)^4 = 1.385 \times 10^7$ , which is  $10^{17}$  times smaller than the original space. If the conveyors are split into three even sections each with eight conveyors, there could be eighty-one in-section states. The system would have  $(8 \times 10 + 1)^3 = 5.314 \times 10^5$  possible states. Fewer experiments could thus cover the smaller state space if it is found reasonable through analysis.

We investigate whether the states of all buffers could be reduced to the states of several sections of the line and how many sections we should have. We evenly split the whole system and every section contains the same number of buffers. A pilot test is designed with the simulation model where section states are limited to three extreme cases and each has a set of certain buffer states (Table 4.3). At the same time, the total content in each section is controlled to be the same. The test with the 3-section design has eight buffers in each section and the starting state comes with 30 products in each section. Other input system states remain the same in the test, all machines are available at the beginning of the simulation and the repair policy is fixed at FIFO.

The capacity of the buffer is ten. Three extreme cases are products piling up and congested at the beginning or the end of the section and products evenly distributed in the buffers inside each section respectively. The assignment of the buffers for three

TABLE 4.3: The assignment of the buffers for three sets of configurations within each section in the pilot test (3-section design)

Extreme Case	Buffer States
High Start; Low End	10, 10, 10, 0, 0, 0, 0, 0
Low Start; High End	0, 0, 0, 0, 0, 10, 10, 10
Balanced	3, 4, 4, 4, 3, 4, 4, 4

in-section extreme cases for the 3-section design can be found in Table 4.3. These three extreme cases are designed to be three situations that have larger differences from each other than from the other cases that are not tested. Thus, if the test results from the extreme cases are reasonable, we will have confidence that the variable reduction by sectioning the manufacturing system works.

TABLE 4.4: 27 scenarios of the 3-section design

	Section 1	Section 2	Section 3
Scenario 1 (S1)	High Start; Low End	High Start; Low End	High Start; Low End
Scenario 2 (S2)	High Start; Low End	High Start; Low End	Low Start; High End
Scenario 3 (S3)	High Start; Low End	High Start; Low End	Balanced
Scenario 4 (S4)	High Start; Low End	Low Start; High End	High Start; Low End
Scenario 5 (S5)	High Start; Low End	Low Start; High End	Low Start; High End
Scenario 6 (S6)	High Start; Low End	Low Start; High End	Balanced
Scenario 7 (S7)	High Start; Low End	Balanced	High Start; Low End
Scenario 8 (S8)	High Start; Low End	Balanced	Low Start; High End
Scenario 9 (S9)	High Start; Low End	Balanced	Balanced
Scenario 10 (S10)	Low Start; High End	High Start; Low End	High Start; Low End
Scenario 11 (S11)	Low Start; High End	High Start; Low End	Low Start; High End
Scenario 12 (S12)	Low Start; High End	High Start; Low End	Balanced
Scenario 13 (S13)	Low Start; High End	Low Start; High End	High Start; Low End
Scenario 14 (S14)	Low Start; High End	Low Start; High End	Low Start; High End
Scenario 15 (S15)	Low Start; High End	Low Start; High End	Balanced
Scenario 16 (S16)	Low Start; High End	Balanced	High Start; Low End
Scenario 17 (S17)	Low Start; High End	Balanced	Low Start; High End
Scenario 18 (S18)	Low Start; High End	Balanced	Balanced
Scenario 19 (S19)	Balanced	High Start; Low End	High Start; Low End
Scenario 20 (S20)	Balanced	High Start; Low End	Low Start; High End
Scenario 21 (S21)	Balanced	High Start; Low End	Balanced
Scenario 22 (S22)	Balanced	Low Start; High End	High Start; Low End
Scenario 23 (S23)	Balanced	Low Start; High End	Low Start; High End
Scenario 24 (S24)	Balanced	Low Start; High End	Balanced
Scenario 25 (S25)	Balanced	Balanced	High Start; Low End
Scenario 26 (S26)	Balanced	Balanced	Low Start; High End
Scenario 27 (S27)	Balanced	Balanced	Balanced

Table 4.4 gives out the design detail of all twenty-seven ( $3^3$ ) scenarios for the 3-section design. If the scenarios show similar performance, the splitting method is more likely to preserve the information from the original system state. Each scenario is simulated 100 times to test with different instances of the scenario and common random

numbers are applied to instances across various scenarios. The scenarios are then compared with the balanced line (scenario 27 / S27 in Figure 4.6) using the difference between them with 95% confidence intervals to determine the significance.

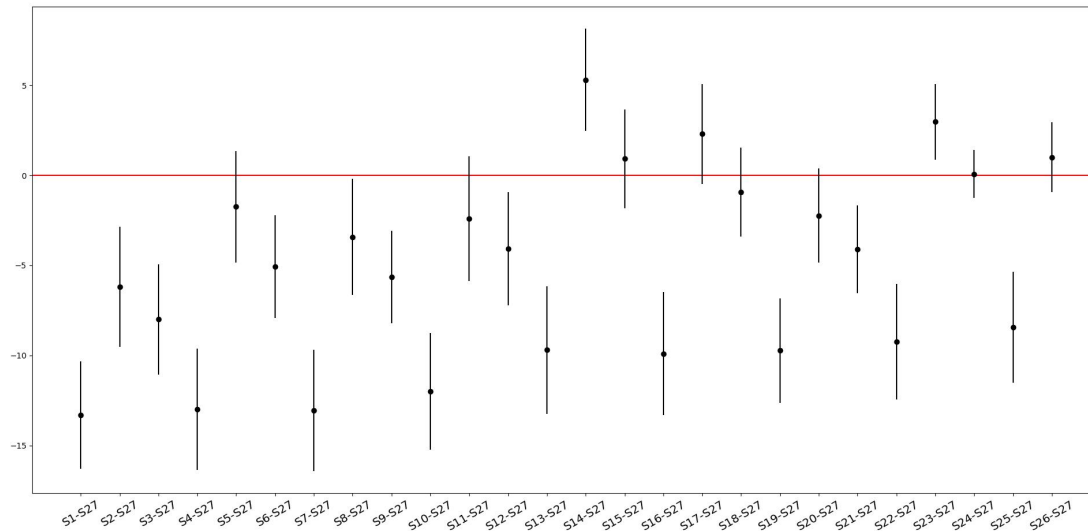


FIGURE 4.6: 95% CI of the differences between scenarios with even split of 3 sections

The differences and the confidence intervals (e.g. S1-S27 (the balanced line), S2-S27) are plotted in Figure 4.6. Eight of twenty-six intervals contain zero which means these scenarios are not statistically different from S27, while the other eighteen scenarios are significantly different from S27. The averages of the differences lie between -13 and +5. Two scenarios that are significantly better than the balanced case are scenarios 14 and 23. They both have ‘Low Start; High End’ for section 2 and section 3. The scenarios having ‘High Start; Low End’ in section 3 have lower throughputs than all other scenarios (e.g. S1, S4, S7). A cyclical pattern appears in the plot. Every three scenarios form a cycle which aligns with the status designed for section 3. In each cycle, scenarios follow the same sequence of ‘High Start; Low End’, ‘Low Start; High End’ and ‘Balanced’ for their third section respectively. It leads to the thought that more attention may need to be given towards the end of the production line. Splitting the production line into three sections does not work very well. We continue to repeat the test with the 4-section design. Figure 4.7 plots the differences between  $3^4 - 1 = 80$  scenarios and the balanced line. Like the experiments for testing the 3-section split, one hundred instances are simulated for all 81 scenarios.

TABLE 4.5: The assignment of the buffers for three sets of configurations within each section in the pilot test (4-section design)

Extreme Case	Buffer Status
High Start; Low End	10, 10, 0, 0, 0, 0
Low Start; High End	0, 0, 0, 0, 10, 10
Balanced	3, 3, 4, 3, 3, 4

TABLE 4.6: 81 scenarios of the 4-section design

	Section 1	Section 2	Section 3	Section 4
Scenario 1 (S1)	High Start; Low End	High Start; Low End	High Start; Low End	High Start; Low End
Scenario 2 (S2)	High Start; Low End	High Start; Low End	High Start; Low End	Low Start; High End
Scenario 3 (S3)	High Start; Low End	High Start; Low End	High Start; Low End	Balanced
Scenario 4 (S4)	High Start; Low End	High Start; Low End	Low Start; High End	High Start; Low End
Scenario 5 (S5)	High Start; Low End	High Start; Low End	Low Start; High End	Low Start; High End
Scenario 6 (S6)	High Start; Low End	High Start; Low End	Low Start; High End	Balanced
Scenario 7 (S7)	High Start; Low End	High Start; Low End	Balanced	High Start; Low End
Scenario 8 (S8)	High Start; Low End	High Start; Low End	Balanced	Low Start; High End
Scenario 9 (S9)	High Start; Low End	High Start; Low End	Balanced	Balanced
Scenario 10 (S10)	High Start; Low End	Low Start; High End	High Start; Low End	High Start; Low End
Scenario 11 (S11)	High Start; Low End	Low Start; High End	High Start; Low End	Low Start; High End
Scenario 12 (S12)	High Start; Low End	Low Start; High End	High Start; Low End	Balanced
Scenario 13 (S13)	High Start; Low End	Low Start; High End	Low Start; High End	High Start; Low End
Scenario 14 (S14)	High Start; Low End	Low Start; High End	Low Start; High End	Low Start; High End
Scenario 15 (S15)	High Start; Low End	Low Start; High End	Low Start; High End	Balanced
Scenario 16 (S16)	High Start; Low End	Low Start; High End	Balanced	High Start; Low End
Scenario 17 (S17)	High Start; Low End	Low Start; High End	Balanced	Low Start; High End
Scenario 18 (S18)	High Start; Low End	Low Start; High End	Balanced	Balanced
Scenario 19 (S19)	High Start; Low End	Balanced	High Start; Low End	High Start; Low End
Scenario 20 (S20)	High Start; Low End	Balanced	High Start; Low End	Low Start; High End
Scenario 21 (S21)	High Start; Low End	Balanced	High Start; Low End	Balanced
Scenario 22 (S22)	High Start; Low End	Balanced	Low Start; High End	High Start; Low End
Scenario 23 (S23)	High Start; Low End	Balanced	Low Start; High End	Low Start; High End
Scenario 24 (S24)	High Start; Low End	Balanced	Low Start; High End	Balanced
Scenario 25 (S25)	High Start; Low End	Balanced	Balanced	High Start; Low End
Scenario 26 (S26)	High Start; Low End	Balanced	Balanced	Low Start; High End
Scenario 27 (S27)	High Start; Low End	Balanced	Balanced	Balanced
Scenario 28 (S28)	Low Start; High End	High Start; Low End	High Start; Low End	High Start; Low End
Scenario 29 (S29)	Low Start; High End	High Start; Low End	High Start; Low End	Low Start; High End
Scenario 30 (S30)	Low Start; High End	High Start; Low End	High Start; Low End	Balanced
Scenario 31 (S31)	Low Start; High End	High Start; Low End	Low Start; High End	High Start; Low End
Scenario 32 (S32)	Low Start; High End	High Start; Low End	Low Start; High End	Low Start; High End
Scenario 33 (S33)	Low Start; High End	High Start; Low End	Low Start; High End	Balanced
Scenario 34 (S34)	Low Start; High End	High Start; Low End	Balanced	High Start; Low End
Scenario 35 (S35)	Low Start; High End	High Start; Low End	Balanced	Low Start; High End
Scenario 36 (S36)	Low Start; High End	High Start; Low End	Balanced	Balanced
Scenario 37 (S37)	Low Start; High End	Low Start; High End	High Start; Low End	High Start; Low End
Scenario 38 (S38)	Low Start; High End	Low Start; High End	High Start; Low End	Low Start; High End
Scenario 39 (S39)	Low Start; High End	Low Start; High End	High Start; Low End	Balanced
Scenario 40 (S40)	Low Start; High End	Low Start; High End	Low Start; High End	High Start; Low End
Scenario 41 (S41)	Low Start; High End	Low Start; High End	Low Start; High End	Low Start; High End
Scenario 42 (S42)	Low Start; High End	Low Start; High End	Low Start; High End	Balanced
Scenario 43 (S43)	Low Start; High End	Low Start; High End	Balanced	High Start; Low End
Scenario 44 (S44)	Low Start; High End	Low Start; High End	Balanced	Low Start; High End
Scenario 45 (S45)	Low Start; High End	Low Start; High End	Balanced	Balanced
Scenario 46 (S46)	Low Start; High End	Balanced	High Start; Low End	High Start; Low End
Scenario 47 (S47)	Low Start; High End	Balanced	High Start; Low End	Low Start; High End
Scenario 48 (S48)	Low Start; High End	Balanced	High Start; Low End	Balanced
Scenario 49 (S49)	Low Start; High End	Balanced	Low Start; High End	High Start; Low End
Scenario 50 (S50)	Low Start; High End	Balanced	Low Start; High End	Low Start; High End
Scenario 51 (S51)	Low Start; High End	Balanced	Low Start; High End	Balanced
Scenario 52 (S52)	Low Start; High End	Balanced	Balanced	High Start; Low End
Scenario 53 (S53)	Low Start; High End	Balanced	Balanced	Low Start; High End
Scenario 54 (S54)	Low Start; High End	Balanced	Balanced	Balanced
Scenario 55 (S55)	Balanced	High Start; Low End	High Start; Low End	High Start; Low End
Scenario 56 (S56)	Balanced	High Start; Low End	High Start; Low End	Low Start; High End
Scenario 57 (S57)	Balanced	High Start; Low End	High Start; Low End	Balanced
Scenario 58 (S58)	Balanced	High Start; Low End	Low Start; High End	High Start; Low End
Scenario 59 (S59)	Balanced	High Start; Low End	Low Start; High End	Low Start; High End
Scenario 60 (S60)	Balanced	High Start; Low End	Low Start; High End	Balanced
Scenario 61 (S61)	Balanced	High Start; Low End	Balanced	High Start; Low End
Scenario 62 (S62)	Balanced	High Start; Low End	Balanced	Low Start; High End
Scenario 63 (S63)	Balanced	High Start; Low End	Balanced	Balanced
Scenario 64 (S64)	Balanced	Low Start; High End	High Start; Low End	High Start; Low End
Scenario 65 (S65)	Balanced	Low Start; High End	High Start; Low End	Low Start; High End
Scenario 66 (S66)	Balanced	Low Start; High End	High Start; Low End	Balanced
Scenario 67 (S67)	Balanced	Low Start; High End	Low Start; High End	High Start; Low End
Scenario 68 (S68)	Balanced	Low Start; High End	Low Start; High End	Low Start; High End
Scenario 69 (S69)	Balanced	Low Start; High End	Low Start; High End	Balanced
Scenario 70 (S70)	Balanced	Low Start; High End	Balanced	High Start; Low End
Scenario 71 (S71)	Balanced	Low Start; High End	Balanced	Low Start; High End
Scenario 72 (S72)	Balanced	Low Start; High End	Balanced	Balanced
Scenario 73 (S73)	Balanced	Balanced	High Start; Low End	High Start; Low End
Scenario 74 (S74)	Balanced	Balanced	High Start; Low End	Low Start; High End
Scenario 75 (S75)	Balanced	Balanced	High Start; Low End	Balanced
Scenario 76 (S76)	Balanced	Balanced	Low Start; High End	High Start; Low End
Scenario 77 (S77)	Balanced	Balanced	Low Start; High End	Low Start; High End
Scenario 78 (S78)	Balanced	Balanced	Low Start; High End	Balanced
Scenario 79 (S79)	Balanced	Balanced	Balanced	High Start; Low End
Scenario 80 (S80)	Balanced	Balanced	Balanced	Low Start; High End
Scenario 81 (S81)	Balanced	Balanced	Balanced	Balanced

The design for the experiments with 4 sections is altered slightly to include six buffers in each section, where we have  $3^4 = 81$  scenarios. The buffer content in each section is adapted to 20 so that the start and the end buffer of the section could be full in the extreme case while the total buffer content on the production line is close to the 3-section design. The buffer state of the three extreme cases for the 4-section design is given in Table 4.5. We record the design detail of all eighty-one scenarios for the 4-section design in Table 4.6. In the graph of the differences between scenarios and the balanced line (Figure 4.7), forty-three in eighty of the scenarios contain zero in the confidence interval, which is more than half of the scenarios in the test. Although no scenario performs significantly better than the balanced line (S81) in the 4-section design, there are four scenarios (i.e. S76, S77, S78 and S79) that have higher average throughputs than the compared S81. Three of them (S76, S77 and S78) share the same settings for the first three sections, where section 1 and section 2 are balanced, and section 3 is 'Low Start; High End'. The cyclic pattern still exists in the 4-section design, there are three scenarios that always have lower average performance than the others in a cycle of nine scenarios. These scenarios all have section 3 set to 'High Start; Low End'. The averages of the differences in Figure 4.7 lie between -6 and +0.5.

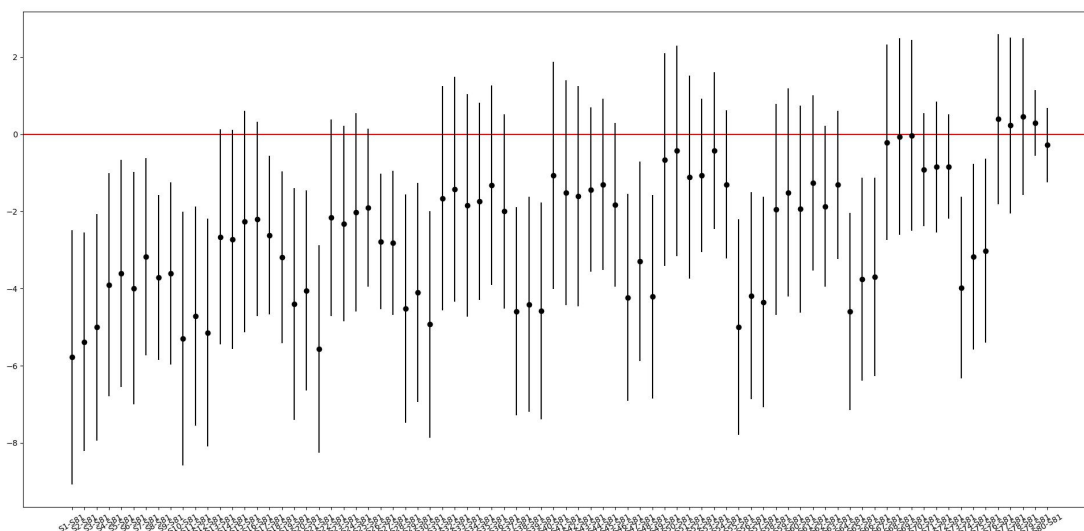


FIGURE 4.7: 95% CI of the differences between scenarios with even split of 4 sections

From the findings, the 4-section design has over half of the scenarios (43/80) that are not significantly different from the balanced scenario which is a higher proportion than the 3-section design (8/26). It shows that more extreme scenarios in the 4-section design could be represented by each other in terms of the short-term throughputs. The 4-section design also has a much smaller average difference ([-6,+0.5]) than the 3-section design ([-13, +5]). For these less extreme scenarios in the 4-section design, we believe they would have more similarities and less difference in their performances. We thus deduce the 4-section design works better than the 3-section design. We do not increase the sections of the production line further to five as more sections would



bring a larger number of extreme scenarios and would take a longer time to carry out the test. Also, the possible states would increase while the simplification that we are looking for becomes less significant. Therefore, the 4-section design has been applied to reduce the state space for the metamodel while losing less information from the original system even though the 3-section design could reduce more state space.

In addition to the buffers, the variables concerning the machines are also grouped into four parts which align with the sections divided for the buffers. So that the locations of the failures are preserved with fewer variables as well. Since breakdowns are the major factors directly linked to the repair order and how the system would recover from the disruptions, they are further divided into finer groups based on whether they belong to a single or parallel station. Specifically, we count how many single or parallel machines fail in four sections respectively. A total of thirty-one machines are turned into four sections of parallel machines that contain six, three, two, and two machines respectively and four sections of non-parallel machines that have three, five, five and five machines respectively.

Furthermore, the adjacency of the broken machines is included to offer more information about the failures, which aims to further narrow down the possible locations of the breakdowns with reduced variables. Two elements are used to describe the adjacency and they are designed in a heuristic way. One counts the number of adjacent broken machines and another counts the number of groups that adjacent failures form. The adjacent information works better in some cases than others. For example, when one single machine and one parallel machine fail in the system shown in Figure 4.3, there are twelve options since three single machines and four parallel machines are in the section. If the adjacency is (2,1) at the same time, the options are reduced to two since machine 3 and machine 4 / 6 are the two only combinations of parallel and non-parallel machines that are adjacent to each other. If the adjacency is (0,0), there are ten possible options where we exclude the two above cases. A total of ten variables are used for the availability of machines which is one-third of thirty-one machine states. As we limit the number of failures to five, most location information could be preserved with the proposed ten reduced variables. Lastly, the decision variable repair policy is an input to the metamodel as well.

Five groups of variables with the 4-section design are thus considered to describe the situation on the production line, they are the final choices of the input variables for our metamodel based on the investigations in this section. They are listed below and clarified in the following paragraph.

- Variable 1 - The sum of buffer content in each of 4 sections
- Variable 2 - The number of parallel machines that are broken down in each of the 4 sections

- Variable 3 - The number of single machines that have broken down in each of the 4 sections
- Variable 4 - A measure of the adjacency of breakdowns
- Variable 5 - The repair policy

Variables 1,2 and 3 are 4-dimensional. Each dimension of variable 1 stands for the total number of products in the buffers in a section. Each dimension of variable 2 and variable 3 is the number of failures in the corresponding section for parallel and non-parallel machines respectively.

Variable 4 uses two dimensions to describe the adjacency of the interrupted machines across the whole line. The first element shows how many of the machines among the failures meet the requirement of adjacency. The second one is the number of groups the adjacent machines could form. The adjacency defined in this research is limited to 1-station adjacency, which means machines in two neighbouring stations are regarded as adjacent. For example, if machines 2, 6, and 7 in Figure 4.3 are broken simultaneously,  $V_4$  (Variable 4) would be (2,1) because machine 6 and machine 7 are two adjacent machines and they form one adjacent group.

Variable 5 is 9-dimensional and each dimension refers to a distinct repair strategy (e.g. FIFO, LIFO, ... see Section 4.2). It is transferred from the categorical variable of repair strategy through dummy encoding (Suits, 1957) where the corresponding dimension will be set to '1' when the repair policy is in use. When all the dimensions are zero, it represents SRT (Shortest Repair Time first). Hence, the 9-dimension variable is used for ten repair policies.

To summarise, the variables used to describe the status of the production line are reduced to 5 groups (23 dimensions in all) and the relationship between them is organised better than directly listing the state of each buffer and machine. Among the input variables, one variable represents the situation of the line, three of them describe the attributes of breakdowns and another one is the decision variable.

#### 4.4.2 Experimental Design

In this section, we elaborate on how the data for fitting the metamodel is collected from the simulation model. In the previous Section 4.4.1, we reduced the high dimensional input variable for the simulation used to train the metamodel to a lower dimension with twenty-three elements in five groups. The experiments are designed for this re-formed metamodel input variable space. Since the production line simulation is stochastic, multiple simulation replications are expected to be run at each single data point so that we could have the average performance which alleviates

the stochastic effect on the performance evaluation. For some sample points in the input variable space, more data is needed than others to keep dispersion at a similar level. The unbalanced sampling design could save the computing budget for those points with low variability and make the best use of the computing budget. As a result, the one-shot design is not the best choice since different areas could need different replications in our task and it is not possible to foresee in advance.

Sequential sampling is then adopted instead of a one-shot design to assign need-based unbalanced samples. Under sequential sampling, the sampling procedure is separated into several iterations. The later rounds are designed based on the previous samplings. We develop our own sequential sampling approach for a stochastic system which is inspired by the ideas in the literature. [Eason and Cremaschi \(2014\)](#) introduced and applied their adaptive sequential sampling method to model chemical processes with the neural networks where the authors assume the process they are dealing with is deterministic. Nevertheless, their proposed method is still useful and valuable for bringing multiple criteria into consideration. Sequential sampling is also investigated in some research regarding stochastic kriging ([Ankenman et al., 2010](#)). For example, [Binois et al. \(2019\)](#) takes both exploring and exploiting into consideration in their paper. These methods limit their application to stochastic kriging only but the idea of balancing exploitation and exploration processes is useful for our application.

### **Adaptive Sequential Sampling Method**

The heuristic sampling method applied in the project combines sequential sampling, the space-filling method and explore-exploit balancing to form our adaptive sequential sampling for stochastic models. The whole process consists of a series of iterations which begins with an initial sampling. The sampling budget is a fixed number of simulation runs for both each one of the sampling iterations and the whole process based on the way of transferring the samples in the metamodel input variables to the simulation input variables (Equation 4.1). This means that exploring a new area in the state space and exploiting the sampled area are going to share the sampling budget in each iteration when they are both included in the procedure. A flowchart of the algorithm is given in Figure 4.8. We also attach the pseudocode in Algorithm 3.

The whole sampling procedure is initialised by applying Latin Hypercube Design (LHD) in the metamodel input variable space of V1, V2 and V3 to obtain 20 samples. Each one of the samples is then randomly instantiated to three sets of input for the simulation model (buffer content of every conveyor and the breakdown machines) where each instance is paired with ten repair policies (V5). This comes from the application where repair policies are compared with the system state (V1, V2 and V3). The system state and V5 being crossed variables in the data collection stage is believed to have the benefit of reducing the confounding in the metamodel estimation. By

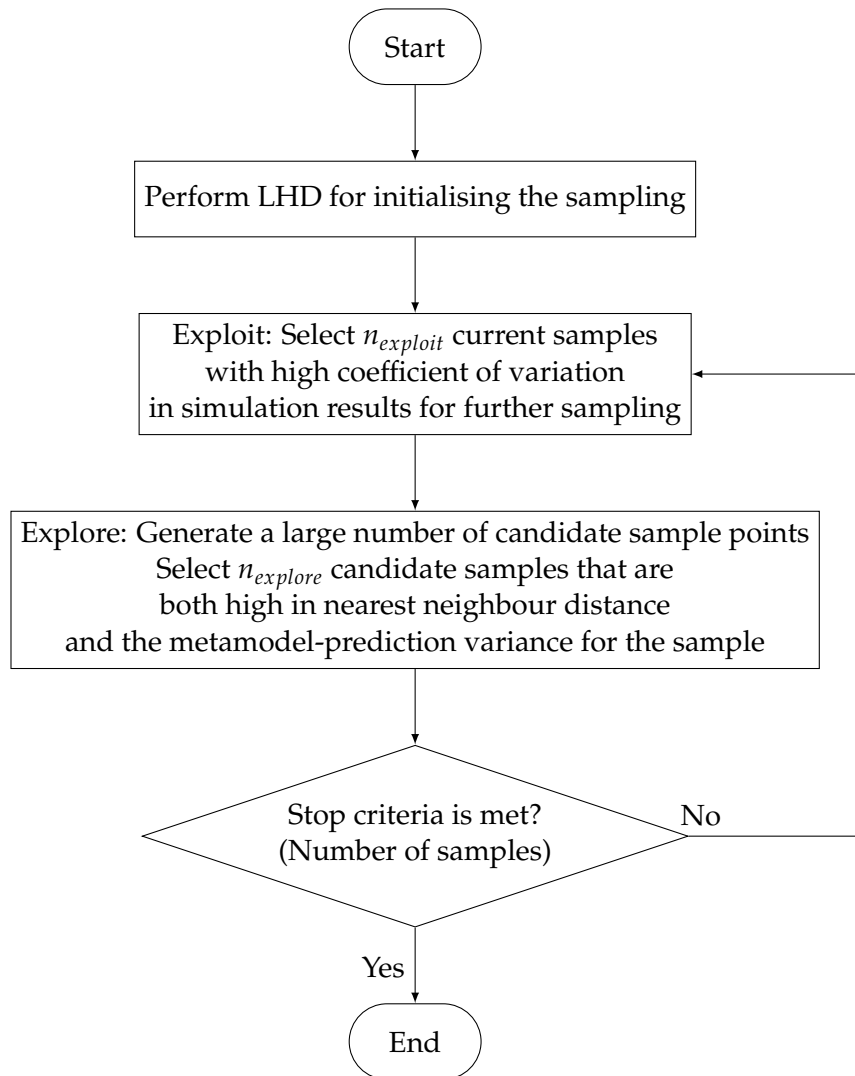


FIGURE 4.8: Flowchart of the Experimental Design

simulating the performances of different repair policies under the same system state and the same random stream, the effects caused by factors other than the decision variable are excluded. We focus on the difference in the performance of the repair policies. Each instance combination for simulation is assigned one replication so every set of metamodel variables ( $V1$ ,  $V2$ ,  $V3$  &  $V5$ ) has three replications.

The variables concerning machine availability ( $V2$  &  $V3$ ) limit the feasibility of the measure of adjacency of breakdowns ( $V4$ ) due to the structure of the production line. So that  $V4$  is not included in the space-filling design. It is generated from the instances of  $V2$  and  $V3$ . A simplified expression of instantiation is given in Equation 4.1 to show how 20 samples are turned into 600 simulation runs in each iteration. (Hence, we are looking for 20 samples in each round of sampling.) The corresponding simulation is carried out following this first round of sampling and they form the initial impression of the model.

$$600 \text{ simulation-budget} \leftarrow 20 \text{ samples} \times 3 \text{ instances} \times 10 \text{ repair policies} \quad (4.1)$$

A neural network model with a single hidden layer is built after each sampling round to check the performance of the model built from the sampled data. The sampling is continued until the size of the data meets the one in ten rule (Peduzzi et al., 1996) which requires data for training to be at least ten times as large as the number of parameters the metamodel has.

---

**Algorithm 3** Proposed algorithm for experimental designs

---

```

1: procedure ADAPTIVE SEQUENTIAL SAMPLING METHOD (N,n)
2:                                     ▷ N: Total samples, n: samples for each iteration
3:   SampleInitial ← Run Latin Hypercube Design                               ▷ initialisation
4:   Collect SampleInitial                                     ▷ instantiate sample and simulate instances
5:   Sampled ← SampleInitial
6:   while len(Sampled) < N do
7:                                     ▷ Exploiting process
8:     NextExploit ← [sample if CV (sample)>0.5 for sample in Sampled] ▷ CV:
     Coefficient of Variation
9:      $n_{exploit} \leftarrow \text{len}(\text{NextExploit})$ 
10:                                     ▷ Exploring process
11:      $n_{explore} \leftarrow n - n_{exploit}$ 
12:     Randomly Generate a large number of ( $L$ ) NextExploreCandidate
13:     NN[l] ← the distance of NextExploreCandidate[l] and its nearest neighbour
     in Sampled
14:     Build 10 models from Sampled with jackknife resampling method
15:     variances = [ ]
16:     for candidate in NextExploreCandidate do
17:       Estimates = [Estimate candidate with 10 models]
18:       Append variance of Estimates to variances
19:     end for
20:     score ← [ NN[l]/Max(NN)+variances[l]/Max(variances) for l in  $\mathcal{L}$  ]
21:                                     ▷ sum the nearest neighbour distance and the estimation variance
22:     NextExplore ← [candidate in NextExploreCandidate if its score is top
     n_explore high]
23:     Collect (NextExploit + NextExplore)
24:     textbfAppendSampled NextExploit + NextExplore to Sampled
25:   end while
26: end procedure

```

---

In the following iterations, data sampling would be implemented from an exploiting

process and an exploring process. In the process of exploiting, the points that have already been sampled and have a large coefficient of variation are selected and receive more replications, while in the process of exploring, we search for new positions for sampling. The computing budget  $n$  for each iteration is fixed at 20 samples, which could turn into 600 simulation runs. In each iteration of the sampling procedure, the exploitation is carried out first and the rest of the budget would be spent on exploration.

For the process of exploiting, we first group the sampled data points by the metamodel input so that the samples in the same group will have the same input variables (V1, V2, V3, V4 and V5). The Coefficient of Variation ( $CV = \frac{\text{standard deviation}}{\text{mean}}$ ) is computed for each group over the simulated throughput. The coefficient of variation is also known as the relative standard deviation. Those groups having a large CV receive one more round of instantiation for the sample combination (V1, V2 and V3), which means each of them will be used to generate another three instances for simulation inputs.

The criterion of CV is determined by the simulation data that is collected to form a gold standard for the metamodel. The gold standard is prepared for being the test set for the metamodel. It includes one hundred data points in the metamodel input space and each one has got one hundred replications (which is 10,000 simulation runs in total). According to the analysis of the gold standard, the average CV for the same set of inputs is 0.298. Based on this, the criterion of CV for re-instantiating the sample is set to 0.5. It is slightly larger than the average of the gold standard because a much smaller replication per sample than the gold standard is expected in the experimental design. If the value is too small, the budget left for exploring would shrink. If it is too large, we will lose the desired control of data dispersion. The CV needs to be carefully selected and catered to the target model and data.

A higher CV means the data spread in a wider range, hence the system is more stochastic in that area. More efforts (replications) are then given to exploit these areas with higher stochasticity. In the new round, each sample for exploiting will have another three instances, where the instances and the repair policies (V5) are crossed (Equation 4.2).

$$\text{Exploiting budget} \leftarrow n_{\text{exploit}} \text{ samples} \times 3 \text{ instances} \times 10 \text{ repair policies} \quad (4.2)$$

After that, we move on to the exploring part with the rest sample budget (Equation 4.3).

$$n_{\text{explore}} = n \text{ samples} - n_{\text{exploit}} \quad (4.3)$$

where  $n$  is the sampling budget in each iteration (Algorithm 3). The samples for exploration will go through the same steps as the samples for exploitation before getting simulated (Equation 4.4).

$$\text{Exploring budget} \leftarrow n_{\text{explore}} \text{ samples} \times 3 \text{ instances} \times 10 \text{ repair policies} \quad (4.4)$$

$$600 \text{ simulation-budget} \leftarrow \text{Exploiting-budget} + \text{Exploring-budget} \quad (4.5)$$

In order to obtain the samples for exploration, we first need to randomly generate a large number ( $L$ , e.g. 500) of candidate samples where the samples for exploration will be selected. Two criteria are applied to decide which points would get sampled. One is the distance between the candidate point and the sampled points, the other one is the metamodel-prediction variance. This part of the sampling method is termed the mixed adaptive sampling algorithm in the literature (Eason and Cremaschi, 2014).

The first criterion is the nearest neighbour distance ( $d_l$ ). For each candidate point, the Euclidean distances between itself and all the points in the existing data set are calculated, the minimum of them is the nearest neighbour distance of the candidate point and the existing samples.

The metamodel-prediction variance (Kleijnen and Beers, 2004) is evaluated through the jackknife resampling method. The jackknife method is a statistical technique that utilises subsamples to estimate bias and variance.  $K$  subsamples are obtained while leaving  $K$  equal-size disjoint sets out. Each subsample is  $(K-1)/K$  of the size of the parent set. We apply the jackknife resampling to the collected data and train one surrogate model for each of the  $K$  subsamples. The surrogate model is a neural network model with a single hidden layer that has the same size as the input layer. The surrogates have Gelu activation functions (Hendrycks and Gimpel, 2016) and are trained using Adam optimiser (Kingma and Ba, 2015) with a step size of 0.001 for 1000 epochs. From this, we would obtain  $K$  surrogates. These surrogates have the same function as the final metamodel which predicts the throughput of the production line. The candidate samples are sent to  $K$  surrogates and receive  $K$  throughput estimations. Variance ( $\hat{\delta}_l^2$ ) for each candidate among these  $K$  estimations is calculated, which is an estimation for the variance of the sampled data (collected data) at the given candidate sample.

The candidate sample which has a higher prediction variance sample would have a higher priority to be simulated for data collection. The prediction variances ( $\hat{\delta}_l^2$ ) would decline after the corresponding design space has been thoroughly explored.

$$\eta_l = \frac{d_l}{\max_{l \in \mathcal{L}}(d_l)} + \frac{\hat{\delta}_l^2}{\max_{l \in \mathcal{L}}(\hat{\delta}_l^2)}, \text{ for } l \in \mathcal{L} = \{1, \dots, l, \dots, L\}. \quad (4.6)$$

An indicator  $\eta_l$  is used to combine the criteria and measure the priority for candidate sample points (Equation 4.6). For each candidate sample point  $l$  in the  $L$ -member candidate set  $\mathcal{L}$ ,  $d_l$  is the nearest neighbour distance and  $\hat{\delta}_l^2$  is the estimated variance of metamodel-prediction. The indicator  $\eta_l$  sums the normalised nearest neighbour distance and the normalised estimated predictor variance. Therefore, the range of the indicator is constrained to  $0 \leq \eta_l \leq 2$ . The samples have  $\eta_l$  closer to zero shows that these points are low in both nearest neighbour distance ( $d_l$ ) and metamodel-prediction variance ( $\hat{\delta}_l^2$ ), thus they have lower priority for sampling. In contrast, the points with high  $\eta_l$  that is closer or equal to two are high in both nearest neighbour distance ( $d_l$ ) and metamodel-prediction variance ( $\hat{\delta}_l^2$ ). These points are assigned high priority to get sampled. The  $n_{explore}$  candidate points with the largest  $\eta_l$  get sampled. The samples for exploitation and exploration are instantiated and simulated. This is one iteration of the adaptive sequential sampling. The process is repeated until the sampling budgets are exhausted.

### 4.4.3 Test Set Preparation

The metamodel will be applied to rank the candidate repair policies and guide the search with the high-fidelity simulation model. Ten test sets are separately designed and collected in order to mimic the application scenario and to form a gold standard. Each test set begins with sampling from the metamodel input variable space (i.e. V1, V2, V3) randomly, the test set is then translated to an input instance for simulation (buffer content of every conveyor and the breakdown machines) randomly. For example, each dimension of V1 would be randomly mapped to a six-value set standing for the levels of six in-section buffers with its summation equal to the corresponding value in V1. The process of preparing the test sets is followed by combining with ten repair policies after which, 100 replications are assigned to each combination (Equation 4.7). The whole process of obtaining the test set is similar to the data collection for training or fitting the metamodel. Besides, more replications are applied to single instances for the gold standard in order to have a stable and converged result for the stochastic system. The test sets allow us not only to check the error between predictions and true values (for example, mean absolute error (MAE) or root mean square error (RMSE)) but also enable the possibility of verifying the ranking capability of the metamodel. Also, the test sets are applied to check how the multi-fidelity simulation optimisation framework works for the production line. The sample in Equation 4.7 refers to the randomly sampled metamodel input.



10 test sets  $\leftarrow$  10 samples  $\times$  (1 instances  $\times$  10 repair policies  $\times$  100 replications) (4.7)

The V1 (buffer states) samples for ten test sets are visualised in Figure 4.9. Four plots are made for four sections of the production line which are marked as V11, V12, V13 and V14 respectively. Each section could have the sum of its buffer content between 0 and 60 (x-axis). Bars are plotted in various colours for different test sets. Number labels are attached to the right of the bars for the sampled buffer levels. The y-axis shows the identification number of the test set. By putting the plots horizontally, the buffer levels of test sets are displayed as they would be on the production line.

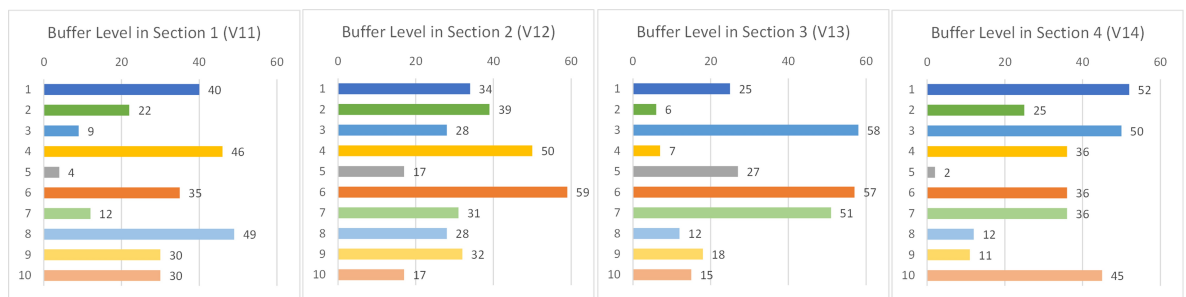


FIGURE 4.9: Production line section buffer level (V1) of 10 test sets

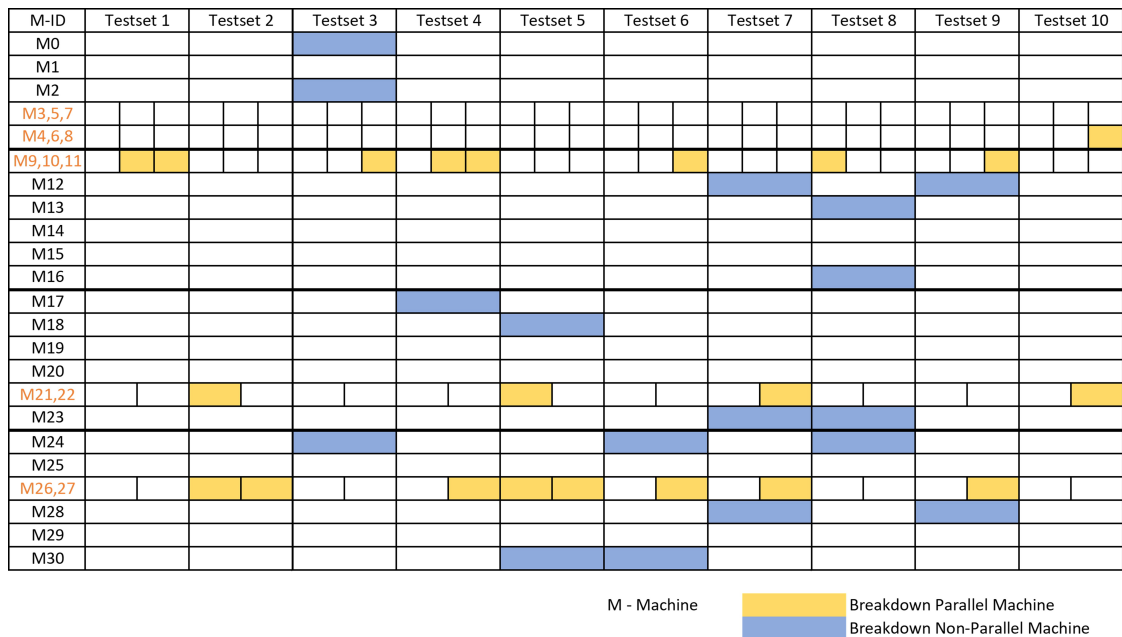


FIGURE 4.10: Breakdowns (instances of V2 & V3) of 10 test sets

The breakdown instances of the V2 & V3 for test sets are given in a grid table (Figure 4.10). The leftmost column contains the machine identification numbers. Each column shows the machine availability in the test sets. The cells that correspond to the broken-down machines are filled with colours. The blue cells are for the non-parallel

broken machines and the yellow ones are for the parallel broken machines. Additionally, the stations that are composed of parallel machines are arranged in a single row. Three thickened lines (below M4,6,8, M16 and M23 respectively) divide the machines into four parts according to the variable design. The instances of the breakdown states are plotted. While this allows us to observe the availability of parallel and single stations, it also enables us to demonstrate the adjacency of breakdowns (V4) visually. For example, in test set 1, M10 and M11 are adjacent breakdowns and in test set 7, M22 and M23 are adjacent breakdowns.

TABLE 4.7: Average throughput of the collected data for test

Rank \ Test set	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	9 <sup>th</sup>	10 <sup>th</sup>
1	320.98 LIFO	316.18 NOSQ	315.94 FIFO	315.75 AP	315.67 MOR	315.63 SRT	315.59 COLQ	315.05 NAP	315.03 FOR	314.66 LRT
2	260.87 AP	259.55 FOR	259.49 COLQ	256.7 FIFO	254.97 SRT	249.15 LIFO	236.92 NOSQ	232.49 LRT	225.68 NAP	221.93 MOR
3	278.24 AP	276.95 FOR	276.23 NAP	264.53 SRT	261.38 LIFO	260.81 COLQ	259.94 FIFO	243.06 NOSQ	242.73 LRT	239.77 MOR
4	280.69 NOSQ	274.4 AP	273.12 COLQ	271.73 SRT	271.69 NAP	271.03 FIFO	270.62 LRT	270.3 FOR	269.61 MOR	269.41 LIFO
5	171.13 AP	169.7 FOR	169.61 NOSQ	163.4 COLQ	159.85 SRT	158.8 FIFO	135.8 LRT	131.85 LIFO	121.07 NAP	108.88 MOR
6	270.5 SRT	269.69 NAP	269.39 COLQ	267.5 FIFO	266.35 MOR	265.71 LRT	264.85 LIFO	258.48 AP	258.47 NOSQ	255.3 FOR
7	243.65 COLQ	243.17 AP	240.38 NAP	240.09 FOR	234.43 SRT	229.2 FIFO	227.68 LIFO	225.51 NOSQ	224.46 LRT	219.03 MOR
8	177.66 COLQ	174.66 SRT	174.49 LRT	174.36 MOR	173.93 NOSQ	172.78 LIFO	172.73 AP	171.62 FOR	170.25 FIFO	169.77 NAP
9	246.62 COLQ	244.35 AP	243.88 LIFO	243.05 SRT	242.5 LRT	242.41 FIFO	242.13 NOSQ	241.72 NAP	241.5 MOR	241.04 FOR
10	276.24 SRT	270.76 MOR	269.72 NOSQ	268.25 COLQ	266.5 FIFO	265.16 LIFO	264.7 AP	260.99 LRT	257.53 FOR	256.27 NAP

The average throughput (Table 4.7) is computed from 1000 replications for every test set, where each repair policy gets an allocation of 100 replications. The highest average value of ten repair policies is the optimal solution and is regarded as the 'true best'. Each row in the table shows the result for one test set and each column is the ranked repair policy and its average throughput. The test sets are treated as a 'gold standard' since they consist of a lot more data than the data for fitting the metamodel or the simulations made during the optimisation.

#### 4.4.4 Fitting the Metamodel

The metamodel projects the modified state of a manufacturing system to its corresponding expectation of the throughput. We chose to use the neural network regression model for the metamodel because it performed well in the inventory system example. The production line model is also a non-linear system but comes with higher dimensions, larger variable spaces, more stochastic processes (which include machine operating time, the time between failures and time to repair) and

higher variability in the output. The neural network regression model has the flexibility in its model structure to cope with our task.

Following the design of experiments in the previous Section 4.4.2, we simulate the specified instances of the samples. The simulated throughputs are averaged over samples to obtain their expectations before being sent to train the metamodel.

The stopping criterion of the experimental design is the number of samples reaching ten times the number of parameters in the metamodel. We assume the model to be a neural network with a single hidden layer and the hidden layer has the same size (number of hidden neurons) as the input layer. The model has one output neuron for the expected throughput. The input layer has 23 neurons to accept data from five groups of input variables. This single-layer neural network would have  $23 \times 23 + 23 \times 1 = 552$  parameters and consequently, we need at least  $552 \times 10 = 5520$  samples to fit the metamodel. In the experiments, it took 35 iterations of sequential sampling to attain 6,910 samples from 21,000 simulations.

The model is tuned to minimise the prediction error (mean absolute error) through grid search. The final neural network has one hidden layer with 15 neurons. The activation function is Gelu (Gaussian Error Linear Unit) (Hendrycks and Gimpel, 2016) which has been used in the inventory system example. The metamodel is trained using Adam optimiser (Kingma and Ba, 2015) with a step size of 0.0005 and 7000 epochs. The metamodel is tested on the designated ten test sets described in Section 4.4.3. Metamodel accuracy and ranking relevant metrics are utilised to evaluate the metamodel and are presented in the following paragraphs.

The mean absolute error between the metamodel predictions and the simulations is 21.92. The error is 9.05% of the average throughput as the average throughput for all sampled test sets is 242.26. The overall ranking capability of the metamodel is measured with the ranking correlation coefficient Kendall's tau following its use in the previous chapter. Although the ranking correlation coefficient is useful, it does not reflect the quality of ranking in the specific area (e.g. the top solutions). The high-rank area is where more effort is put in by the aggression parameter and should receive emphasis.

Therefore, three other metrics are designed in addition to Kendall's tau. They are 'Top3\_captured', 'top1rank' and 'actual\_rank\_pred'. 'Top3\_captured' indicates how many true top-3 solutions from the test set are captured by the metamodel. It could be a value between zero and three, where zero is the situation that none of the true top three solutions is ranked top three by the metamodel. 'Top1rank' is the rank of the true best solution by the metamodel. 'Actual\_rank' is the true rank of the repair policy that is predicted the best by the metamodel. For example, in test set 1 (Table 4.8), one of the true top-3 solutions in the 'gold standard' is estimated to have top-3 performance by the metamodel. The solution that performs the best in the 'gold

standard' ranks nine among the metamodel estimations. And the best solution from the metamodel estimation ranks seven in the 'gold standard'.

TABLE 4.8: Metrics recorded for the metamodel

	Testset 1	Testset 2	Testset 3	Testset 4	Testset 5	
Kendal's tau	0.155	0.644	0.466	0.644	0.555	
top3_captured	1	2	1	2	2	
top1rank	9	1	1	2	1	
actual_rank	7	1	1	2	1	
	Testset 6	Testset 7	Testset 8	Testset 9	Testset 10	Average
Kendal's tau	-0.155	0.511	0.244	0.155	0.422	0.3641
top3_captured	1	1	2	1	1	1.4
top1rank	5	4	2	5	1	3.1
actual_rank	8	2	2	2	1	2.7

The performance of the metamodel for ten test sets on these three metrics is recorded in Table 4.8 with their average values calculated. The performance of the metamodel is shown very differently among the ten test sets. The average Kendall's tau is 0.36 which is not high. This could be caused by having extremely low values in some tests, such as test set 6 (-0.155), test set 1 (0.155), and test set 9 (0.155). Four tests have Kendall's tau over 0.5, they are test set 2, set 4, set 5 and set 7. The average of the top3\_captured is 1.4 which is close to the median (1.5). At least one of the true top-3 solution got captured by the metamodel prediction and four of them (test set 2, 4, 5 and 8) caught two of the top-3 solutions. From the third and the fourth row, we can find that test 2, test 3, test 5 and test 10 have both values found 1 which indicates the metamodel correctly predicts the true best for these four test sets. From 'actual\_rank', four other tests rank the second-best solution as 'the best' repair policy. The metamodel average predicts the solution that ranks 2.7 as the best from the metamodel in ten tests.

We find that the performance of the metamodel in terms of ranking capability in the high-rank area is acceptable although the overall ranking correlation coefficient Kendall's tau is not satisfactorily high. According to the discussion of the metamodel evaluation with the proposed metrics, the fitted neural network regression metamodel is believed to be able to work well with the multi-fidelity simulation optimisation framework.

## 4.5 Summary

In this chapter, we describe the production line simulation model and the construction of a neural network regression model as a metamodel for the simulation. The models are structured to work with the multi-fidelity simulation optimisation framework so

that we can optimise the repair order in near real-time and mitigate the effect of disruption on the system throughput as we discuss in the next chapter.

The main contributions of this chapter are proposing an approach to building a metamodel for the production line simulation model and the experiment design developed for fitting the metamodel. A secondary contribution is tailoring metrics for evaluating the performance of metamodel for a multi-fidelity optimisation framework.

The repair policy is included in the simulation model to order the machines that are waiting for the repair. We reduce the dimensions of metamodel inputs by carefully selecting the input variables. This is achieved by using variables relating to different sections of the line rather than each machine and by considering what features of a breakdown are important in governing subsequent behaviour. Fewer experiments are thus required to cover the feasible space and fit the metamodel.

An adaptive sequential sampling method is proposed to design experiments for fitting the metamodel based on the simulation output. The uneven sampling balances the need to exploit the sampled area and exploring the new region so that it makes the best use of the limited computing budget.

The metamodel is evaluated via estimates of prediction error and metrics that measure the accuracy of the ranking of repair strategies. The metamodel is tested on ten designated test sets. These test sets are prepared to offer a 'gold standard', which will be used for testing the optimisation as well.

The analysis suggests that the metamodel produces reasonable predictions and rankings for the repair policies. The contributions in this chapter bridge the multi-fidelity optimisation algorithm and the scenario for industrial application.



## Chapter 5

# Real-Time Optimisation of the Production Line

In this chapter, we describe an application of the multi-fidelity simulation optimisation (MFSO) algorithm to the production line model. The algorithm aims to find the optimal repair policy when experiencing multiple machines breakdown simultaneously. The optimal repair policy is defined as the policy that when applied results in the highest expected throughput in the next three hours. We intend to find the optimal solution in a comparably short time so that it is possible to support real-time on-site decision-making. When the repair requests are raised, it takes time to assign the task, collect the tools and move to the site before the actual repair could happen. In the meantime, the optimisation algorithm could be carried out to offer suggestions on the ordering of the repairs. The maximum duration of the optimisation algorithm is thus defined as 10 minutes in the experiments to fit the scenario.

The optimal repair policy is selected from ten candidate repair policies as described in Section 4.3. This is a slightly different problem from the inventory system example discussed in Section 3.4. First, the numbers of candidate solutions are on very different scales. The inventory system example has more than 10,000 solutions while the production line example has only 10 repair policies to choose from. Although the MFSO framework is able to work with a large number of alternatives, reducing the state space reduces the number of input variables for the metamodel. This reduces the data required for building the metamodel. This is essential especially when the model is complex, large and slow to run. Besides, the production line model describes a real-life system that is more complicated than the inventory model, which means that it takes longer to run one replication. When designing a metamodel, an additional difficulty is that the current state of the system is more complex to define. Last, but not least, one suggestion is given out by the optimisation algorithm in this repair policy selection problem while top-five solutions are suggested in the inventory system

example. Consequently, the multi-fidelity simulation optimisation framework proposed in Chapter 3.4 is altered to adapt to the application scenario as described in the next section. This also shows the flexibility of the framework.

## 5.1 Multi-fidelity Framework

The optimisation algorithm introduced in Section 3.1 is revised in order to adapt to the production line repair policy selection problem. The idea behind the algorithm remains the same, which is that high-ranked solutions according to the metamodel estimations would have more chances to be sampled for simulation in the online optimisation stage. As there are only ten alternatives to choose between, the clustering of solutions step is removed and we group alternatives directly based on the metamodel estimates of throughput rank. The flowchart of the modified algorithm is shown in Figure 5.1.

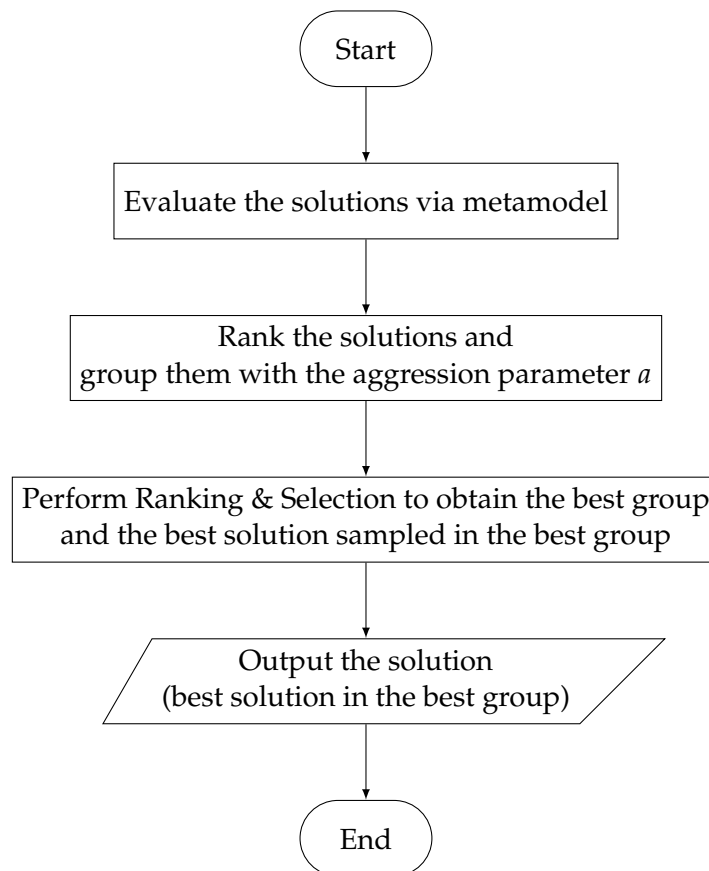


FIGURE 5.1: Flowchart of the Multi-fidelity Simulation Optimisation Framework returning single solution

The solution that has the highest average simulation output from the best group obtained from the ranking and selection procedure is selected as the final result of our multi-fidelity simulation optimisation (MFSO) framework.



The repair policies are grouped according to the ranks of predicted throughput. The ratio of group members follows the rule ( $1^a : 2^a : 3^a \dots$ ) with the aggression parameter  $a$  where  $a$  is a natural number. The optimisation algorithm with three possible settings of the aggression parameter, 1, 2 and 3, is tested. The exact number of solutions in groups while applying the aggression parameter to ten repair policies are given in Table 5.1.

TABLE 5.1: Structure of the groups

Group	No. of solutions (a=1)	No. of solutions (a=2)	No. of solutions (a=3)
1	1	1	1
2	2	$2^2=4$	$2^3+1=9$
3	3	5	
4	4		

The first thing we notice in Table 5.1 is that the numbers of groups are not fixed for different  $a$  in this case. Secondly, the solutions are not always able to be grouped strictly following the proposed ratio since there are only ten repair policies. In the second column of the table, the solutions are divided into four groups and fit well with the grouping ratio when  $a = 1$ . While in the third column ( $a = 2$ ), the third group has only five solutions when we have a larger group 2. The size of the second and the third groups does not meet the expectation under this aggression parameter setting as these two groups should have a larger difference than the one between the first and the second group. The leftover solutions for the last group are merged with the previous group when the leftover solutions are less than the solutions in the previous group. It would violate the design of increasing the group size while the ranks of solutions rise if the last solution is placed into an individual group instead of being merged. For example, the last solution is merged into group two in the column  $a = 3$  (Table 5.1).

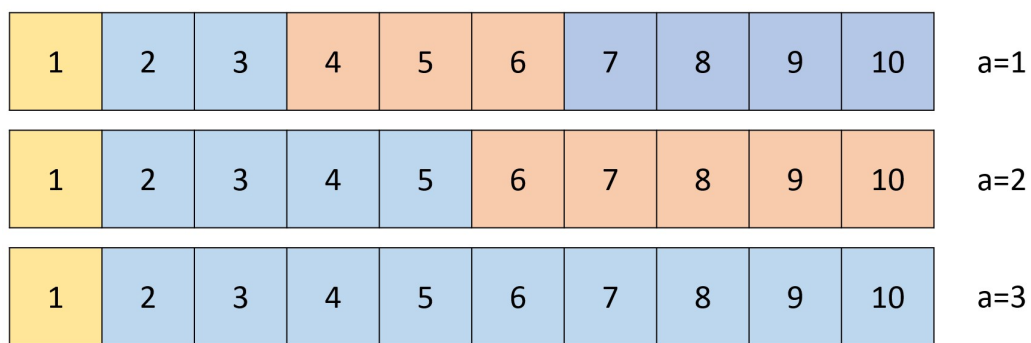


FIGURE 5.2: Illustration of aggression parameter for 10 repair policies

Three sets of grouping rules (aggression parameter  $a = 1, 2, 3$ ) for ten solutions are also visualised in Figure 5.2. Three rows show the groupings under three aggression parameters. The numbers in the figure represent the ranks of repair policies based on the metamodel predictions. Solutions in different groups are coloured differently. The

difference in group sizes is meant to increase. However, the difference may not be obvious due to the limited total solution number (e.g. group 2 and group 3 for  $a = 2$ ).

When multiple machines are broken down on the production line, we first evaluate the situation (V1, V2, V3 & V4 (Section 4.4.1)) with ten candidate repair policies (RPs; V5) using the metamodel (Section 4.4.4) to estimate the expectation of the system throughput in three hours. The RPs are then ranked according to the predicted throughput and sorted into fewer groups following the strategies with the selected aggression parameter  $a$ . After that, a ranking and selection (R&S) method is applied to these groups and each group is treated as a single solution. OCBA (Optimal Computing Budget Allocation) (Section 3.3) is the R&S procedure used in the experiments with the production line simulation model like in the inventory system example. Replications are allocated to the groups based on the average performance of the solutions sampled in each group. Next, a solution from that selected group is randomly sampled to be simulated. The ranking and selection procedure will give out the group that has the highest average in-group sample performance when the online computing budget is exhausted. Finally, the solution that has the largest mean value of the simulation outputs in that selected group is returned as the final result of the optimisation (Figure 5.1).

We set the null hypothesis  $H_0$  to be that the average throughput for the repair policy obtained by the proposed MFSO framework is not different from the average throughput from the true optimum for the system, modelled here by the gold standard. The averages are taken over the subsequent three hours. This can be written as  $\mu_d = 0$ . The alternative hypothesis  $H_1$  is that the true difference between the throughputs is not equal to zero:  $\mu_d \neq 0$ .

## 5.2 Numeric Experiments

In this section, we compare approaches for selecting the optimal repair policy for the production line model. The multi-fidelity simulation optimisation framework is hence compared with the default policy, FIFO (First in first out), which is regarded as no optimisation, the uniform assignment (UA) where all the online simulation budget is uniformly assigned to each repair policy and the AP (Adjacent Process first) policy which is the most frequently best-performing repair policy (more explanation in the following paragraph). We also compare these results with making a decision directly from the metamodel estimates.

The repair policy AP is the strategy that most frequently outperforms the other repair policies in the collected simulation data for fitting the metamodel (Figure 5.3). When the instantiated sample is simulated with ten repair policies, the one that performs the best with the highest throughput is recorded. In the figure, we can see that, in addition

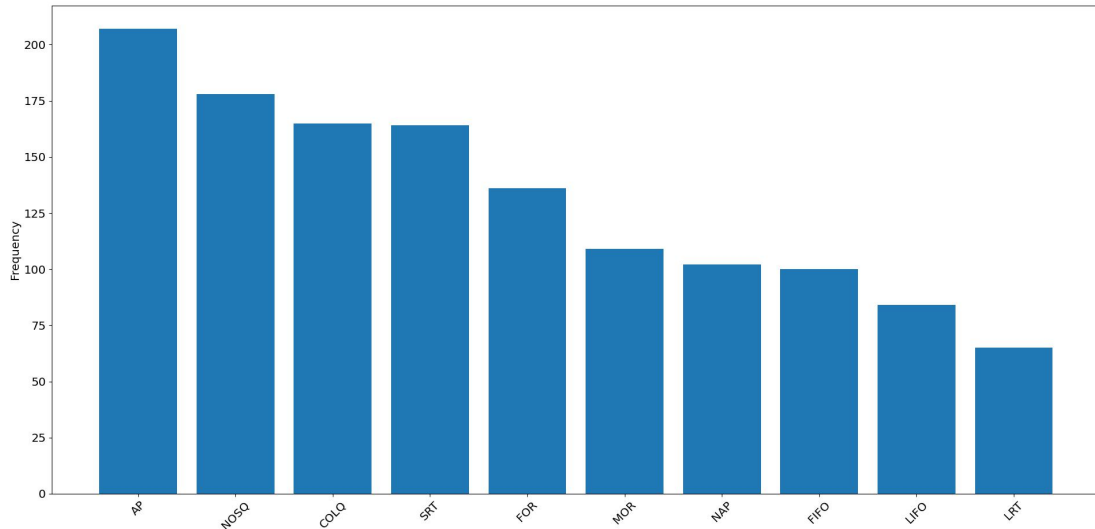


FIGURE 5.3: Histogram for the situation when the repair policy performs the best from the collected simulation data

to AP, the queue-length-based repair policies (i.e. NOSQ (Next Operation with Shortest Queue first) and COLQ (Current Operation with Longest Queue first)) are also performing well, while LIFO (Last In First Out) and LRT (Longest Repair Time first) have the lowest chance of being the best strategy. All-time AP policy is thus compared against other methods.

The algorithms that have an online simulation optimisation stage are tested with a small amount of budget since we are aiming to support real-time or near-real-time decision-making. Two online budgets are tested in the experiments, they are 50 simulation-run and 100 simulation-run for the whole procedure. They take roughly five and ten minutes respectively to work with the production line model. Under each of the two different budgets, the MFSO algorithm and the UA algorithm are carried out to select the optimal repair policy. Since the MFSO and UA incorporate some randomness in how they choose the optimal strategy, they may not produce the same result every time. In order to obtain a stable performance of the algorithms, they are repeated ten times for each of the ten test sets and the average result is reported.

The result (optimal repair policy) obtained from the trial  $m$  of the optimisation algorithm for test set  $t$  is measured by its performance in the ‘gold standard’ which is obtained from the  $N$ -replication test sets. The best solution ( $b$ ) in the ‘gold standard’ is regarded as the optimal solution. In order to compare the repair policies with the optimum, we first calculate the elementwise difference  $d_{tmm}$  to the best from all  $N$  replications.

Let  $M = 10$  ( $m = 1, \dots, M$ ) be the number of times we run the process for each test set. We assume that we have  $T$  testsets,  $t = 1, \dots, T$  and for each we run  $N$  replications,  $n = 1, \dots, N$ . Let  $N = 100$  be the number of replications in the gold standard, and

$T = 10$  be the number of test sets. And let  $d_{tnm}$  be the difference between the output of the chosen strategy  $y_{tnm}$  and the best strategy  $y_{tnb}$  on the  $m^{th}$  trial with the test set  $t$  on the  $n^{th}$  replication of the 'gold standard'.

$$d_{tnm} = y_{tnb} - y_{tnm} . \quad (5.1)$$

As the performances of repair policies are compared,  $d_{tnm}$  could be negative since  $y_{tnb}$  is not guaranteed to be larger than  $y_{tnm}$  in every replication  $n$ . The difference between the output of the chosen strategy and the best strategy on the  $m^{th}$  trial with the test set  $t$  obtained from the algorithm is assumed to follow a normal distribution

$D_m \sim N(\mu_m, \sigma_m^2)$ , where

$$\mu_m = \frac{1}{N} \sum_{n=1}^N d_{tnm} , \quad (5.2)$$

$$\sigma_m^2 = \frac{1}{N-1} \sum_{n=1}^N (d_{tnm} - \mu_m)^2 . \quad (5.3)$$

The obtained mean ( $\mu_m$ ) and variance ( $\sigma_m^2$ ) of the difference between the optimal repair policy and other alternatives for all ten tests can be found in Appendix A. We find that alternative repair policies could behave very differently in the randomly generated tests. The difference in throughput between the best and worst strategy could be as small as six in test 1 or as large as sixty in test 5. Thus the alternatives that have better performance (higher throughput) might be easier to identify in test 5 than in test 1.

Then, the confidence interval (Equation 5.4) is computed for  $D_m$

$$CI_m = \mu_m \pm z_{1-\alpha/2} \sqrt{\frac{\sigma_m^2}{N}} . \quad (5.4)$$

We calculate the 95% ( $\alpha = 0.05$ ) confidence interval of the solution for  $T$  test sets on  $M$  trials using the  $N$ -replication simulation data. The obtained repair policy in  $m^{th}$  trial is not statistically different from the optimum in test  $t$  if  $CI_m$  contains zero in the range of the confidence interval.

The optimisation results for the production line on one of  $M$  trials are shown in Table 5.2. Their specific confidence intervals are documented in Table 5.3. The full results of the tests are available in Appendix B. The first three methods in Table 5.2 and 5.3 (i.e. FIFO, AP and metamodel) do not involve multiple trials as they do not have an online simulation stage in the process and the results are deterministic. The optimal repair policy obtained by MFSO and UA varies for different trials and they are not guaranteed to be the best.

	Testset 1	Testset 2	Testset 3	Testset 4	Testset 5	Testset 6	Testset 7	Testset 8	Testset 9	Testset 10
Gold Standard	AP	AP	AP	NOSQ	AP	SRT	COLQ	COLQ	COLQ	SRT
FIFO (Default)	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO
AP	AP	AP	AP	AP	AP	AP	AP	AP	AP	AP
Metamodel	COLQ	AP	AP	AP	AP	AP	AP	SRT	AP	SRT
100 Sim Run										
UA	FIFO	FIFO	FOR	FIFO	FIFO	FIFO	FOR	COLQ	FOR	FIFO
MFSO a →1	NOSQ	FOR	AP	LRT	AP	AP	AP	COLQ	MOR	SRT
MFSO a →2	SRT	AP	AP	AP	AP	AP	COLQ	FOR	LIFO	LRT
MFSO a →3	COLQ	AP	AP	NOSQ	AP	COLQ	AP	SRT	AP	NAP
50 Sim Run										
UA	AP	AP	AP	AP	LIFO	LIFO	AP	AP	AP	AP
MFSO a →1	FIFO	AP	AP	AP	AP	COLQ	AP	LIFO	AP	COLQ
MFSO a →2	LRT	NOSQ	AP	AP	AP	AP	AP	SRT	AP	SRT
MFSO a →3	AP	AP	AP	AP	FOR	COLQ	AP	SRT	MOR	SRT

TABLE 5.2: Optimisation result for the production line model on one trial (yellow cells capture the optimal repair policies and the light blue cells capture the repair policies that are not significantly different from the optimum)

Each row in Table 5.2 shows the results for one method and each column contains the results for one of the ten prepared test sets. Table 5.2 is split into three parts by the horizontal lines below the ‘gold standard’. From the top to the bottom of the table, they are the records for non-online-optimisation methods and the records for the methods with online simulation optimisations with 100 and 50 simulation runs respectively. For the first group, we have the default first in first out (FIFO) rule which is the situation where no optimisation is involved in the repair process. The AP (Adjacent Process first) is the repair policy that has the highest frequency to perform the best in the data collected from the simulation (Figure 5.3). The performance of optimisation solely with metamodel prediction is on the third row. For the other two groups, each has the same four methods: the uniform allocation (UA) and the proposed MFSO framework with three aggression parameters  $a = 1, 2$  & 3.

	Testset 1	Testset 2	Testset 3	Testset 4	Testset 5	Testset 6	Testset 7	Testset 8	Testset 9	Testset 10
FIFO (Default)	5.04 ±3.439	4.17 ±4.689	18.3 ±5.142	9.66 ±4.51	12.33 ±6.445	3 ±3.221	14.45 ±5.694	7.41 ±4.519	4.21 ±3.48	9.74 ±4.71
AP	5.23 ±3.443	0	0	6.29 ±5.357	0	12.02 ±4.743	0.48 ±6.163	4.93 ±4.592	2.27 ±3.97	11.54 ±5.049
Metamodel	5.39 ±3.814	0	0	6.29 ±5.357	0	12.02 ±4.732	0.48 ±6.163	3 ±4.385	2.27 ±3.97	0
100 Sim Run										
UA	5.04 ±3.439	4.17 ±4.689	1.29 ±2.292	9.66 ±4.51	12.33 ±6.445	3 ±3.221	3.56 ±5.442	0	5.58 ±5.796	9.74 ±4.71
MFSO a →1	4.8 ±3.554	1.32 ±4.727	0	10.7 ±5.293	0	12.02 ±4.732	0.48 ±6.163	0	5.12 ±5.349	0
MFSO a →2	5.35 ±3.841	0	0	6.29 ±5.357	0	12.02 ±4.743	0	6.04 ±4.758	2.74 ±6.338	15.25 ±7.099
MFSO a →3	5.39 ±3.814	0	0	0	0	1.11 ±2.517	0.48 ±6.163	3 ±4.385	2.27 ±3.97	19.97 ±5.831
50 Sim Run										
UA	5.23 ±3.443	0	0	6.29 ±5.357	39.28 ±10.868	5.65 ±3.813	0.48 ±6.163	4.93 ±4.592	2.27 ±3.97	11.54 ±5.049
MFSO a →1	5.04 ±3.439	0	0	6.29 ±5.357	0	1.11 ±2.517	0.48 ±6.163	4.88 ±5.642	2.27 ±3.97	7.99 ±4.501
MFSO a →2	6.32 ±3.902	23.95 ±6.26	0	6.29 ±5.357	0	12.02 ±4.732	0.48 ±6.163	3 ±4.385	2.27 ±3.97	0
MFSO a →3	5.23 ±3.443	0	0	6.29 ±5.357	1.43 ±4.807	1.11 ±2.517	0.48 ±6.163	3 ±4.385	5.12 ±5.349	0

TABLE 5.3: Difference between selected solution and optimum on one trial

Some cells in Table 5.2 are filled with colour. The cells are highlighted in yellow when the methods give out the true optimum for the corresponding test. The light blue cells are the test results that do not match the true optima but have no significant difference from the optimal repair policies. The yellow cells have their corresponding cells in Table 5.3 zero ( $CI_m = 0$ ) and the corresponding cells of blue cells in the same table contain zero in the confidence intervals.

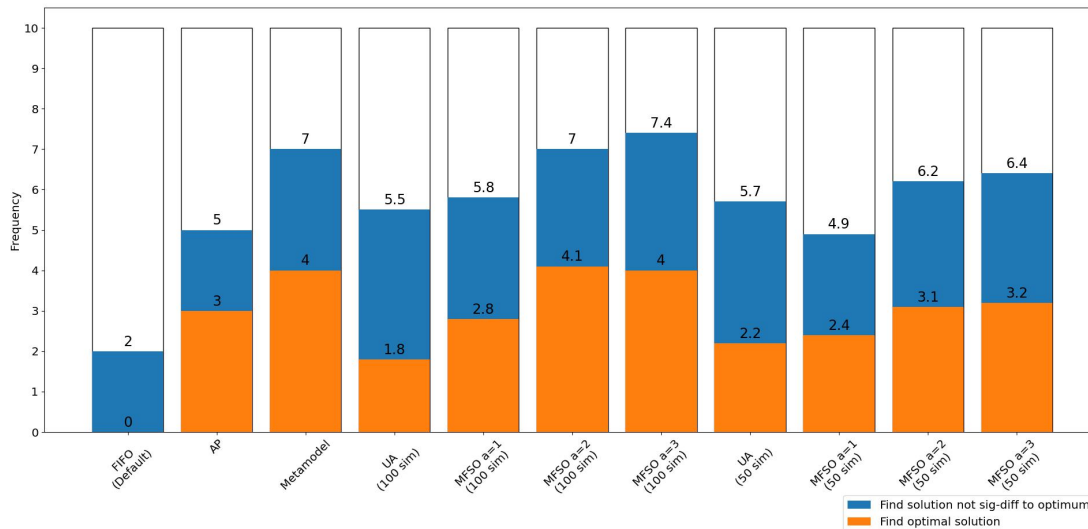


FIGURE 5.4: Aggregated frequency of optimisation results meet the optimal repair policies and the alternatives that are not significantly different from the optimum

We calculate how often different methods would catch the true optimal solution and the solutions that are not significantly different from the optimum in ten tests. This measures the performance of the methods across various test sets. For example, MFSO  $a \rightarrow 3$  (100 Sim run) catches four optima in yellow and four solutions that are not statistically different to the optima in blue on the trial recorded in Table 5.2. We average the counts among ten trials and plot the results in Figure 5.4. The orange bar shows the average number of tests that find the optimal solution and the blue bar stands for the average number of tests that find a solution that has no significant difference from the optimum. Summed together, they constitute the average performance of each method (included as the number above the blue bar). Since there are ten designated test sets in all, the maximum each bar could possibly reach is ten, which is plotted in white as the background of the bars.

In Figure 5.4, the three tallest orange bars that capture the four true best solutions are three MFSOs with a 100 simulations online budget and the metamodel. The MFSO with a budget of 50 replications and the AP are slightly lower with three optimum catches. On the other hand, the lowest orange bar is FIFO which does not have any result in 10 tests equal to the best solution. However, when considering results that are not significantly different from the best (blue bars), we see a slightly different ordering. MFSO ( $a = 3$ , 100 sim) slightly outperforms other methods with 7.4 out of

10. MFSO ( $a = 2, 100$  sim) and metamodel come second at 7. They are followed by two MFSOs ( $a = 2, 3$ ) with 50 online simulation budgets and reaching 6.2 and 6.4 respectively. Generally speaking, MFSO (100 sim) performs better than MFSO (50 sim) with an average raise of 0.7 for the chance to catch the optimal repair policies in our ten tests and 0.9 for the chance to catch not only the optimal solutions but also those repair strategies that are not significantly different from the optimum. When having the same online budget, MFSO ( $a = 2, 3$ ) make better suggestions than UA in terms of finding both the optimal solutions alone and with the solutions that are not significantly different from the optimum. FIFO remains the least ideal method with the lowest bar which has an overall height of two.

To sum up, the MFSO ( $a = 3$ ) with a 100-simulation budget achieves the best performance among the methods in comparison for being the highest bar for both with only the orange part and with the stacked blue part in Figure 5.4. Both a larger aggression parameter  $a$  and a higher online budget can benefit the MFSO framework to perform better. What is more, metamodel alone also shows strong performance in the tests. We find that the default FIFO is the least ideal method. This implies that applying any optimisation in the comparison could make an improvement in the throughput of the manufacturing system.

After analysing Figure 5.4, we further structure a way to quantify the overall performance of the optimisation procedures. We assume that the differences for each test set among  $M$  trials follow a normal distribution such that the random variable  $D_t \sim N(\mu_t, \sigma_t^2)$ , where:

$$\mu_t = \frac{1}{M} \sum_{m=1}^M \mu_m, \quad (5.5)$$

$$\sigma_t^2 = \frac{1}{M^2} \sum_{m=1}^M \sigma_m^2. \quad (5.6)$$

When reporting results, we first sum the differences in  $T$  tests,

$$D_\Sigma = \sum_{t=1}^T D_t. \quad (5.7)$$

Assuming normality, we have  $D_\Sigma \sim N(\mu, \sigma^2)$ . We can therefore estimate the mean and variance of the sum,  $D_\Sigma$ ,

$$\mu_\Sigma = \sum_{t=1}^T \mu_t, \quad (5.8)$$

$$\sigma_\Sigma^2 = \sum_{t=1}^T \sigma_t^2. \quad (5.9)$$

We then look for the mean error over all of the test sets,  $D_{\Sigma}/T$ . In this case, the mean of the normal distribution is

$$\mu_T = \mu_{\Sigma}/T, \quad (5.10)$$

and the variance is

$$\sigma_T^2 = \sigma_{\Sigma}^2/T^2. \quad (5.11)$$

From the mean and the variance of  $D_{\Sigma}/T$ , the confidence interval could be derived as:

$$CI = \mu_T \pm z_{1-\alpha/2} \sqrt{\sigma_T^2}. \quad (5.12)$$

Method	Aggregated difference
FIFO (Default)	$8.831 \pm 4.687$
AP	$4.276 \pm 4.042$
Metamodel	$2.945 \pm 3.723$
UA (100 Sim Run)	$4.920 \pm 4.427$
MFSO a $\rightarrow 1$ (100 Sim Run)	$4.416 \pm 4.039$
MFSO a $\rightarrow 2$ (100 Sim Run)	$3.400 \pm 3.814$
MFSO a $\rightarrow 3$ (100 Sim Run)	$2.638 \pm 3.659$
UA (50 Sim Run)	$4.718 \pm 4.389$
MFSO a $\rightarrow 1$ (50 Sim Run)	$5.865 \pm 4.502$
MFSO a $\rightarrow 2$ (50 Sim Run)	$3.919 \pm 4.013$
MFSO a $\rightarrow 3$ (50 Sim Run)	$3.984 \pm 4.031$

TABLE 5.4: Aggregated difference between selected solution and optimum

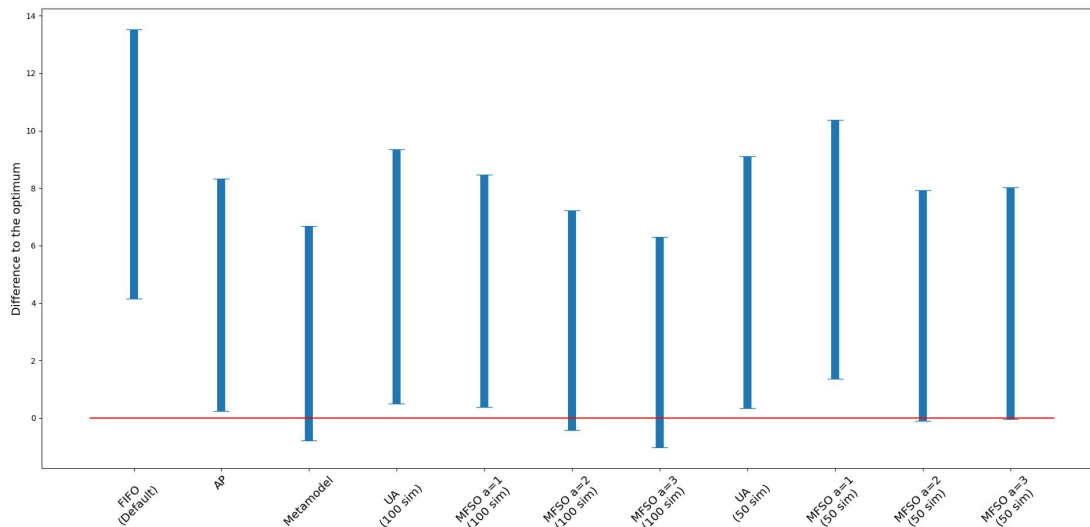


FIGURE 5.5: Aggregated difference to the optimum

The 95% confidence intervals of the aggregated difference for all methods are recorded in Table 5.4 and plotted in Figure 5.5. The vertical blue bars in the diagram are the confidence intervals of the methods marked on the x-axis. The y-axis shows the value of differences to the optimum. The red line parallel to the x-axis is zero difference.



The methods that have confidence intervals intersect the zero line are able to select the repair policy that is not significantly different to the best solution in the ‘gold standard’. Five methods have zero in their intervals in Figure 5.5. They are the proposed MFSO framework with the aggression parameter  $a = 2$  and 3 with both 100 and 50 simulation budgets and the metamodel. These five methods are showing better performance than other methods in the comparison. We find that MFSO ( $a = 3$ , 100 sim) has the lowest aggregated mean difference at 2.638 and the narrowest interval at  $\pm 3.659$  in Table 5.3. It is also observed that MFSOs with higher online budgets come with narrower confidence intervals. We can not reject the null hypothesis that MFSO can find either the right repair policy or the policy that is not significantly different from the best when the aggression parameter  $a = 2$  or 3.

To round up, the results shown in Figure 5.4 and 5.5 agree with each other. The five methods having the highest bars in Figure 5.4 also have confidence intervals that intersect the zero line in Figure 5.5. MFSO ( $a = 3$  100 sim) performs the best in both plots.

We are aware that using the metamodel on its own for optimisation, without running the ranking and selection step also generates acceptable solutions. This makes the simulation seem not to offer extra information and doubt could be raised on whether the online simulation should remain in the optimisation procedure. However, this should not be a surprise since the average ‘actual\_rank’ of the metamodel in Table 4.8 is 2.7. The final optimisation result would not differ a lot from the suggestion based on the metamodel estimations when the metamodel is of high quality. A high-quality metamodel is highly accurate in terms of both the prediction and the ranking ability, especially in the high-rank area. Online simulation optimisation offers extra confidence in the optimisation result.

### 5.3 Conclusions

In this chapter, we apply the multi-fidelity simulation optimisation (MFSO) framework proposed in Section 3 to a real-life problem with the simulation model and the metamodel that has been described in Section 4. During the application, we have the following contributions: 1. The application of the MFSO framework to a practical example in manufacturing for real-time optimisation; 2. The design of an MFSO framework that can be used for real-time decision-making with a complex simulation model. Through analysing the test results, we find that the proposed MFSO framework works well for the real-time optimisation of a manufacturing system, especially when the aggression parameter  $a$  is 3 and has a 100 online simulation budget.

The proposed MFSO framework works alongside the simulation model and the metamodel, which forms a digital twin to select the optimal repair policy in near real time. The stochasticity of the system is preserved in the optimisation procedure by keeping the simulation in the online phase. On the other hand, the metamodel helps to focus more on the promising alternatives and saves computing time. The positive findings show that the proposed MFSO is able to quickly optimise a stochastic process. This implies the potential of the MFSO framework to have broad applications across a range of different application areas.

In order to achieve real-time or near real-time optimisation, only a small number of replications should be made in the online phase. The online budget could be understood as being used to double-check the metamodel prediction since MFSO puts emphasis on the solutions with better metamodel performance. Estimations from the deterministic metamodel could be corrected in the online ranking and selection with simulation. Simulation has its value in securing the quality of the optimisation. In situations where there is not sufficient time for online simulation, using the metamodel alone could be an alternative.

## Chapter 6

# Conclusions

We explore digital twin technology in real-time/near-real-time optimisation for stochastic systems. A multi-fidelity framework, where a low-fidelity metamodel is included to work with the high-fidelity simulation model, is proposed in order to achieve fast optimisation. The research on the multi-fidelity simulation optimisation (MFSO) framework and its application in managing a production line in real-time has been elaborated on in the previous chapters. Deploying the proposed MFSO to the production line could boost the throughput as the experiment results suggest. We conclude the thesis in this chapter by highlighting the contributions to the academic literature in Section 6.1, describing the implications for practice in Section 6.2 and showing our outlook for some future works in the field of real-time simulation optimisation for digital twins in Section 6.3.

### 6.1 Contributions

The research is carried out for a real-life problem on the production line where the priorities of corrective maintenance need to be determined and assigned for multiple simultaneous breakdowns. An MFSO framework has been proposed to solve the problem. It can be fit into a larger framework of a digital twin where the simulation model is complex and computationally expensive and/or has a large number of alternatives. The method is tested on two examples, a textbook example and an example of a production line.

The work in this project makes contributions both to academia and to industry. The major methodological contributions are the proposed optimisation framework for real-time simulation optimisation and a design of experiments (DoE) method for stochastic systems while the budget for sampling is limited.

### 6.1.1 Methodological Contributions

The multi-fidelity simulation optimisation framework is proposed in Chapter 3. MFSO extends the MO<sup>2</sup>TOS framework (Xu et al., 2014), a leading multi-fidelity optimisation framework, to work with an uneven grouping strategy for the alternatives. The proposed framework forms our solution for real-time simulation optimisation and answers research question (RQ) 1 (which is raised for improving existing ranking and selection procedures for producing results in near real-time). The optimisation is designed to focus more on the solutions that are estimated to work well by the metamodel. The uneven grouping strategy ( $1^a : 2^a : 3^a \dots$ ) can be adjusted by an aggression parameter  $a$ . Two applications of the proposed framework have been demonstrated. The framework is able to work with situations that have a very large number of alternatives by clustering them first before grouping. The example of an inventory system with over 10,000 alternatives is tested. The results suggest a high probability of the MFSO generating optimisation output that is not significantly different from the true optimal solution, where MFSO outperforms the MO<sup>2</sup>TOS.

The proposed adaptive sequential sampling method has been described in Section 4.4.2. It efficiently assigns the sampling budget for a stochastic system. Unlike many methods in the literature, our approach does not rely on specific modelling techniques (e.g. stochastic kriging). The experimental design combines an exploiting process and an exploring process in each iteration to collect data from the feasible area. The uncertainty at different locations could be different so extra replications are assigned only for the locations that are highly varied in the exploiting process. In the exploring process, we seek new locations for sampling, which are both far from the existing samples and having high prediction variance. The adaptive sequential sampling forms part of the solution for RQ3 which is about training the metamodel efficiently.

### 6.1.2 Empirical Contributions

The MFSO framework has been applied to two examples, the inventory system example and a production line example. The inventory system evaluates MFSO and the proposed method performs very well in dealing with the optimisation problem with a large number of alternatives. This is the basis for applying the proposed method to the manufacturing system. In the production line case, we are looking for the optimal repair policy when multiple machines break down simultaneously. The simulation model of the manufacturing system is structured to enable hot-start and repair policies, which answers RQ5 for simulating a production line. The input variables are transformed before fitting the metamodel in order to reduce its dimension. The system is segmented into four sections, the section states are used to describe the system instead of taking the states of every buffer and machine. A

measurement for the adjacency of the failures is proposed to provide extra information to reduce information loss caused by transforming the state space. The experiments required to collect data for training the metamodel are then reduced so that we can fit the metamodel with less time and computing budget (RQ3).

For quantifying the performance of the metamodel (RQ4), we measure the ranking performance of the metamodel in addition to estimating the loss function (e.g. mean absolute error) which measures the accuracy of a machine learning model. The ranking correlation coefficient Kendall's tau is applied in the inventory system example. The high value indicates that the metamodel is working well for ranking the predictions. In the production line case, the metrics for the quality of ranking the top solutions are introduced in section 4.4.4 (`top3_captured`, `top1rank` and `actual_rank`) besides Kendall's tau since this area would cause a greater impact on the optimisation.

The MFSO framework is modified to the application of the production line (RQ2) with ten repair policies by directly grouping the alternatives and adapting the grouping ratio. It is able to select a repair policy that has no significant difference from the optimal solution.

Our work contributes both to academia, by proposing methods to solve problems from the industry and to practice by implementing the proposed methods in a real-life setting.

## 6.2 Implications for Practice

Our proposed MFSO is a method that could assist decision-making and solve manufacturing problems in near real time. The method could support fast feedback when the system is abnormal with the suggestion from the framework which helps to bring the system back to normality faster than using methods not based on optimisation. The throughput is thus increased by minimising the losses which are caused by the machine breakdowns.

The proposed MFSO framework is developed based on the MO<sup>2</sup>TOS. In MFSO, we use a metamodel that is higher in estimation precision than MO<sup>2</sup>TOS to guide the online simulation optimisation. In the online phase, the solutions that the metamodel predicts will have better performance have higher chances of getting sampled for simulation. It is achieved by having a smaller group for highly ranked solutions, while MO<sup>2</sup>TOS evenly groups solutions that will have various performances predicted by the metamodel. By doing that, we would have a higher probability of identifying the optimal solution with a limited online computing budget for simulation.

## Limitations

Since simulation is involved in the MFSO framework as a key component in the online phase, the time available for running the optimisation is constrained by the running time of the simulation model. A well-built simulation model that simulates the behaviour of a complex manufacturing system easily gets large in size and can be computationally expensive to run. In the experiments, we assume that a ten-minute response time is allowed for the algorithm to run and return the optimisation result as this will not cause extra time costs for on-site staff. The staff will receive the suggestions produced by the optimisation algorithm before they arrive at the location of the failed machines. In other application scenarios, the computing budget may be different and this is something that needs to be decided by the problem owner. For example, in a situation where the maximum allowed response time for the optimisation algorithm is very short, such as one minute. If a single run of the simulation model takes ten seconds, there would not be enough time for the MFSO to give out a proper result. In these cases, making the decision with metamodel estimations alone could be an alternative but more research is needed to determine its reliability. Also, a more accurate metamodel might be needed since the solutions are not going to be assessed through the high-fidelity simulation model.

## Cost

Deploying the proposed MFSO algorithm to the manufacturing system could boost the throughput while no big updates on the facilities are needed. However, there might still need some extra expenses associated with collecting, transmitting, processing and storing information from the manufacturing system. This forms the infrastructure of the digital twin that connects the physical system, the virtual models and the optimisation algorithms. The capability of the existing system significantly affects what and how many modifications would be needed for the transformation.

We need at least the real-time production line buffer state to enable the optimisation. The sensor is normally used to attain such information. Cameras with computer vision could be a substitute (Panahi et al., 2022). The collected data is then transmitted via cable or wireless communication to the server where the data could be processed to remove the noise from both the system and the environment for sensor data or be processed by computer vision algorithms for camera data. After that, the processed data can then be used as the system state and input to the optimisation procedures. The devices that are necessary for the data collection and preparation stage are the cost of deploying the MFSO algorithm to the production line.

## 6.3 Future Work

The proposed MFSO framework works well for the tested inventory problem and a production line repair order problem. We believe the proposed framework could be applied to more optimisation problems in manufacturing; for example, designing precautional maintenance plans, making production plans and arranging labour schedules. For a different production line, in addition to another sequential production line, it is also worth exploring the application to a flexible or reconfigurable line (Wang et al., 2016). It refers to a production line that is designed to accommodate multiple tasks where the workstations can be reconfigured and adapted to handle different jobs so that different product variants or types are able to share one production line.

There are two other directions for future research. One is to enable the proposed algorithm to deal with multiple objectives and the other is to test different modelling techniques for fitting the metamodel of the simulation model. More details are elaborated in Section 6.3.1 and 6.3.2 respectively.

### 6.3.1 Fusing multi-objective optimisation into the multi-fidelity simulation optimisation framework

The proposed MFSO frame has been designed and tested to optimise a single objective, however, in reality, there are many situations in which more than one objective function need to be considered when performing the optimisation for the manufacturing system. Different objectives could conflict with each other which means improving one objective would compromise the other objectives. For example, a shorter cycle time of the machine might reduce the quality of the job. The multi-objective optimisation balances various demands and finds the optimal trade-off among the conflicting objectives. There are a few methods that have been developed to deal with multi-objective optimisation, such as multiple attribute utility theory (Butler et al., 2001) and Pareto front (Lee et al., 2004).

Multiple attribute utility theory sets a weight coefficient for each objective and the interaction of objectives. This satisfies the situation where the objectives have different priorities. These weights directly affect the optimisation and thus they are not easy to determine and need expert knowledge for the specific scenario.

On the other hand, the Pareto front equally treats all the conflict objectives and regards them as nontrivial. A solution is non-dominated when it is not able to be improved while not sacrificing any objective. There normally exists more than one solution that meets the requirement of being non-dominated. The qualified solutions form the Pareto front. The concept of the Pareto front has been applied to the ranking

and selection procedure in the research like Lee et al. (2004), Applegate et al. (2020) and Hunter et al. (2019). Li et al. (2016) developed an variant of MO<sup>2</sup>TOS to cope with multiple objectives. We would like to incorporate the Pareto front into the proposed MFSO framework. The solutions could be sorted into a number of Pareto layers based on the metamodel estimations where the solutions in each layer are not dominated by each other. The Pareto front is the layer with solutions that are estimated to perform the best. The Pareto layers could then be grouped in accordance with the proposed grouping ratio with aggression parameter  $a$ . The specific rule may need adjustment to adapt to the multi-objective optimisation. We believe this could extend the ability of MFSO to deal with multi-objective optimisation problems close to real time.

### 6.3.2 Working with different types of metamodels

In our proposed MFSO framework, the metamodel is used as a low-fidelity model to guide the optimisation via simulation so that the simulation replications required for the optimisation are reduced. Feedforward neural network regression model has been used and tested as the metamodel in the proposed MFSO. It works well in the tests. We believe it is worth trying other modelling techniques and it would be valuable to check their performance working with our proposed framework. For example, the Gaussian processes model (Rasmussen and Williams, 2005) is also a popular choice for regression as a simulation metamodel.

MO<sup>2</sup>TOS (Xu et al., 2014) used a Jackson network (Meyn and Down, 1994) to simplify the high-fidelity simulation model. Their low-fidelity model has large biases in the estimations, yet the ordinal rankings are regarded as 'largely correct' by the authors. This could be the next thing to explore, whether and how the balance between the accuracy in estimation and the model complexity could be achieved in a low-fidelity queue theory model.

Since the information extracted from the low-fidelity model is the rankings of the solutions, we consider building a metamodel of the high-fidelity simulation that suggests the rankings directly instead of estimating the solutions first and then ranking the estimations. Learn-to-rank (Liu et al., 2009) is a group of algorithms that have been widely used by web search engines to give out a sequence of results based on the keywords given by the users. The idea was introduced by Fuhr (1992) in 1992. There are now three major groups of methods for learn-to-rank, the pointwise approach (Crammer and Singer, 2001), the pairwise approach (Burges et al. (2005), Liu et al. (2020)) and the listwise approach (Xu and Li (2007), Cao et al. (2007), Swezey et al. (2021)). Unlike early research concentrated on the pointwise approach, most recent research in the field focuses on the pairwise approach and listwise approach. Although the major application of the learn-to-rank is still web searching, it is also an essential technology for electronic commerce applications (Huzhang et al., 2020).



Apart from them, [Forman \(2021\)](#) adapt the learn-to-rank method for geolocation in package delivery. There is a lack of research to apply the learn-to-rank methods to more industries such as manufacturing, which would make it a meaningful attempt to build a metamodel for production line simulation model using learn-to-rank methods. The collected data needs to be re-organised to fit a listwise learn-to-rank metamodel. Data should be transformed into a series of lists that contain the same system state with ten repair policies. The output of the metamodel would be the performance rank of the input repair policies, which also need extra data processing from the simulation output. In addition, more data may need to be collected to fit the metamodel, which depends on the number of parameters contained in the learn-to-rank metamodel. Although little research applies learn-to-rank methods for simulation metamodel, it is still worth it to explore the potential of a different class of modelling techniques.



## Appendix A

# Difference Between the Optimal Repair Policy and Other Alternatives in 100-replication Gold Standard ( $D_m$ )

TABLE A.1: Difference between the optimal repair policy and other alternatives ( $D_m$ )  
Part 1

Testset 1	Mean	Var	Testset 2	Mean	Var
FIFO	5.04	307.958	FIFO	4.17	572.244
LIFO	0	0	LIFO	11.72	797.658
SRT	5.35	384.048	SRT	5.90	561.263
LRT	6.32	396.260	LRT	28.38	2284.501
FOR	5.95	346.351	FOR	1.32	581.614
MOR	5.31	422.519	MOR	38.94	2352.845
NOSQ	4.80	328.889	NOSQ	23.95	1020.008
COLQ	5.39	378.685	COLQ	1.38	501.026
AP	5.23	308.543	AP	0	0
NAP	5.93	351.500	NAP	35.19	2298.559
Testset 3	Mean	Var	Testset 4	Mean	Var
FIFO	18.30	688.192	FIFO	9.66	529.499
LIFO	16.86	847.798	LIFO	11.28	977.658
SRT	13.71	725.521	SRT	8.96	988.806
LRT	35.51	807.869	LRT	10.07	729.298
FOR	1.29	136.693	FOR	10.39	609.392
MOR	38.47	767.989	MOR	11.08	943.488
NOSQ	35.18	754.452	NOSQ	0	0
COLQ	17.43	402.773	COLQ	7.57	800.955
AP	0	0	AP	6.29	747.157
NAP	2.01	179.808	NAP	9.00	744.525

TABLE A.2: Difference between the optimal repair policy and other alternatives ( $D_m$ )  
Part 2

Testset 5	Mean	Var	Testset 6	Mean	Var
FIFO	12.33	1081.193	FIFO	3.00	270.000
LIFO	39.28	3074.567	LIFO	5.65	378.553
SRT	11.28	1796.466	SRT	0	0
LRT	35.33	3888.789	LRT	4.79	549.703
FOR	1.43	601.520	FOR	15.20	626.202
MOR	62.25	4777.301	MOR	4.15	334.795
NOSQ	1.52	718.798	NOSQ	12.03	596.130
COLQ	7.73	2630.664	COLQ	1.11	164.867
AP	0	0	AP	12.02	585.616
NAP	50.06	2664.865	NAP	0.81	248.115
Testset 7	Mean	Var	Testset 8	Mean	Var
FIFO	14.45	843.987	FIFO	7.41	531.618
LIFO	15.97	1252.292	LIFO	4.88	828.632
SRT	9.22	1349.729	SRT	3.00	500.505
LRT	19.19	1052.499	LRT	3.17	346.062
FOR	3.56	770.916	FOR	6.04	589.433
MOR	24.62	1275.491	MOR	3.30	261.101
NOSQ	18.14	873.576	NOSQ	3.73	491.472
COLQ	0	0	COLQ	0	0
AP	0.48	988.697	AP	4.93	548.854
NAP	3.27	793.633	NAP	7.89	505.372
Testset 9	Mean	Var	Testset 10	Mean	Var
FIFO	4.21	315.299	FIFO	9.74	577.507
LIFO	2.74	1045.689	LIFO	11.08	614.297
SRT	3.57	606.005	SRT	0	0
LRT	4.12	459.682	LRT	15.25	1311.886
FOR	5.58	874.387	FOR	18.71	797.905
MOR	5.12	744.794	MOR	5.48	1083.727
NOSQ	4.49	584.838	NOSQ	6.52	639.606
COLQ	0	0	COLQ	7.99	527.485
AP	2.27	410.239	AP	11.54	663.564
NAP	4.90	1052.899	NAP	19.97	884.979

## **Appendix B**

# **Complete Experiment Results in Production Line Example**

TABLE B.1: Complete experiment results for 10 testsets in 10 trials

Algorithm	Trial	Testset 1	Testset 2	Testset 3	Testset 4	Testset 5	Testset 6	Testset 7	Testset 8	Testset 9	Testset 10
UA (100 Sim)	1	FIFO	FIFO	FOR	FIFO	FIFO	FIFO	FOR	COLQ	FOR	FIFO
	2	LRT	LIFO	LIFO	LRT	FOR	SRT	COLQ	AP	SRT	LRT
	3	SRT	SRT	FOR	MOR	FOR	SRT	COLQ	MOR	LIFO	SRT
	4	LIFO	FOR	FOR	MOR	FIFO	LIFO	AP	LIFO	LIFO	LIFO
	5	SRT	FOR	AP	SRT	NOSQ	FIFO	AP	SRT	LRT	SRT
	6	AP	AP	AP	AP	SRT	AP	AP	NOSQ	AP	AP
	7	FOR	FOR	FOR	MOR	NOSQ	MOR	SRT	COLQ	MOR	MOR
	8	LIFO	LIFO	FOR	AP	AP	AP	COLQ	FOR	LIFO	LIFO
	9	FIFO	FIFO	NAP	FIFO	FIFO	FIFO	COLQ	SRT	FIFO	FIFO
	10	FIFO	AP	NAP	FIFO	AP	FIFO	FIFO	FIFO	FIFO	FIFO
MFSO a → 1 (100 Sim Run)	1	NOSQ	FOR	AP	LRT	AP	AP	AP	COLQ	MOR	SRT
	2	MOR	COLQ	FOR	SRT	AP	AP	FIFO	COLQ	FIFO	NOSQ
	3	NOSQ	COLQ	AP	FOR	FIFO	SRT	COLQ	MOR	FOR	MOR
	4	NOSQ	AP	FOR	SRT	AP	COLQ	NOSQ	AP	SRT	COLQ
	5	FIFO	FOR	FOR	LIFO	AP	SRT	FOR	NOSQ	LIFO	FIFO
	6	COLQ	AP	AP	FOR	AP	AP	NOSQ	SRT	COLQ	LIFO
	7	AP	COLQ	FOR	SRT	NOSQ	FIFO	FOR	AP	NOSQ	LIFO
	8	NOSQ	SRT	AP	NOSQ	FIFO	SRT	AP	COLQ	SRT	COLQ
	9	COLQ	AP	AP	AP	AP	AP	FIFO	NOSQ	COLQ	SRT
	10	COLQ	AP	FOR	COLQ	AP	FIFO	AP	FIFO	AP	LIFO
MFSO a → 2 (100 Sim Run)	1	SRT	AP	AP	AP	AP	AP	COLQ	FOR	LIFO	LRT
	2	SRT	COLQ	FOR	NAP	AP	MOR	AP	AP	COLQ	COLQ
	3	FIFO	AP	AP	LIFO	NOSQ	FIFO	AP	LIFO	SRT	MOR
	4	NOSQ	AP	AP	NOSQ	NOSQ	AP	COLQ	LRT	SRT	MOR
	5	LIFO	AP	AP	AP	AP	FIFO	COLQ	LIFO	AP	FOR
	6	MOR	AP	AP	AP	COLQ	AP	SRT	AP	COLQ	COLQ
	7	LIFO	AP	AP	AP	NOSQ	COLQ	FOR	MOR	FOR	SRT
	8	COLQ	AP	AP	COLQ	AP	FOR	COLQ	COLQ	COLQ	SRT
	9	COLQ	NAP	AP	AP	AP	AP	COLQ	LRT	AP	SRT
	10	COLQ	AP	AP	AP	AP	COLQ	AP	SRT	AP	SRT
MFSO a → 3 (100 Sim Run)	1	COLQ	AP	AP	NOSQ	AP	COLQ	AP	SRT	AP	NAP
	2	COLQ	FOR	AP	FOR	AP	AP	AP	SRT	FOR	SRT
	3	COLQ	AP	AP	NOSQ	FOR	COLQ	AP	SRT	FOR	SRT
	4	AP	AP	AP	AP	AP	COLQ	AP	SRT	FIFO	SRT
	5	COLQ	AP	AP	AP	AP	AP	AP	SRT	AP	SRT
	6	FOR	AP	AP	AP	AP	MOR	AP	SRT	FOR	SRT
	7	COLQ	COLQ	AP	NAP	AP	NOSQ	AP	FIFO	AP	SRT
	8	COLQ	FOR	AP	AP	AP	SRT	AP	COLQ	AP	SRT
	9	COLQ	AP	AP	AP	AP	AP	COLQ	SRT	AP	SRT
	10	COLQ	AP	AP	AP	AP	COLQ	AP	SRT	AP	SRT
UA (50 Sim Run)	1	AP	AP	AP	AP	LIFO	LIFO	AP	AP	AP	AP
	2	COLQ	COLQ	COLQ	COLQ	COLQ	COLQ	COLQ	COLQ	COLQ	COLQ
	3	SRT	FOR	FOR	FOR	FOR	FOR	FOR	NOSQ	FOR	SRT
	4	LRT	AP	NAP	AP	AP	SRT	AP	LRT	LRT	SRT
	5	FOR	COLQ	NAP	SRT	NOSQ	LIFO	COLQ	MOR	LIFO	SRT
	6	LIFO	FOR	SRT	COLQ	FOR	COLQ	COLQ	SRT	SRT	SRT
	7	MOR	AP	AP	MOR	SRT	MOR	MOR	MOR	MOR	MOR
	8	COLQ	COLQ	AP	COLQ	AP	COLQ	COLQ	AP	LRT	COLQ
	9	LIFO	FIFO	FOR	LIFO	FIFO	MOR	NAP	MOR	FIFO	LIFO
	10	LIFO	FOR	NAP	FIFO	SRT	LIFO	FOR	FIFO	FIFO	LIFO
MFSO a → 1 (50 Sim Run)	1	FIFO	AP	AP	AP	AP	COLQ	AP	LIFO	AP	COLQ
	2	FOR	NAP	FIFO	NAP	SRT	NOSQ	SRT	COLQ	NAP	COLQ
	3	MOR	SRT	FOR	COLQ	NOSQ	FIFO	AP	NOSQ	FIFO	MOR
	4	NAP	FIFO	AP	SRT	FIFO	SRT	COLQ	MOR	SRT	SRT
	5	COLQ	SRT	AP	AP	FIFO	COLQ	NOSQ	COLQ	NOSQ	FIFO
	6	AP	AP	AP	AP	AP	AP	SRT	FOR	SRT	AP
	7	FIFO	FOR	AP	FOR	SRT	FOR	LIFO	LIFO	FOR	SRT
	8	COLQ	FOR	AP	LRT	AP	AP	COLQ	FOR	LRT	SRT
	9	COLQ	AP	NOSQ	AP	FIFO	SRT	COLQ	SRT	AP	AP
	10	COLQ	COLQ	AP	AP	FIFO	FOR	NOSQ	AP	AP	SRT
MFSO a → 2 (50 Sim Run)	1	LRT	NOSQ	AP	AP	AP	AP	AP	SRT	AP	SRT
	2	FIFO	COLQ	SRT	LIFO	SRT	NOSQ	FOR	COLQ	COLQ	MOR
	3	NAP	FOR	AP	NOSQ	NOSQ	FIFO	FOR	LRT	FIFO	SRT
	4	LRT	COLQ	AP	LIFO	NOSQ	MOR	COLQ	AP	NOSQ	COLQ
	5	LIFO	COLQ	FOR	AP	AP	COLQ	COLQ	MOR	FOR	NOSQ
	6	FOR	AP	SRT	AP	FIFO	MOR	COLQ	FIFO	AP	SRT
	7	MOR	FOR	AP	AP	AP	NAP	AP	AP	COLQ	SRT
	8	COLQ	AP	AP	AP	AP	AP	AP	FOR	AP	FOR
	9	FIFO	AP	LIFO	NOSQ	AP	AP	AP	SRT	SRT	SRT
	10	COLQ	AP	AP	AP	NOSQ	COLQ	FOR	COLQ	AP	SRT
MFSO a → 3 (50 Sim Run)	1	AP	AP	AP	AP	FOR	COLQ	AP	SRT	MOR	SRT
	2	FOR	SRT	AP	NAP	FOR	COLQ	COLQ	SRT	NAP	MOR
	3	MOR	AP	LIFO	AP	AP	NAP	FOR	SRT	COLQ	NAP
	4	MOR	FIFO	AP	LIFO	AP	LRT	AP	COLQ	AP	SRT
	5	COLQ	AP	LIFO	LIFO	AP	AP	SRT	LRT	AP	NAP
	6	FIFO	AP	AP	AP	AP	AP	FOR	SRT	AP	SRT
	7	SRT	AP	AP	AP	AP	AP	AP	SRT	AP	SRT
	8	COLQ	LIFO	AP	LIFO	AP	AP	NAP	SRT	AP	NOSQ
	9	COLQ	AP	AP	AP	AP	AP	AP	SRT	AP	SRT
	10	NOSQ	AP	AP	COLQ	AP	LIFO	AP	FIFO	AP	SRT
FIFO (default)		FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO	FIFO
AP		AP	AP	AP	AP	AP	AP	AP	AP	AP	AP
Metamodel		COLQ	AP	AP	AP	AP	AP	AP	SRT	AP	SRT

## References

- Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat Abd Elatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018. ISSN 24058440. .
- Abdullah Alrabghi and Ashutosh Tiwari. State of the art in simulation-based optimisation for maintenance systems. *Computers Industrial Engineering*, 82:167–182, 2015. ISSN 0360-8352. .
- Bruce Ankenman, Barry L. Nelson, and Jeremy Staum. Stochastic kriging for simulation metamodeling. *Operations Research*, 58(2):371–382, 2010. ISSN 0030364X, 15265463.
- Eric A. Applegate, Guy Feldman, Susan R. Hunter, and Raghu Pasupathy. Multi-objective ranking and selection: Optimal sampling laws and tractable approximations via SCORE. *Journal of Simulation*, 14(1):21–40, 2020. ISSN 17477786. .
- Russell R. Barton. Tutorial: Metamodeling for simulation. In *Proceedings - Winter Simulation Conference*, 2020. ISBN 9788578110796. .
- Russell R. Barton and Martin Meckesheimer. Chapter 18 Metamodel-based simulation optimization. In *Handbooks in Operations Research and Management Science*, volume 13, pages 535–574. 2006. .
- Robert E. Bechhofer. A single-sample multiple decision procedure for ranking means of normal populations with known variances. *The Annals of Mathematical Statistics*, 25(1):16–39, mar 1954. ISSN 0003-4851. .
- Clément Bénard, Gérard Biau, Sébastien Da Veiga, and Erwan Scornet. Interpretable random forests via rule extraction. In *International Conference on Artificial Intelligence and Statistics*, pages 937–945. PMLR, 2021.
- Gérard Biau and Erwan Scornet. A random forest guided tour. *Test*, 25(2):197–227, 2016. ISSN 11330686. .
- Mickaël Binois, Jiangeng Huang, Robert B. Gramacy, and Mike Ludkovski. Replication or exploration? sequential design for stochastic simulation experiments. *Technometrics*, 61(1):7–23, 2019. .

- Elisa Bjerre, Michael N. Fienen, Raphael Schneider, Julian Koch, and Anker L. Højberg. Assessing spatial transferability of a random forest metamodel for predicting drainage fraction. *Journal of Hydrology*, 612:128177, 2022. ISSN 0022-1694. .
- Juergen Branke and Wen Zhang. A new myopic sequential sampling algorithm for multi-objective problems. In *Proceedings - Winter Simulation Conference*, pages 3589–3598. IEEE, 2015.
- Juergen Branke, Wen Zhang, and Yang Tao. Multiobjective ranking and selection based on hypervolume. In *Proceedings - Winter Simulation Conference*, volume 0, pages 859–870, 2016. ISBN 9781509044863. .
- Jürgen Branke, Stephen E. Chick, and Christian Schmidt. Selecting a selection procedure. *Management Science*, 53(12):1916–1932, 2007. .
- Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning, ICML '05*, page 89–96, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931805.
- John Butler, Douglas J. Morrice, and Peter W. Mullarkey. A multiple attribute utility theory approach to ranking and selection. *Management Science*, 47(6):800–816, 2001. ISSN 00251909. .
- Rodrigo Neves Calheiros, Rajiv Ranjan, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *ArXiv*, abs/0903.2525, 2009.
- Yiyun Cao, Christine Currie, Bhakti Stephan Onggo, and Michael Higgins. Simulation optimization for a digital twin using a multi-fidelity framework. In *Proceedings - Winter Simulation Conference*, pages 1–12, 2021. .
- Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning, ICML '07*, page 129–136, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937933. .
- Chun Hung Chen, Jianwu Lin, Enver Yücesan, and Stephen E. Chick. Simulation budget allocation for further enhancing the efficiency of ordinal optimization. *Discrete Event Dynamic Systems: Theory and Applications*, 10(3):251–270, 2000. ISSN 09246703. .
- E Jack Chen. Using parallel and distributed computing to increase the capability of selection procedures. In *Proceedings - Winter Simulation Conference*, pages 9–pp. IEEE, 2005.



- Yow-Mow Chen, T. Fujisawa, and H. Ōsawa. Availability of the system with general repair time distributions and shut-off rules. *Microelectronics Reliability*, 33(1):13–19, 1993. ISSN 0026-2714. .
- Stephen E. Chick and Koichiro Inoue. New two-stage and sequential procedures for selecting the best simulated system. *Operations Research*, 49(5):732–743, 2001. .
- Stephen E. Chick, Jürgen Branke, and Christian Schmidt. Sequential sampling to myopically maximize the expected value of information. *INFORMS Journal on Computing*, 22(1):71–80, 2010. .
- Francois Chollet. Keras. <https://keras.io/>, 2015.
- Koby Crammer and Yoram Singer. Pranking with ranking. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.
- Gabriele D’Angelo and Moreno Marzolla. New trends in parallel and distributed simulation: From many-cores to cloud computing. *Simulation Modelling Practice and Theory*, 49:320–335, 2014. .
- William H.E. Day and Herbert Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1(1):7–24, 1984. ISSN 01764268. .
- Benjamin Durakovic. Design of experiments application, concepts, examples: State of the art. *Periodicals of Engineering and Natural Sciences (PEN)*, 5(3), 2017.
- John Eason and Selen Cremaschi. Adaptive sequential sampling for surrogate model generation with artificial neural networks. *Computers Chemical Engineering*, 68: 220–232, 2014. ISSN 0098-1354. .
- Raed El-Khalil. Simulation analysis for managing and improving productivity: A case study of an automotive company. *Journal of Manufacturing Technology Management*, 26(1):36–56, 2015.
- Khairy Elsayed and Chris Lacor. Robust parameter design optimization using Kriging, RBF and RBFNN with gradient-based and evolutionary optimization techniques. *Applied Mathematics and Computation*, 236:325–344, 2014. ISSN 00963003. .
- Benjamin Epstein. The exponential distribution and its role in life testing. Technical report, Wayne State Univ Detroit MI, 1958.
- Kai-Tai Fang, Chang-Xing Ma, and Peter Winker. Centered l2-discrepancy of random sampling and latin hypercube design, and construction of uniform designs. *Mathematics of Computation*, 71(237):275 – 296, 2002. .

- Guy Feldman and Susan R. Hunter. SCORE allocations for bi-objective ranking and selection. *ACM Transactions on Modeling and Computer Simulation*, 28(1), 2018. ISSN 15581195. .
- Daniel J Fonseca, Daniel O Navarrese, and Gary P Moynihan. Simulation metamodeling through artificial neural networks. *Engineering Applications of Artificial Intelligence*, 16(3):177–183, 2003.
- George Forman. Getting your package to the right place: Supervised machine learning for geolocation. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part IV 21*, pages 403–419. Springer, 2021.
- Linda Weiser Friedman and Israel Pressman. The metamodel in simulation analysis: Can it be trusted? *Journal of the Operational Research Society*, 39(10):939–948, 1988.
- Michael C. Fu, editor. *Handbook of Simulation Optimization*. Springer, 2014. ISBN 978-1-4939-1383-1.
- Norbert Fuhr. Probabilistic models in information retrieval. *The Computer Journal*, 35(3):243–255, 1992.
- Edward Glaessgen and David Stargel. The digital twin paradigm for future nasa and us air force vehicles. In *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*, page 1818, 2012.
- Heitor M. Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10): 1469–1495, 2017. ISSN 15730565. .
- Heitor Murilo Gomes, Jean Paul Barddal, Luis Eduardo Boiko, and Albert Bifet. Adaptive random forests for data stream regression. In *ESANN 2018 - Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, number April, pages 267–272, 2018. ISBN 9782875870476.
- Travis Goodwin, Jie Xu, Nurcin Celik, and Chun-Hung Chen. Real-time digital twin-based optimization with predictive simulation learning. *Journal of Simulation*, pages 1–18, 2022.
- Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). pages 1–9, 2016.
- Michael Higgins and John Ladbrook. Ford’s power train operations: Changing the simulation environment. In *Proceedings - Winter Simulation Conference*, pages 3308–3318. IEEE, dec 2018. ISBN 978-1-5386-6572-5. .

- Tin Kam Ho. Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1:278–282, 1995.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. .
- L. Jeff Hong and Guangxin Jiang. Offline simulation online application: A new framework of simulation-based decision making. *Asia-Pacific Journal of Operational Research*, 36(6), 2019. ISSN 02175959. .
- L. Jeff Hong and Barry L. Nelson. A brief introduction to optimization via simulation. In *Proceedings - Winter Simulation Conference*, number 2002, pages 75–85, 2009. ISBN 9781424457700. .
- Shih-Cheng Horng and Shin-Yeu Lin. Ordinal optimization of G/G/1/K polling systems with k-limited service discipline. *Journal of Optimization Theory and Applications*, 140(2):213–231, 2009. ISSN 00223239. .
- Shih Cheng Horng and Shin Yeu Lin. Evolutionary algorithm assisted by surrogate model in the framework of ordinal optimization and optimal computing budget allocation. *Information Sciences*, 233:214–229, 2013. ISSN 00200255. .
- Shih Cheng Horng, Shin Yeu Lin, Loo Hay Lee, and Chun Hung Chen. Memetic algorithm for real-time combinatorial stochastic simulation optimization problems with performance analysis. *IEEE Transactions on Cybernetics*, 43(5):1495–1509, 2013. ISSN 21682267. .
- Deng Huang, Theodore T Allen, William I Notz, and R Allen Miller. Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization*, 32:369–382, 2006a.
- Deng Huang, Theodore T Allen, William I Notz, Ning Zeng, et al. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3):441–466, 2006b.
- Hsiang-Hsi Huang, Wen Pei, Horng-Huei Wu, and Ming-Der May. A research on problems of mixed-line production and the re-scheduling. *Robotics and Computer-Integrated Manufacturing*, 29(3):64–72, 2013. ISSN 0736-5845. . Extended Papers Selected from FAIM 2011.
- Carolynne Hultquist, Gang Chen, and Kaiguang Zhao. A comparison of Gaussian process regression, random forests and support vector regression for burn severity assessment in diseased forests. *Remote Sensing Letters*, 5(8):723–732, 2014. .
- Susan R. Hunter, Eric A. Applegate, Viplove Arora, Bryan Chong, Kyle Cooper, Oscar Rincón-Guevara, and Carolina Vivas-Valencia. An introduction to multiobjective

- simulation optimization. *ACM Transactions on Modeling and Computer Simulation*, 29(1):1–35, 2019. ISSN 15581195. .
- Guangda Huzhang, Zhen-Jia Pang, Yongqing Gao, Yawen Liu, Weijie Shen, Wen-Ji Zhou, Qing Da, An-Xiang Zeng, Han Yu, Yang Yu, and Zhi-Hua Zhou. Aliexpress learning-to-rank: Maximizing online model performance without going online, 2020.
- Paul Hyden and Lee Schruben. Improved decision processes through simultaneous simulation and time dilation. In *Proceedings - Winter Simulation Conference*, volume 1, pages 743–748. IEEE, 2000.
- Dmitry Ivanov, Alexandre Dolgui, Ajay Das, and Boris Sokolov. Digital supply chain twins: Managing the ripple effect, resilience and disruption risks by data-driven optimization, simulation, and visibility. In *Handbook of Ripple Effects in the Supply Chain*, volume 276, pages 309–332. 2019. ISBN 978-3-030-14301-5. .
- Danny J. Johnson. A spreadsheet method for calculating work completion time probability distributions of paced or linked assembly lines. *International Journal of Production Research*, 40(5):1131–1153, 2002. .
- Mark E Johnson, Leslie M Moore, and Donald Ylvisaker. Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, 26(2):131–148, 1990.
- O.A. Joseph and R. Sridharan. Effects of flexibility and scheduling decisions on the performance of an fms: simulation modelling and analysis. *International Journal of Production Research*, 50(7):2058–2078, 2012. .
- Vengazhiyil Roshan Joseph. Space-filling designs for computer experiments: A review. *Quality Engineering*, 28(1):28–35, 2016. .
- Maurice George Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81, jun 1938. ISSN 00063444. .
- Seong Hee Kim and Barry L. Nelson. A fully sequential procedure for indifference-Zone selection in simulation. *ACM Transactions on Modeling and Computer Simulation*, 11(3):251–273, 2001. ISSN 10493301. .
- Seong-Hee Kim and Barry L. Nelson. Chapter 17 selecting the best system. 2006a.
- Seong-Hee Kim and Barry L Nelson. On the asymptotic validity of fully sequential selection procedures for steady-state simulation. *Operations Research*, 54(3):475–488, 2006b.
- Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

- EMRE Kiyak. The importance of preventive maintenance in terms of reliability in aviation sector. *Recent Advances in Manufacturing Engineering*, 2011.
- Jack PC Kleijnen and WCM Van Beers. Application-driven sequential designs for simulation experiments: Kriging metamodelling. *Journal of the operational research society*, 55(8):876–883, 2004.
- Mirko Kück, Jens Ehm, Torsten Hildebrandt, Michael Freitag, and Enzo M. Frazzon. Potential of data-driven simulation-based optimization for adaptive scheduling and control of dynamic manufacturing systems. In *Proceedings - Winter Simulation Conference*, pages 2820–2831, 2016. .
- Averill M Law. *Simulation Modeling and Analysis*. McGraw-Hill Education, fifth edit edition, 2015. ISBN 9781259254383.
- Loo Hay Lee, Ek Peng Chew, Suyan Teng, and David Goldsman. Optimal computing budget allocation for multi-objective simulation models. In *Proceedings - Winter Simulation Conference*, volume 1, pages 586–593. IEEE, 2004. .
- Loo Hay Lee, Ek Peng Chew, Suyan Teng, and David Goldsman. Finding the non-dominated Pareto set for multi-objective simulation models. *IIE Transactions (Institute of Industrial Engineers)*, 42(9):656–674, 2010a. ISSN 0740817X. .
- Loo Hay Lee, Ek Peng Chew, Suyan Teng, and David Goldsman. Finding the non-dominated Pareto set for multi-objective simulation models. *IIE Transactions (Institute of Industrial Engineers)*, 42(9):656–674, 2010b. ISSN 0740817X. .
- Haobin Li, Yueqi Li, Loo Hay Lee, Ek Peng Chew, Giulia Pedrielli, and Chun Hung Chen. Multi-objective multi-fidelity optimization with ordinal transformation and optimal sampling. In *Proceedings - Winter Simulation Conference*, volume 2016-Febru, pages 3737–3748. IEEE, 2016. ISBN 9781467397438. .
- Haobin Li, Chenhao Zhou, Byung Kwon Lee, Loo Hay Lee, Ek Peng Chew, and Rick Siow Mong Goh. Capacity planning for mega container terminals with multi-objective and multi-fidelity simulation optimization. *IIE Transactions*, 49(9): 849–862, 2017.
- Haitao Liu, Yew-Soon Ong, and Jianfei Cai. A survey of adaptive sampling for global metamodelling in support of simulation-based complex engineering design. *Structural and Multidisciplinary Optimization*, 57(1):393–416, jan 2018. ISSN 1615-147X. .
- Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.
- Weiwen Liu, Qing Liu, Ruiming Tang, Junyang Chen, Xiuqiang He, and Pheng Ann Heng. Personalized re-ranking with item relationships for e-commerce. In

- Proceedings of the 29th ACM International Conference on Information Knowledge Management, CIKM '20*, page 925–934, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368599.
- Yuh-Chuyn Luo, Chun-Hung Chen, E. Yucesan, and Insup Lee. Distributed web-based simulation optimization. In *Proceedings - Winter Simulation Conference*, volume 2, pages 1785–1793 vol.2, 2000. .
- Andrew March and Karen Willcox. Provably convergent multifidelity optimization algorithm not requiring high-fidelity derivatives. *AIAA Journal*, 50(5):1079–1089, 2012. ISSN 00011452. .
- Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, dec 1943. ISSN 0007-4985. .
- Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979. ISSN 00401706.
- George Meghabghab and George Nasr. Iterative RBF neural networks as metamodels of stochastic simulations. *Proceedings of the 2nd International Conference on Intelligent Processing and Manufacturing of Materials, IPMM 1999*, 2:729–734, 1999. .
- Sean P Meyn and Douglas Down. Stability of generalized jackson networks. *The Annals of Applied Probability*, pages 124–148, 1994.
- Roxanne A. Moore, David A. Romero, and Christiaan J.J. Paredis. Value-based global optimization. *Journal of Mechanical Design, Transactions of the ASME*, 136(4):1–14, 2014. ISSN 10500472. .
- Max D Morris and Toby J Mitchell. Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, 43(3):381–402, feb 1995. ISSN 03783758. .
- Dimitris Mourtzis. Simulation in the design and operation of manufacturing systems: State of the art and new trends. *International Journal of Production Research*, 58(7): 1927–1949, 2020. .
- Misbah Mubarak, Christopher D. Carothers, Robert B. Ross, and Philip Carns. Enabling parallel simulation of large-scale hpc network systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):87–100, 2017. .
- Ashkan Negahban and Jeffrey S. Smith. Simulation for manufacturing system design and operation: Literature review and analysis. *Journal of Manufacturing Systems*, 33 (2):241–261, 2014. ISSN 0278-6125. .

- Duy Nguyen-Tuong, Jan Peters, and Matthias Seeger. Local Gaussian process regression for real time online model learning and control. *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*, pages 1193–1200, 2009.
- Eric C. Ni, Dragos F. Ciocan, Shane G. Henderson, and Susan R. Hunter. Efficient ranking and selection in parallel computing environments. *Operations Research*, 65(3):821–836, 2017. .
- Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. InterpretML: A Unified Framework for Machine Learning Interpretability. 2019.
- Aroonsri Nuchitprasittichai and Selen Cremaschi. An algorithm to determine sample sizes for optimization with artificial neural networks. *American Institute of Chemical Engineers Journal*, 59(3):805–812, 2013. .
- Varun Kumar Ojha, Ajith Abraham, and Václav Snášel. Metaheuristic design of feedforward neural networks: A review of two decades of research. *Engineering Applications of Artificial Intelligence*, 60(December 2016):97–116, 2017. ISSN 09521976. .
- Bhakti Stephan Onggo, Angel A. Juan, Navonil Mustafee, Andi Smart, and Owen Molloy. Symbiotic simulation system: Hybrid systems model meets big data analytics. In *Proceedings - Winter Simulation Conference*, pages 1358–1369, 2018. ISBN 9781538665725. .
- Néstor Ordaz, David Romero, Dominic Gorecky, and Héctor R Siller. Serious games and virtual simulator for automotive manufacturing education & training. *Procedia Computer Science*, 75:267–274, 2015.
- Roshan Panahi, Joseph Louis, Ankur Podder, and Colby Swanson. Tracking volumetric units in modular factories for automated progress monitoring using computer vision. In *Construction Research Congress 2022*, pages 822–829, 2022.
- Shrikant S Panwalkar and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.
- Jeong-Soo Park. Optimal latin-hypercube designs for computer experiments. *Journal of Statistical Planning and Inference*, 39(1):95–111, 1994. ISSN 0378-3758. .
- Edward Paulson. Sequential procedures for selecting the best one of several binomial populations. *The Annals of Mathematical Statistics*, 38(1):117–123, feb 1967. ISSN 0003-4851. .
- Giulia Pedrielli, K. Selcuk Candan, Xilun Chen, Logan Mathesen, Alireza Inanalouganji, Jie Xu, Chun Hung Chen, and Loo Hay Lee. Generalized ordinal

- learning framework (GOLF) for decision making with future simulated data. *Asia-Pacific Journal of Operational Research*, 36(6):1–35, 2019. ISSN 02175959. .
- Peter Peduzzi, John Concato, Elizabeth Kemper, Theodore R Holford, and Alvan R Feinstein. A simulation study of the number of events per variable in logistic regression analysis. *Journal of clinical epidemiology*, 49(12):1373–1379, 1996.
- Linda Pei, Barry L Nelson, and Susan Hunter. A new framework for parallel ranking & selection using an adaptive standard. In *Proceedings - Winter Simulation Conference*, pages 2201–2212. IEEE, 2018.
- Linda Pei, Barry L Nelson, and Susan R Hunter. Parallel adaptive survivor selection. *Operations Research*, 2022.
- Riccardo Pellegrini, Umberto Iemma, Cecilia Leotardi, Emilio F Campana, and Matteo Diez. Multi-fidelity Adaptive global metamodel of expensive computer simulations. *2016 IEEE Congress on Evolutionary Computation, CEC 2016*, pages 4444–4451, 2016. .
- Yijie Peng, Jie Xu, Senior Member, Loo Hay Lee, Senior Member, Jianqiang Hu, and Chun-hung Chen. Efficient simulation sampling allocation using multifidelity models. *IEEE Transactions on Automatic Control*, 64(8):3156–3169, 2019. .
- Luc Pronzato. Minimax and maximin space-filling designs: some properties and methods for construction. *Journal de la Société Française de Statistique*, 158(1):7–36, 2017.
- John Ross Quinlan. Semi-autonomous acquisition of pattern-based knowledge. *Basel Department of Computer Science, University of Sydney*, pages 159–172, 1980.
- Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6: 1939–1959, 2005. ISSN 15337928.
- Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005. ISBN 9780262256834. .
- Luke Rhodes-Leader, David J. Worthington, Barry L. Nelson, and Bhakti Stephan Onggo. Multi-fidelity simulation optimisation for airline disruption management. In *Proceedings - Winter Simulation Conference*, pages 2179–2190. IEEE, 2018. ISBN 9781538665725. .
- Roland Rosen, Stefan Boschert, and Christoph Heinrich. Next generation digital twin. In *Proceedings of TMCE*, volume 60, page 86, 2018. ISBN 9789461869104. .
- Susan M. Sanchez, Paul J. Sanchez, and Hong Wan. Work smarter, not harder: A tutorial on designing and conducting simulation experiments. WSC '20, page 1128–1142. IEEE Press, 2021. ISBN 9781728194998.



- Mahmood Shafiee. Maintenance strategy selection problem: an mcdm overview. *Journal of Quality in Maintenance Engineering*, 2015.
- Qinshuo Shen, Faridaddin Vahdatikhaki, Hans Voordijk, Jeffrey van der Gucht, and Lex van der Meer. Metamodel-based generative design of wind turbine foundations. *Automation in Construction*, 138:104233, 2022.
- SimPy. Simpy. <https://simpy.readthedocs.io>, 2020.
- Jeffrey S. Smith. Survey on the use of simulation for manufacturing system design and operation. *Journal of Manufacturing Systems*, 22(2):157–171, 2003. ISSN 0278-6125. .
- Jie Song, Yunzhe Qiu, Jie Xu, and Feng Yang. Multi-fidelity sampling for efficient simulation-based decision making in manufacturing management. *IIEE Transactions*, 51(7):792–805, 2019. ISSN 24725862. .
- Daniel B. Suits. Use of dummy variables in regression equations. *Journal of the American Statistical Association*, 52(280):548–551, 1957. ISSN 01621459.
- Robin Swezey, Aditya Grover, Bruno Charron, and Stefano Ermon. Pirank: Scalable learning to rank via differentiable sorting, 2021.
- Simon JE Taylor, Anastasia Anagnostou, Nura Tijjani Abubakar, Tamas Kiss, James DesLauriers, Gabor Terstyanszky, Peter Kacsuk, Jozsef Kovacs, Shane Kite, Gary Pattison, et al. Innovations in simulation: Experiences with cloud-based simulation experimentation. In *Proceedings - Winter Simulation Conference*, pages 3164–3175. IEEE, 2020.
- Matthias Thürer, George Huang, Mark Stevenson, Cristovao Silva, and Moacir Godinho Filho. The performance of due date setting rules in assembly and multi-stage job shops: An assessment by simulation. *International Journal of Production Research*, 50(20):5949–5965, 2012. .
- Edwin R. van Dam, Bart Husslage, Dick den Hertog, and Hans Melissen. Maximin latin hypercube designs in two dimensions. *Operations Research*, 55(1):158–169, 2007. .
- Gaofeng Gary Wang. Adaptive response surface method using inherited latin hypercube design points. *Journal of Mechanical Design*, 125(2):210–220, 06 2003.
- Shiyong Wang, Jiafu Wan, Di Li, and Chunhua Zhang. Implementing smart factory of industrie 4.0: An outlook. *International Journal of Distributed Sensor Networks*, 12(1): 3159805, 2016.
- Yuanhang Wang, Chao Deng, Jun Wu, Yingchun Wang, and Yao Xiong. A corrective maintenance scheme for engineering equipment. *Engineering Failure Analysis*, 36: 269–283, 2014. ISSN 1350-6307. .

- Jie Xu, Si Zhang, Edward Huang, Chun-Hung Chen, Loo Hay Lee, and Nurcin Celik. Efficient multi-fidelity simulation optimization. In *Proceedings - Winter Simulation Conference*, pages 3940–3951. IEEE, dec 2014. ISBN 978-1-4799-7486-3. .
- Jie Xu, Edward Huang, Liam Hsieh, Loo Hay Lee, Qing Shan Jia, and Chun Hung Chen. Simulation optimization in the era of Industrial 4.0 and the Industrial Internet. *Journal of Simulation*, 10(4):310–320, 2016a. ISSN 17477786. .
- Jie Xu, Si Zhang, Edward Huang, Chun Hung Chen, Loo Hay Lee, and Nurcin Celik. MO2TOS: Multi-iffdelity optimization with ordinal transformation and optimal sampling. *Asia-Pacific Journal of Operational Research*, 33(3):1–26, 2016b. .
- Jun Xu and Hang Li. Adarank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '07*, page 391–398, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595935977. .
- Te Xu, Dug Hee Moon, and Seung Geun Baek. A simulation study integrated with analytic hierarchy process (ahp) in an automotive manufacturing system. *Simulation*, 88(4):450–463, 2012.
- Taejong Yoo, Hyunbo Cho, and Enver Yücesan. Web services-based parallel replicated discrete event simulation for large-scale simulation optimization. *Simulation*, 85(7): 461–475, 2009. ISSN 00375497. .
- Si Zhang, Loo Hay Lee, Ek Peng Chew, Jie Xu, and Chun-Hung Chen. A simulation budget allocation procedure for enhancing the efficiency of optimal subset methods. *IEEE Transactions on Automatic Control*, 61(1):95–27267, 2016.
- Si Zhang, Jie Xu, Loo Hay Lee, Ek Peng Chew, Wai Peng Wong, and Chun Hung Chen. Optimal computing budget allocation for particle swarm optimization in stochastic optimization. *IEEE Transactions on Evolutionary Computation*, 21(2):206–219, 2017. ISSN 1089778X. .
- Xuqian Zhang and Wenhua Zhu. Application framework of digital twin-driven product smart manufacturing system: A case study of aeroengine blade manufacturing. *International Journal of Advanced Robotic Systems*, 16(5):1–16, 2019. ISSN 17298814. .
- Zhengmin Zhang, Zailin Guan, Yeming Gong, Dan Luo, and Lei Yue. Improved multi-fidelity simulation-based optimisation: application in a digital twin shop floor. *International Journal of Production Research*, 60(3):1016–1035, 2022.
- Yinchao Zhu, Giulia Pedrielli, and Loo Hay Lee. TD-OCBA: Optimal computing budget allocation and time dilation for simulation optimization of manufacturing systems. *IIEE Transactions*, 51(3):219–231, 2019. ISSN 24725862. .

---

Aleksei V. Zhukov, Denis N. Sidorov, and Aoife M. Foley. Random forest based approach for concept drift handling. *Communications in Computer and Information Science*, 661:69–77, 2017. ISSN 18650929. .