Master Thesis Business Analytics

Solving the Buffer Allocation Problem in Tandem Lines with Machine Learning Techniques

Nanne Dieleman
January 2021

Avanade Netherlands bv
Orteliuslaan 1000, 3528 BD
Utrecht
Supervisor: Jordy Gierveld

Vrije Universiteit Amsterdam
Faculty of Science, Business Analytics
De Boelelaan 1081a, 1081 HV
Amsterdam
Supervisor: Joost Berkhout
Second Reader: Alessandro Zocca

Preface

This thesis serves as the conclusion of the Master Business Analytics. The research is presented that was conducted throughout an internship period at Avanade, a joint venture of Microsoft and Accenture. Avanade is a leading Microsoft technology consultancy that helps businesses by providing innovative solutions. This research was conducted at the Analytics department of Avanade Netherlands.

This thesis is about solving the Buffer Allocation Problem (BAP), an NP-hard combinatorial optimisation problem, that is often solved to design queuing systems. A thorough study is conducted on the development of a neural network that estimates the throughput of tandem lines, and this neural network is combined with an evolutionary algorithm to solve BAP. The developed solution methods can be used by Avanade to optimise cloud network designs of customers.

I would like to sincerely thank my thesis supervisor at the VU, Joost Berkhout, and my supervisor at Avanade, Jordy Gierveld. Their help and support throughout my internship period was invaluable and much appreciated. I would also like to thank my second reader, Alessandro Zocca, for his time and feedback.

Keywords: Buffer Allocation Problem, Machine Learning, Neural Networks, Evolutionary Algorithms, Metamodeller, Stochastic Systems, Queuing Theory.

Abstract

The Buffer Allocation Problem (BAP) is an NP-hard combinatorial optimisation problem that is often solved to design queuing networks. One of the problems of BAP is that the throughput of a queuing network cannot be determined analytically in most cases. Therefore, a BAP solution method consists of an evaluative method and a generative method. The evaluative method is used to evaluate the throughput of the system, and the generative method is used to find a (near) optimal buffer allocation. Basically any optimisation algorithm can be used as generative method. Simulations are often used as evaluative methods, but these are difficult to build and have extensive running times. Other evaluative methods usually only work for very specific queuing network or are complicated to implement. Machine learning techniques such as neural networks present new opportunities to create evaluative methods.

In this thesis, a neural network metamodeller is developed that estimates the throughput of many *different* tandem lines (with different rates, number of stations, etc.). It is used in combination with an Evolutionary Algorithm (EA) to find a good buffer allocation. A thorough study is conducted on what kind of data is required to create a neural network that can estimate the throughput of many different tandem lines. Moreover, theoretical queuing knowledge is added to the data to analyse whether this has a positive impact on the performance of the neural network.

The results indicate that the neural network metamodeller is very well capable of estimating the throughput of tandem lines. The proposed method therefore expands the current state-of-the-art BAP solution methods by being able to estimate the throughput of many different tandem lines with a high accuracy. Moreover, adding theoretical data to the data sets improves the performance of the metamodeller. A good buffer allocation can be found by combining the developed metamodeller with an EA. Additionally, it reaches these solutions in a fraction of the time that a classic simulation-based EA requires to find a solution. The proposed methods can be used by Avanade to optimise the design of cloud networks.

Summary

In this thesis, a solution method based on machine learning techniques is developed for the Buffer Allocation Problem (BAP). The buffer allocation problem is an NP-hard combinatorial optimisation problem that arises in the design of queuing systems. Given a certain number of available buffers, the goal is to allocate these buffers over the system such that the steady-state throughput (the number of products that are produced per time unit) is maximised. The buffers serve as slots where products can wait if they are not in service. If all the buffers in front of a station are full, blocking of newly arriving products occurs. This reduces the throughput of the system and should therefore be avoided.

A specific type of queuing systems is considered, namely the tandem line. A tandem line consists of I stations in tandem. A product has to visit all stations before it can leave the system. Blocking at a station occurs if all buffers in front of the station are full. In this thesis, the Single Server Buffer Allocation Problem (SSBAP) and Multi-Server Buffer Allocation Problem (MSBAP) are considered. SSBAP only considers single server stations, whereas MSBAP also considers multi-server stations. This makes MSBAP a more complex problem than SSBAP.

A BAP solution method consists of two elements: an evaluative method and a generative method. The evaluative method determines the throughput that corresponds to a certain buffer allocation, and the generative method tries to find the best buffer allocation by using the throughput (estimate) of the evaluative method.

In case of a tandem line with buffers, the throughput achieved by a certain buffer allocation cannot be determined analytically. This means that estimation methods have to be used to estimate the throughput. Simulations are often used, but these are complex to build and have long running times. In the literature, also other methods are proposed, but these are usually designed for specific queuing systems, and therefore not generally applicable.

Machine learning techniques can be leveraged to create new evaluative methods. There are currently a few papers that also use machine learning techniques as evaluative methods, but these assume very specific queuing systems and do not provide a thorough study on what kind of data is required to create an evaluative method. In this thesis, neural networks are developed that can estimate the throughput of a wide variety of different tandem lines (e.g. different number of stations, servers, service rates, etc.), which has not been studied before. Furthermore, a study is conducted to determine what kind of data is suitable as training data for the neural network. The results suggest that it is best to create a data set where each data row corresponds to a different tandem line. The test set contains total enumerations of a few tandem lines. There is thus a difference between the kind of data that is used to train the neural network and to test the neural network, which is not conform "classical" machine learning in which the test set is very similar to the train set. However, the mean Relative Absolute Error (RAE) of a train set consisting of total enumerations lies around 10-20%, whereas it lies around 2% for a train set where each row corresponds to a different tandem line.

Theoretical knowledge based on queuing theory is added to the data sets as features, which aids the performance of the neural networks enormously. Experiments are also conducted that use the estimates of multiple neural networks to create a final throughput estimate. The results suggest that using fourteen neural networks and taking the mean

of the estimates results in a lower error than using a single neural network. The final result is a metamodeller that can accurately estimate the throughput of many different unseen tandem lines. The mean RAE values of the throughput approximations of the final metamodellers lie around 0.70% on validation sets.

In principle, any optimisation algorithm can be used as generative method. In this thesis, an Evolutionary Algorithm (EA) is used. EAs are randomised algorithms that can be applied in many different settings. Various methods exist that can be used to optimise the algorithm itself (for example crowding methods). The developed evolutionary algorithm uses an integer representation, uniform crossover, an *inc_dec* mutation operator and a crowding strategy. The algorithm is very effective: in a test problem with more than 100000 different buffer allocations, the algorithm finds the true maximum by considering only 0.67% of all solutions.

The metamodeller and the evolutionary algorithm are also combined and tested on several test instances. The results suggest that this combination is effective. Moreover, the results are compared to a simulation-based evolutionary algorithm. The metamodeller-based EA often has a better accuracy, and the simulation-based EA is also much slower than the metamodeller-based EA. For example, on one test instance the metamodeller combined with the EA takes 2 minutes to run (including training of the metamodeller which takes one and a half minute), whereas the simulation-based EA takes 13 minutes to run. The mean error of the simulation-based EA is 0.0248, whereas it is equal to 0.0147 for the metamodeller-based EA on this particular instance.

As an experiment, a binary neural network is also developed. Given data on two different buffer allocations of the same tandem line, this neural network chooses which of the two allocations is better. The results of the experiment suggest that this method performs worse than the developed regression neural network, though.

This research was conducted at Avanade, a large global consultancy that helps customers to improve their processes based on the Microsoft platform. Currently, an important topic for Avanade is to help customers migrate to the cloud. A key advantage of the cloud is that it is easy to scale resources when the load exceeds the capacity. However, this also has a downside. Customers tend to not optimise the design of their cloud solutions anymore, as capacity can easily (and even automatically) be upgraded if required. This thus increases the costs of the cloud solutions. The results of this thesis can be used to help customers improve the design of these cloud solutions, and hence reduce the costs.

An example use case is the design of an Extract, Transform, Load (ETL) process of a customer of Avanade. This customer recently moved to the cloud, which resulted in a shift from a single resource (a database) to multiple cloud solutions (data loading factories, databases, etc), each with their own capacities and buffers. This new network of cloud solutions can be modelled as a (FIFO) queue, and is ideal to be optimised by solving BAP. Currently, when the capacity of the system is not sufficient, extra data warehouses/databases/etc. are added to the network, whereas the capacity itself can also be allocated much more effectively. The costs will then be reduced considerably. Moreover, optimising the capacity will also increase the throughput of the network, which will reduce the costs even further. Thus, the developed methods presented in this thesis can be used to reduce the operating costs of the clients of Avanade, and thus increase the value delivered by Avanade.

A real-time dashboard has also been developed that further demonstrates the value of this thesis. It provides insights into a real-time production line, and the developed metamodeller-based EA can be run within the dashboard to optimise the production line.

To conclude, in this thesis it is demonstrated that neural networks can be used to estimate the throughput of a large variety of tandem lines. Moreover, adding theoretical knowledge as features to the data sets helps the training of the neural network. Thirdly, the neural network combined with the EA has comparable (and usually better) performance to a simulation-based EA, whilst having a much lower running time. The developed solution method can also be used to solve other problems, such as the Server Allocation Problem (SAP).

Contents

1 Introduction

The Buffer Allocation Problem (BAP) is a widely studied combinatorial optimisation problem. It is often used to optimise the design of production lines of, for example, factories (Qudeiri, Yamamoto, Ramli, and Al-Momani, 2007). In the buffer allocation problem one tries to find an optimal distribution of buffer slots in a queuing network to attain the highest possible throughput (i.e. the number of products that are produced per time unit). BAP is an NP-hard problem (Weiss et al., 2019). The number of possible buffer allocations rapidly increases with a larger number of stations and buffer slots, and total enumeration of the solution space is then not a feasible solution method. Moreover, the throughput corresponding to a certain buffer allocation cannot be determined analytically in most cases. This means that computationally expensive simulations or other estimation methods have to be used to estimate the throughput. As these methods only provide *estimates* of the throughput, it is always unknown whether the true maximum has been found by a solution method. This type of (stochastic) problem is thus much more difficult to solve than a deterministic combinatorial optimisation problem.

These aspects make solving BAP particularly difficult, and many methods have been proposed in the literature to do so. To date, however, few studies have examined the use of machine learning to solve BAP. Nonetheless, techniques such as neural networks do provide new opportunities to tackle the classic problem in different ways. Therefore, the objective of this thesis is to use machine learning techniques, and in particular neural networks, to solve BAP.

Parallel tandem lines such as the one depicted in Figure 1 are analysed. In front of each station, buffer slots are available. Products wait at these buffers if all machines at the station are in service. If the buffer places at a station are full, then the station is said to be *blocked*. New products cannot wait at a buffer slot and have to wait at the previous machine until a slot becomes available. This previous machine cannot produce a new product, and the blocking of stations thus reduces the number of products that can be produced per time unit, i.e. the throughput, of the tandem line. This is disadvantageous and should therefore be avoided.

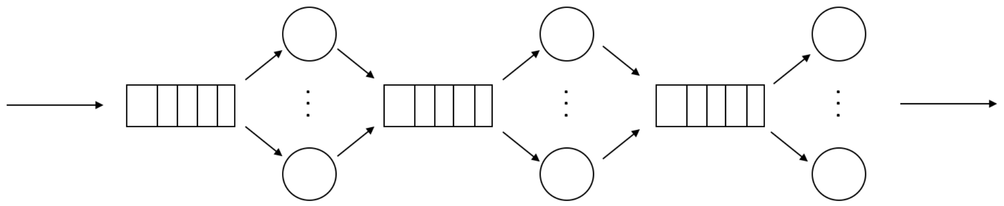


Figure 1: Schematic illustration of a tandem line with three stations.

A solution method of BAP usually consists of an evaluative method that estimates the throughput corresponding to a certain buffer allocation, and a generative method that tries to find the optimal allocation by analysing and considering different buffer allocations (see Figure 2). Many different BAP solution methods have been proposed in the literature, and these will be discussed thoroughly.

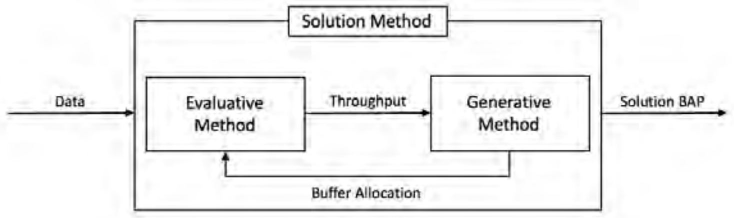


Figure 2: The general outline of a BAP solution method.

In this thesis, a neural network metamodeller is developed that estimates the throughput of a tandem line given a certain buffer allocation. This neural network is then used in combination with an Evolutionary Algorithm (EA) to find an allocation of the buffers that maximises the throughput. The neural network that is developed is able to provide throughput estimates for a wide variety of tandem lines, which sets the developed method apart from the current literature, as only neural networks that consider one specific tandem line have been developed up til now. This will make it possible to use the same neural network to generate estimates of many different tandem lines.

A lot of mathematical theory on queues and tandem lines is available. This knowledge may help the neural network to learn how to estimate the throughput of a tandem line. Therefore, features based on mathematical relations from queuing theory are added to the training data. It is then investigated whether this indeed has a positive influence.

As an experiment, a binary neural network is also developed. Given data on two different buffer allocations of the same tandem line, this neural network chooses which of the two allocations is better.

All in all, the following research questions will be answered in this thesis:

- Main question: Is it possible to use neural networks to estimate the throughput of a variety of tandem lines?
- Sub-question 1: Can the training of the neural network be improved by using theoretical knowledge on queuing systems?
- Sub-question 2: Does the solution of a neural network combined with an evolutionary algorithm outperform the solution of a simulation-based evolutionary algorithm?

This research was conducted at Avanade, a leading Microsoft technology consultancy. Avanade helps companies to improve their processes by offering services and solutions based on the Microsoft platform (such as cloud computing in Azure or Dynamics 365). Avanade was founded in 2000 as a joint venture of Microsoft and Accenture. Avanade is based in 24 countries all over the world and has more than 39000 employees. In the Netherlands, Avanade helps large companies in, for example, the financial, retail and infrastructure sector.

Avanade helps their customers by improving their business processes. An important effort of Avanade is to help customers to migrate to the cloud. Important (data) processes are then migrated from on-premises data centres to the solutions offered by Microsoft (for example SQL databases or virtual machines). This results in a network of software solutions, and these networks can be modelled as mathematical queuing networks.

Migrating to the cloud has many advantages, and one of them is that scaling of resources is incredibly easy to do. In the cloud, if the load of a certain service becomes too high, one can simply (even automatically) allocate extra resources. Traditionally in an on-premises data centre, scaling of resources is much more difficult. By moving to the cloud, companies have changed the way in which they develop their solutions, as extra capacity can be bought almost immediately. However, allocating extra resources also has a downside, as it is quite costly to do so. Therefore, instead of buying extra resources if the capacity is reached, it is better to allocate the capacity of existing resources carefully. The methods that are developed in this thesis can be used to do so. This will reduce the operation costs of the cloud solution, and results in Avanade providing extra value to their customers. The author was part of the Analytics department throughout her internship period.

In Section 2, general theory about tandem networks is explained. In Section 3, the buffer allocation problem is discussed in-depth, an extensive literature review is provided and the used Discrete Event Simulator (DES) is considered. The methodologies for creating the neural networks and the evolutionary algorithm are discussed in Section 4 and 5, respectively. In Sections 6 and 7 the developed methods are applied to the single server and multi-server buffer allocation problem, respectively. A real-time dashboard has been developed, which will be considered in Section 8. An analysis about how to apply the developed methods to cloud networks is provided in Section 9. The results of an experiment with a binary neural network are provided in Section 10. This thesis is then concluded in Sections 11 and 12 with a conclusion and discussion, and ideas for future research, respectively. If the reader has little time, then it is advised to read the summary and Section 6.

2 Problem Description: Tandem Networks

2.1 Notation

The following notation is used throughout this thesis:

- I = the number of stations of the tandem network.
- S_i = the number of servers at station i.
- C_i = the number of buffers at station i.
- $K_i = S_i + C_i$.
- B = the total number of available buffers.
- μ_i = the service rate per server at station i.
- λ = the external arrival rate at the first station.
- N = the number of data points of a data set.
- π = the stationary distribution of the number of customers at a station (including in-service customers).
- X_i = random variable of the stationary distribution of the number of customers at station i (including in-service customers).

This notation is also incorporated in Figure 3 that will be considered in Section 2.3.

2.2 Queuing Theory

Queuing models are stochastic models of queuing systems. They are often used to model real-life systems and they can be used to analyse, monitor and optimise these systems. They are, for instance, used to improve health care (Palvannan and Teow, 2012), manufacturing (Govil and Fu, 1999) and even cloud computing (C. Chen et al., 2015).

A queuing model consists of four elements: the number of servers, the service distribution and rate, the arrival process and rate, and (possibly) the available buffer space. In many cases, the arrival process is a homogeneous Poisson process. A Poisson process is a counting process with elegant properties, such as independent and exponentially distributed interarrival times. The service distribution is often exponential, but it can in principle be any continuous or discrete distribution. The available buffer spaces determine how many customers can be in the system. If all servers are used and all buffer spaces are taken by customers, the queuing system is said to be *blocked*, and newly arriving customers immediately leave the system. A queuing system may also be *starved*, which means that a station cannot serve new customers due to a lack of customers. In many cases, an infinite buffer space is used for modelling purposes, but in this thesis the emphasis is on queues with finite buffer spaces.

The regular notation for queues is Kendall's notation, which quantifies the four described elements of a queue. A queue with homogeneous Poisson arrivals, exponential service rates, S servers and C buffer places is indicated by $M/M/S/S+C$, or in shorthand $M/M/S/K$, where $K = S + C$. The same queue without a buffer restriction is indicated by $M/M/S$. More combinations are possible, but these are not used in this thesis.

An important element of queuing theory is the determination of the stationary distribution of a queue. This stationary distribution can be used to determine many different steady-state performance measures, such as the mean waiting time or the mean number of customers in the system. This may be very useful when one tries to optimise systems. In the following, the stationary distributions of the queues that are considered in this thesis are provided. Let $\pi(x)$ be the probability that there are x customers in the system. Appendix E contains the derivations.

M/M/1 The stationary distribution π of an $M/M/1$ queue with service rate μ_1 is given by:

$$\pi(x) = \Big(1 - \frac{\lambda}{\mu_1}\Big)\Big(\frac{\lambda}{\mu_1}\Big)^x, \text{ for } x \geq 0. \tag{1}$$

M/M/S The stationary distribution π of an $M/M/S$ queue with service rate μ_1 is given by:

$$\pi(x) = \begin{cases} \big(\frac{\lambda}{\mu_1}\big)^x \frac{1}{x!}\pi_0 & \text{for } x = 0,1, ..., S-1 \\ \big(\frac{\lambda}{\mu_1}\big)^x \frac{1}{S!}\frac{1}{S^{x-S}}\pi_0 & \text{for } x \geq S, \end{cases} \tag{2}$$

where

$$\pi_0 = \left(\left(\sum_{x=0}^{S-1}\Big(\frac{\lambda}{\mu_1}\Big)^x \frac{1}{x!}\right) + \Big(\frac{\lambda}{\mu_1}\Big)^S \frac{1}{S!}\frac{1}{1-\frac{\lambda}{S\mu_1}}\right)^{-1}. \tag{3}$$

M/M/1/K The stationary distribution π of an $M/M/1/K$ queue with service rate μ_1 is given by:

$$\pi(x) = \frac{1-\frac{\lambda}{\mu_1}}{1-\Big(\frac{\lambda}{\mu_1}\Big)^{K+1}}\Big(\frac{\lambda}{\mu_1}\Big)^x, \text{ for } 0 \leq x \leq K. \tag{4}$$

M/M/S/K The stationary distribution π of an $M/M/S/K$ queue with service rate μ_1 is given by:

$$\pi(x) = \begin{cases} (\frac{\lambda}{\mu_1})^x \frac{1}{x!}\pi_0 & \text{for } x = 1, 2, ..., S-1 \\ (\frac{\lambda}{\mu_1})^x \frac{1}{S!}\frac{1}{S^{x-S}}\pi_0 & \text{for } x = S, ..., K, \end{cases} \tag{5}$$

where

$$\pi_0 = \left(\left(\sum_{x=0}^{S-1}\Big(\frac{\lambda}{\mu_1}\Big)^x \frac{1}{x!}\right) + \frac{1}{S!}\sum_{x=S}^{K}\Big(\frac{\lambda}{\mu_1}\Big)^x \frac{1}{S^{x-S}}\right)^{-1} \tag{6}$$

2.3 Tandem Network Theory

A tandem network is a network of queues. Figure 3 is a schematic illustration of a tandem line with three stations with the notation that is used in this thesis. A customer arrives at the network and joins the buffer (also known as queue) of the first station. If there are no other customers in the queue and if a server is available, the customer is served by a server. If the service is finished, the customer joins the next buffer, and this process repeats itself until the customer has visited all stations. In this thesis, $M/M/S_i$ and $M/M/S_i/K_i$ (where $K_i = S_i + C_i$) tandem networks are considered. This indicates that the number of servers and buffers may be different at each station i. A tandem network consisting of $M/M/S_i$ queues has infinite buffer capacity at each station, and a network consisting of $M/M/S_i/K_i$ queues has finite buffer capacity $C_i = K_i - S_i$. A tandem line is said to be *unbalanced* if the service distributions and rates are not equal at all stations. Most tandem lines that are considered in this thesis are unbalanced.

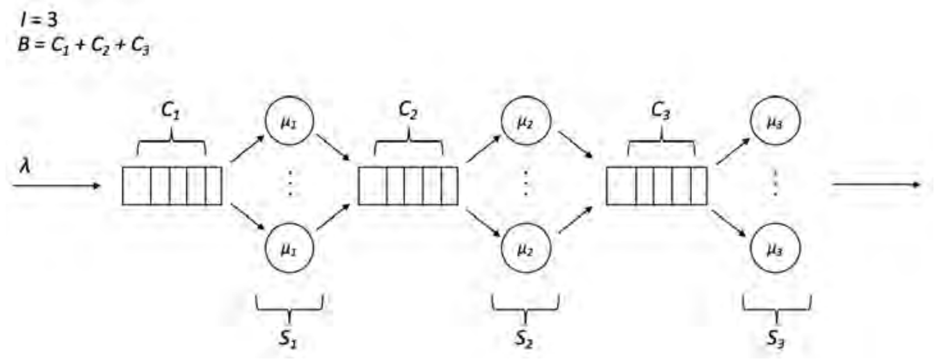


Figure 3: Schematic illustration of a tandem network with three stations.

2.4 $M/M/S_i$ Tandem Networks

In the case of $M/M/1$ and $M/M/S_i$ tandem lines, relatively easy product forms exists that quantify the stationary distributions of these queues. Moreover, stationary performance measures are easy to define. This is due to Burke's theorem (Burke, 1956), which states that the output of a stable ($\lambda < \mu$) and steady-state $M/M/S_i$ and $M/M/\infty$ queue also follows a Poisson process with the same rate as the arrival rate and that stations in a $M/M/S_i$ tandem queue are independent.

The stationary distribution of an $M/M/S_i$ tandem network is simply the product of the marginal distributions (see for example Tijms (2003) for a proof). Let $\pi(x_1, ..., x_I)$ be the probability that there are x_1 customers at station 1, x_2 customers at station 2, etc. By using (1) in case of an $M/M/1$ tandem line, its stationary distribution π is thus given by:

$$\pi(x_1, ..., x_I) = \prod_{i=1}^{I} \Big(1 - \frac{\lambda}{\mu_i}\Big)\Big(\frac{\lambda}{\mu_i}\Big)^{x_i}, \text{ for } x_i \geq 0. \tag{7}$$

By using (2) in case of an $M/M/S_i$ tandem line the stationary distribution π is given by:

$$\pi(x_1, ..., x_I) = \prod_{i=1}^{I} \pi_i(x_i), \tag{8}$$

where:

$$\pi_i(x_i) = \begin{cases} \big(\frac{\lambda}{\mu_i}\big)^{x_i} \frac{1}{x_i!} \pi_{0_i} & \text{for } x_i = 0,1, \ ..., \ S_i - 1 \\ \big(\frac{\lambda}{\mu_i}\big)^{x_i} \frac{1}{S_i!} \frac{1}{S_i^{x_i - S_i}} \pi_0 & \text{for } x_i \geq S_i, \end{cases} \tag{9}$$

where π_{0_i} is equal to:

$$\pi_{0_i} = \left(\left(\sum_{x=0}^{S_i-1} \left(\frac{\lambda}{\mu_i}\right)^x \frac{1}{x!} \right) + \left(\frac{\lambda}{\mu_i}\right)^{S_i} \frac{1}{S_i!} \frac{1}{1-\frac{\lambda}{S_i\mu_i}} \right)^{-1}. \tag{10}$$

It is also easy to formulate performance measures, as the performance measures of single $M/M/S_i$ queues can be used. An example is the mean waiting time at the queue of station i of a tandem line of $M/M/1$ queues:

$$E[W_i^q] = \frac{\frac{\lambda}{\mu_i}}{\mu_i(1-\frac{\lambda}{\mu_i})}, \tag{11}$$

which is the same as the mean waiting time at the queue of a single $M/M/1$ queue. Performance measures will form an important part of this thesis, as they will be used as features to train the neural networks. This will be further discussed in Section 4.5.

2.5 $M/M/S_i/K_i$ Tandem Networks

No closed-form solutions are available for the distribution or throughput of $M/M/S_i/K_i$ tandem networks. This makes it difficult to evaluate the performance of the networks and many algorithms and methods have been developed to analyse this type of tandem queues. These methods will be further discussed in the literature review of Section 3.2.

An important difference between $M/M/S_i$ and $M/M/S_i/K_i$ tandem networks is that blocking of customers occurs in case of a $M/M/S_i/K_i$ tandem. This is the case if upon completion of a service at station i, station $i+1$ is full. Several blocking types exist, and the most well-known ones are Blocking After Service (BAS) and Blocking Before Service (BBS). These types are also known as type-2 and type-3 blocking, respectively.

With BBS, a customer that is admitted to a server checks whether the next station is blocked, if not, then the service starts. If the station is blocked, the service cannot start and the customer waits until the next station is unblocked. With BAS, once a customer has been served, he checks whether the next station is blocked. If so, then this customer stays at his current server until the next station is unblocked. Service cannot be started for a new customer at this server, only when the blocked customer can proceed to the next station. This blocking type is used most often in manufacturing lines (Dallery and Gershwin, 1992), and therefore also used in this thesis. Equivalences exist between the two blocking types, see for example the work of Onvural and Perros (1986) for relations between blocking types in case of tandem lines and several other queues.

3 Problem Description: The Buffer Allocation Problem (BAP)

3.1 Problem Definition Buffer Allocation Problem (BAP)

The Buffer Allocation Problem (BAP) is a frequently studied problem in combinatorial optimisation. In BAP, a queuing network is considered that has limited slots for products, customers, etc. at each station. The goal is to allocate the buffers to each station in such a way to achieve a certain objective, which is usually to maximise the throughput of the system.

There are two main formulations of BAP. The first formulation is called the Primal problem, which is also known as BAP 2 and BAP B (Demir et al. (2014) and Weiss et al. (2019)). In this Primal problem, the objective is to minimise the number of buffers that are used to achieve a certain predefined throughput. The most basic formulation is as follows:

$$\begin{aligned} \min \quad & \sum_{i=1}^{I} C_i \\ \text{s.t.} \quad & \tau((C_1, ..., C_I)) = \tau^* \\ & C_i \in \mathbb{N}^0, \quad i = 1, ..., I, \end{aligned}$$

where τ^* is the predefined throughput and $\tau((C_1, ..., C_I))$ the throughput corresponding to buffer allocation $(C_1, ..., C_I)$. The second formulation is known as the Dual problem, BAP 1 and BAP A. The goal is to maximise the throughput given a predefined number of buffers. The formulation is as follows:

$$\begin{aligned} \max \quad & \tau((C_1, ..., C_I)) \\ \text{s.t.} \quad & \sum_{i=1}^{I} C_i = B \\ & C_i \in \mathbb{N}^0, \quad i = 1, ..., I \end{aligned}$$

One may make some extra assumptions on the problems, which results in additional constraints in both formulations. For example, a minimum number of buffers at each station may be added as an extra constraint.

BAP often arises in manufacturing problems, when a production line is designed or optimised, for example in the optimisation of an ink-jet printers manufacturing system (Burman et al., 1998). Buffers are used in production lines to reduce the influence of starvation and blocking in the line, which increases the number of products that can be produced per time unit. However, simply allocating a large number of buffers is not satisfactory, as this is costly or not even feasible due to space constraints in the physical factory. Therefore, it is important to find a good buffer allocation.

The high complexity of BAP is expressed in the fact that it is an NP-hard problem (Demir et al., 2014). For small instances, a total enumeration of the solutions is still feasible, but this rapidly becomes impossible with larger systems. Therefore, over the years many scientific methods have been proposed in the literature to solve BAP. These will be discussed in the following.

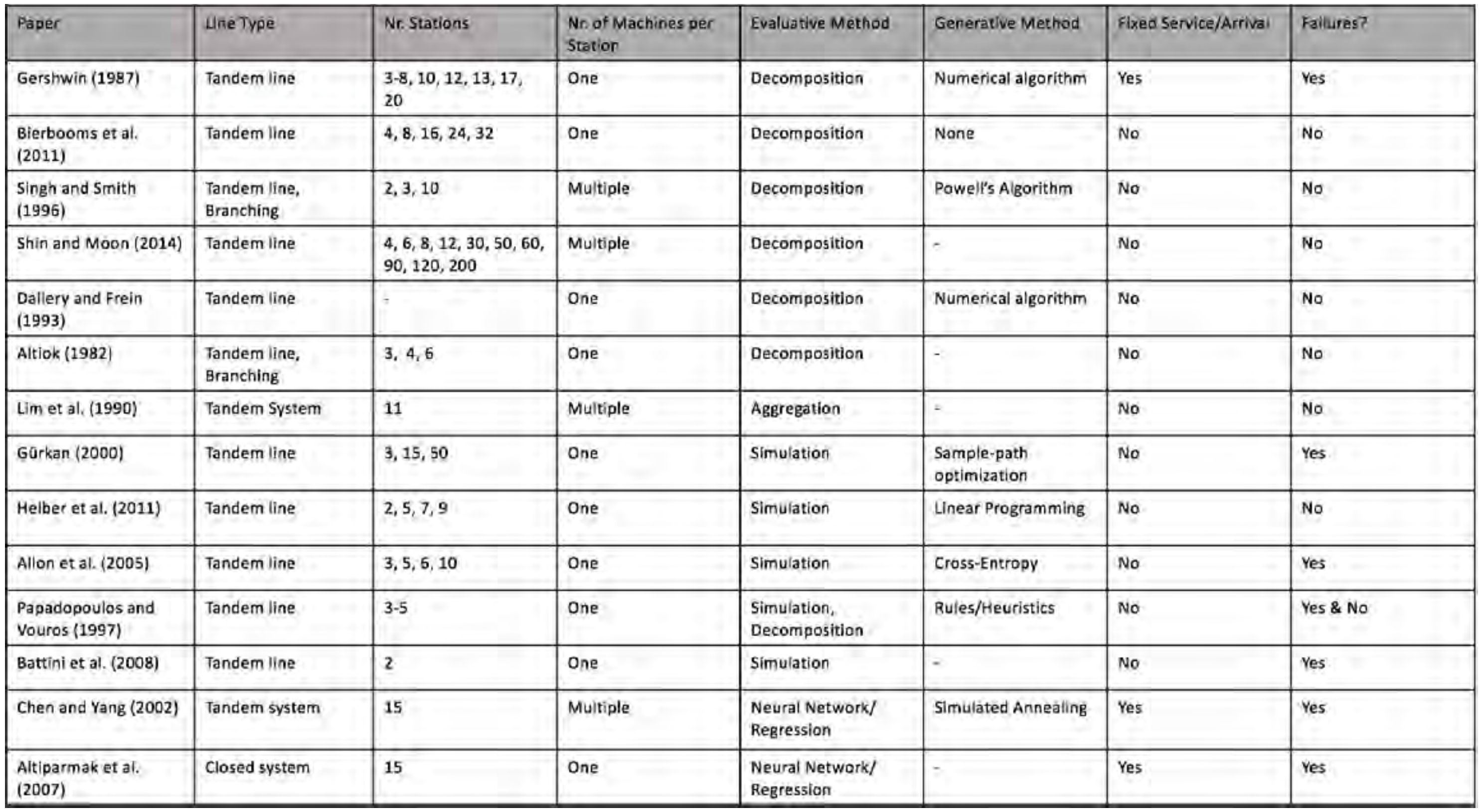
In this thesis, BAP will be considered for tandem lines of $M/M/S_i/K_i$ queues. A single server type is considered (thus with $M/M/1/K_i$ queues), which is referred to as the Single Server Buffer Allocation Problem (SSBAP), and a multi-server type is considered (with $M/M/S_i/K_i$ queues) which is called the Multi- Server Buffer Allocation Problem (MSBAP).

3.2 Literature Overview

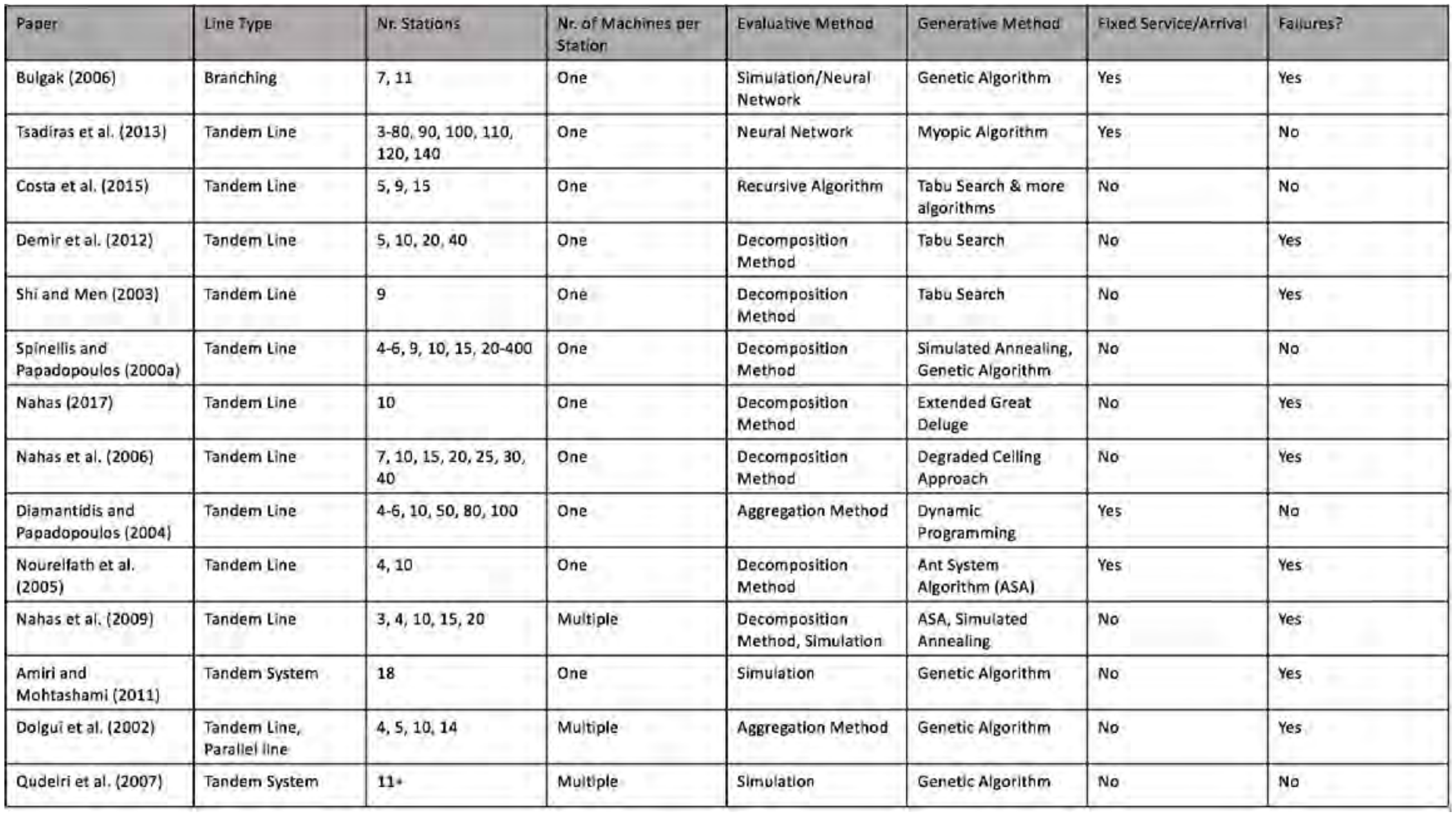
A rich literature is available on solving BAP. In general, a solution method consists of an evaluative method and a generative method. The evaluative method is used to evaluate the value of the function that is optimised, which is usually the throughput of the system, and the generative method is then used to explore the search space. This division of the solution methods is often made in the literature (Demir et al. (2014), Spinellis and C. Papadopoulos (2000a), H. Papadopoulos and Vouros (1997)) and is therefore also adopted in this thesis.

Table 4 contains an overview of the considered papers and their characteristics. The difference between a *"Tandem Line"* and *"Tandem System"* in the *"Line Type"* column is that a tandem system is a larger and more complicated system than a tandem line. It may, for example, be a model of a whole factory with multiple branched tandem lines. A tandem line is the same system as described in Section 2.3.

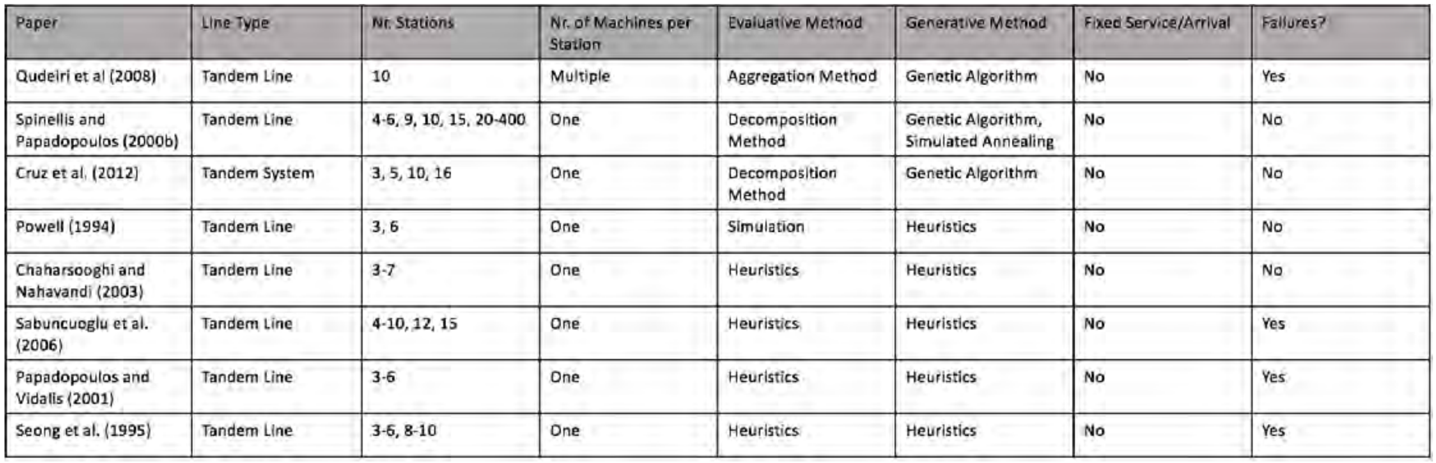
Most of the papers consider tandem lines with one machine per station (thus SSBAP). Whether machine failures are or are not considered is equally divided over the papers. Four main evaluative methods are observed, namely the decomposition method, aggregation method, simulation and neural networks. Most papers use the decomposition method as evaluative method, and a wide variety of generative methods is considered. Only a handful of papers compare the performance of various generative methods (see for example Spinellis and C. Papadopoulos (2000b)) and most papers thus consider the performance of only one generative method. The size of the tandem lines is usually between two and ten, but some papers consider tandem lines of up to 400 stations. There are no real benchmark sets of buffer allocation problems, although some papers consider the problems of other papers. In the following paragraphs, a summary is provided of each paper, and the various methods are compared. Moreover, concluding remarks are provided that set the proposed solution method apart from the current state-of-the-art solution methods.



Paper	Line Type	Nr. Stations	Nr. of Machines per Station	Evaluative Method	Generative Method	Fixed Service/Arrival	Failures?
Gershwin (1987)	Tandem line	3-8, 10, 12, 13, 17, 20	One	Decomposition	Numerical algorithm	Yes	Yes
Bierbooms et al. (2011)	Tandem line	4, 8, 16, 24, 32	One	Decomposition	None	No	No
Singh and Smith (1996)	Tandem line, Branching	2, 3, 10	Multiple	Decomposition	Powell's Algorithm	No	No
Shin and Moon (2014)	Tandem line	4, 6, 8, 12, 30, 50, 60, 90, 120, 200	Multiple	Decomposition	-	No	No
Dallery and Frein (1993)	Tandem line	-	One	Decomposition	Numerical algorithm	No	No
Altiok (1982)	Tandem line, Branching	3, 4, 6	One	Decomposition	-	No	No
Lim et al. (1990)	Tandem System	11	Multiple	Aggregation	-	No	No
Gürkan (2000)	Tandem line	3, 15, 50	One	Simulation	Sample-path optimization	No	Yes
Helber et al. (2011)	Tandem line	2, 5, 7, 9	One	Simulation	Linear Programming	No	No
Alon et al. (2005)	Tandem line	3, 5, 6, 10	One	Simulation	Cross-Entropy	No	Yes
Papadopoulos and Vouros (1997)	Tandem line	3-5	One	Simulation, Decomposition	Rules/Heuristics	No	Yes & No
Battini et al. (2008)	Tandem line	2	One	Simulation	-	No	Yes
Chen and Yang (2002)	Tandem system	15	Multiple	Neural Network/ Regression	Simulated Annealing	Yes	Yes
Altiparmak et al. (2007)	Closed system	15	One	Neural Network/ Regression	-	Yes	Yes



Paper	Line Type	Nr. Stations	Nr. of Machines per Station	Evaluative Method	Generative Method	Fixed Service/Arrival	Failures?
Bulgak (2006)	Branching	7, 11	One	Simulation/Neural Network	Genetic Algorithm	Yes	Yes
Tsadiras et al. (2013)	Tandem Line	3-80, 90, 100, 110, 120, 140	One	Neural Network	Myopic Algorithm	Yes	No
Costa et al. (2015)	Tandem Line	5, 9, 15	One	Recursive Algorithm	Tabu Search & more algorithms	No	No
Demir et al. (2012)	Tandem Line	5, 10, 20, 40	One	Decomposition Method	Tabu Search	No	Yes
Shi and Men (2003)	Tandem Line	9	One	Decomposition Method	Tabu Search	No	Yes
Spinellis and Papadopoulos (2000a)	Tandem Line	4-6, 9, 10, 15, 20-400	One	Decomposition Method	Simulated Annealing, Genetic Algorithm	No	No
Nahas (2017)	Tandem Line	10	One	Decomposition Method	Extended Great Deluge	No	Yes
Nahas et al. (2006)	Tandem Line	7, 10, 15, 20, 25, 30, 40	One	Decomposition Method	Degraded Ceiling Approach	No	Yes
Diamantidis and Papadopoulos (2004)	Tandem Line	4-6, 10, 50, 80, 100	One	Aggregation Method	Dynamic Programming	Yes	No
Nourelfath et al. (2005)	Tandem Line	4, 10	One	Decomposition Method	Ant System Algorithm (ASA)	Yes	Yes
Nahas et al. (2009)	Tandem Line	3, 4, 10, 15, 20	Multiple	Decomposition Method, Simulation	ASA, Simulated Annealing	No	Yes
Amiri and Mohtashami (2011)	Tandem System	18	One	Simulation	Genetic Algorithm	No	Yes
Dolgui et al. (2002)	Tandem Line, Parallel line	4, 5, 10, 14	Multiple	Aggregation Method	Genetic Algorithm	No	Yes
Qudeiri et al. (2007)	Tandem System	11+	Multiple	Simulation	Genetic Algorithm	No	No



Paper	Line Type	Nr. Stations	Nr. of Machines per Station	Evaluative Method	Generative Method	Fixed Service/Arrival	Failures?
Qudeiri et al (2008)	Tandem Line	10	Multiple	Aggregation Method	Genetic Algorithm	No	Yes
Spinellis and Papadopoulos (2000b)	Tandem Line	4-6, 9, 10, 15, 20-400	One	Decomposition Method	Genetic Algorithm, Simulated Annealing	No	No
Cruz et al. (2012)	Tandem System	3, 5, 10, 16	One	Decomposition Method	Genetic Algorithm	No	No
Powell (1994)	Tandem Line	3, 6	One	Simulation	Heuristics	No	No
Chaharsooghi and Nahavandi (2003)	Tandem Line	3-7	One	Heuristics	Heuristics	No	No
Sabuncuoglu et al. (2006)	Tandem Line	4-10, 12, 15	One	Heuristics	Heuristics	No	Yes
Papadopoulos and Vidalis (2001)	Tandem Line	3-6	One	Heuristics	Heuristics	No	Yes
Seong et al. (1995)	Tandem Line	3-6, 8-10	One	Heuristics	Heuristics	No	Yes

Figure 4: Literature table.

3.2.1 Decomposition Methods

Decomposition methods are often used as the evaluative method to solve BAP. They are based on the idea to decompose a tandem queue into a sequence of single or two server queues, and to evaluate the performance of each component of this sequence. How to decompose the tandem queue into single components constitutes the main effort of this method. A disadvantage is that specific assumptions about the queue have to be made to be able to use a decomposition method, for example on the service distribution or arrival process. However, these methods are not as computationally intensive as, for example, simulation methods.

Pioneering work in this area was produced by Gershwin (1987) already in 1987. In the proposed method, a single server queue with I stations is decomposed into $I - 1$ queues with two servers. A numerical algorithm determines the parameters of the $I - 1$ new queues.

Another good example of a decomposition method is the work of Bierbooms et al. (2013). In this work, single-server tandem queues are decomposed into smaller subsystems, while taking blocking and starvation into account. The smaller subsystems contain a server, a buffer and another server. The service rates of both stations also include the effects of starvation and blocking. These service rates are found by using an iterative algorithm.

Singh and J. M. Smith (1997) use a different decomposition of the system. They model the blocking at a station as an $M/M/S/\infty$ queue with rerouting if the next $M/M/S/K$ station is still full after completion of a service at this artificial $M/M/S/\infty$ blocking queue. An algorithm estimates the parameters of the decomposition. They apply this method on several different queuing networks, amongst which are a tandem queue and a triangular queue.

Shin and Moon (2014) consider yet another decomposition method. They decompose the tandem network into smaller components consisting of an artificial server, buffer, server, buffer and another server. Their method is best applied to small tandem networks, as the speed of convergence of their algorithm is lower for larger tandem networks.

Lastly, Dallery and Frein (1993) provide an overview of the working of decomposition methods. Moreover, they discuss and analyse several different numerical algorithms for using the decomposition method. They prove several properties of these algorithms, such as convergence properties.

There are many more papers on the decomposition method, see for example Altiok (1982). The discussed papers demonstrate that a large variety of decomposition methods exist. Each method is specifically designed for a certain queuing network, which makes a specific decomposition method not generally applicable.

3.2.2 Aggregation Methods

Aggregation methods are closely related to decomposition methods. However, instead of decomposing a queuing network into smaller subnetworks, the queuing network is aggregated into a queue with only one station. Examples can be found in Dolgui et al. (2002) and Lim et al. (1990).

Lim et al. (1990) propose an aggregation method for reliable production systems with complex routing. The asymptotic theory that they develop is not computationally intensive, and quite accurate.

Qudeiri, Yamamoto, Ramli, and Jamali (2008) consider a tandem line with multiple machines per station. They first aggregate these multiple machines into one machine, such that the tandem line consists of single machine stations, and then use the aggregation method proposed in J. Li (2005) to evaluate the throughput of the network.

3.2.3 Simulation Optimisation

Simulations are often used to evaluate unknown functions, and are also evaluative methods. Simulations are simple to comprehend, but building a realistic and error-free simulation is time consuming. Moreover, they are often computationally expensive, which usually makes them less suitable, especially in case of large problems. Stochastic simulations (a.k.a. Monte Carlo simulations) make it possible to take the randomness of the modelled process into account.

Simulation optimisation is a field in which optimisation problems are solved by simulating the function that has to be optimised. An in-depth review on simulation optimisation in general is given by Amaran et al. (2016). The works of Nsoesie et al. (2013) and Schwartz et al. (2006) are interesting examples of using simulation optimisation in case of health care and inventory management, respectively. Discrete Event Simulation (DES) and Continuous Simulation (CS) are the most often used simulation methods. In a DES, events are scheduled at discrete time points, and the simulator jumps through these scheduled events. A continuous simulator considers time on a continuous scale.

Gürkan (2000) uses simulation optimisation to solve BAP. He considers a continuous version of BAP, in which the products that are produced are continuous instead of discrete (e.g. producing fluids). He uses a specific type of simulation, namely sample path simulation. With this method, one (albeit very long) sample path is considered as an approximation of the function that is optimised, and the derivative that is required for optimisation is computed by using this deterministic sample path. He is able to determine the derivative by using a generalised semi-Markov process as a model of the evolution of a discrete-event simulation. The advantage of this method is that only a few evaluations are necessary to optimise the function.

Helber et al. (2011) propose another method to use simulations as evaluative method, namely to combine DES with linear programming. A Linear Program (LP) is solved that combines producing the output of a simulation and optimising the buffer allocation. They formulate a discrete-time LP with decision variables for the quantity of products that are produced in a time period, the buffer allocation, and the number of waiting products at the end of a time period. In order to be able to solve the LP, they assume a certain number of products that can be produced per time period. This value is determined by sampling many (random) production times and counting the number of completed products per time period. The resulting model provides relatively accurate approximations of the throughput of the queue. However, the method is not very effective in case of a small number of available buffers, or in case of large variability in the service times.

Allon et al. (2005) combine simulation with the cross-entropy method. Their algorithm randomly generates buffer allocations conform a cross-entropy minimisation mechanism, which is updated based on simulation results of the generated buffer allocations.

Battini et al. (2009) use simulations to gain insight into the influence of unreliable machines in a tandem line. They propose to allocate buffers in such a way as to minimise the down-time of the tandem line instead of maximising the throughput.

A model management system that optimises production systems is created by H. Papadopoulos and Vouros (1997). One of the solution methods within this management system is also a simulation. The papers of Amiri and Mohtashami (2012) and Qudeiri, Yamamoto, Ramli, and Al-Momani (2007) also use simulation optimisation to solve BAP. Amiri and Mohtashami (2012) create a metamodeller of a discrete event simulator by a two-level factorial design. They use this metamodeller to estimate the throughput corresponding to a certain buffer allocation. Qudeiri, Yamamoto, Ramli, and Al-Momani (2007) use a discrete event simulator to estimate the throughput. The papers are further discussed in Section 3.2.5.

3.2.4 Machine Learning Models

The popularity and effectiveness of machine learning models in the scientific literature has also given rise to the use of machine learning models to solve BAP. Most papers use a neural network as metamodeller that estimates the throughput of the considered system. In this way, considerable computation time is saved that would normally be used by the simulator to evaluate the throughput.

Already in 1999, Kilmer, A. Smith, and Shuman (1999) used a neural network metamodeller as substitute of a simulation of an (s, S) inventory system. They compare the performance of their neural network with a simulator, and report quite reasonable results. Kuo et al. (2007) use a Neural Network (NN) as metamodeller of a wafer fabrication assembly system. They use a very small training data set, as they want to create a time-efficient system. This results in a relatively large error on the validation set, but also in a model that is fast to train and to use. Lechevalier et al. (2015) use a NN as metamodeller to estimate the energy consumed by a machine. Moreover, they propose a framework to create a NN metamodeller in general. Can and Heavy (2012) compare using a NN as metamodeller to using a Genetic Program (GP) performing symbolic regression as metamodeller. They consider three different use cases, amongst which is a production line. The GP provides slightly better results, especially extrapolation-wise. However, it is also more computationally intensive.

M. Chen and Yang (2002) combine a neural network metamodeller with simulated annealing to optimise the configuration on a manufacturing plant. There are six parameters that can be set to maximise the production rate of the plant, which makes it a difficult optimisation problem. They establish that regression metamodellers perform worse than NN metamodellers.

Altiparmak et al. (2007) use a NN to solve BAP in a closed tandem system. They consider three types of systems, one with identical stations, one with several fallible stations and one with only fallible stations. They create a separate NN for each system type. It should be noted that only the failure rates of the machines (may) have different values for each machine, and that for example the service rates are fixed.

Bulgak (2006) combines a Genetic Algorithm (GA) with a neural network to solve BAP in a split-and-merge assembly system. They compare the performance of a GA in combination with a simulator to the performance of a GA with a neural network. The failure rates are also the only parameters that are not identical in this paper. The GA and NN combination is much faster than the GA and simulator combination, whereas the performance is comparable.

A Decision Support System (DSS) that solves BAP and uses a NN as metamodeller is created by Tsadiras et al. (2013). They consider 9382 experiments that are used to train, test and validate the DSS. This means that the total computation time that is required to create the DSS is considerable (running times of more than 164 days are reported). However, once these computations have been performed and the DSS created, new users can use it right-away.

To conclude, neural networks have often been used as metamodellers as a substitute for simulations. Reasonable to excellent results have been achieved in the literature. However, the current literature mainly focuses on *using* the neural network as metamodeller instead of providing a thorough analysis of what does or does not work when such a metamodeller is created. Moreover, BAP is solved for specific systems with fixed parameters (except for failure rates), which means that the same neural network cannot be used for different systems and a new one should be created for each system. In this thesis, a metamodeller is created that is able to estimate the throughput of many different systems. Furthermore, a thorough analysis is provided to determine what kind of data is suitable to develop such a metamodeller.

3.2.5 Search Algorithms

The second component of a BAP solution method is the generative method. Usually search algorithms are used as the generative component. Search algorithms help to explore the solution space in a particular, and often efficient way.

Costa et al. (2015) consider the primal buffer allocation problem and apply Tabu search as their search algorithm. Tabu search is based on a local search algorithm. The algorithm evaluates solutions in a so-called neighbourhood and determines whether the best solution in a neighbourhood is better than the current solution. Worse solutions can be accepted by the algorithm to escape from local optima and a Tabu list is created that saves visited solutions. This list is then used to prohibit the algorithm from considering these solutions again. The Tabu algorithm developed in Costa et al. (2015) is parallelised: instead of considering one solution trajectory, three different solution trajectories are considered. These trajectories evolve independently, but occasionally transfer information. This algorithm can also be implemented by using parallel computing, which makes it a fast algorithm.

Demir et al. (2012) also use Tabu search to solve BAP. In contrast to Costa et al. (2015), Demir et al. (2012) consider an unreliable production line in which machines may break down. Moreover, they propose an adaptive Tabu search algorithm that changes its behaviour on the current state of the search. New solutions are created by swapping a certain number of buffers from one station to another. The number of buffers that are swapped depends on the size of the instance, but is also determined by the search algorithm. If a certain solution remains to be the best solution for several iterations, then the number of buffers that are swapped are reduced. However, if more diverse solutions are needed, a diversification method is used that restarts the search.

Shi and Men (2003) propose yet another method based on Tabu Search to solve BAP. They combine Nested Partitions (NP) with Tabu Search. The NP method is a search method that divides regions into (promising) subsets that are evaluated by a random sampling scheme. If one of the subsets is better than the current solution then this subset will be partitioned. If none of the subsets is better, then the method backtracks to a larger region. They use Tabu Search on top of the random sampling scheme to find better solutions in each subset.

Spinellis and C. Papadopoulos (2000a) use simulated annealing in combination with a decomposition method to solve BAP. The decomposition method of Dallery and Frein (1993) is used as evaluative method. Simulated Annealing is a search method that tries to reduce the probability that a local optimum is found. It does so by accepting worse solutions with a certain probability, which reduces over time. In Spinellis and C. Papadopoulos (2000a), new solutions are created by moving a randomly selected number of buffers from one station to another.

Extended Great Deluge (EGD) is another search algorithm that has been applied to solve BAP (Nahas, 2017). It is a local search method that uses a specific approach to accept or reject new solutions. In Nahas (2017) the buffer allocation problem is combined with a maintenance problem and a solution is provided that maximises the throughput while also taking machine failures into account.

In Nahas, Ait-Kadi, et al. (2006) the degraded ceiling approach is used to solve BAP in an unreliable environment. The degraded ceiling algorithm is a local search algorithm that accepts solutions below a certain ceiling C value, which decreases over time. They compare their method to a simulated annealing implementation, which it outperforms.

Diamantidis and C. Papadopoulos (2004) use Dynamic Programming (DP) to solve BAP. They use the aggregation method proposed by Lim et al. (1990) to evaluate the throughput and they combine this with a DP algorithm. The algorithm is a classic DP algorithm that uses a recursive equation to find the solution. The algorithm does not consider bad solutions, which reduces the computation time. Both the error and the running time of the proposed method is small for the considered tandem system.

The ant system algorithm is closely related to evolutionary algorithms, and used by M. Nourelfath and Ait-Kadi (2005) to solve BAP. They use a decomposition method to evaluate the throughput. An ant system algorithm is based on pheromone trails that ants produce to lead their colony towards food. New solutions are created based on a heuristic, and changed according to the pheromone trails. Better solutions will have stronger pheromone trails, which pushes the search towards better solutions. The buffer allocation problem they consider is slightly more difficult than the classic problem, as they also allow for machine selection. In Nahas, Nourelfath, et al. (2009), the any system algorithm is compared to simulated annealing. They find that simulated annealing outperforms the general ant system algorithm, but that an altered version of the ant system algorithm outperform simulated annealing.

Several publications on using evolutionary algorithms to solve BAP are also available. Examples are the works of Amiri and Mohtashami (2012), Dolgui et al. (2002), Qudeiri, Yamamoto, Ramli, and Al-Momani (2007), Qudeiri, Yamamoto, Ramli, and Jamali (2008), Spinellis and C. Papadopoulos (2000b), and Cruz et al. (2012). An interesting observation is that most of these papers consider tandem systems instead of tandem lines. Tandem systems are larger queuing networks with (possibly) rerouting, branching and parallelism. Evolutionary algorithms turn out to be highly effective to solve BAP, as they are flexible and can consider a large part of the search space in a relatively short period of time. Moreover, they can also be used to solve a multi-objective problem, as for instance Amiri and Mohtashami (2012) do. They want to maximise throughput and minimise the number of used buffers at the same time. Most papers either use an integer or binary encoding of the solutions. However, for example Qudeiri, Yamamoto, Ramli, and Al-Momani (2007) consider an advanced production line and cannot use integer and binary encoding. They therefore use a matrix encoding method in which the columns and rows resemble specific parts of the production line. Tailored recombination and mutation parameters are required when this encoding is used.

3.2.6 Heuristics

Heuristics are simple rules or algorithms that may help to solve a (difficult) problem. Heuristics do not solve a problem analytically, but provide a simple way to reach reasonably good solutions.

Powell (1994) provides several rule of thumbs for allocating buffer storage to unbalanced lines. He considers both μ and σ unbalance in the service rate, and finds, for example, that placing a buffer close to a station with σ-unbalance and the other buffers alternately works well.

Chaharsooghi and Nahavandi (2003) propose a heuristic algorithm to solve SSBAP. They find a buffer configuration by considering each station of the network consecutively. The number of buffers at each station is selected by finding the number of buffers that make that the station behaves as an $M/M/1/\infty$ queue. Their algorithm is relatively easy to implement and shows good solutions. However, the number of buffers is not part of the input of the algorithm, which is a disadvantage if a manufacturer has a specific number of buffers available. They find that their approach is especially valuable in case of large queuing systems.

Sabuncuoglu, Erel, et al. (2006) propose a heuristic algorithm for tandem lines with failures. They compare their numerical algorithm to six other heuristics. Moreover, they state general observations on how to allocate buffers.

Lastly, H. Papadopoulos and Vidalis (2001) also propose a heuristic algorithm for unreliable production lines. The method works for small lines of up to six stations. The method can also be used with larger lines, but then it needs to be combined with, for example, a decomposition method. The algorithm starts by finding a good initial buffer allocation by considering the parameters of the line, and then this initial allocation is improved in several steps.

More papers on heuristics that solve BAP have been written. See for example the work of Seong et al. (1995).

3.2.7 Contribution to the State-of-the-art

Evidently, a rich literature exists on solving BAP. Nevertheless, this thesis contributes to the literature in several ways. First of all, a neural network will be created that can estimate the throughput of many different queuing systems (differing in arrival rate, service rates, number of stations, number of servers, etc.). In the current literature, BAP is only solved by metamodellers for queuing systems with fixed parameters. Secondly, the method will be applicable to both single and multi-server tandem lines, and also to, for example, the Server Allocation Problem (SAP). Especially multi-server tandem lines have not been investigated extensively in the literature. Thirdly, a thorough study is conducted to determine what kind of training data is suitable for training such a neural network. Fourthly, the influence of adding theoretical queuing data to the training data is analysed. Fifthly, as an experiment, a binary neural network is created that does not perform regression, but directly tries to choose the best of two different buffer allocations. This has not been done before. Sixthly, by using an evolutionary algorithm in combination with a neural network, the opportunity is created to use the developed method on many more problems, as evolutionary algorithms are versatile optimisation algorithms. An example would be to solve the joint BAP and Server Allocation Problem (SAP), in which both the buffer allocation and the server allocation of a queuing system is optimised.

3.3 Discrete Event Simulator

The data that is used in this thesis is generated by using a Discrete Event Simulator (DES). The reason for this is two-fold, first of all, no data was available at the company, and secondly, a DES provides the flexibility to create all types and quantities of data, which is crucial to develop the neural network.

3.3.1 The Implementation

The DES was implemented in Python with the use of the *SimPy* package (Lünsdorf and Scherfke, 2020). A schematic illustration of the DES is provided in Figure 5. The upper part of the figure presents a small queuing network with two stations. The first station has two servers and the second station has three servers. Customers arrive according to a homogeneous Poisson process. If the buffer at the first station is full upon the arrival of a new customer, then the customer immediately leaves the system. If not, he/she joins the buffer at station one. If a server at station one becomes available and there are no customers in front of the customer, the customer proceeds to a server and is helped. If upon completion of this service, the next station is not blocked, the customer proceeds to the next station. If the next station is blocked, the customer waits at the server until the next station becomes available again. Note that this means that a service of a new customer can only be started if the customer proceeds to the next station, in accordance with BAS blocking.

The lower part of the figure shows the used Python components and methods. An arrival is simulated by generating an exponential random number as interarrival time. This interarrival time is subsequently scheduled by *SimPy* by the `env.timeout()` command. Once the interarrival time has passed, a customer arrives at the system and is added to a *SimPy* Store object if the first buffer is not full, otherwise the new customer immediately leaves the system. Once a server is available, the customer is removed from the *SimPy* store object and assigned to a *SimPy* resource object. A service time is simulated by generating an exponential random number. This service time is then also scheduled by `env.timeout()`. Once the service is finished, the customer is either moved to the second *SimPy* Store object, or, if the next station is blocked, waits until the next station is unblocked and is then moved to the second *SimPy* Store object. This process repeats until the customer leaves the system.

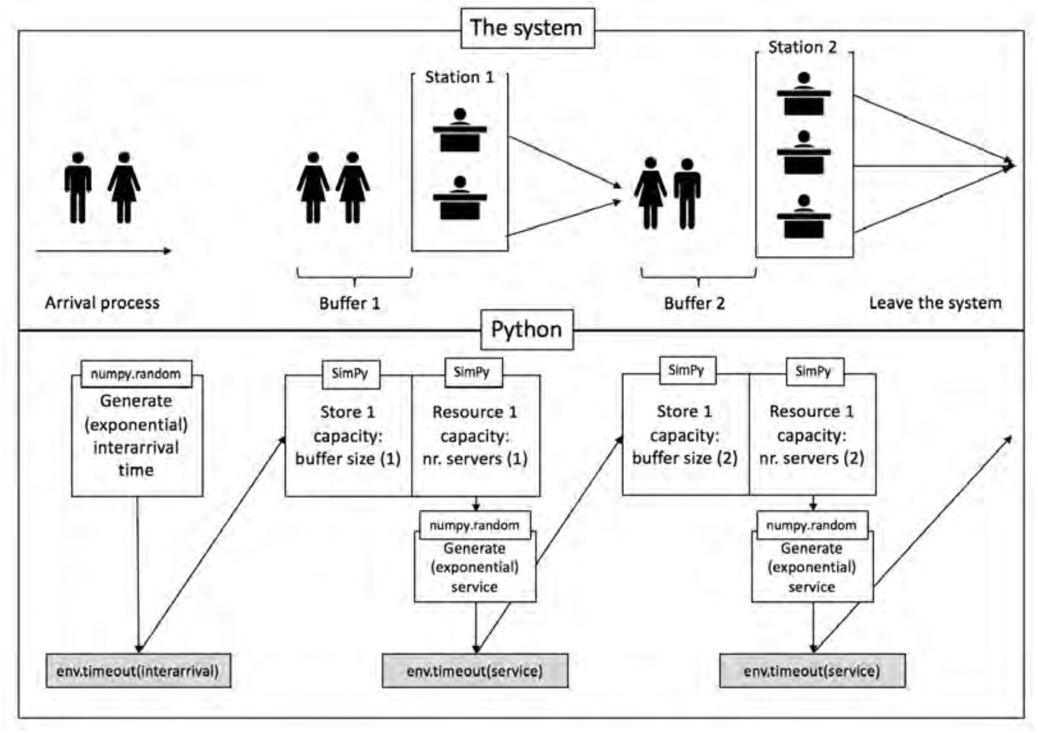

The system
Station 1
Station 2
Arrival process
Buffer 1
Buffer 2
Leave the system
Python
numpy.random
Generate (exponential) interarrival time
SimPy
Store 1 capacity: buffer size (1)
SimPy
Resource 1 capacity: nr. servers (1)
SimPy
Store 1 capacity: buffer size (2)
SimPy
Resource 1 capacity: nr. servers (2)
numpy.random
Generate (exponential) service
numpy.random
Generate (exponential) service
env.timeout(interarrival)
env.timeout(service)
env.timeout(service)

Figure 5: Schematic illustration of a network and the Python components and actions.

The most important task of the DES is to estimate the throughput of the considered queuing networks given a certain buffer allocation. The DES is used as an aid to generate problem instances and data sets. For example, test sets are generated by executing a total enumeration of all the possible buffer configurations of a certain queuing system and saving the throughput that is calculated by the DES corresponding to each buffer configuration. The calculated throughput will be considered the ‘true’ throughput to be able to assess the performance of the neural networks and the evolutionary algorithms.

3.3.2 Throughput and Stability

The throughput of a certain queuing network is determined by the DES by monitoring and saving the number of customers that finish every time unit and by taking the average over these saved numbers. In order to create a reliable estimate of the throughput, it is important that only the average is taken over the data when the queuing network has become stable. Figure 6 provides two examples of the average throughput over time for six different queuing networks. The left plot only considers single server queuing networks, whereas the right plot is for multi-server queuing networks. An unstable average can clearly be observed in the first 500 time units of both plots. Therefore, to estimate the throughput, the average is taken over the number of finished customers per time unit of time units 750 - 2000.

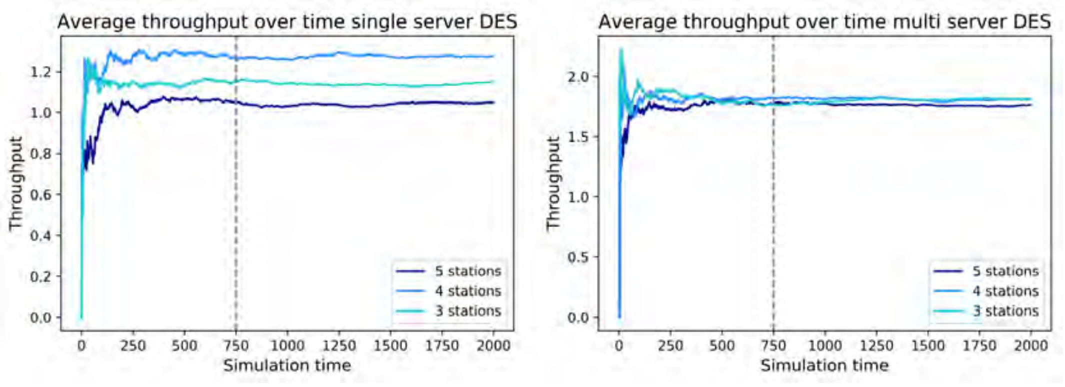


Figure 6: The throughput over time for different queuing networks.

To create a reliable estimate of the throughput, it is best to take the mean of the average throughputs of multiple independent simulations. Figure 7 presents boxplots of the average throughput for different simulation settings. The first boxplot, for example, was created by taking the average over the average throughputs obtained by 10 independent simulation runs of 1000 time units and performing this 100 times. The running time of each experiment is also reported. Clearly, simulations of 2000 time units are more reliable than simulations of 1000 time units, as the width of the boxplots is smaller. Moreover, taking the average over more independent simulation runs also increases the reliability. However, the running time also increases. Table 1 presents the 95% confidence intervals of the simulation settings. Except for simulation setting [10, 1000], all confidence intervals are reasonably small. All in all, a balance between the accuracy and running time has to be found. The running time is an important constraint, as the DES will be used in many occasions. Therefore, the simulation setting [10, 2000] is selected, which indicates that each simulation run takes 2000 time units (of which 750 time units form the warm-up period), and the average is taken over 10 independent simulation runs.

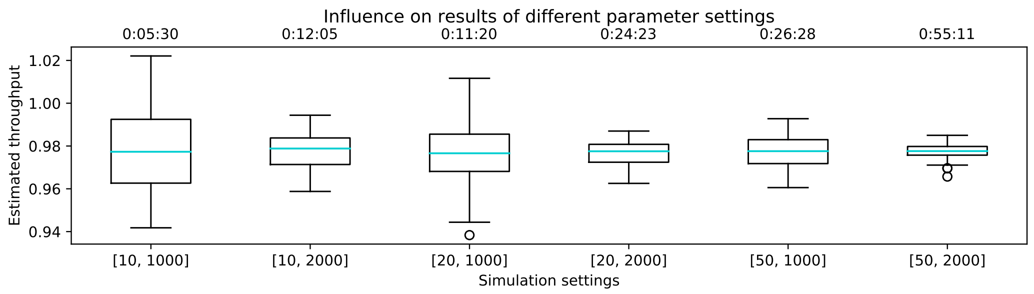


Figure 7: Boxplots of different parameter settings.

Table 1: 95% confidence intervals of the different simulation settings.

Simulation Setting	Lower Bound CI	Upper Bound CI
10, 1000	0.975	0.982
10, 2000	0.977	0.980
20, 1000	0.974	0.979
20, 2000	0.976	0.978
50, 1000	0.976	0.979
50, 2000	0.977	0.978

3.3.3 Multiprocessing

In general, it takes a lot of time to generate the required data sets. Generating one data set of 25000 data rows takes around 54 hours on a computer with a 2.7 GHz Quad-Core Intel Core i5 processor and 16 GB of memory. As many different data sets are required for this thesis, the standard multiprocessing package of Python is used to generate the data in a much faster way. The implemented solution uses all cores of the machine, and each core runs the required simulations for a particular buffer allocation. The used machine has 4 cores, and the time to generate a data set of 25000 rows reduces with multiprocessing to roughly 13 hours. The code is also transformed to an Azure Databricks workbook that uses multiprocessing. Running this workbook on the cloud computing virtual machines of Microsoft also reduces the running time. However, due to costs constraints this could not be used to a huge extent.

4 Methodology: Neural Networks (NN)

This part treats estimating the throughput of a tandem line by using a neural network. As previously discussed, no closed-form solution exists to estimate the throughput of a tandem line of $M/M/S_i/K_i$ queues. Therefore, usually simulations or other mathematical methods are used to estimate the throughput of tandem lines. However, simulations are computationally costly, and other methods may be difficult to implement, which makes developing other methods to estimate the throughput worthwhile (also see Section 3.2 for a review of the current literature). In this thesis, a neural network is developed to estimate the throughput of tandem lines of $M/M/S_i/K_i$ queues. Several neural networks will be developed and a thorough study is conducted to determine what kind of data is most appropriate to create such a neural network.

4.1 Literature

Neural Networks are perhaps the most well-known machine learning models. They are used to solve many different problems; both classification and regression problems. Examples can be found in finance ((Yu et al., 2007) and (Angelini et al., 2008)), consumer behaviour (Badea, 2014), health care (Bagnasco et al., 2015) and (micro)production (Kussul et al., 2010). Already in 1993 it was proven that feedforward neural networks with non-polynomial activation functions are universal approximators (Leshno et al., 1993), which means that these neural networks are able to approximate any function.

Neural Networks are used to perform a variety of regression tasks. For example, Leung et al. (2000) use neural networks to forecast exchange rates, Vasconcelos et al. (2001) predict the probability on fire ignition in Portugal and Cigizoglu (2003) combines ARMA models with neural networks to estimate the flow of rivers.

Neural networks have also been used to solve BAP. These methods have already been discussed in Section 3.2. In general, they have been used regularly as metamodellers, i.e. models that estimate the outcome of a simulation. Figure 8 presents a schematic illustration of the working of a metamodeller. The real system is modelled by a simulation, and the simulation is modelled by a neural network (or other method).

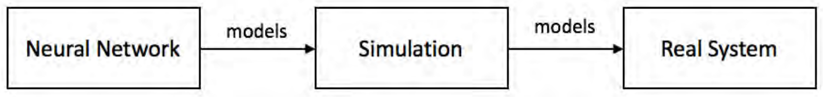


Figure 8: Schematic illustration of the working of a metamodeller.

The advantage of metamodellers is that they reduce the computation time extensively (Fonseca et al., 2003), while still attaining a relatively good accuracy. Neural networks have the advantage that they can estimate non-linear functions and that they do not require a lot of computational effort to be used (Pierreval, 1996). However, the disadvantage of metamodellers is that it may be difficult and time-consuming to develop one (Sabuncuoglu and Touhami, 2002).

The literature is full of examples of metamodellers that estimate the output of simulations and that are used to optimise systems. S. Li et al. (2021), for example, use gradient boosted decision trees as metamodellers of a simulation. They estimate service levels to optimise the staffing and scheduling at contact centres. Aussem and Hill (2000) create a neural network metamodeller that forecasts the propagation of green algae in the Mediterranean twelve years in advance. They report good accuracy that is comparable to the simulation results. Hurrion (1997) use a neural network to optimise the optimum number of kanbans in a manufacturing system. They first use the neural network to find a good initial solution, and subsequently use a simulation to find a better optimum. Metamodellers have also been used in health care: Kilmer and A. Smith (1997) use a neural network to estimate the patient time in an emergency department. Lastly, neural networks have also been used to optimise the design of large and complex systems. Shourian et al. (2008), for example, use a neural network to estimate the outcome of a network flow model to optimise the design of a river basin.

Another well-known method to estimate the outcome of a simulation is Kriging. It is an interpolation method that takes the weighted average of Gaussian stochastic processes to create an estimate. Matta et al. (2012) use Kriging as a metamodeller of a production system. Ankenman et al. (2010) develop stochastic Kriging, which can be used to create metamodellers of stochastic simulations. In general, Kriging yields good results. However, the disadvantage is that Kriging does not scale well (S. Li et al., 2021).

4.2 Neural Network Theory

Figure 9 is an illustration of a simple neural network. It consists of three types of layers: input, hidden and output layers. The number of input nodes is equal to the number of features of the data set. A neural network may have zero or multiple hidden layers and each layer contains a number of nodes. This number is flexible and may range between one to an arbitrary large number. In case of a regression neural network, the output layer usually consists of one node, as the estimated value is the only output.

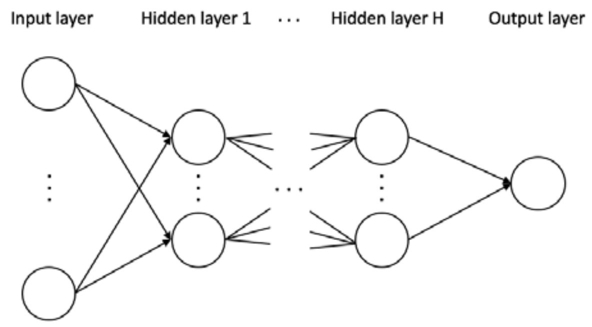


Figure 9: An illustration of a simple neural network.

The input is transformed to output by a mathematical operator. To do so, each hidden node contains an activation function that is responsible for transforming the input data into an estimate. The ReLu function is used in this thesis, which is given by:

$$f(x) = \max(0, x). \tag{12}$$

It was introduced in 2011 by Glorot et al. (2011) and is since then the most commonly used activation function for several reasons. First of all, the solution becomes sparse, because nodes may receive the value zero, and sparsity makes models simpler. Secondly, it helps to solve the vanishing gradients problem (Glorot et al., 2011), in which the gradient that is used in the backpropagation algorithm becomes almost equal to zero, halting the training of neural networks. Other activation functions also exist, such as the sigmoid and tanh functions.

The links between all nodes have weights and the goal of training a neural network is to select the weights that minimise the error function. In the case of regression, usually the squared loss function is used as error function, which is given by:

$$L = \sum_{n=1}^{N} (y_n - f_n)^2, \tag{13}$$

where y_n is the true value and f_n the estimate. A neural network is generally trained through the back-propagation algorithm. The main idea behind the back-propagation algorithm is that an input is fed to the neural network, after which the output of the neural network is compared to the true value. The error that is made is back-propagated through the network and the weights are adjusted such that the error is reduced for this input. There are several methods that can be used to run the backpropagation algorithm, and the Adam optimiser is used in this thesis. This is a stochastic gradient solver introduced by Kingma and Ba (2017). The solver has many advantages over other methods, such as computational efficiency and easy implementation. Moreover, it works well with large data sets and parameter spaces.

The neural networks are implemented by using the *Sklearn* Python package (Pedregosa et al., 2011).

4.3 Training the Neural Networks

The neural networks are trained by data sets that are generated by the DES discussed in Section 3.3. The data sets contain information on the considered queuing network (like number of servers, number of stations, arrival rates, etc.), and the throughput. Altiparmak et al. (2007), Bulgak (2006) and Tsadiras et al. (2013) all use neural networks to estimate the throughput of queuing networks. However, all these papers consider queuing networks with fixed service and arrival rates. In other words, the service rates and arrival rates are not part of the input of the neural network. In case of Tsadiras et al. (2013), for example, the service rates are assumed to be equal to 1.0 for all machines. This reduces the applicability of the neural network greatly. It is therefore worthwhile to investigate whether it is possible to train a well-performing neural network that can handle variable service rates, arrival rates and number of stations. This is not trivial to do, as a neural network cannot handle variable input size, for example.

4.3.1 The Train Sets

Several types of training data sets have been generated. With regard to the considered queuing systems, the following assumptions have been made:

- The number of stations ranges from two to five.
- The number of buffers ranges from the number of stations plus one to thirty.

- The arrivals follow a Poisson process.
- The arrival rate ranges between 0.5 and 2.5.
- The service times are exponentially distributed.
- The service rates range between the selected arrival rate and 3.0 (thus $\rho < 1$).

The reasoning behind these assumptions is that the considered systems/instances should not become too large, while still providing a complicated optimisation problem. For instance, five stations with 30 buffers already result in 23751 different buffer allocations for one specific system. Total enumeration in combination with a simulation is not feasible in this case (except if one accepts long running times of several hours or even days depending on the instance and simulation accuracy) which makes it an interesting optimisation problem.

A list of used features is provided in Appendix F. As previously indicated, it should be taken into account while generating the training data that the neural network should be able to handle variable arrival and service rates and a variable number of stations. This will influence the structure of the training data set. As the queuing networks may consist of two to five stations, not all features will be useful for all queuing networks. For example, if the queuing network consists of five stations, then all stations have buffers and service rates. However, if the network consists of only two stations, then no service rate is available for stations three to five. Feed-forward neural networks cannot handle a variable number of input nodes, as the number of input nodes is fixed by definition. Zero-padding is a natural method to handle a variable input size. With zero-padding, the features that are not available receive the value '0'. In the case of the two station network, this would mean that the features for stations three to five will receive the value '0'. In this thesis, a thorough study is performed to determine what the most suitable data is to train a neural network for this problem. In Sections 6 and 7 the performed experiments and used train sets are discussed.

4.3.2 The Test Sets

The neural network will be applied in combination with the evolutionary algorithm to find the optimal buffer configuration of a specific queuing network, with a specified number of servers, buffers and arrival and service rates. As the neural network should be able to estimate the throughput of many different queuing networks (with different service, rates, number of servers, etc.), the test set should also contain several instances of different queuing networks. Therefore, the generated test sets contains the data of multiple total enumerations. A total enumeration is an enumeration of all the possible buffer configurations for a specific queuing network, with a fixed number of stations, arrival and service rates, and number of available buffers. The throughput corresponding to each buffer configuration is determined by using the DES of Section 3.3. The test sets contain 15000 data rows of total enumerations of randomly selected queuing networks.

4.3.3 The Grid Search

In order to find the best parameters for the neural networks, grid searches are performed. The features and target are first normalised, as it is known that neural networks in general perform better with normalised features (Hoogendoorn and Funk, 2018).

Both single layer and multiple layer neural networks are considered. A 5-fold cross-validation grid search is used to select the parameters. Several performance measures are used to assess the performance of the neural network. These performance measures are based on standard regression measures, and also on ranking measures. As standard regression measures, the Mean Absolute Error (MAE), Mean Squared Error (MSE) and Relative Absolute Error (RAE) are used. The MAE is defined as:

$$MAE = \frac{\sum_{n=1}^{N} |y_n - f_n|}{N}, \tag{14}$$

where f_n is the estimate and y_n the true value. The MSE is defined as:

$$MSE = \frac{\sum_{n=1}^{N} (y_n - f_n)^2}{N}, \tag{15}$$

The RAE is defined as:

$$RAE = \frac{1}{N} \sum_{n=1}^{N} |\frac{y_n - f_n}{y_n}| * 100\%, \tag{16}$$

These three measures were selected as they are among the most popularly used for regression tasks (Botchkarev, 2019), and because they convey different knowledge. The MAE provides information on the absolute size of the errors, whereas the MSE puts more emphasis on the outliers of the estimates because the errors are squared. The RAE provides a percentage measure that enables to easily compare the performance across data sets.

Three ranking measures are considered, the Mean Absolute Deviation (MAD), the Mean Squared Deviation (MSD) and the Maximum Deviation (MD). The idea behind the ranking measures is that it is important that the neural network provides the correct ranking of the solutions when the neural network is combined with the evolutionary algorithm (as we are trying to find the buffer allocation corresponding to the optimum in the optimisation problem, not the maximum throughput itself). Moreover, it is most important that the neural network is able to correctly predict the ranks of the solutions with the highest throughput. Therefore, the MAD and MSD are scaled in such a way so that the solutions that have the highest throughput (and are thus most important) receive the most importance in these measures. The ranking measures are defined as follows:

$$MAD = \frac{\sum_{n=1}^{N} \left(\frac{N-R_n}{N}\right) |R_n - NeR_n|}{N}, \tag{17}$$

$$MSD = \frac{\sum_{n=1}^{N} \left(\frac{N-R_n}{N}\right) (R_n - NeR_n)^2}{N}, \tag{18}$$

$$MD = \max |R_n - NeR_n|, \tag{19}$$

with R_n equal to the true rank of configuration n and NeR_n equal to the rank estimated by the neural network.

4.4 Confidence Intervals

Neural networks are probabilistic models. When a neural network is trained, random initial weights are generated. During training, these weights are adapted to reduce the error. It may therefore be the case that a local minimum has been found when the training has stopped. This indicates that depending on the initial weights, different neural networks are trained. Reporting a single performance measure of a single neural network will then not provide enough insight into the performance of the neural network. Therefore, methods have been developed to create Confidence Intervals (CI) and Prediction Intervals (PI) in the literature.

As discussed in Pearce et al. (2018), the difference between CIs and PIs is the type of uncertainty that they convey. With regression, one tries to fit a function $f(x)$ to data (y) that is produced by the true function and some noise ϵ (with zero mean):

$$y = f(x) + \epsilon. \tag{20}$$

This results in three types of uncertainty, namely σ_y^2, $\sigma_{f(x)}^2$, σ_ϵ^2, that follow the relationship $\sigma_y^2 = \sigma_{f(x)}^2 + \sigma_\epsilon^2$. CIs consider $\sigma_{f(x)}^2$, whereas PIs also incorporate the uncertainty of the noise, which means that they consider σ_y^2. As PIs incorporate more uncertainty, PIs are wider than CIs (Pearce et al., 2018).

Hoogendoorn and Funk (2018) gain insights into the performance of a neural network by training X neural networks with different initial seeds for the weights and by calculating the performance measure over a test set for each trained neural network. This results in X values of the performance measure and these can then be used to create CIs.

Srivastav et al. (2007) use bootstrapping to create CIs. They sample with replacement from the train set and train a neural network on this sample. They then define a bootstrap regression function which is the average of the individual fitted bootstrap functions. The uncertainty can then be quantified by calculating a bootstrap variance that compares the individual bootstrap functions to the average bootstrap function.

Another well-known method is the Delta Method. This method uses Taylor-Expansions to quantify model uncertainty (Hoef, 2012). The disadvantage of this method is that the Hessian matrix quickly becomes too large for (deep) neural networks. Therefore, Nilsen et al. (2019) propose a new approximate delta method that also works for deep neural networks.

Gal and Ghahramani (2016) use dropout to quantify model uncertainty in deep learning. Dropout is a regularization technique that prevents a neural network to become too complex by randomly dropping neurons during training. The method of Gal and Ghahramani (2016) can be used for both regression and classification to create PIs.

Pearce et al. (2018) develop a new loss function for training the neural networks in combination with gradient descent. By using this new loss function, the upper limit of a 95% PI is simply created by taking the mean estimate of X trained neural networks, calculating the variance over these estimates, and adding 1.96 times the standard deviation to the mean.

All in all, ample methods exist to create prediction and confidence intervals. Some methods are quite complex to use and require adaptation of already implemented functions of *Sklearn*. In this thesis, the method of Hoogendoorn and Funk (2018) is used, which is chosen because of its clear interpretation and straightforward implementation.

In all cases, X is set to 100, which means that 100 neural networks with different initial weight seeds are trained.

4.5 Theoretical Data

One of the research questions is about whether adding theoretical knowledge about the queuing network to the data set helps training the neural network. In order to investigate this, theoretical knowledge has to be added to the data sets. The used features are introduced in this section.

In the following, the features are presented based on whether SSBAP or MSBAP is considered, and which queue type is used to create the features ($M/M/1$, $M/M/S_i/K_i$, etc). However, the features can also be classified in another way: buffer dependent and buffer independent features. The buffer dependent features have a different value depending on the considered buffer allocation, whereas buffer independent features do not have different values when the buffer allocation differs.

4.5.1 SSBAP

In case of SSBAP, theoretical data based on $M/M/1$ and $M/M/1/K_i$ (where $K_i = C_i+1$) queues is added to the data set. Appendix E contains the derivations of the introduced features.

$M/M/1$ **features** For the first set of features, an $M/M/1$ tandem queue is considered with I stations, arrival rate λ, service rates $(\mu_1, ..., \mu_I)$ and $\rho_i = \frac{\lambda}{\mu_i}, i = 1, ..., I$. Recall that it is assumed that $\rho_i < 1 \ \forall i$. Moreover, notice that by Burke's output theorem (Burke, 1956), the output of each stationary station is equal to λ. The operator $\sigma(x_1, ..., x_N)$ represents calculating the sample standard deviation over a data vector $(x_1, ..., x_N)$.

The following quantities based on $M/M/1$ queue steady-state performance measures are added to the data:

- The expected number of customers at station i:

$$E[L_i] = \frac{\rho_i}{1-\rho_i}. \tag{21}$$

- The expected number of customers at the queue of station i:

$$E[L_i^q] = \frac{\rho_i^2}{1-\rho_i}. \tag{22}$$

- The expected waiting time at station i:

$$E[W_i^q] = \frac{\rho_i}{\mu_i(1-\rho_i)}. \tag{23}$$

- The expected system time at station i:

$$E[S_i] = \frac{1}{\mu_i} + \frac{\rho_i}{\mu_i(1-\rho_i)}. \tag{24}$$

- The probability that more than K_i customers are present at station i:

$$P(X_i > K_i) = 1 - \sum_{x=0}^{K_i} (1-\rho_i)\rho_i^x. \tag{25}$$

- The probability that station i is idle:

$$P(X_i = 0) = 1 - \rho_i. \tag{26}$$

$$P(X_i = 0)_\sigma = \sigma(P(X_1 = 0), ..., P(X_I = 0)). \tag{27}$$

- The probability that there are exactly K_i customers at station i:

$$P(X_i = K_i) = (1-\rho_i)\rho_i^{K_i}. \tag{28}$$

$$P(X_i = K_i)_\sigma = \sigma(P(X_1 = K_1), ..., P(X_I = K_I)). \tag{29}$$

- The throughput of an $M/M/1$ tandem line:

$$t_{tp} = \min(\mu_1, ..., \mu_I). \tag{30}$$

A drawback of these features is that they are equal to zero if I is smaller than the maximum number of stations that is considered. This would, for example, be the case for a two station tandem line. The features 'corresponding' to stations three to five are all equal to zero. These features will then not be very useful for training the neural network, as a large part of these features will have value zero. Therefore, summarising features are added that summarise the behaviour of the whole tandem line by a single quantity.

$$EL_{sum} = \sum_{i=1}^{I} \frac{\rho_i}{1-\rho_i}, \tag{31}$$

$$EL_\sigma = \sigma(E[L_1], ..., E[L_I]), \tag{32}$$

$$EL^q_{sum} = \sum_{i=1}^{I} \frac{\rho_i^2}{1-\rho_i}, \tag{33}$$

$$EL^q_\sigma = \sigma(E[L^q_1], ..., E[L^q_I]), \tag{34}$$

$$EW^q_{sum} = \sum_{i=1}^{I} \frac{\rho_i}{\mu_i(1-\rho_i)}, \tag{35}$$

$$EW^q_\sigma = \sigma(E[W^q_1], ..., E[W^q_I]), \tag{36}$$

$$ES_{sum} = \sum_{i=1}^{I} \frac{1}{\mu_i} + \frac{\rho_i}{\mu_i(1-\rho_i)}, \tag{37}$$

$$ES_\sigma = \sigma(E[S_1], ..., E[S_I]). \tag{38}$$

Notice that it is an indication that the buffer allocation created an unbalanced line if the standard deviation features are high. When the features based on the performance measures of $M/M/1$ tandem lines were added to the data, it was observed that high correlations were present between the features EL_{sum}, EL^q_{sum}, EW^q_{sum} and ES_{sum}, between EL_σ, EL^q_σ, EW^q_σ and ES_σ, and between the features $E[L_i]$, $E[L^q_i]$, $E[W^q_i]$ and $E[S_i]$ $\forall i$ (see Figure 10). This will not benefit the neural network, and therefore only the EL_{sum}, $E[L_i]$ $\forall i$, EL_σ features are added to the data set, instead of all 24 features.

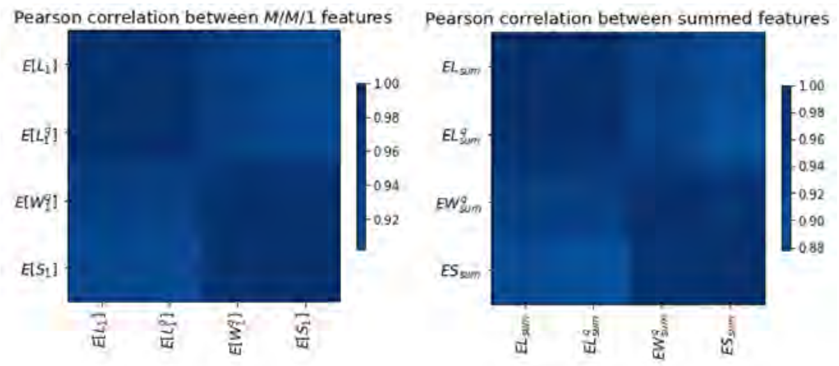


Figure 10: Correlations between several performance measures.

$M/M/1/K_i$ **features** Preliminary experiments showed the importance of the arrival rate and blocking properties of the tandem lines as features. Therefore, for the second set of features, features were designed that incorporate the blocking behaviour of the tandem queue. Consider an $M/M/1/K_i$ tandem queue with I stations, C_i buffers at station i ($K_i = C_i + 1$), arrival rate λ, service rates $(\mu_1, ..., \mu_I)$ and $\rho_i = \frac{\lambda}{\mu_i}, i = 1, ..., I$. The following features are added to the data:

- The blocking probability at station i:

$$\pi^{K_i} = \frac{1-\rho_i}{1-\rho_i^{K_i+1}}\rho_i^{K_i}, \tag{39}$$

$$\pi^K_\sigma = \sigma(\pi^{K_1}, ..., \pi^{K_I}). \tag{40}$$

- The probability of an idle server at station i:

$$\pi^{0_i} = \frac{1-\rho_i}{1-\rho_i^{K_i+1}}, \tag{41}$$

$$\pi^0_\sigma = \sigma(\pi^{0_1}, ..., \pi^{0_I}). \tag{42}$$

- The number of customers at an $M/M/1/K_i$ station:

$$E[L_i]^{MM1K} = \frac{\rho_i(K_i\rho_i^{K_i+1} - (K_i+1)\rho_i^{K_i} + 1)}{(\rho_i - 1)(\rho_i^{K_i+1} - 1)}, \tag{43}$$

$$E[L]^{MM1K}_\sigma = \sigma(E[L_1]^{MM1K}, ..., E[L_I]^{MM1K}). \tag{44}$$

As previously indicated, the arrival rate is an important feature. Notice that Burke's output theorem does not hold for tandem lines of $M/M/1/K_i$ queues, and by assuming that the arrival rate at each station is equal to λ in the previous features, a higher arrival rate is assumed than the actual arrival rate at each station. An algorithm (see Algorithm 1 below and Appendix E for the derivation) is used that incorporates the blocking probability of an $M/M/1/K_i$ queue into the arrival rate. The resulting values are referred to as effective throughput rates.

Algorithm 1: Effective throughput algorithm

```
$\gamma_0 = \lambda$
For i = 1,...,I:
    $\rho_i = \frac{\gamma_{i-1}}{\mu_i}$
    $\gamma_i = \gamma_{i-1}\Big(1 - \frac{(1-\rho_i)\rho_i^{K_i}}{1-\rho_i^{K_i+1}}\Big)$
```

The features based on $M/M/1/K_i$ tandem lines now use $\rho_i = \frac{\lambda}{\mu_i}$ as utilisation. However, due to the blocking behaviour of an $M/M/1/K_i$ queue, this utilisation is in general too high. The newly introduced effective throughput rates could also be used instead of the arrival rates. Define $\hat{\rho}_i = \frac{\gamma_{i-1}}{\mu_i}$ and consider the following features:

- The blocking probability at station i:

$$\pi_\gamma^{K_i} = \frac{1-\hat{\rho}_i}{1-\hat{\rho}_i^{K_i+1}}\hat{\rho}_i^{K_i}, \tag{45}$$

$$\pi_{\gamma\sigma}^{K} = \sigma(\pi_\gamma^{K_1}, ..., \pi_\gamma^{K_I}). \tag{46}$$

- The probability of an idle server at station i:

$$\pi_\gamma^{0_i} = \frac{1-\hat{\rho}_i}{1-\hat{\rho}_i^{K_i+1}}, \tag{47}$$

$$\pi_{\gamma\sigma}^{0} = \sigma(\pi_\gamma^{0_1}, ..., \pi_\gamma^{0_I}). \tag{48}$$

- The number of customers at an $M/M/1/K_i$ station:

$$E[L_i]_\gamma^{MM1K} = \frac{\hat{\rho}_i(K_i\hat{\rho}_i^{K_i+1} - (K_i+1)\hat{\rho}_i^{K_i} + 1)}{(\hat{\rho}_i - 1)(\hat{\rho}_i^{K_i+1} - 1)}, \tag{49}$$

$$E[L]_{\gamma\sigma}^{MM1K} = \sigma(E[L_1]_\gamma^{MM1K}, ..., E[L_I]_\gamma^{MM1K}). \tag{50}$$

All in all, for the SSBAP, 66 theoretical features are created. Appendix F contains a table with the mathematical notation and the corresponding feature names.

4.5.2 MSBAP

For MSBAP, similar features are added. However, instead of considering single server queuing systems, multi-server systems have to be considered. These queuing systems are more involved and require more mathematics. Appendix E contains all the derivations.

$M/M/S_i$ **features** For the first set of features, an $M/M/S_i$ tandem queue is considered with I stations, S_i servers per station, arrival rate λ, service rates $(\mu_1, ..., \mu_I)$ and utilisation $\rho_i = \frac{\lambda}{S_i \mu_i}$, for $i = 1, ..., I$. Define $K_i = S_i + C_i$, where S_i is the number of servers at station i and C_i the number of buffers at station i. Define the following quantity:

$$\Pi_i = \Big(\frac{\lambda}{\mu_i}\Big)^{S_i} \frac{1}{S_i!} \Bigg((1 - \rho_i) \sum_{x=0}^{S_i - 1} \Big(\frac{\lambda}{\mu_i}\Big)^x \frac{1}{x!} + \Big(\frac{\lambda}{\mu_i}\Big)^{S_i} \frac{1}{S_i!} \Bigg)^{-1} \tag{51}$$

The following features are added:

- The expected number of customers at station i:

$$E[L_i] = \frac{\rho_i}{1 - \rho_i} \Pi_i + \frac{\lambda}{\mu_i}, \tag{52}$$

$$EL_{sum} = \sum_{i=1}^{I} E[L_i], \tag{53}$$

$$EL_\sigma = \sigma(E[L_1], ..., E[L_I]). \tag{54}$$

- The probability that station i is idle:

$$\pi_{0_i}^{MMS} = \Bigg(\sum_{x=0}^{S_i - 1} \Big(\frac{\lambda}{\mu_i}\Big)^x \frac{1}{x!} + \Big(\frac{\lambda}{\mu_i}\Big)^{S_i} \frac{1}{S_i!} \frac{1}{1 - \rho_i} \Bigg)^{-1}, \tag{55}$$

$$\pi_0^{MMS_\sigma} = \sigma(\pi_{0_1}^{MMS}, ..., \pi_{0_I}^{MMS}). \tag{56}$$

- The probability on $K_i = S_i + C_i$ customers at station i:

$$\pi_{K_i}^{MMS} = \Big(\frac{\lambda}{\mu_i}\Big)^{K_i} \frac{1}{S_i!} \frac{1}{S_i^{K_i - S_i}} \pi_{0_i}^{MMS}, \tag{57}$$

$$\pi_K^{MMS_\sigma} = \sigma(\pi_{K_1}^{MMS}, ..., \pi_{K_I}^{MMS}). \tag{58}$$

- The probability of more than $K_i = S_i + C_i$ customers at station i:

$$P(X_i > K_i) = 1 - \sum_{x=0}^{S_i - 1} \Big(\frac{\lambda}{\mu_i}\Big)^x \frac{1}{x!} \pi_{0_i}^{MMS} - \sum_{x=S_i}^{K_i} \Big(\frac{\lambda}{\mu_i}\Big)^x \frac{1}{S_i!} \Big(\frac{1}{S_i}\Big)^{x - S_i} \pi_{0_i}^{MMS}. \tag{59}$$

$M/M/S_i/K_i$ **features** For the second set of features, consider an $M/M/S_i/K_i$ tandem queue with I stations, S_i servers at station i, $K_i = S_i + C_i$ with C_i the number of buffers at station i, arrival rate λ, service rates $(\mu_1, ..., \mu_I)$ and utilisation $\rho_i = \frac{\lambda}{S_i \mu_i}$. The following features are considered:

- The probability that station i is idle:

$$\pi_{0_i}^{MMSK} = \Bigg(\sum_{x=0}^{S_i - 1} \Big(\frac{\lambda}{\mu_i}\Big)^x \frac{1}{x!} + \frac{1}{S_i!} \sum_{x=S_i}^{K_i} \frac{1}{S_i^{x - S_i}} \Big(\frac{\lambda}{\mu_i}\Big)^x \Bigg)^{-1}, \tag{60}$$

$$\pi_0^{MMSK_\sigma} = \sigma(\pi_{0_1}^{MMSK}, ..., \pi_{0_I}^{MMSK}). \tag{61}$$

- The probability that station i is blocked:

$$\pi_{K_i}^{MMSK} = \frac{1}{S_i!}\frac{1}{S_i^{K_i-S_i}}\Big(\frac{\lambda}{\mu_i}\Big)^{K_i}\pi_{0_i}^{MMSK}, \tag{62}$$

$$\pi_{K}^{MMSK_\sigma} = \sigma(\pi_{K_1}^{MMSK}, ..., \pi_{K_I}^{MMSK}). \tag{63}$$

- The number of customers at an $M/M/S_i/K_i$ station:

$$E[L_i]^{MMSK} = \sum_{x=0}^{S_i-1} x\Big(\frac{\lambda}{\mu_i}\Big)^x \frac{1}{x!}\pi_{0_i}^{MMSK} + \sum_{x=S_i}^{K_i} x\Big(\frac{\lambda}{\mu_i}\Big)^x \frac{1}{S_i!}\frac{1}{S_i^{x-S_i}}\pi_{0_i}^{MMSK}, \tag{64}$$

$$E[L]_\sigma^{MMSK} = \sigma(E[L_1]^{MMSK}, ..., E[L_I]^{MMSK}). \tag{65}$$

Algorithm 2 presents the algorithm for the effective throughput rates for a multi-server tandem queue.

Algorithm 2: Effective throughput algorithm multi-server type

```
$\gamma_0 = \lambda$
For i = 1,...,I:
    $\gamma_i = \gamma_{i-1}\Big(1 - (\frac{\gamma_{i-1}}{\mu_i})^{K_i}\frac{1}{S_i!}\frac{1}{S_i^{K_i-S_i}}\Big(\sum_{x=0}^{S_i-1}\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^x\frac{1}{x!} +$
        $\frac{1}{S_i!}\sum_{x=S_i}^{K_i}\frac{1}{S_i^{x-S_i}}\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^x\Big)^{-1}\Big)$
```

In line with the single server case, features based on the effective throughputs instead of the arrival rates are also added to the data set. Define $\hat{\rho}_i = \frac{\gamma_{i-1}}{S_i\mu_i}$.

$$\pi_{\gamma 0_i}^{MMSK} = \left(\sum_{x=0}^{S_i-1}\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^x\frac{1}{x!} + \frac{1}{S_i!}\sum_{x=S_i}^{K_i}\frac{1}{S_i^{x-S_i}}\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^x\right)^{-1}, \tag{66}$$

$$\pi_{\gamma 0}^{MMSK_\sigma} = \sigma(\pi_{\gamma 0_1}^{MMSK}, ..., \pi_{\gamma 0_I}^{MMSK}) \tag{67}$$

$$\pi_{\gamma K_i}^{MMSK} = \frac{1}{S_i!}\frac{1}{S_i^{K_i-S_i}}\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^{K_i}\pi_{\gamma 0_i}^{MMSK}, \tag{68}$$

$$\pi_{\gamma K}^{MMSK_\sigma} = \sigma(\pi_{\gamma K_1}^{MMSK}, ..., \pi_{\gamma K_I}^{MMSK}) \tag{69}$$

$$E[L_i]_\gamma^{MMSK} = \sum_{x=0}^{S_i-1} x\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^x \frac{1}{x!}\pi_{\gamma 0_i}^{MMSK} + \sum_{x=S_i}^{K_i} x\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^x \frac{1}{S_i!}\frac{1}{S_i^{x-S_i}}\pi_{\gamma 0_i}^{MMSK}, \tag{70}$$

$$E[L]_{\gamma\sigma}^{MMSK} = \sigma(E[L_1]_\gamma^{MMSK}, ..., E[L_I]_\gamma^{MMSK}). \tag{71}$$

All in all, 65 theoretical features are added to the data set. Appendix F contains a table with the mathematical names and the feature names.

4.6 Feature Importance

In order to determine whether the training of the neural network is aided by theoretical knowledge, several methods are used.

First of all, the Pearson Correlation between all features and the target is calculated. Pearson correlation can be used to assess the level of dependency between the features. A high correlation with the target may be an indication of a useful feature.

Secondly, forward and backward selection are used. These methods can be used to assess which feature has the most influence during the training of the models. With forward selection, models are first trained on all available single features. The feature that yields the best model is saved. Next, models with two features are created, of which the first feature is the previously selected feature. This repeats itself until all features have been added to the model. The order of addition to the model thus provides information on the importance of the feature. Backward selection is similar to forward selection, but works in reversed order. First a model is trained on all features, and then the feature whose removal results in the least performance reduction is removed. Both algorithms are presented in Hoogendoorn and Funk (2018).

The third method is based on the weights of the trained neural network. The idea behind this method is that more important features receive larger weights in the trained neural network. One of the first measures that was proposed in the literature is Garson's Similarity (Garson, 1991). It is one of the most often used measures, but after its publication drawbacks have been reported. The measure uses the absolute values of the weights, which means that information is lost on the sign of the weights. Other measures have also been introduced (see for example Gedeon (1997)). Olden, Joy, et al. (2004) conduct a simulation study with nine different performance measures (amongst which is Garson's) to assess which performance measure provides the best results. They report that the Connection Weight Approach (CWA) proposed by Olden and Jackson (2002) is the most reliable measure, and that in fact Garson's similarity performs worst. The formula for CWA importance is given by:

$$CWA_j = \sum_{h=1}^{H} \omega_{jh} o_h, \tag{72}$$

where j is the j^{th} input feature, ω_{jh} the weight between the j^th input node and the h^{th} hidden node and o_h the weight between the h^{th} hidden node and the output node. Notice that this method is only valid for single hidden layer neural networks, although one can adjust the measure in case of multiple hidden layers. The experiment and evidence for the excellence of CWA that is provided in Olden, Joy, et al. (2004) is convincing, and therefore this measure is adopted in this thesis. It is slightly adapted to take the random nature of neural networks into account. One hundred different neural networks are trained with different initial seeds (that determine the weights). For a feature j, the CWA_j importance is calculated for each trained neural network, and the average is taken over these hundred values to obtain a mean importance of feature j.

The last method is to train a model on a data set that does not contain any of the theoretical features, and to train a model on a data set with all features, including the theoretical features. If the performance of the model without theoretical features is similar, then adding theoretical features is of minimal influence. This will be assessed by statistical tests.

4.7 Multiple Neural Networks

It might be beneficial to use the estimates of multiple trained neural networks to create a final estimate. Then, each neural network provides an estimate, and the final estimate is created by taking the mean over the individual estimates. This can be seen as a form of "wisdom of the crowd", in which multiple individual estimates are combined into one final estimate. Several experiments were conducted to determine whether taking the median over the individual estimates performed better, but this did not provide better results. The neural networks can be created in multiple ways. First of all, it should be noted that a neural network is a randomised model, so using a different seed to generate the random initial weights will also yield a slightly different neural network. Thus, multiple neural networks can be created by using the same parameters, but a different initial seed. Secondly, the grid search also provides information on the performance of the parameter combinations that are not the best. Thus, one neural network could be trained on the best parameter combination, the second one could be trained on the second best combination, etc. Experiments will be conducted that analyse what the influence of using multiple neural networks is on the estimation accuracy.

4.8 Error Insights

The fact that multiple neural networks can be used to create a final throughput estimate can also be leveraged to obtain an idea about how certain the throughput estimate is. Moreover, confidence intervals can be created for the estimates. A method is proposed here to do so.

Algorithm 3 contains the steps of the method. These steps can be followed for a certain test data set. This results in an empirical distribution $\hat{\sigma}$ (the standard deviation), an empirical distribution E_{abs} (the absolute error between the mean estimate and the true estimate) and an empirical distribution E (the error between the mean estimate and the true estimate). Subsequently, a 95%-quantile regression is performed with E_{abs} as dependent variable and $\hat{\sigma}$ as independent variable, which yields the function:

$$E_{abs} = a + b\hat{\sigma} \tag{73}$$

This 95%-quantile regression line can then be used in case of a new data point D to make statements such as "It is 95% certain that the absolute error is not greater than y for data point D". This is done by calculating $\hat{\sigma}^D$ and plugging it into the fitted function (73), which yields a value for y. Note that also other quantile regression lines could be made (for example a 90%-quantile regression line).

Algorithm 3: Error Insights Algorithm

```
for all data_points d in test do
    Let each neural network $j$ output a throughput estimate $\tau_j^d$
    Calculate the sample standard deviation $\hat{\sigma}^d$ of all estimates $\tau_j^d$
    Calculate the sample mean $\hat{\mu}^d$ of all estimates $\tau_j^d$
    Calculate the error measure $E_{abs}^d = |\hat{\mu}^d - \tau_{true}^d|$
    Calculate the error measure $E^d = \hat{\mu}^d - \tau_{true}^d$
end
```

To create a 95%-confidence interval for the error, the distribution of E is used. Both a 97.5%-quantile and 2.5%-quantile regression are performed with E as dependent variable and $\hat{\sigma}$ as independent variable. This results in the functions:

$$E = a_{lower} + b_{lower}\hat{\sigma} \tag{74}$$

$$E = a_{upper} + b_{upper}\hat{\sigma} \tag{75}$$

Given a new data point D, the confidence interval for the error is then given by:

$$[a_{lower} + b_{lower}\hat{\sigma}^D, a_{upper} + b_{upper}\hat{\sigma}^D]. \tag{76}$$

This method will be further clarified with an example in Section 6.2.12.

4.9 Benchmark Models

To compare the performance of the neural network to other models, two regression models are considered: a linear regression model and an exponential regression model. These two regression models are also, for example, used by Altiparmak et al. (2007) to compare their developed neural network to.

The function for the linear regression model is given by:

$$y = \beta_0 + \sum_{i=1}^{F} \beta_i X_i, \tag{77}$$

with X_i the i^{th} feature, β_i the fitted coefficients, and y the target. This model assumes that there exists a linear relationship between the independent and dependent variables. The function for the exponential regression model is given by:

$$\ln(y) = \beta_0 + \sum_{i=1}^{F} \beta_i X_i. \tag{78}$$

This model assumes that there exists an exponential relationship between the independent and dependent variables.

5 Methodology: Evolutionary Algorithm (EA)

Evolutionary Algorithms (EAs) are randomised optimisation algorithms. These algorithms date back to the 1940s, when the first steps were made to develop these algorithms (Eiben and J. Smith, 2015). The algorithms are based on evolutionary processes, such as reproduction and survival of the fittest. They have been proven to be versatile optimisation algorithms that can deal with many different problems. Furthermore, many methods exist that can be used to optimise the algorithm itself (for example crowding methods). The developed EA is discussed in this section.

5.1 Components

An EA usually consists of the same components. A pseudo code of a general EA algorithm is provided in Algorithm 4. This pseudo code is based on the pseudo codes in Eiben and J. Smith (2015) and Hartmann (2001). The first steps boil down to randomly generating a population of encoded solutions and evaluating the fitness of these solutions. Subsequently, a while loop is started in which a new generation is created in each iteration. Parents are first selected from the current population according to a certain mechanism. These parents form new encoded solutions by a crossover operation. Each new candidate is mutated and its fitness is evaluated. The new generation is then selected from the old population together with all the new candidates according to a survival selection mechanism.

Algorithm 4: Evolutionary Algorithm

```
Generate random initial population $POP$.
Evaluate the fitness of all candidates $Q \in POP$.
$nr\_gen := 1$.
while $nr\_gen < GEN$ do
    $nr\_gen$ += 1.
    Select parents $P$ from population $POP$.
    Perform crossover on $P$ to generate children $Q$.
    Perform mutation on children $Q$.
    Evaluate the fitness of $Q$.
    Select the new generation $POP$ by survival selection.
end
```

One of the most important components of an EA is the representation of a solution. This representation serves as an encoding of a solution, and the mutation and crossover operations are performed on this encoding, not on the solutions themselves. Common representations are binary and integer representations. In a binary representation, a solution is encoded as a bit string and in an integer representation, a solution is encoded as a list of integers. Both types will be used in this thesis.

Crossover operators are designed to combine two (or possibly more) solutions into a new, and hopefully better, solution. This operator is based on the evolutionary concept of reproduction. The mutation operator is designed to make small changes to the solution, and thus to the encoding. It is therefore important that a small change in the encoding also yields a reasonably small change in the fitness. This is referred to as the strong causality principle, and is, for example, discussed by Rosca and Ballard (1995).

The difference between crossover and mutation operators is also often explained by the exploration versus exploitation principle. Crossover makes larger changes to the encoding than mutations, which indicates that crossover performs the exploration of the solution space, whereas mutation is used to search smaller areas in the solution space (i.e. exploitation of good solutions).

5.2 Literature Overview

A rich literature exists on evolutionary algorithms. Over the years, many types of algorithms were developed and used to solve many different types of problems. In this subsection, an overview is given on the existing literature on EAs.

EAs can be subdivided into several types, for example Genetic Algorithms, Genetic Programming and Evolutionary Programming. These types are instances of evolutionary algorithms and make certain design decisions, for example on the used representation. In this thesis, no distinction is made between the different types of evolutionary algorithms.

Applications of evolutionary algorithms can be found in many areas. An example of an application in health care is Vinterbo and Ohno-Machado (1999), who use an EA to select variables of a logistic regression model to estimate the presence of myocardial infarction (a cardiovascular condition). Their method is significantly better than, for example, standard backward or forward selection. Qaid and Talbar (2013) use EAs to encrypt and decrypt images. In the groundbreaking work of Schmidt and Lipson (2009) natural laws are extracted from experimental data by EAs, which creates opportunities to discover mathematical laws that have not been discovered yet. The fact that EAs are not limited by human conventionality even makes them good designers. In Hornby et al. (2011), for example, a new X-band antenna for the NASA's space technology 5 spacecraft is designed by an EA. Two different antennas are designed by two EAs and both outperform the human-designed antenna. EAs are nowadays also applied in the area of evolutionary robotics. A robot is then designed by using an EA, which creates improved robots (Doncieux et al., 2015).

The buffer allocation problems that are considered in this thesis are combinatorial optimisation problems. EAs have also been used in this area. Liao et al. (2012) apply them on the travelling salesman problem, Prins (2004) apply them on the vehicle routing problem and Mesghouni et al. (2004) apply them on scheduling problems. Mühlenbein et al. (1988) combines parallel computing with EAs to solve combinatorial problems. There is even a yearly conference on using evolutionary algorithms to solve these types of problems, which is called EvoCOP. See the proceedings of 2019 as an example (Liefoogh and Paquete, 2019).

EAs have also been often used to solve BAP. These papers have been discussed in Section 3.2.

5.3 The Developed EA

5.3.1 The Representation

Two different representations are used to encode the different solutions, and they are subject to several conditions. In particular, the representation should convey the number of stations, the number of buffers and the configuration of the buffers.

A binary and an integer representation is considered. Figure 11 contains the encoding of the same queuing network in the two representations. The network consists of four stations and there are eleven buffers to be distributed.

In the integer representation, the length of the encoding represents the number of stations. The i^{th} number represents the number of buffers at station i. Thus, in this example, station three has four buffers and station four has two buffers.

In the binary representation, a network is encoded by a bit string with a size equal to the maximum number of buffers minus one. The ones represent the stations and the zeroes represent the number of buffers at each station. Two things should be noted here. First of all, the number of ones is equal to the number of stations minus one, as the last one would be obsolete. Secondly, the number of buffers at each station is equal to the number of zeroes plus one. This is done to simplify the creation of the mutation operators. In the case of the example in Figure 11, there are three buffers at station one and two buffers at station two.

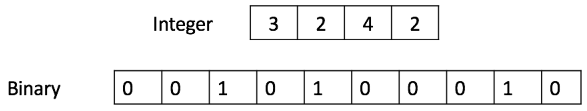


Figure 11: Example of the integer and binary representations.

5.3.2 The Mutation Operators

Six different mutation operators are considered. Four different operators are used for the integer representation and two different operators are used for the binary representation.

Illustrations of the integer representation mutation operators can be found in Figure 12. The coloured cells resemble the values that are affected by the mutation. The first mutation operator is called the *swap* mutation. Two values are randomly selected and swapped. The second operator is the *inc_dec* operator. Two values are randomly chosen. One value is increased and the other value is decreased. The third operator is the *scramble* operator in which a random subset is chosen and shuffled. The last operator is the *inverse* operator in which a random subset is inverted.

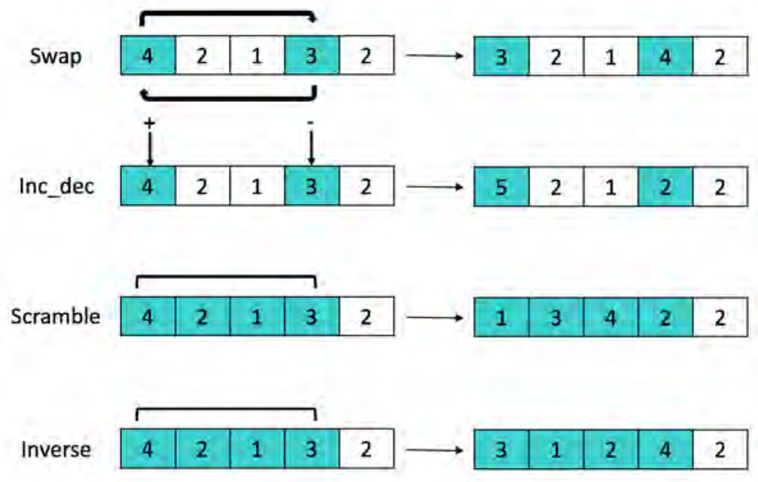


Figure 12: The integer mutation operators.

The illustrations of the two binary mutation operators can be found in Figure 13. The first operator is the *Move* operator. X ones are randomly moved one step forward or backward. X is randomly chosen and ranges between 1 and $I-1$. The second operator is the *Flip* operator. X ones are randomly moved. X is again randomly chosen and ranges between 1 and $I-1$.

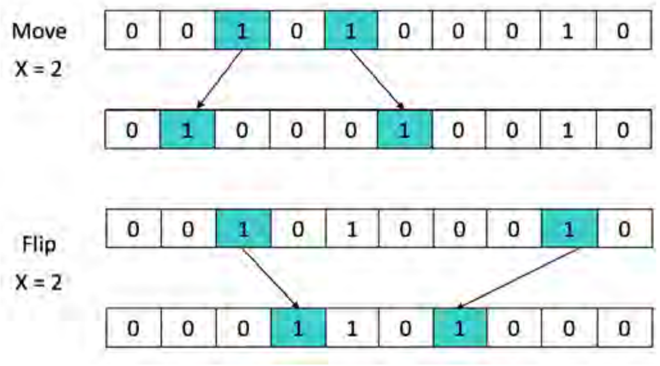


Figure 13: The binary mutation operators.

5.3.3 The Crossover Operators

Three different crossover operators are considered. Two for the integer representation and one for the binary representation.

A standard integer crossover cannot be used, as the sum of the values has to be equal to the number of available buffers, and standard crossovers do not guarantee this. Therefore, two known crossover types are changed such that the sum of the solution is always equal to the number of available buffers.

Figure 14 presents two examples of the application of the uniform crossover. The coloured cells indicate which value stems from which parent. The method is based on the uniform crossover for binary chromosomes discussed by Eiben and J. Smith (2015). The child is filled from left to right by randomly selecting a value of one of the parents for each cell. If the selected value is not suitable (e.g. because the sum of the buffers becomes greater than the maximum), the value of the other parent is selected.

In the left example of Figure 14, the (random) selection is such that the sum of the cells is equal to the number of available buffers (in this case 12). However, in the right example a problem arises when the fourth cell is considered, as selecting either four or three will increase the sum of the first four values above 12. Therefore, neither of the two values is chosen and the cell receives the temporary value 0. It may also be the case that assigning a value to a cell results in a sum lower than the available number of buffers, but a sum so high that the rest of the chromosome cannot be filled by values without exceeding the available number of buffers. This cell then also receive the value 0. The method then moves to the next cells. When all cells have been considered, the cells with value 0 receive the value 1 (recall that it is assumed that there is always a buffer at a station). If the sum of the cells is not equal to the available number of buffers, then one of the cells that did not receive a value of one of the parents is replaced by a value such that the sum becomes equal to the available number of buffers. In the example this means that the fourth cell receives the value 12 - 10 = 2.

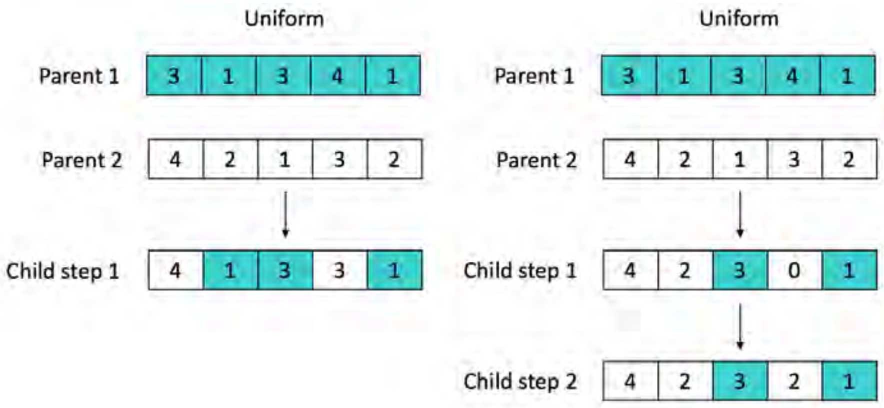


Figure 14: The integer uniform crossover.

The schematic representation in Figure 15 is for the point crossover, which is based on the 1-point crossover for binary representations discussed in Eiben and J. Smith (2015). This crossover type is comparable to the uniform crossover, with the exception that instead of uniformly selecting the values, a split is made in the encoding of the two parents. The first part of one of the parents is added as the first part of the child, and the second part of the other parent is added to the child as second part of the child. The same mechanism is used as with the uniform crossover if selecting a value of the second parent results in either too many or too few buffers. This is illustrated in the right example of Figure 15, which accidentally results in the same child as was created with the uniform crossover in Figure 14. This crossover is less disruptive than the uniform crossover, as whole parts of the parent are copied to the child.

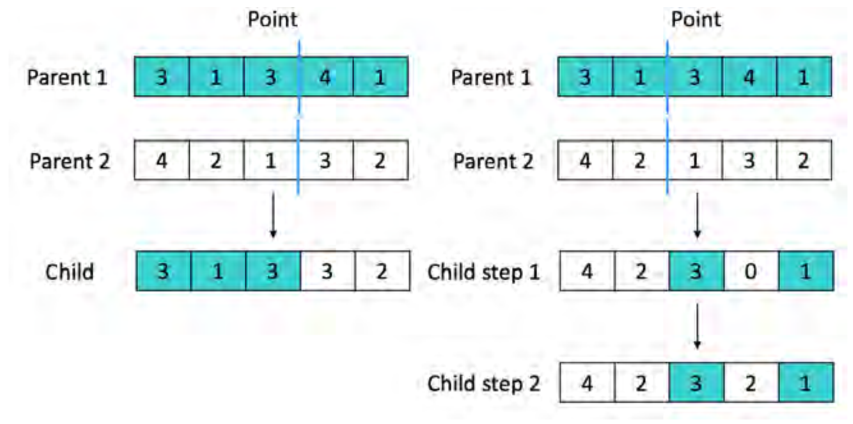


Figure 15: The integer point crossover.

A schematic representation of the binary crossover is provided in Figure 16. The child is created by randomly choosing indices that contain an one in the chromosomes of the parents. Thus, in the case of the example, three indices are selected from the indices [0, 2, 4, 6, 8].

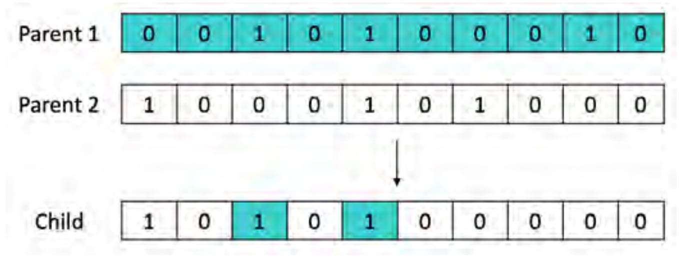


Figure 16: illustration of the binary crossover.

5.3.4 Parent and Survivor Selection

The parent selection mechanism is used to select the parents that are used to create new solutions by the crossover operator. Many different mechanisms are available, like tournament selection and roulette wheel selection (Jebari et al., 2013). Some mechanisms deliberately select good solutions as parents, whereas others do not. Multiple techniques were considered, but uniform parent selection provided the best results. This mechanism randomly selects two parents, and thus does not display a bias towards good solutions.

The survivor selection mechanism is used to select the solutions that are part of the new generation. The first mechanism that is implemented is a tournament mechanism. This means that the new generation is selected by randomly selecting X candidate solutions from the created population and the old population and adding the best candidate to the new population. This, however, created problems since the population was quickly taken over by the current best individual, which indicates that the algorithm had the tendency to get stuck in local optima. In the literature, various methods are proposed to deal with premature convergence, like fitness sharing, niching and crowding (Sareni and Krähenbühl, 1998). Crowding mechanisms are methods that try to create an even distribution of solutions over multiple optima. Plots of the solution space of the SSBAP showed a rugged landscape, which indicates that a crowding mechanism may be beneficial. Crowding was introduced by DeJong (1975) and later improved by other scientists.

The firstly considered mechanism is based on the Diversity-Guided EA (DGEA) of Ursem (2002). In his method, the algorithm alternates between periods of diversification and exploitation. During the diversification phase, only mutations are performed, and during the exploitation phase, only selection mechanisms and crossovers are used. It depends on the diversity of the population in which phase the algorithm operates. This method provided slightly better results, but a plot of the diversity over the generations showed more of a random walk than a clear direction. The secondly implemented crowding mechanism is based on Mahfoud (1995), in which parent-child tournaments are performed. However, this crowding mechanism assumes that two children are created by the crossovers, which is not the case in the used crossovers. Therefore, the mechanism is slightly adapted. A child is created by performing crossover and mutation, and its fitness is calculated. Subsequently, one of its parents is randomly selected and a parent-child tournament is performed. If the child is better, the child replaces the parent in the population. This mechanism provided better results and was therefore used. Furthermore, the algorithm was programmed in such a way so that a generation has no duplicate solutions. This also increases the diversity of the population.

5.3.5 The Grid Search

As multiple representations, crossover and mutation operators are created, the best configuration of the algorithm has to be determined. This is done by performing a grid search over all the different mutation, crossover and representation configurations. The performance of the algorithms is examined in four different ways. First of all, 100 runs are performed with a small population and a relatively high number of generations, and boxplots are created of the mean and maximum achieved fitness during the runs. These boxplots can be used to assess in how many cases a high fitness is achieved by the configurations. Secondly, 100 runs are performed with a large population and a relatively low number of generations, and the mean number of visited different solutions and the mean number of revisited solutions is calculated. This helps to assess whether the configurations are able to consider a wide range of solutions. Thirdly and fourthly, single runs are performed and plots are created of the mean and maximum fitness over the generations during this run. This helps to analyse the speed of convergence of the various configurations.

It should be noted that it is common in evolutionary algorithms to have more meta parameters. For example, it is common that a mutation is performed on a candidate with a probability $0 \leq M \leq 1$. Experiments where the values of these kind of parameters were changed were performed, and it turned out that these parameters did not have a significant positive impact on the performance of the EA. Keeping these results into account and to keep the algorithm as simple as possible, these kind of parameters were "hard-coded" into the algorithm. The value of M was set to 1, for instance.

6 Application: Single Server Buffer Allocation Problem (SSBAP)

In this section, the developed solution method is applied to the Single Server Buffer Allocation Problem (SSBAP).

6.1 Problem Definition

In the Single Server Buffer Allocation Problem (SSBAP) tandem networks of $M/M/1/K_i$ (where $K_i = C_i + 1$) queues are considered. This means that each station of the network has one server, customers arrive at the first station according to a homogeneous Poisson Process and are served according to an exponential distribution at each station. The queuing network has B buffers at its disposal and the goal is to maximise the throughput by placing the buffers at each station. The SSBAP is the simplest form of BAP.

6.2 Neural Network

The neural network is created by using the MLPRegressor of the Python package *Sklearn* (Pedregosa et al., 2011).

6.2.1 The First Experiment: Classic Machine Learning

The first type of training data is a data set that contains multiple total enumerations, and is generated in a similar way as the test sets. Recall that a total enumeration data set contains a total enumeration of all buffer configurations for a certain queuing system together with the throughput estimated by the DES (see also Section 3.3). This data set is therefore closest to a common machine learning data set, in which the training and test sets are similar. A list of features that are used in this type of data set can be found in Appendix F.

Two training data sets are generated, each with 25000 data rows. In the subsequent, these data sets are called $Test\_1\_train\_1$ and $Test\_1\_train\_2$, respectively. $Test\_1\_train\_1$ contains data on twelve total enumerations (and thus twelve different queuing systems), and $Test\_1\_train\_2$ contains eighteen total enumerations. The service and arrival rates of each total enumeration is randomly generated from a uniform distribution. A test set with 15000 data rows is also generated, which is called $Test\_1\_test$. Appendix A contains the parameters of these data sets.

A grid search as introduced in Section 4.3.3 is performed to select the parameters of the neural network by using 5-fold cross-validation on each training set. The grid search tries to find a parameter set that minimises MAE. Table 2 contains the considered values for the selected parameters. The parameter *max_iterations* determines for how many iterations the neural network is trained. The parameter *alpha* is a regularisation parameter of Ridge regression. A higher value means higher regularisation and thus a simpler model. When *early_stopping* is set to True, training of the neural network is stopped when there is no improvement on a validation set. Both single and double hidden layers are considered. The other parameters are set to their initial values as provided by *Sklearn*.

Table 2: The considered grid search parameters.

Parameter	Values
max_iterations	{200, 500, 1000, 2000}
alpha	{0.01, 0.001, 0.0001, 0.00001}
early_stopping	{True, False}
hidden_layer_sizes	{(10), (20), (30), (40), (50), (60), (70), (80), (90) (100), (10,10), (20,10), (10,20), (20,20), (20,30) (30,20), (30,30), (30,10), (10,30), (40,10), (10,40) }

Table 3 contains the best found parameters for each training set. The MAE on $Test\_1\_train\_1$ is 0.00743, and the MAE on $Test\_1\_train\_2$ is 0.00870, corresponding to a RAE of 0.643% and 0.695% respectively. The performance on the train sets is thus quite good. However, the picture changes entirely when the performance on the test set is considered. Figure 17 contains the MAE values for the test set together with 95% confidence intervals. The performance on each individual total enumeration that combined create the total test set is also reported. Clearly, the MAE values on the test set are much larger than those on the train set. The mean RAE on the total test set by using the first train set as training data is equal to 19.46% and the RAE on the total set by using the second train set as training data is equal to 11.74%. Overfitting could be one of the reasons for this bad performance, but similar values were observed for other parameter sets of the neural network (for example when a lower number of hidden nodes was used). This is an indication that perhaps the neural network is not able to learn the essence of the problem by using this type of training sets. The plots for the MSE and RAE values can be found in Appendix B.

Table 3: The selected grid search parameters for each training set.

Parameter	$Test\_1\_train\_1$	$Test\_1\_train\_2$
max_iterations	2000	1000
alpha	0.00001	0.00001
early_stopping	True	True
hidden_layer_sizes	100	100

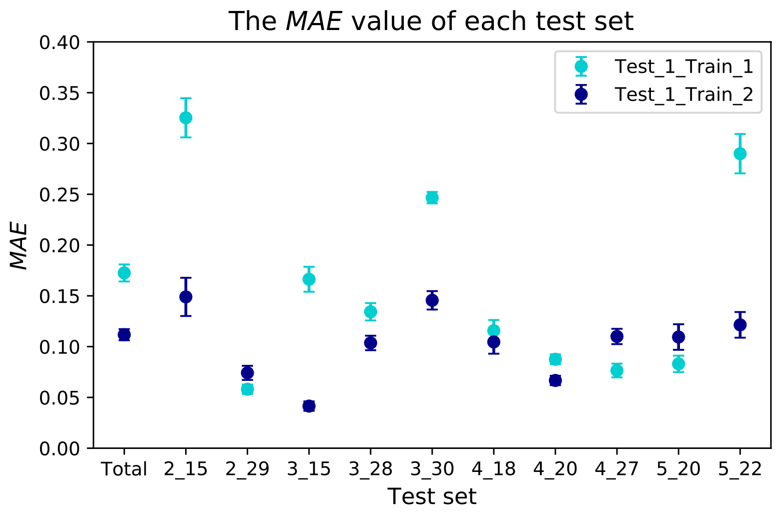


Figure 17: The MAE values together with 95% confidence intervals for each test set.

6.2.2 The Second Experiment: Discretisation

When the performance of the first type of training sets was analysed, it was observed that in some cases very unstable queues were part of the data sets. This created outliers in the considered features, which made it more difficult for the neural network to learn the essence of the problem. For example, the feature $MM1\_EL\_sum$ of train set 1 has values ranging from 2.2 to 72.32, and in other training sets larger outliers were observed, for example values of 600.

Therefore, the arrival and service rates are discretised. The arrival and service rates are still within the same ranges as the first data set, but now in steps of Δ. Different values of Δ are considered, namely 0.025, 0.05, 0.10 and 0.25.

Data sets similar to the first type of training set are generated. The number of servers and buffers is similar to the first experiment, but the arrival and service rates are now randomly selected from discretised rates with $\Delta \in \{0.025, 0.05, 0.10, 0.25\}$. This creates in total eight new training data sets. The parameters of these eight training data sets can be found in Appendix A. It is now important to assess what the influence of the discretisation is on the performance of the neural network. This will be done in two ways: by using a total enumerated test set, and by assessing the inter- and extrapolation capabilities of the neural network.

Grid searches are again performed to select the parameters, and the different values of Table 3 are again considered. Tables 48 and 49 in Appendix B contain the selected parameters for each train set.

Figure 18 presents the RAE values on the test set $Test\_1\_test$ for each of the eight discretised training sets, and also the training sets of the first type without discretisation. In general, the higher the value of Δ, the greater the RAE. Both the train sets with Δ equal to 0.025 and 0.05 attain comparable values for RAE.

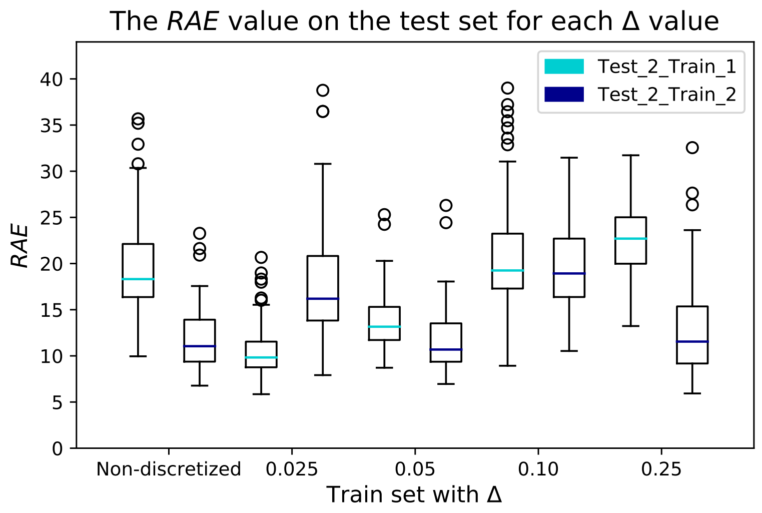


Figure 18: RAE values for train sets with different values of Δ.

It is known that neural networks can struggle with extrapolation (Xu et al., 2020), and in some cases also with interpolation. To assess the influence of the discretisation on the inter- and extrapolation capabilities of the neural network, a new test set is generated. The test set contains 15000 rows of random queuing systems with randomly selected buffer configurations, with uniformly distributed arrival rates between 0.5 and 3.0 and uniformly distributed service rates between the selected arrival rate and 3.5. This data set thus provides insight in the inter- and extrapolation capabilities of the neural network.

Figure 19 presents plots of the absolute errors on this test set for different values of Δ. Clearly, the larger Δ, the higher the absolute errors. However, surprisingly, the discretised version with $\Delta = 0.025$ has lower errors than the non-discretised version. This is an indication that discretising the data set is a good idea. Another test set was generated that only considered the interpolation capabilities of the neural network as the arrival rates were uniformly distributed between 0.5 and 2.5 and the service rates were uniformly distributed between the arrival rate and 3.0. The results on this test set were similar to those presented, and therefore these are not considered here.

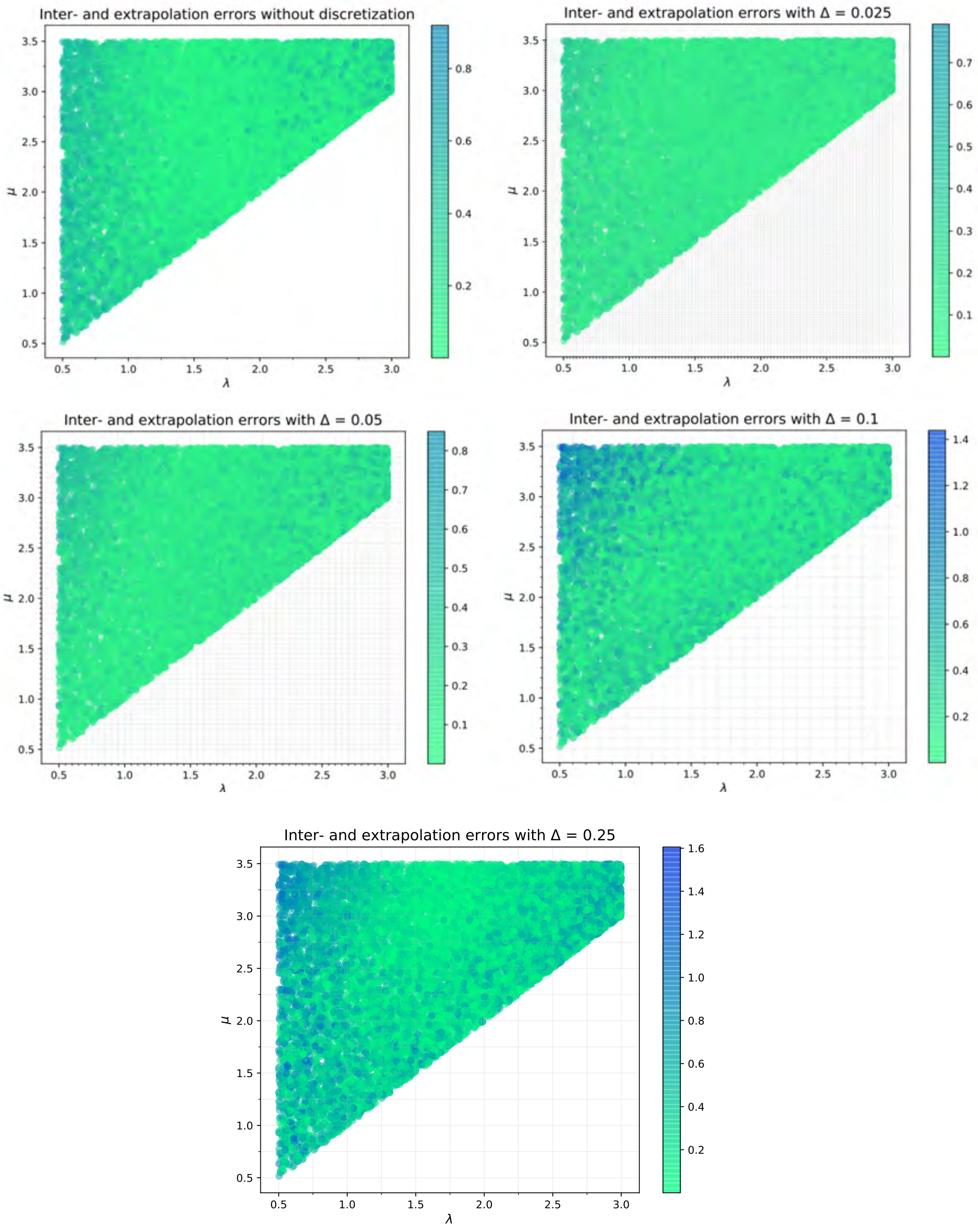


Figure 19: The inter- and extrapolation capabilities of the neural network for different values of Δ.

6.2.3 The Third Experiment: Discretisation and Classic Machine Learning

All in all, we observe that discretising the rates work well, as discretisation makes sure that no highly unstable queues are part of the data set, and the extra- and interpolation capabilities of the neural network seem to be either similar or even better than those trained by the non-discretised set. It seems that $\Delta = 0.025$ provides the best results. Notice that discretising the arrival and service rates does not limit the total data space size extensively. This will be further addressed in Section 6.2.9. A new test set ($Test\_3\_test$) is generated that also has discretised parameters. These values can be found in Appendix A.

Neural networks are trained with two data sets similar to the first experiment, but then with discretised parameters with $\Delta = 0.025$. The MAE on the first train set is 0.0067 and the MAE on the second train set is 0.0076, which is comparable to those of the first experiment. Figure 20 presents the MAE values together with 95% confidence intervals on the discretised total test set, and on each individual test set that together make up the total test set, for neural networks trained on the generated data sets with a Δ of 0.025. In general, the MAE values in this plot are lower than those of Figure 17. However, the MAE values on the total test set are still quite high. This is also reflected in RAE scores of around 10%, which is not a satisfactory error percentage (also see the plots in Appendix B).

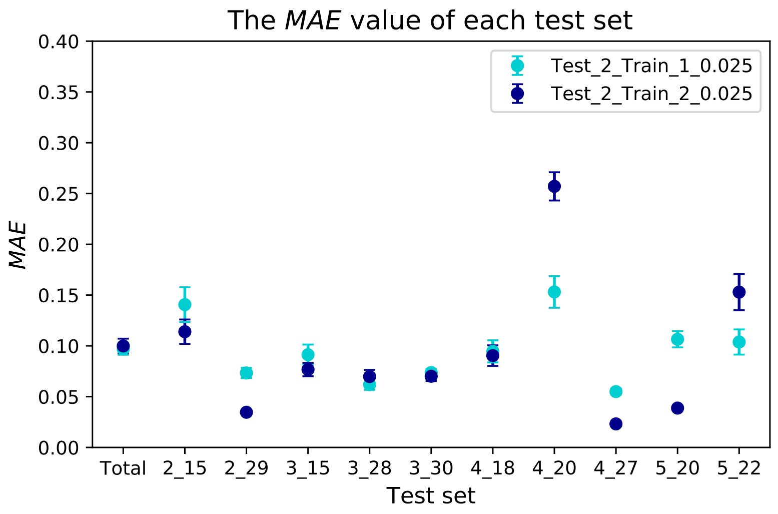


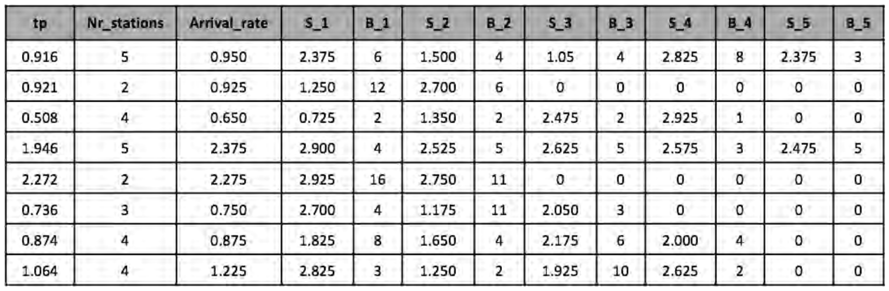
Figure 20: The MAE values together with 95% confidence intervals for each test set.

6.2.4 The Fourth Experiment: More Tandem Lines

Using a train set that contains multiple total enumerations does not provide satisfactory results. Therefore, a new type of training set is developed. Instead of using the data of a limited number of total enumerations, a train set is created that contains the data of many different queuing systems with different parameters (arrival rate, service rates, etc.). This may help the neural network to learn what the influence of the different queuing parameters is on the throughput.

Two training data sets with 25000 data rows are generated. Each row corresponds to a queuing network with a randomly selected number of stations, service rates, arrival rate and buffer configuration. The arrival and service rates are still between the considered boundaries (see Section 4.3.1) and are discretised with $\Delta = 0.025$.

Figure 21 contains a snapshot of the first eight rows of a data set that has been generated according to the above method. Clearly, information on many different queuing systems is now part of the data set.



tp	Nr_stations	Arrival_rate	S_1	B_1	S_2	B_2	S_3	B_3	S_4	B_4	S_5	B_5
0.916	5	0.950	2.375	6	1.500	4	1.05	4	2.825	8	2.375	3
0.921	2	0.925	1.250	12	2.700	6	0	0	0	0	0	0
0.508	4	0.650	0.725	2	1.350	2	2.475	2	2.925	1	0	0
1.946	5	2.375	2.900	4	2.525	5	2.625	5	2.575	3	2.475	5
2.272	2	2.275	2.925	16	2.750	11	0	0	0	0	0	0
0.736	3	0.750	2.700	4	1.175	11	2.050	3	0	0	0	0
0.874	4	0.875	1.825	8	1.650	4	2.175	6	2.000	4	0	0
1.064	4	1.225	2.825	3	1.250	2	1.925	10	2.625	2	0	0

Figure 21: A snapshot of the first eight data rows generated according to the new method.

Two grid searches are again performed to select the parameters for the two new train sets. The selected parameters can be found in Appendix B. The train sets are referred to as $Test\_4\_Train\_1$ and $Test\_4\_Train\_2$ in the following. The MAE value on the first train set is 0.0124 and on the second train set 0.0132. These values are slightly higher than those of the first two types of train sets. Figure 22 presents the MAE values on the test set. Clearly, the errors are much lower than was the case with the first two types of train sets. The RAE values lie around 2%, which is much better than the previously reported 10% (also see the plots in Appendix B). Thus, it seems very beneficial to incorporate information of many different queuing systems in the data set instead of information on only a couple of total enumerations. The performance of the two train sets is also quite comparable, which is an advantage as this indicates that the method is generalisable. In the following experiments, one (instead of two) training set is generated to analyse a certain hypothesis, as there is not a lot of difference between the performance of training sets of the same type (Figure 22).

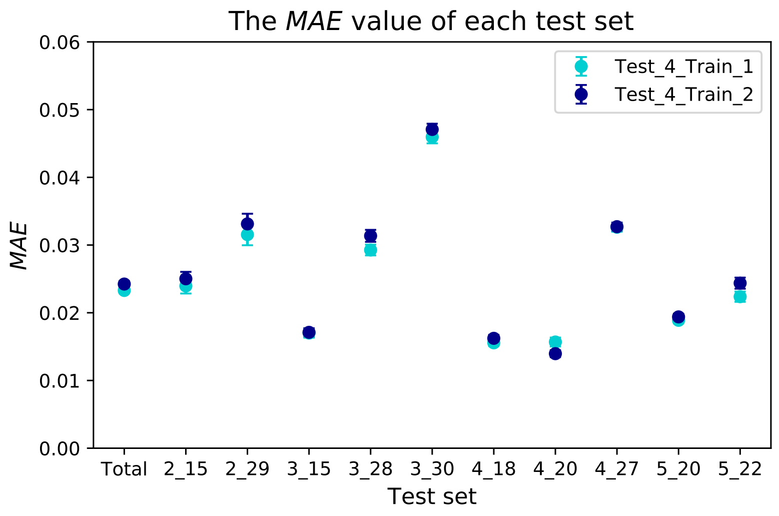
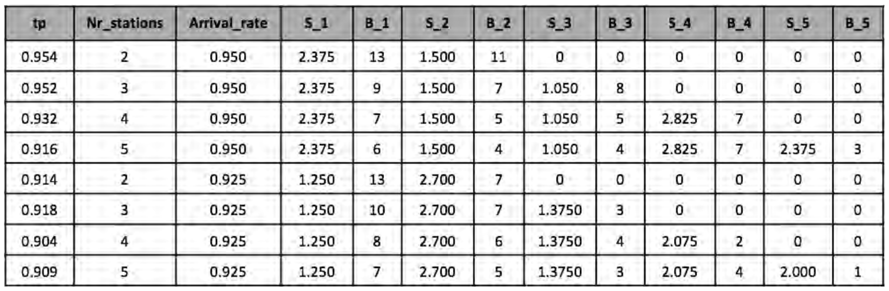


Figure 22: The MAE values together with 95% confidence intervals for each test set.

6.2.5 The Fifth Experiment: A Balanced Data Set

In the previous data set, each row corresponded to a different queuing network with a randomly selected number of stations. This may create an unbalanced data set, as in some data sets queues with a specific number of stations are more frequently present than others. Therefore, a new training set is generated that, for a given arrival rate and service rates, contains the data of a queuing network with two, three, four and five stations, with randomly selected buffer allocations. This also has the advantage that the data set now contains information on what the influence of a longer tandem line is on the throughput for a queuing network with the same parameters.

Figure 23 contains a snapshot of the first eight rows of a data set that has been generated according to the above method. Notice the difference between Figure 21 and Figure 23: in Figure 23 four rows of data contain the information about a queuing system with the same arrival and service rates, instead of only one. Moreover, notice that the service and arrival rates of the two data sets coincide, which was achieved by using seeds for the random numbers. This was done to ensure that the two data sets are really comparable, as they contain information on the same tandem lines.



tp	Nr_stations	Arrival_rate	S_1	B_1	S_2	B_2	S_3	B_3	S_4	B_4	S_5	B_5
0.954	2	0.950	2.375	13	1.500	11	0	0	0	0	0	0
0.952	3	0.950	2.375	9	1.500	7	1.050	8	0	0	0	0
0.932	4	0.950	2.375	7	1.500	5	1.050	5	2.825	7	0	0
0.916	5	0.950	2.375	6	1.500	4	1.050	4	2.825	7	2.375	3
0.914	2	0.925	1.250	13	2.700	7	0	0	0	0	0	0
0.918	3	0.925	1.250	10	2.700	7	1.3750	3	0	0	0	0
0.904	4	0.925	1.250	8	2.700	6	1.3750	4	2.075	2	0	0
0.909	5	0.925	1.250	7	2.700	5	1.3750	3	2.075	4	2.000	1

Figure 23: A snapshot of the first eight data rows generated according to the new method.

A grid search is again performed to select the parameters for the new train set. The selected parameters can be found in Appendix B. The train set is referred to as $Test\_5\_Train\_1$ in the following. The MAE value on the train set is 0.0125. Figure 24 presents the MAE values on the test set (Appendix B contains the plots for the MSE and RAE values). The errors are very comparable to the errors of Figure 22, which indicates that this method of generating the data does not improve the performance of the neural network. This is also supported by Figure 25. To statistically assess this, Mann-Whitney U tests are performed with the following hypotheses:

$$H_0 : F = G \tag{79}$$

$$H_1 : F \neq G, \tag{80}$$

with F and G the considered distributions. The Mann-Whitney U test was chosen as probability plots and Shapiro Wilk tests did not provide any indication of normally distributed data. The performed Mann-Whitney U tests also do not indicate that one of the methods is better than the other.

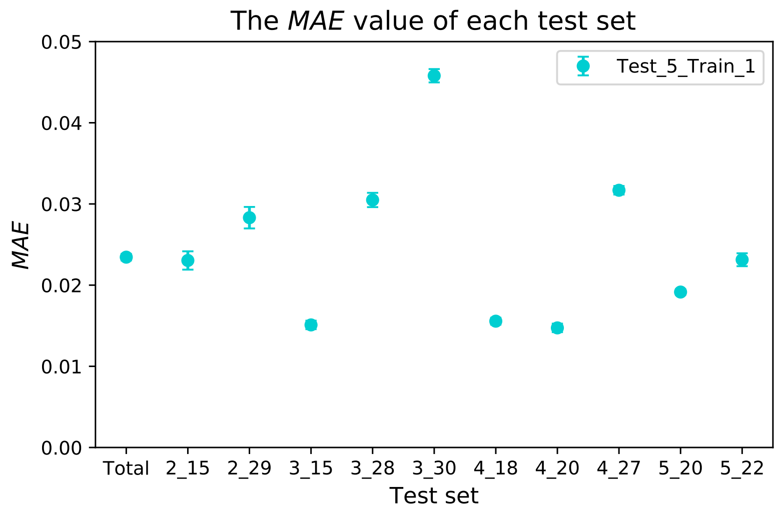


Figure 24: The MAE values together with 95% confidence intervals for each test set.

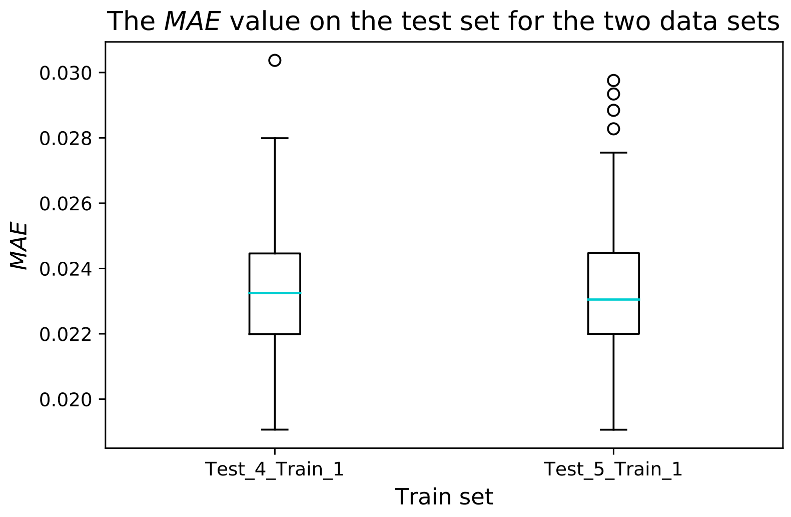
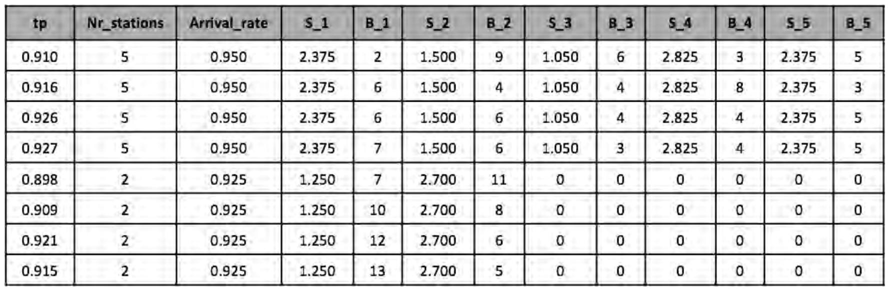


Figure 25: The MAE values on the total test set for $Test\_4\_Train\_1$ and $Test\_5\_Train\_1$.

6.2.6 The Sixth Experiment: Multiple Buffer Configurations

It might be interesting to add data of multiple buffer configurations for the same queuing system instead of only one as was the case in the previous experiments. Therefore, the influence of adding multiple configurations to the data set on the performance is analysed in this sixth experiment.

Four new data sets are created that contain two, three, four and five random buffer configurations of the same queuing network. Each data set contains 25000 data rows. Figure 26 contains a snapshot of the first eight rows of a data set with four buffer configurations per queuing system. Notice that this data set contains more information about the influence of the buffer allocations of a specific queuing system, but less information on different queuing system than the data set with one configuration per queuing system, as each data set contains an equal number of rows. As in the fourth experiment, the number of stations of each queuing system is randomly selected as well as the arrival and service rates. Seeds are again used to make sure that the data sets contain information on the same queuing systems.



tp	Nr_stations	Arrival_rate	S_1	B_1	S_2	B_2	S_3	B_3	S_4	B_4	S_5	B_5
0.910	5	0.950	2.375	2	1.500	9	1.050	6	2.825	3	2.375	5
0.916	5	0.950	2.375	6	1.500	4	1.050	4	2.825	8	2.375	3
0.926	5	0.950	2.375	6	1.500	6	1.050	4	2.825	4	2.375	5
0.927	5	0.950	2.375	7	1.500	6	1.050	3	2.825	4	2.375	5
0.898	2	0.925	1.250	7	2.700	11	0	0	0	0	0	0
0.909	2	0.925	1.250	10	2.700	8	0	0	0	0	0	0
0.921	2	0.925	1.250	12	2.700	6	0	0	0	0	0	0
0.915	2	0.925	1.250	13	2.700	5	0	0	0	0	0	0

Figure 26: A snapshot of the first eight data rows of a data set with four configurations per system.

Grid searches are again performed to select the parameters for each data set. The selected parameters can be found in Appendix B. Figure 27 presents boxplots of the MAE values for each data set (with Ci referring to a data set with i random buffer allocations per queuing system). There does not seem to be a big difference between the performance of the data sets. This is also supported by significant Mann-Whitney U tests between all pairs of data sets. There is not one data set that significantly performs better than the other data sets. Therefore, it is decided to continue with a train set with one buffer allocation per queuing system (C1 in Figure 27). This coincides with the train sets generated in experiment four.

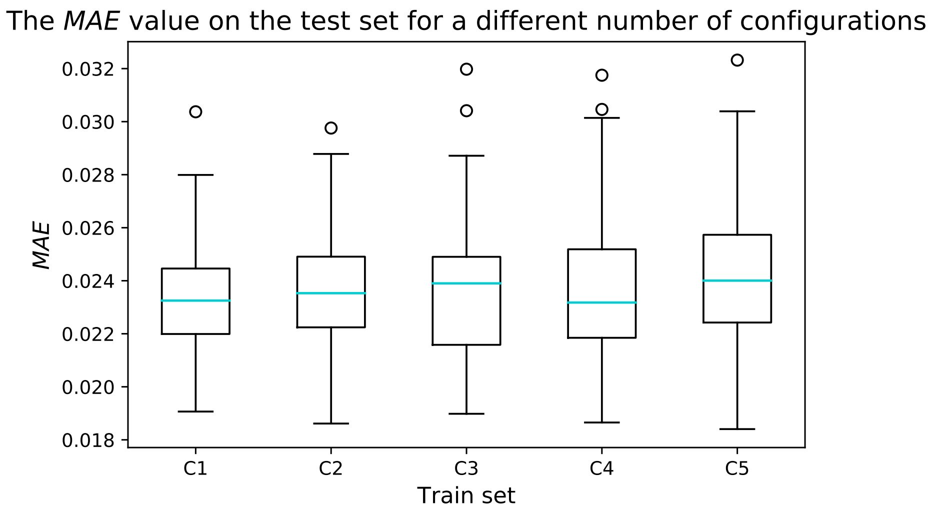


Figure 27: The MAE values for a different number of random buffer configurations.

6.2.7 The Seventh Experiment: Theoretical Data

In this seventh experiment, the influence of theoretical data on the performance of the neural network is investigated. The training data set of the fourth experiment is used ($Test\_4\_Train\_1$) and the 66 theoretical features discussed in Section 4.5.1 are added to this data set. The methods described in Section 4.6 are used to assess whether the theoretical features have a positive influence on the performance.

Figure 28 contains a heatmap of the Pearson correlation with all features. The correlations with the feature *tp* are most important, as this is the target of the neural network. A very high correlation is present between *tp* and $MM1\_t\_tp$, $arrival\_rate$, $gamma\_1$ and $gamma\_2$. The gamma features and $MM1\_t\_tp$ are theoretical features. The $Service\_1$ feature that specifies the service rate at station 1 also has a high correlation with *tp*.

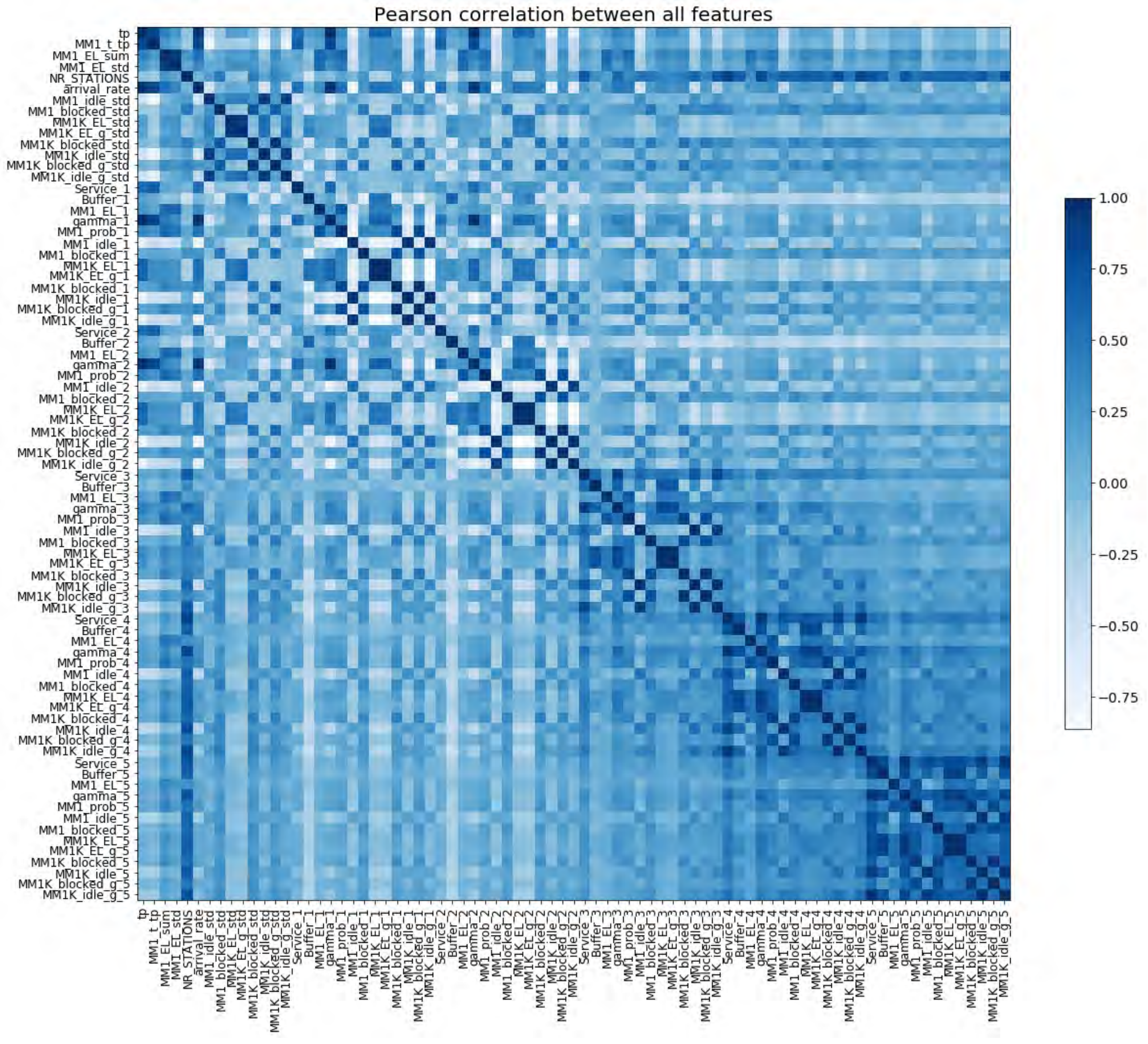


Figure 28: Pearson Correlation between all variables.

A new grid search is performed to find the parameters for the data set that includes theoretical data. Appendix B contains the selected parameters. Interestingly, the best neural network has one hidden layer with 70 neurons, which is smaller than the previously trained neural networks.

Figure 29 presents boxplots of the MAE values of neural networks trained on data sets without and with theoretical features. Notice that a separate grid search has been performed on each data set. The neural networks trained on the theory data set clearly outperform the neural networks trained on the non-theoretical data set. This is also supported by a significant Mann-Whitney U test. The mean RAE of the non-theoretical data set is 1.92%, whereas the mean RAE of the theory data set is 1.49%. A clear improvement.

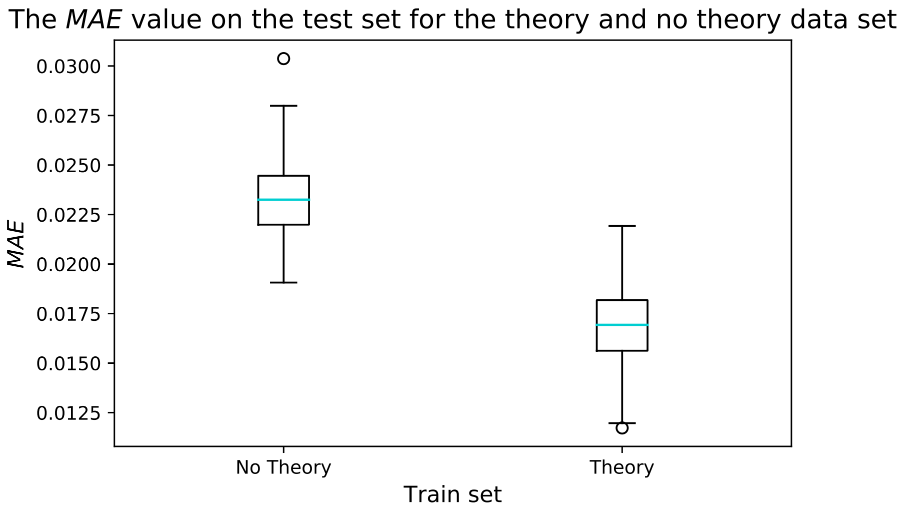


Figure 29: The MAE value on the test set for the theory and no theory data set.

Table 4 contains the importance of the ten most important features according to the Connection Weight Approach (CWA). The $gamma\_2$ feature is the most important feature according to this method. Five of the ten features are theoretical features. Interestingly, although the feature $MM1\_t\_tp$ has a high correlation with the throughput, it does not receive a very high importance according to CWA.

Table 4: The importance of the ten most important features according to CWA.

Feature	Mean Importance
gamma_2	0.395
gamma_1	0.383
arrival_rate	0.343
Service_2	0.188
Service_1	0.186
gamma_3	0.137
MM1_t_tp	0.081
gamma_4	0.067
Service_3	0.066
Service_4	0.054

Figures 30 and 31 present the results of the backward and forward selection algorithms, respectively. In the backward selection algorithm, *MM1K_blocked_g_2* is the last feature that is removed. In the forward selection algorithm, *gamma_1* is the first attribute that is selected. The first non-theoretical feature that is added is the arrival rate, which is the *twelfth* feature that is added. The forward and backward selection algorithms thus also indicate that theoretical features are important.

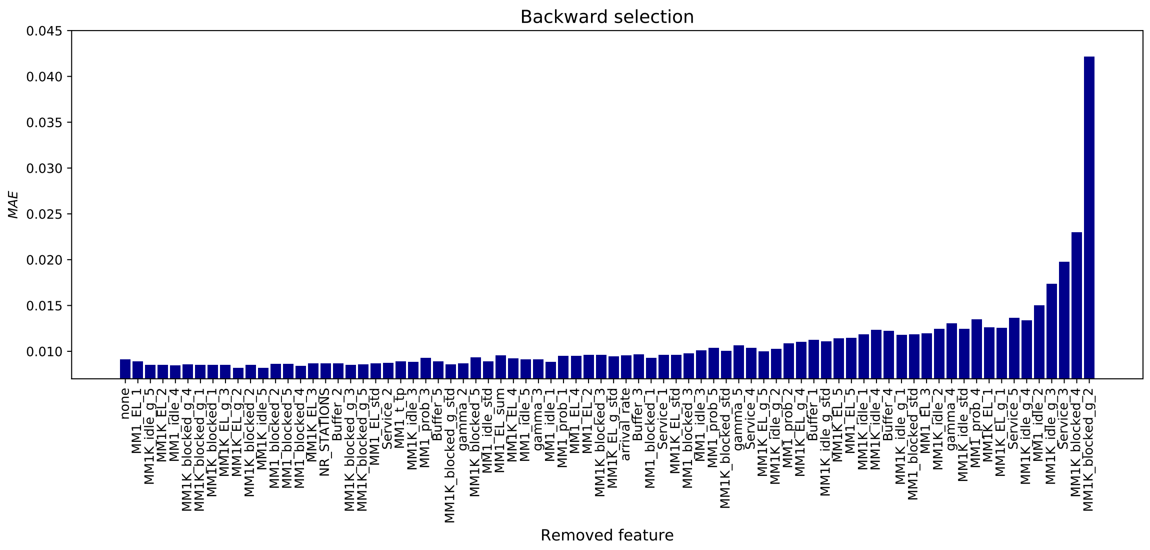


Figure 30: The results of the backward selection algorithm.

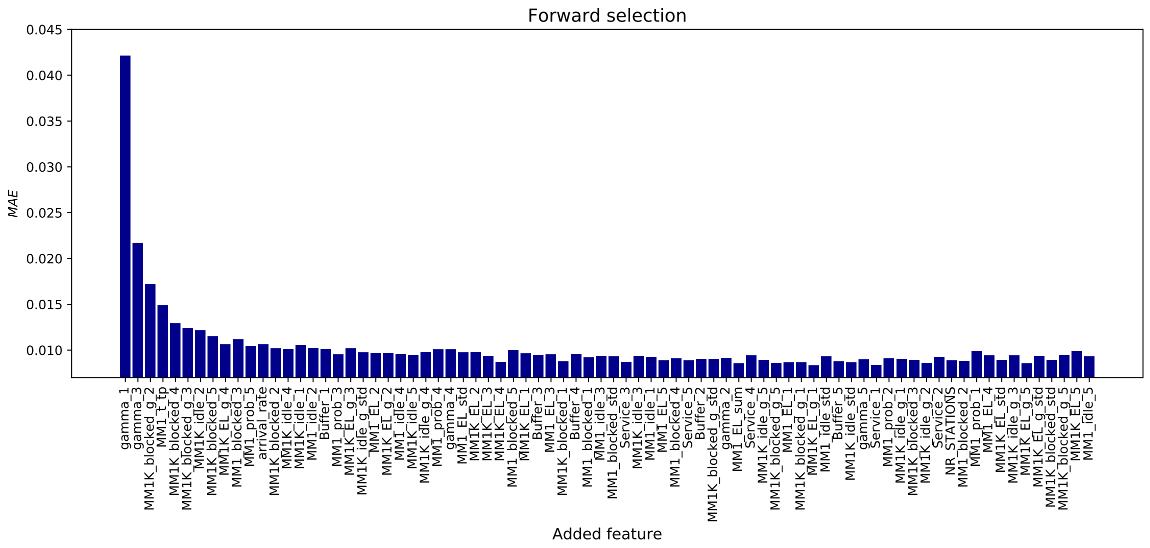


Figure 31: The results of the forward selection algorithm.

To conclude, adding theoretical features to the data set has a significant positive impact on the performance of the neural network. Moreover, it seems that the theoretical features $gamma\_2$ and $gamma\_1$ are most important.

6.2.8 The Eighth Experiment: Multiple Neural Networks

In Section 4.7 it was discussed to use multiple neural networks to create an estimate of the throughput. In this experiment, this is investigated.

The first method of Section 4.7 is considered first. The first data set of the fourth experiment ($Test\_4\_Train\_1$) is used as training data (including the theoretical features). Figure 32 presents boxplots of the MAE values on the test set when multiple neural networks are used. Clearly, using multiple neural networks reduces both the mean and the spread of the MAE values. Significant Mann-Whitney U tests indicate significant differences between all pairs of using one to eight neural networks. The influence on the performance of adding an additional neural network reduces when more neural networks are considered. The difference between using nine and ten neural networks is not significant anymore according to a Mann-Whitney U test, for example. However, the difference between eleven and fourteen neural networks is again significant. It should be noted that using more neural networks also increases the running time. Figure 33 plots the target versus the predictions of a single neural network and the target versus the predictions of fourteen neural networks combined of a test instance with four stations and 18 buffers. The predictions of the fourteen neural networks combined are more concentrated around the line of perfect estimation ($y = x$) and less outliers are observed.

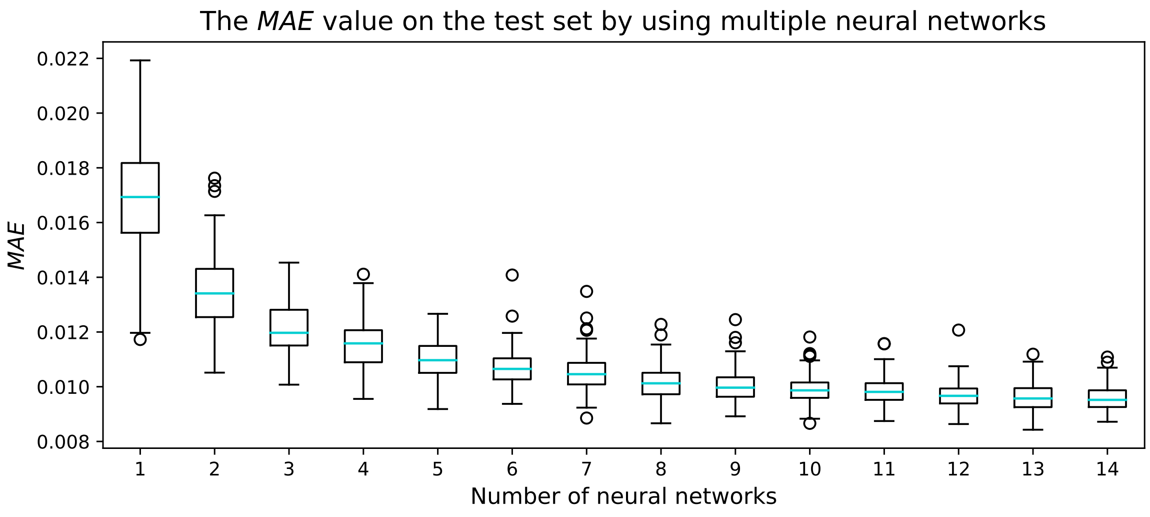


Figure 32: The MAE values on the test set when multiple neural networks are used.

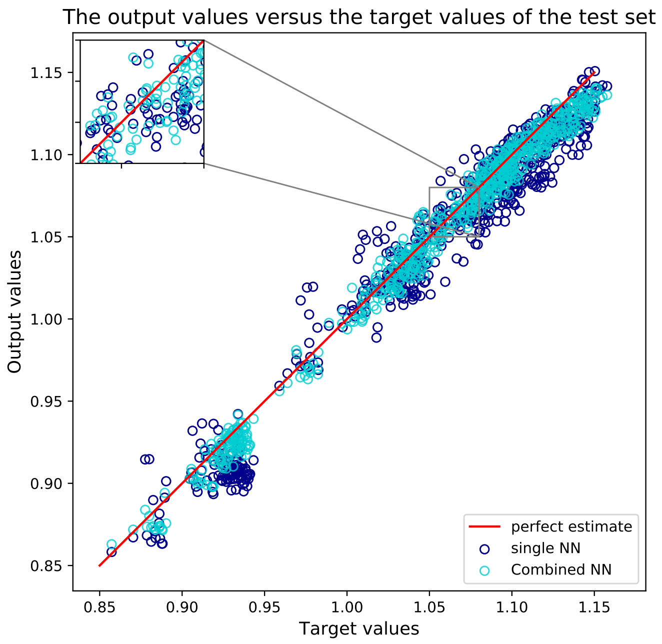


Figure 33: The target versus the predictions of a single neural network and fourteen neural networks combined.

All in all, it is concluded that using more neural networks to create an estimate of the throughput is beneficial. However, the influence on the performance of adding an additional neural network reduces when more neural networks are considered. The influence on the running time when it is combined with the evolutionary algorithm should also be taken into account to reach a final decision on the number of neural networks that should be used. If running time does not have to be taken into account, then using fourteen neural networks is best.

The results indicate that the second method that was discussed in Section 4.7 significantly performs worse than the first method. For brevity, the results are not presented here.

6.2.9 The Ninth Experiment: Data Set Size Influence

Until now only data sets of 25000 rows have been considered. In general it holds that a larger data set reduces the error on unseen data. In this section, the influence of the data set size is further investigated.

The fact that the data sets were discretised in the second experiment results in a reduced data space. Without discretisation, the data space is infinite. With discretisation, it is possible to calculate the data space size with the considered parameters for the arrival rates, service rates, number of stations and buffer size (see Section 4.3.1). When a single row of a data sets is generated, the number of stations i is randomly selected. Then, the number of available buffer slots is randomly selected (ranging between $i+1$ and 30) and one random buffer allocation with B buffers is selected. Subsequently, an arrival rate is selected that can range between 0.5 and 2.5 with $\Delta = 0.025$, and then the service rates at each station are selected. These range from the arrival rate + 0.025 until 3.0 with $\Delta = 0.025$. By counting the number of possibilities in each step, the total data space size can be determined.

Table 5 presents the total number of different buffer allocations that exist for a queuing system of i stations with $i+1$ until 30 available buffer slots. The number of different arrival-service combinations can be determined by using:

$$\sum_{j=20}^{100} j^i. \tag{81}$$

Notice that the number of service rate combinations depends on the selected arrival rate. If, for example, the arrival rate is set at 0.65, there are 94 different service rates available (the number of values between 0.675 and 3.0 with $\Delta = 0.025$). The lowest number of possible service rates occur when the arrival rate is set at 2.5, and is equal to 20. The largest number of possible service rates occur when the arrival rate is set at 0.5, and is equal to 100. This results in the $\sum_{i=20}^{100}$ part of formula 81. Each station may have a different service rate, and therefore the total number of arrival and service rates combinations can be calculated by formula 81. Multiplying these number of combinations with the respective values of Table 5 then results in Table 6 and a total data set space of 2.45×10^{16} for the given parameters of Section 4.3.1 and a discretisation of $\Delta = 0.025$. This means that a data set of 25000 rows represents $1.02 \times 10^{-10}\%$ of the total data space size. A data set of 25000 rows is thus a very small data set compared to the total data space size. The good performance of the neural network then indicates that the neural network really learns a function that estimates the throughput, instead of simply "memorising" the 25000 data rows.

Table 5: The total number of buffer allocations.

Number of Stations	Total Number of Buffer Allocations
2	434
3	4059
4	27404
5	142505

Table 6: The data space size for each number of stations.

Number of Stations	Size Data Space
2	1.46×10^{08}
3	1.03×10^{11}
4	5.62×10^{13}
5	2.45×10^{16}

To analyse the influence of the data set size on the performance of the neural network, one data set of 100000 data rows is generated in a similar way as the data of experiment four and theoretical features are also part of the data set. One hundred single neural networks are then trained on 10000, 20000, 30000, etc. data rows of this data set and the hundred test errors are saved. Figure 34 presents the MAE values together with 95% confidence intervals on the test set for the differently sized train sets. Clearly, the error reduces when a larger train set is considered. Moreover, the spread of the errors also reduces when a larger train set is used.

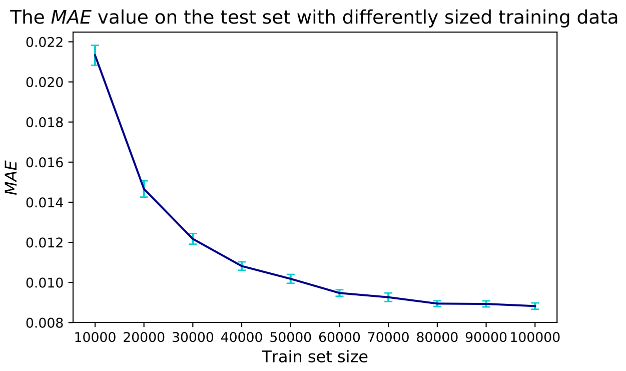


Figure 34: The MAE values together with 95% confidence intervals on the test set for differently sized train sets.

It is also interesting to consider the ranking measures of Section 4.3.3 in combination with the data set size influence. Two types of single neural networks are trained: one on a data set of size 25000 and one on a data set of size 100000. The mean MAE, MAD, MSD and the maximum MD of 100 independent runs on five test set instances are reported in Tables 7, 8, 9 and 10, respectively. The values of the other test instances can be found in Appendix B. The cells with a green colour belong to the neural network with the lowest value of the respective performance measure. The neural network trained on a set of size 100000 has lower values of all performance measures, which was also as expected.

Table 7: The mean MAE on the individual test set instances.

Instance	2_15	3_15	3_30	4_20	5_20
Mean MAE (25000)	0.0152	0.0121	0.0224	0.0124	0.0123
Mean MAE (100000)	0.0100	0.0075	0.0169	0.0114	0.0081

Table 8: The mean MAD on the individual test set instances.

Instance	2_15	3_15	3_30	4_20	5_20
Mean MAD (25000)	0.558	2.590	8.041	94.974	79.807
Mean MAD (100000)	0.474	1.655	5.964	78.224	52.229

Table 9: The mean MSD on the individual test set instances.

Instance	2_15	3_15	3_30	4_20	5_20
Mean MSD (25000)	1.284	23.718	212.397	29920.477	22900.810
Mean MSD (100000)	0.890	9.748	115.652	20572.881	9720.997

Table 10: The maximum MD on the individual test set instances.

Instance	2_15	3_15	3_30	4_20	5_20
Maximum MD (25000)	8	42	109	810	1328
Maximum MD (100000)	4	24	74	680	815

6.2.10 The Final Metamodeller

All in all, the final metamodeller is trained on a train set with randomly selected tandem lines and one random buffer configuration per tandem line (according to the fourth experiment). The theoretical features have a profound influence on the performance of the neural network and these are therefore also part of the data set. Using multiple neural networks is beneficial, and the best metamodeller is obtained by taking the mean estimates of fourteen independent neural network. To obtain insights into the performance of the final developed metamodeller, two training data sets are considered: one of 25000 data rows, and one of 100000 data rows. The previously used test set (see Appendix A) was used to select the number of neural networks in the eight experiment, which means that the test set does no longer provide an unbiased estimate of the error on unseen data. Therefore, a new validation set is generated, that contains 11 total enumerations. Appendix A contains the parameters of these total enumerations.

The parameters that were selected in experiment seven (see Appendix B) are used. The mean RAE values on the validation set with a model trained with 25000 and 100000 rows are equal to 0.811% and 0.614%, respectively. These values are low enough for optimisation purposes. Moreover, they are lower than the errors of the "classical" machine learning approach of the third experiment.

Figure 35 presents a plot of the estimated throughputs versus the true throughputs of a queuing system with 4 stations and 21 buffers for the final metamodeller trained on a data set of 100000 rows. The plot follows the line $x = y$, which indicates that the metamodeller creates good estimates. This can also be seen in Figure 36, which is a plot of the true throughputs versus the estimated throughputs. The RAE value of this instance is equal to 0.461%.

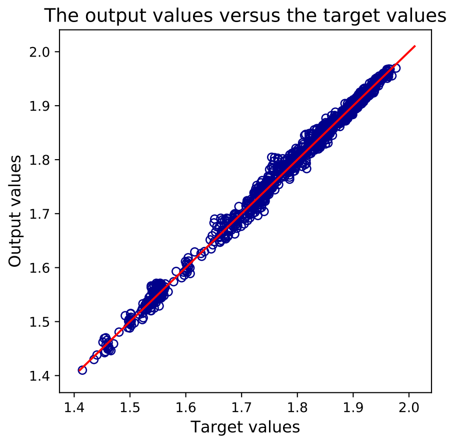


Figure 35: The output values versus the target values of the final metamodeller.

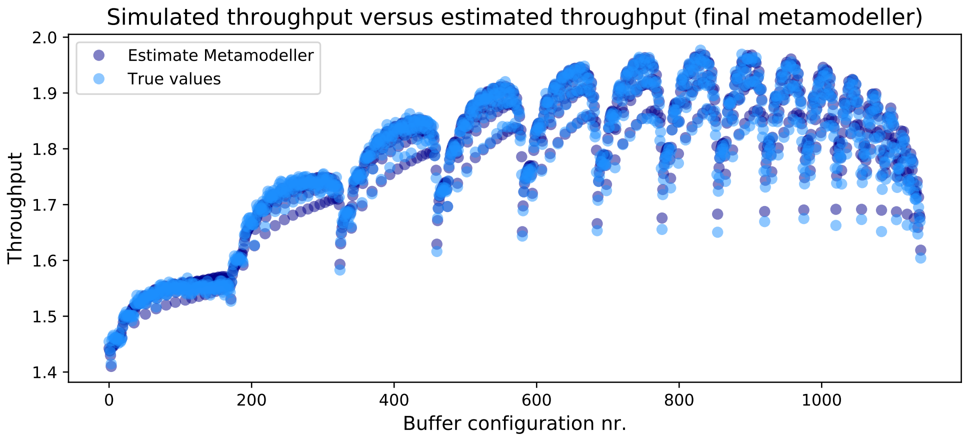


Figure 36: The enumerated values versus the estimated values of the final metamodeller.

6.2.11 The Benchmark Models

In this section, the performance of the developed metamodeller is compared to the performance of two benchmark models. The two regression models as discussed in Section 4.9 are considered. The same training set as in the previous section is used, but both theoretical and non-theoretical versions are considered. The training data set contains 100000 data rows. The validation set introduced in the previous section is used.

The mean *RAE* values on the validation set of the linear and exponential models trained on the non-theoretical data set are equal to 8.517% and 8.147%, respectively. The mean *RAE* values on the validation set of the linear and exponential models trained on the theoretical data set are equal to 1.622% and 2.575%, respectively. These values emphasise yet again the positive impact of the theoretical features on the performance of the models. Nevertheless, these *RAE* values are higher than the ones obtained by the metamodeller. The linear regression model outperforms the exponential regression model in case of a theoretical data set.

Figure 37 presents a plot of the true and estimated throughputs of a queuing system with 4 stations and 21 buffers for the linear regression model trained on the theoretical data set. Clearly, the regression model is not able to satisfactorily estimate the throughput. This is also reflected in Figure 38, which is a plot of the estimated throughputs versus the true throughputs. Notice that this is the same instance as in Section 6.2.10 and that Figures 35 and 36 show much better fits. The *RAE* corresponding to this instance is 1.504%. Considering these graphs, it would not be advised to use this linear regression model to estimate the throughput. Linear regression models were also build with a train set of 25000 data rows. These models performed worse, and for brevity, the results are not presented here.

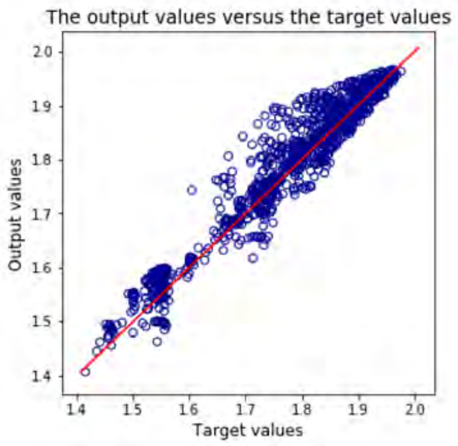


Figure 37: The output values versus the target values of a linear regression model trained on a theoretical data set.

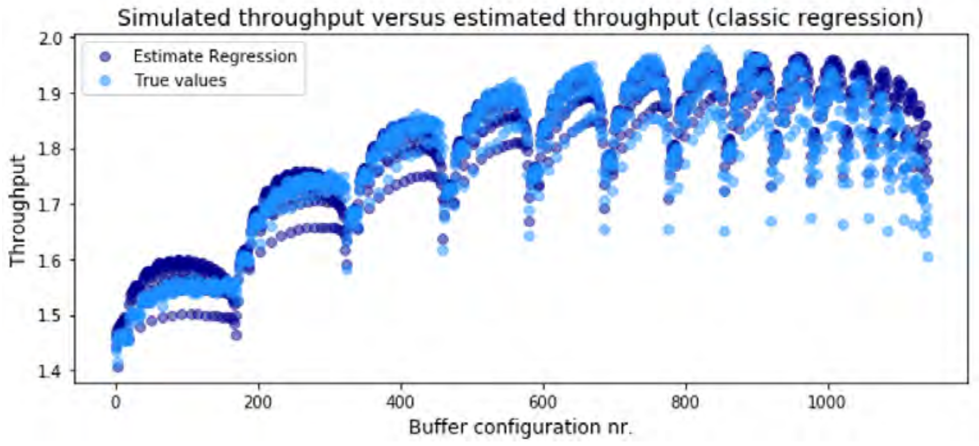


Figure 38: The enumerated values versus the estimated values versus of a linear regression model trained on a theoretical data set.

Table 11 presents the mean RAE values on the validation set of the two benchmark models and the mean RAE values of the two final metamodellers trained on a data set of 25000 rows and 100000 rows (see Section 6.2.10). Clearly, the developed metamodellers outperform the two benchmark models tremendously. Even the metamodeller trained on 25000 data rows outperforms the benchmark models, even though the benchmark models are trained on a data set of 100000 rows.

Table 11: The mean RAE values of the final metamodellers and the benchmark models.

	Linear Model	Exp. Model	Metamodeller (25000)	Metamodeller (100000)
RAE	1.622%	2.575%	0.831%	0.620%

6.2.12 Error Insights

The method proposed in Section 4.8 is used here to gain insights into the certainty of the throughput estimates.

The empirical distributions of E, E_{abs} and $\hat{\sigma}$ are determined by using the test set introduced in Section 6.2.3. The train set with 100000 data rows (including theoretical features) of Section 6.2.10 is used to train the neural networks. Fourteen independent neural networks are used and the steps of Algorithm 3 are followed. Figure 39 plots E_{abs} against $\hat{\sigma}$ for the test set together with the fitted 95%-quantile regression line. The fitted regression line is given by:

$$E_{abs} = 0.0133 + 1.0152\hat{\sigma}. \tag{82}$$

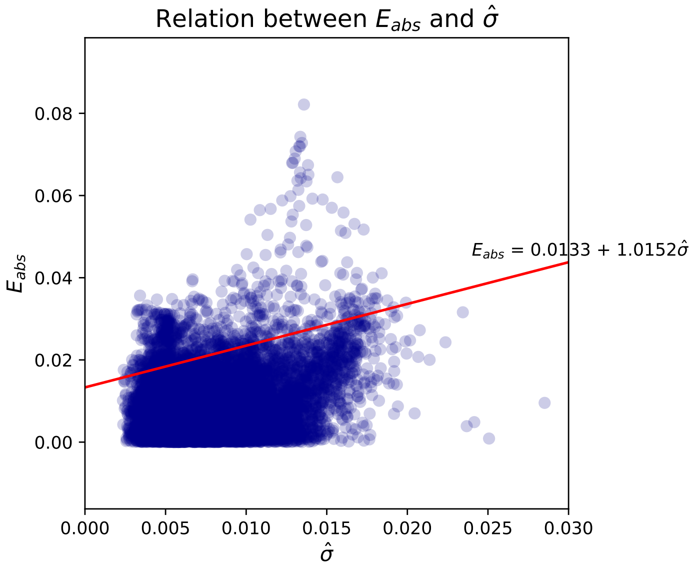


Figure 39: A plot of E_{abs} against $\hat{\sigma}$ together with the fitted 95%-quantile regression line.

This fitted regression line is used together with an unseen data point D to gain insights into the error made by the neural networks. The estimates of the throughput for data point D made by the fourteen neural networks are provided in Table 12. The sample standard deviation $\hat{\sigma}^D$ is given by 0.0100, and filling this into function (82) yields that it is 95% certain that the absolute error is not greater than 0.0234. Figure 40 plots E against $\hat{\sigma}$ and the two fitted 97.5%-quantile and 2.5%-quantile regression lines. These two regression lines can be used to calculate a 95%-confidence interval for the error of point D, and this is given by [-0.0188, 0.0274]. By adding the mean estimate of the throughput for data point D, a 95%-confidence interval for the throughput estimate can be created, which is given by [1.428, 1.474].

Table 12: The throughput estimates of the fourteen neural networks.

Neural Network	τ_j^D
NN_1	1.455
NN_2	1.438
NN_3	1.463
NN_4	1.433
NN_5	1.451
NN_6	1.466
NN_7	1.451
NN_8	1.446
NN_9	1.451
NN_{10}	1.442
NN_{11}	1.431
NN_{12}	1.448
NN_{13}	1.444
NN_{14}	1.438

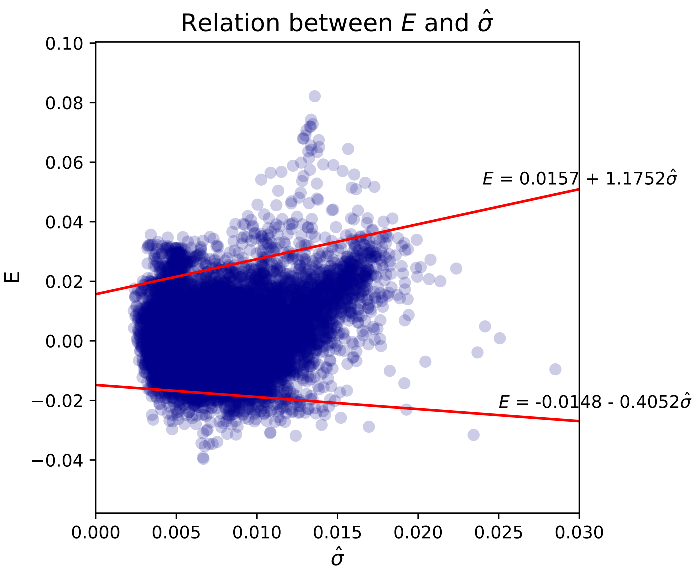


Figure 40: A plot of E against $\hat{\sigma}$ together with the fitted 97.5%-quantile and 2.5%-quantile regression lines.

To validate that this approach is accurate, the algorithm is applied to the validation set of Section 6.2.10. Figure 41 plots E_{abs} against $\hat{\sigma}$ for the data of this validation set. The regression line that is plotted is the 95%-quantile line (82) fitted on the test data. Intuitively, at most 5% of the data points should lie above this line. Of all the data points, 3.92% lies above this line, which is thus good. However, the picture changes when bins of size 0.005 are considered (Figure 42). In bin [0.000, 0.005], 11.92% of the data points lie above the fitted regression line, which is more than the expected 5%. This is thus an indication that the method is not entirely accurate. The method can be used to gain some insights into the error and the accuracy, but further research is required to create a statistically valid approach.

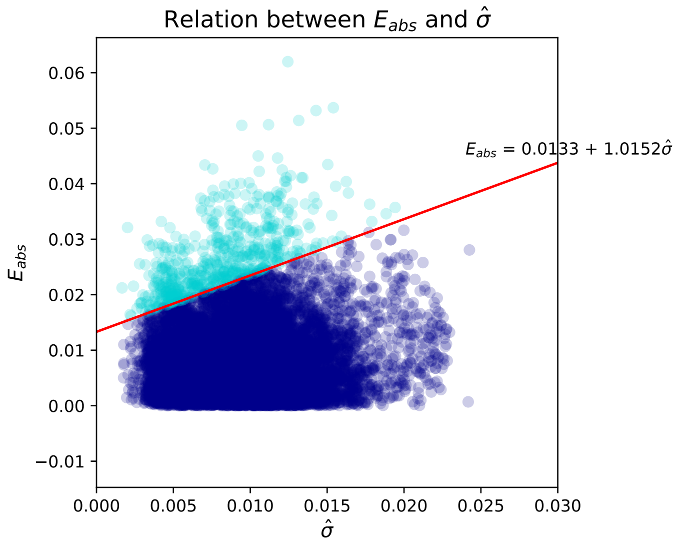


Figure 41: A plot of E_{abs} against $\hat{\sigma}$ on the validation set together with the fitted 95%-quantile regression line.

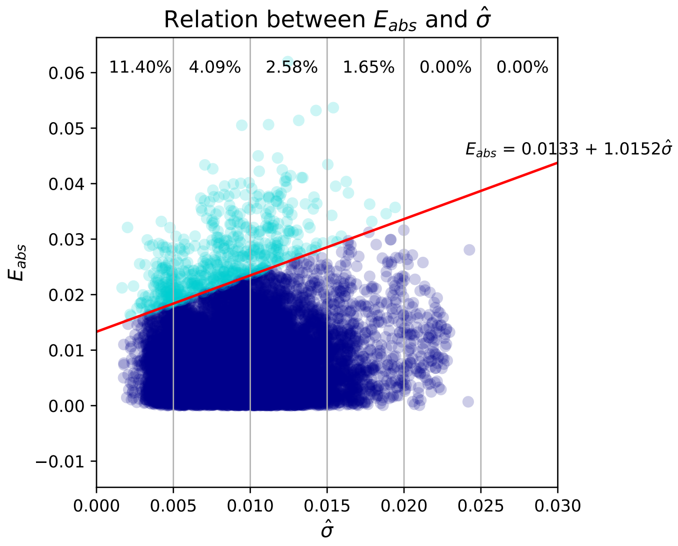


Figure 42: A plot of E_{abs} against $\hat{\sigma}$ on the validation set together with the fitted 95%-quantile regression line.

6.3 Evolutionary Algorithm

To select the best representation, mutation and crossover operators for the EA, a grid search as described in Section 5.3.5 is performed. Two test instances are used to do so, and these are referred to as EA_4_22 and EA_5_24. The parameters of these instances can be found in Appendix A.

The boxplots of the maximum and mean fitness over all the different configurations of test instance EA_4_22 are provided in Figures 43 and 44, respectively. In the figures, the abbreviation $'U'$ stands for uniform crossover, $'P'$ stands for point crossover and $'B'$ stands for binary representation. The integer representation with uniform crossover and *inc_dec* mutation seems to perform best, as it attains the maximum fitness most often, and has the highest mean fitness. This is also the case for test instance EA_5_24 of which the figures can be found in Appendix B.

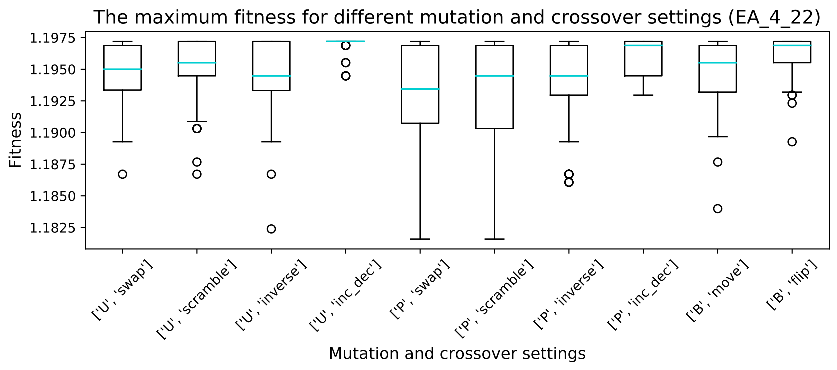


Figure 43: maximum of all configurations.

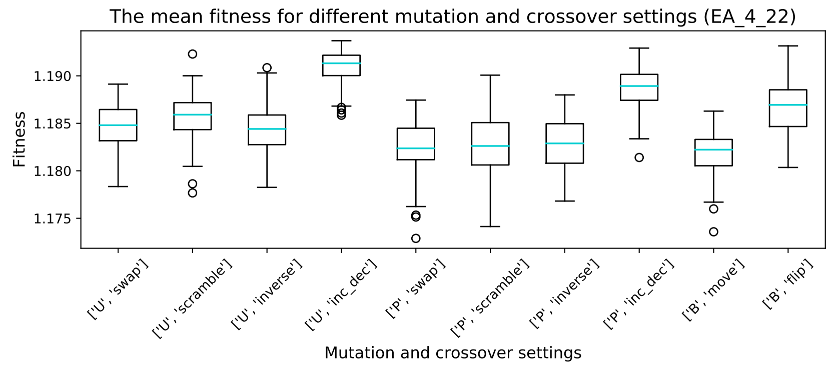


Figure 44: mean of all configurations.

Figure 45 presents the maximum and mean fitness per generation of one EA run for instance EA_4_22. The integer representation seems to outperform the binary representation. The performance of all integer representation configurations seem comparable in these two plots, but the *inc_dec* mutation operator is perhaps slightly better than the others.

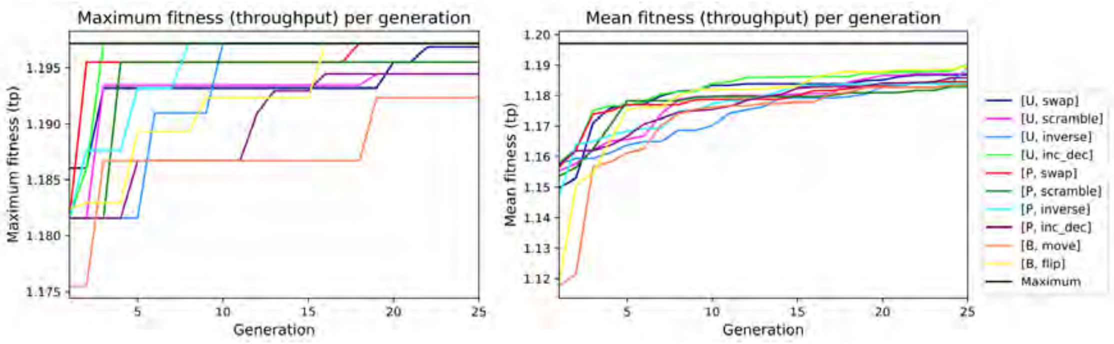


Figure 45: maximum and mean fitness over the generations.

Figures 46 and 47 present the percentage of considered and revisited solutions for each combination of crossover and mutation operators for instance EA_5_24. The figures for instance EA_4_22 can be found in Appendix B. The binary representation configurations seem to consider a higher number of different solutions than the integer representation. This is also accompanied by a lower percentage of revisits. There does not seem to be a significant difference between the percentage of revisits between the mutation and crossover operators of the integer representation.

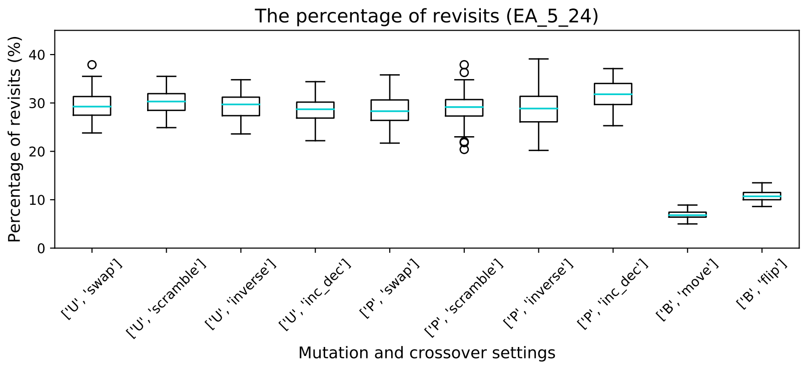


Figure 46: The percentage of revisits for test instance EA_5_24.

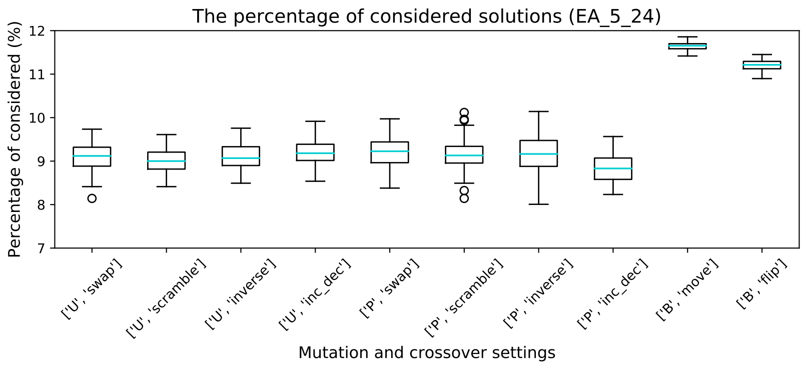


Figure 47: The percentage of considered solutions for test instance EA_5_24.

All in all, the experiments indicate that the integer representation in combination with the uniform crossover and *inc_dec* mutation operator reaches the optimal maximum most often. The binary representation clearly considers a greater number of different solutions, but does not converge very quickly to a maximum.

To really test the performance of the EA, a very large test set is created. The number of stations of this set is equal to five and the number of buffers is 42. This results in 101270 different buffer configurations. The parameters of this large test set can be found in Appendix A. The EA is run with an integer representation, uniform crossover and *inc_dec* mutation operator. The population size is set at 50 and 15 generations are considered. Note that in this experiment the "true" throughputs are used by the EA algorithm as fitness of the solutions.

The algorithm finds the true maximum with a throughput of 1.372 already in generation 10. The algorithm considered 0.67% of all solutions and had 16.27% revisits. It is rather impressive that the algorithm found the maximum so quickly, but this can be explained by looking at Figure 48 that presents all buffer configurations, i.e. solutions, and the solutions that were considered by the algorithm. The crowding behaviour of the algorithm is clearly observable in this figure: the algorithm focuses on exploring solutions with high throughputs, and therefore quickly converges to the maximum.

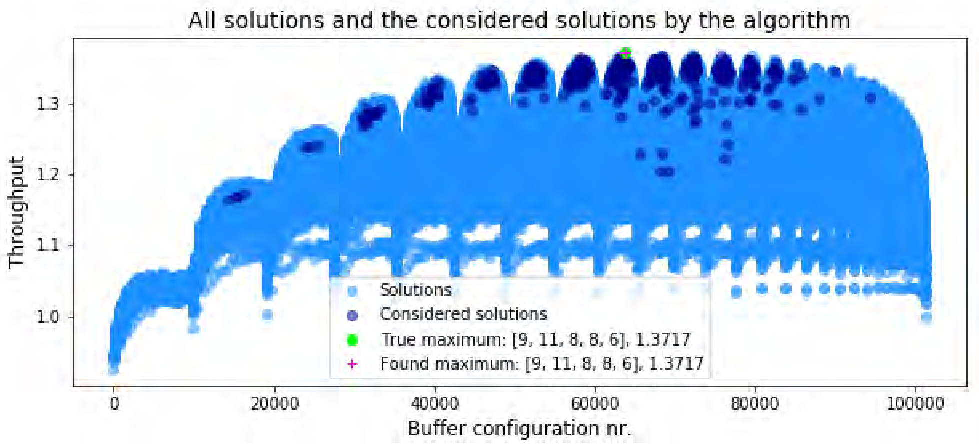


Figure 48: All solutions together with the solutions considered by the algorithm.

6.4 Combining the Metamodeller and the EA

The time has come to combine the evolutionary algorithm with the developed metamodeller. To assess the performance of the combination, the evolutionary algorithm is run 100 times on different test instances. The parameters of these test instances can be found in Appendix A. The fitness of a certain buffer allocation is determined by using the developed metamodeller. The 100 independent EA runs are performed with *pop_size* set to 50 and *nr_generations* set to 25.

It should be noted that we consider two different solution spaces in the following. We have the solution space made up of "true" throughputs and the solution space made up of the throughput estimates of the metamodeller. In the previous section, the EA was used in combination with the "true" throughputs. When the EA is combined with the metamodeller, the EA uses the throughput estimates of the metamodeller as fitness values and the EA tries to find the best solution in the solution space of the metamodeller throughput estimates. It may be the case that the metamodeller deems that the true optimum is not the best solution, which means that the probability is small that the true maximum will be found when the EA is combined with the metamodeller. This should be kept into account when the results are analysed.

In experiment eight it was concluded that using multiple neural networks to create an estimate increases the performance. However, using more neural networks also increases the running time. A short test is conducted to determine what this influence is exactly. One run of the EA is performed on a large instance (EA_5_27 in appendix A) that uses the throughput estimates of a single neural network, and one run is performed that uses the mean throughput estimates of fourteen neural networks. In the first case, a run takes 00:36 minutes and in the second case, a run takes 00:38 minutes. The running time difference between a single neural network and fourteen neural networks is thus negligible, whereas the performance is much better, and therefore in the subsequent always fourteen neural networks are used.

Table 13 presents the results of running the EA combined with a metamodeller trained on a data set of 100000 rows (the data set of Section 6.2.10) on multiple test instances.

The "found solution" that is reported in the table is the solution that is most often found out of the 100 runs. The row "number of times NN max found" represents the number of times that the EA has found the maximum solution according to the throughput estimates of the metamodeller. As previously indicated, this is probably thus not the true maximum. The row "Mean deviation to true max" represents the mean deviation of the *true* throughput of the found solution and the throughput of the true best solution. The mean deviations are quite small, which indicates that the metamodeller combined with the EA provide useful results. The running time of the algorithm is also quite low.

Table 13: The results of the EA combined with a metamodeller trained on a data set of 100000 rows.

Instance	3_30	4_25	4_29	5_25	5_27
Nr. times NN max found	100	97	100	100	100
Mean deviation to true max	0.0118	0.0260	0.0187	0.0169	0.0344
Considered solutions (%)	39.438	24.049	13.073	6.392	4.813
Found solution	[12,12,6]	[18,1,5,1]	[10,10,5,4]	[8,5,6,4,2]	[9,7,5,3,3]
True maximum	[14,9,7]	[12,4,8,1]	[9,10,8,2]	[10,4,5,5,1]	[8,9,5,3,2]
Running time one run (min)	00:19	00:17	00:20	00:37	00:49

Figure 49 is a plot of the true throughputs together with the estimates of the metamodeller for instance 3_30. Moreover, the true maximum and the found solution are also provided. The found solution has a throughput of 2.2605, whereas the metamodeller predicts a throughput of 2.2618 for solution $[12, 12, 6]$.

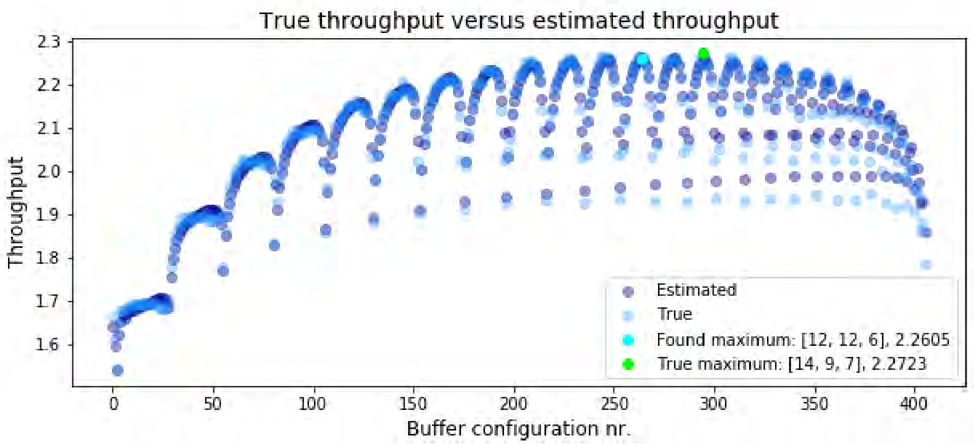


Figure 49: The true throughputs versus the estimates of the metamodeller together with the true and found maximum.

To also show that the metamodeller combined with the EA works well when a smaller data set is used to train the neural networks, the metamodeller consisting of fourteen neural networks is trained on a data set of 25000 data rows (representing only $1.02 \times 10^{-10}\%$ of the total data space). Table 14 presents the results of running the EA combined with this metamodeller. The mean deviations are quite small, which indicates that the metamodeller combined with the EA will also provide useful results when not a large data set is available for training the metamodeller. Interestingly, in some cases the mean deviation to the true max is even smaller than that of the metamodeller trained on a data set of 100000 rows (for example instance 4_29). The reason for this is that the metamodeller performs a regression task, and although the metamodeller trained on a data set of 100000 rows has lower performance measures, it does not mean that it always has a better idea than the metamodeller trained on 25000 rows about which of the allocations is the true maximum.

Table 14: The results of the EA combined with a metamodeller trained on a data set of 25000 rows.

Instance	3_30	4_25	4_29	5_25	5_27
Nr. times NN max found	100	97	100	100	100
Mean deviation to true max	0.0143	0.0241	0.0074	0.0280	0.0250
Considered solutions (%)	38.803	23.463	12.864	6.156	4.675
Found solution	[12,11,7]	[14,2,6,3]	[10,9,6,4]	[9,5,5,4,4,2]	[8,8,5,4,2]
True maximum	[14,9,7]	[12,4,8,1]	[9,10,8,2]	[10,4,5,5,1]	[8,9,5,3,2]

6.5 Simulation versus the Metamodeller

One of the research questions of this thesis is whether the metamodeller combined with the EA outperforms a simulation-based EA. This will be further investigated in this section. The accuracy of the solutions that are found by both solution methods are compared. Moreover, the running times of both methods are also compared.

In case of a simulation-based EA, the fitness of a solution is evaluated by using the DES of Section 3.3. Due to extensive running times of the DES, the fitness of an individual is evaluated by performing one simulation run of 2000 time units (a warm-up period of 750 time units is still considered). The fitness is saved in a table that can be used when the fitness of the same buffer allocation is required in a later step of the algorithm. This is done to further reduce the running time.

In case of both configurations (simulation-based EA and metamodeller-based EA), ten independent EA runs are performed with *pop_size* set to 50 and *nr_generations* set to 25. Due to extensive running times of the simulation-based EA, it was infeasible to consider more than ten EA runs. In each run, fourteen new (and independent) neural networks are trained on a theoretical data set of size 25000 and 100000. Furthermore, the seeds of the simulation are set at other values each run, such that also a "new" simulation is used. Tables 15, 16 and 17 present the results of these runs.

The running times of the simulation-based EA quickly increase when a larger and more difficult instance is considered (Table 15), whereas the running time of the metamodeller-based EA is much more constant for all instances. The running time of the metamodeller-based EA consists of two elements. First of all, the metamodeller has to be trained on a data set. Secondly, the EA has to be run while the trained metamodeller is used to determine the fitness of solutions. The running times of the metamodeller-based EA also include the training of the metamodeller in Table 29, which has only to be done once for all instances. Thus, the actual running time is even shorter. It takes on average

00:53 minutes to train the metamodeller on a data set of size 25000 and 01:40 minutes to train the metamodeller on a data set of size 100000. If the accuracy of the simulations is increased, then the simulation-based EA will outperform the metamodeller-EA in terms of accuracy, in theory. However, the running times will then also increase further which is a significant downside.

The simulation-based EA performs the worst on three instances, and the best on two instances (4_25 and 5_27). Note, however, that the metamodeller-based EA reaches very similar results on instances 4_25 and 5_27 (Table 16). Recall that "Mean deviation to true max" represents the mean deviation of the *true* throughput of the found solution and the throughput of the true best solution. The cells with a green colour have the smallest mean deviation, whereas the cells with a red colour have the largest mean deviation. The standard deviation of the deviation of the simulation-based EA is usually the highest (Table 17), which indicates that the simulation-based EA is less stable than the metamodeller-based EA.

All in all, the results indicate that the metamodeller-based EA provides good solutions, comparable to, and often even better than, those of the simulation-based EA, in a much shorter amount of time.

Table 15: The running times (in minutes) of the EA combined with the metamodeller and the simulations.

Instance	3_30	4_25	4_29	5_25	5_27
Mean running time one run simulation (min)	02:25	04:35	07:09	12:59	13:01
Mean running time one run NN (inc. training large) (min)	01:53	01:56	01:57	02:10	02:18
Mean running time one run NN (inc. training small) (min)	01:07	01:10	01:12	01:24	01:33

Table 16: The mean deviation of the EA combined with the metamodeller and the simulations.

Instance	3_30	4_25	4_29	5_25	5_27
Mean deviation to true max simulation	0.0133	0.0230	0.0180	0.0248	0.0228
Mean deviation to true max NN (large)	0.0120	0.0277	0.0124	0.0147	0.0321
Mean deviation to true max NN (small)	0.0129	0.0259	0.0131	0.0182	0.0251

Table 17: The standard deviation of the deviation of the EA combined with the metamodeller and the simulations.

Instance	3_30	4_25	4_29	5_25	5_27
Std simulation	0.0068	0.0072	0.0071	0.0096	0.0069
Std NN (large)	0.0024	0.0020	0.0023	0.0044	0.0069
Std NN (small)	0.0020	0.0060	0.0036	0.0089	0.0083

7 Application: Multi-Server Buffer Allocation Problem (MSBAP)

In this section, the developed solution method is applied to the Multi-Server Buffer Allocation Problem (MSBAP).

7.1 Problem Definition

In the Multi-Server Buffer Allocation Problem (MSBAP), the considered tandem lines are made from $M/M/S_i/K_i$ queues (where $K_i = S_i + C_i$). The difference with the SSBAP problem is thus that at each station multiple servers may be present. This makes the problem slightly more difficult, as not only the service speed, but also the number of servers determine where the bottleneck of the network is.

7.2 Neural Network

A similar approach as with the SSBAP problem is adopted to develop a neural network. The main difference is that now also data on the number of servers at each station should be part of the data set.

In Section 6.2.2 we saw that discretising the arrival and service rates is beneficial. Therefore, the arrival and service rates are also discretised with MSBAP, and a Δ of 0.025 is again used. The total number of servers that can be present in a tandem line ranges between the number of stations of the line and ten. The reason for this is that the problem should not become too large, while still maintaining an interesting optimisation problem.

7.2.1 The First Experiment: Discretisation and Classic Machine Learning

In this first experiment, two training data sets are generated that contain 25000 rows with total enumerations. The first train set contains data on 12 total enumerations (called $MSBAP\_Test\_1\_train\_1$) and the second train set contains data on 18 total enumerations ($MSBAP\_Test\_1\_train\_2$). The parameters of these two train sets can be found in Appendix C. A list of features of these data sets can be found in Appendix F.

In case of SSBAP we saw that this type of training data does not provide very good results, although it is closest to a common machine learning data set in which the train and test sets are similar. This is probably also the case for MSBAP, and these two train sets thus serve as benchmarks for the amount of improvement that is achieved by the other type of training sets.

Two grid searches are performed to select the parameters for the two train sets. The selected parameters can be found in Appendix D. The MAE value on the first train set is 0.00796 and on the second train set 0.00669. Figure 50 presents the MAE values on the test set. The MAE values are quite high and they are in general also higher compared to the results of SSBAP. This is probably because MSBAP is a more complicated problem than SSBAP. The plots of the MSE and RAE values are provided in Appendix D. The RAE values are on average around 15%.

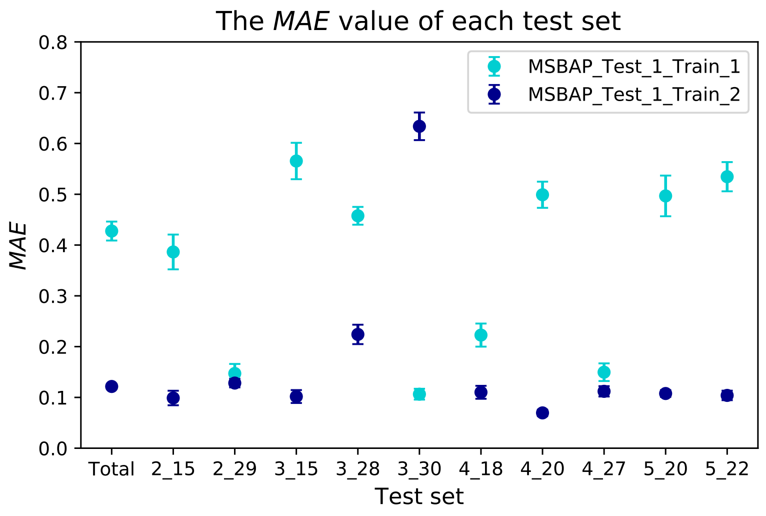


Figure 50: The MAE values together with 95% confidence intervals of each test set.

7.2.2 The Second Experiment: More Tandem Lines

A train set that contains multiple total enumerations does again not provide satisfactory results. In line with experiment four of SSBAP, a new train set is created that contains the data of many different queuing systems with different parameters.

Two training data sets with 25000 data rows are generated. Two grid searches are performed and the selected parameters can be found in Appendix D. The MAE value on the first train set is 0.0162 and on the second train set 0.0160. Figure 51 presents the MAE values on the test set. The plots of the MSE and RAE values are provided in Appendix D. There has been a large improvement compared to the first experiment. The RAE values are on average around 2.0%, which was around 15% in the first experiment. The difference between the performance of the two train sets is also much smaller with this type of training data. Therefore, in the subsequent, only one training data set of each type is used.

In case of SSBAP, two experiments were conducted that analysed the influence of using a balanced data set (Section 6.2.5) and using multiple random buffer allocations per tandem line (Section 6.2.6). The results did not indicate a significant positive influence, and it is expected that this is also the case for MSBAP. Taking the extensive running times of generating data sets into account, these two experiments are not considered in case of MSBAP.

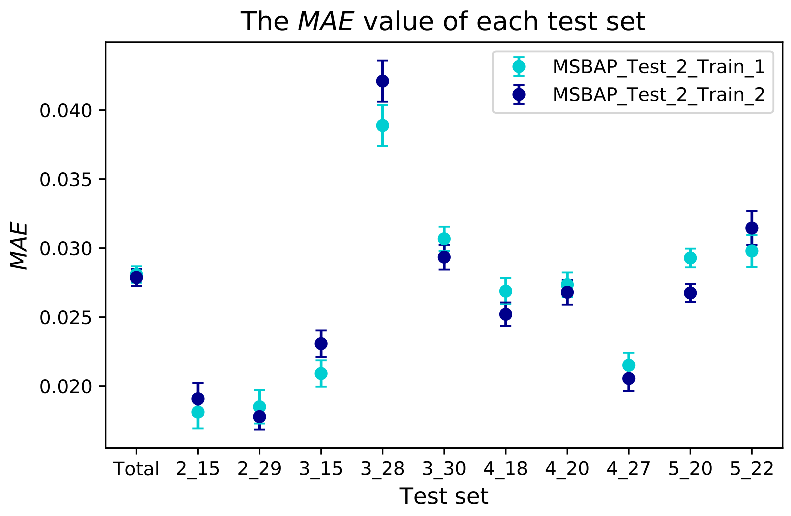


Figure 51: The MAE values together with 95% confidence intervals for each test set.

7.2.3 The Third Experiment: Theoretical Data

In this third experiment, the influence of theoretical features is analysed. The theoretical features that were discussed in Section 4.5.2 are added to the data set of the previous section ($MSBAP\_Test\_2\_Train\_1$), and the methods described in Section 4.6 are used to analyse the performance.

Figure 52 presents the Pearson correlation between all variables. The features $gamma\_1$, $arrival\_rate$, and $gamma\_2$ have a very high correlation with the throughput. The gamma features were also important theoretical features for SSBAP.

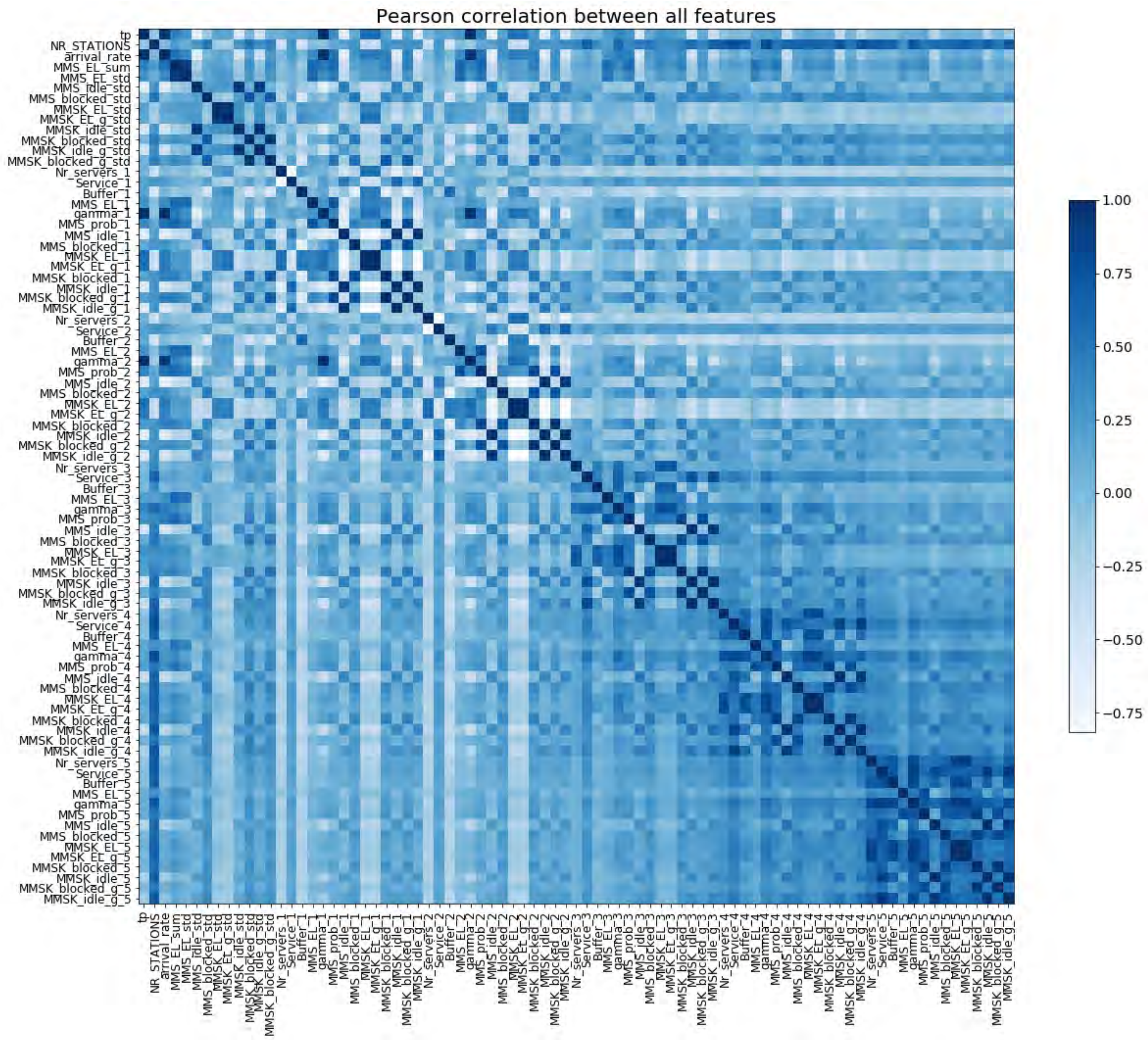


Figure 52: Pearson Correlation between all variables.

A grid search is performed to select the parameters of the neural network in case of the theoretical train set. Appendix D contains the selected parameters. Figure 53 presents a boxplot of the MAE values on the theoretical and non-theoretical data sets. The theoretical data set results in much better performance than the non-theoretical data set. A significant Mann-Whitney U test also confirms this. The average RAE of the theoretical data set is 0.94%, whereas it is equal to 2.00% in case of the non-theoretical data set.

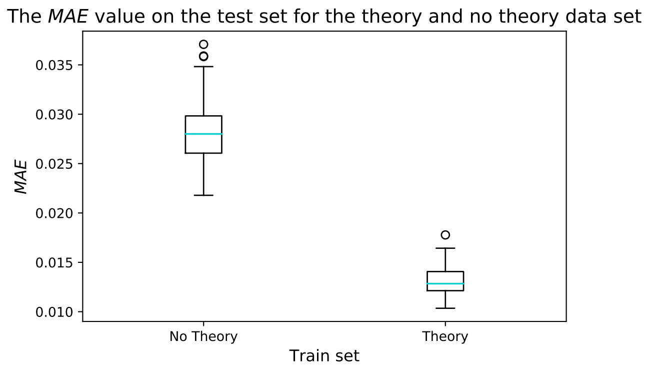


Figure 53: The MAE value on the test set for the theory and no theory data set.

Table 18 shows the ten most important features with their mean importance according to CWA. All features except $arrival\_rate$ and $Nr\_servers\_1$ are theoretical features. The first three most important features are the same as was the case for SSBAP. The importance of the first three features is much higher than that in Table 4.

Table 18: The importance of the ten most important features according to CWA.

Feature	Mean Importance
$gamma\_2$	0.595
$gamma\_1$	0.575
$arrival\_rate$	0.544
$gamma\_3$	0.199
$gamma\_4$	0.128
$gamma\_5$	0.051
$MMS\_blocked\_2$	0.029
MMS_idle_2	0.023
$MMSK\_blocked\_g\_2$	0.023
$Nr\_servers\_1$	0.022

Figures 54 and 55 show the results of the backward and forward selection algorithms, respectively. According to the backward selection algorithm, the feature $MMSK\_blocked\_g\_2$ is most important, whereas the feature $gamma\_1$ is most important according to the forward selection algorithm. Both features are part of the ten most important features according to CWA (Table 18).

To summarise, the theoretical features have a profound positive influence on the performance of the neural network, just as was the case with SSBAP. Especially the gamma features seem important.

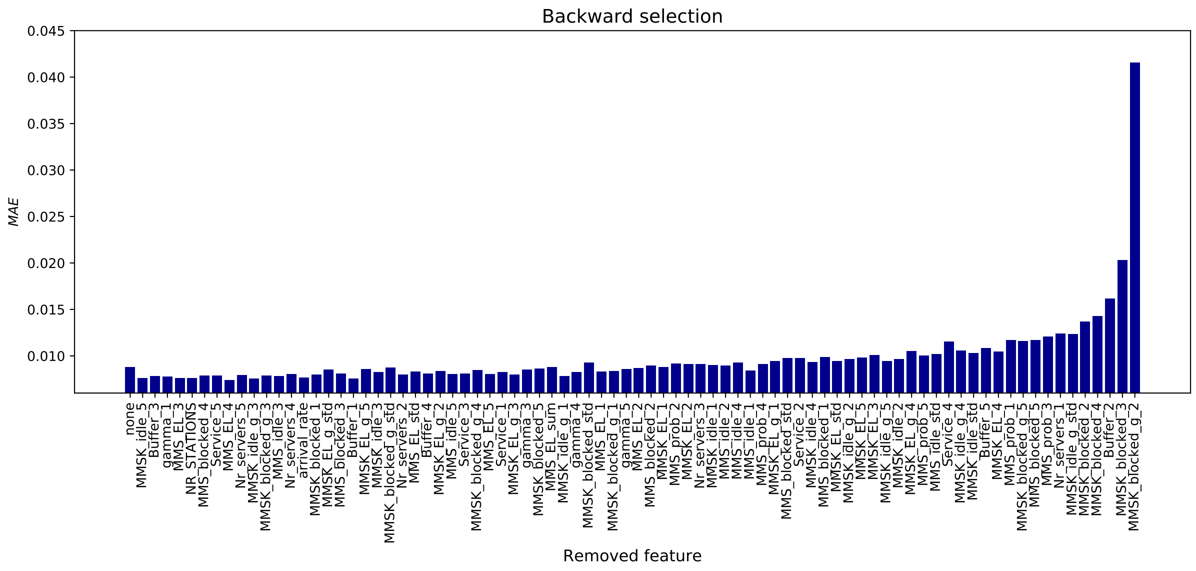


Figure 54: The results of the backward selection algorithm.

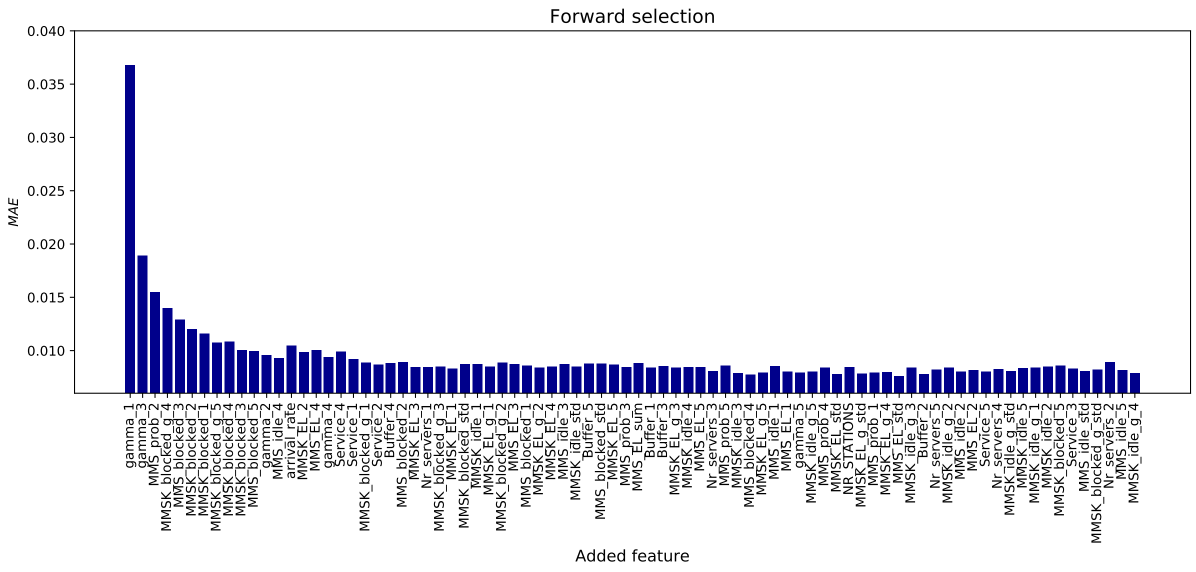


Figure 55: The results of the forward selection algorithm.

7.2.4 The Fourth Experiment: Multiple Neural Networks

The first method of Section 4.7 is used to investigate what the influence of using multiple neural networks is on the performance. The second method is not considered, as it performed worse in the SSBAP case. The data set $MSBAP\_Test\_2\_Train\_1$ is used as training data (including theoretical features). Figure 56 contains the MAE values on the test set for different number of neural networks. Just as was the case with SSBAP, using more neural networks to create an estimate of the throughput is beneficial. This is also confirmed by significant Mann-Whitney U tests. The influence of adding an additional neural network reduces when more neural networks are used. The target versus the predictions created by a single neural network and fourteen neural networks combined is shown in Figure 57. A 4-station tandem line with 20 buffer slots is considered. The estimates of the combined neural network contain less outliers than the estimates of the single neural network. To conclude, also in the case of MSBAP combining the estimates of multiple neural networks is beneficial. Combining fourteen neural networks also seems the best in this case. In case of SSBAP, the reduction in the mean MAE value by using fourteen neural networks instead of one neural network was 43.49%, whereas this is 35.60% in case of MSBAP. It thus seems that the positive influence of using multiple neural networks is slightly smaller than in case of SSBAP.

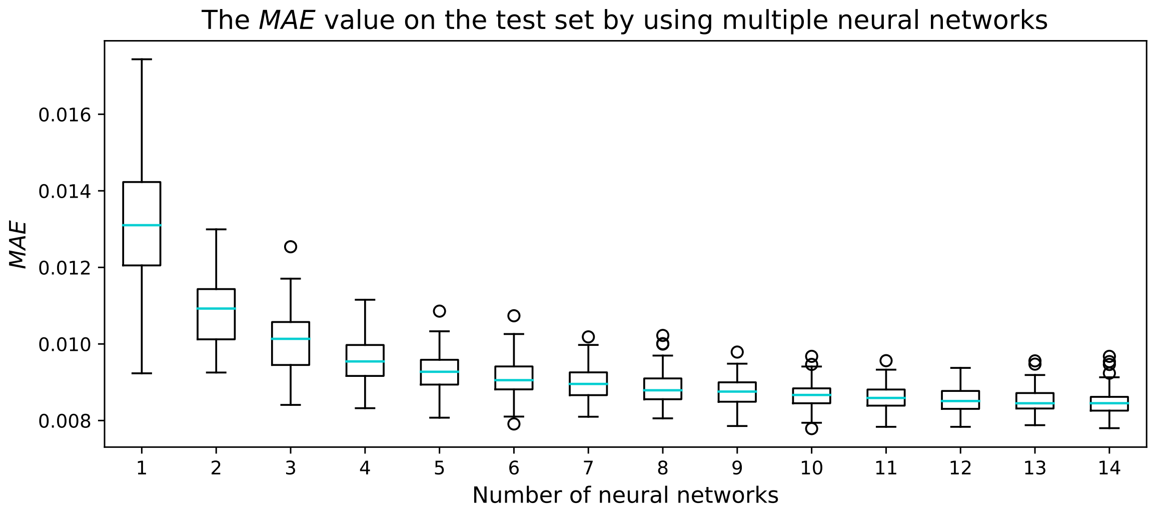


Figure 56: The MAE values on the test set when multiple neural networks are used.

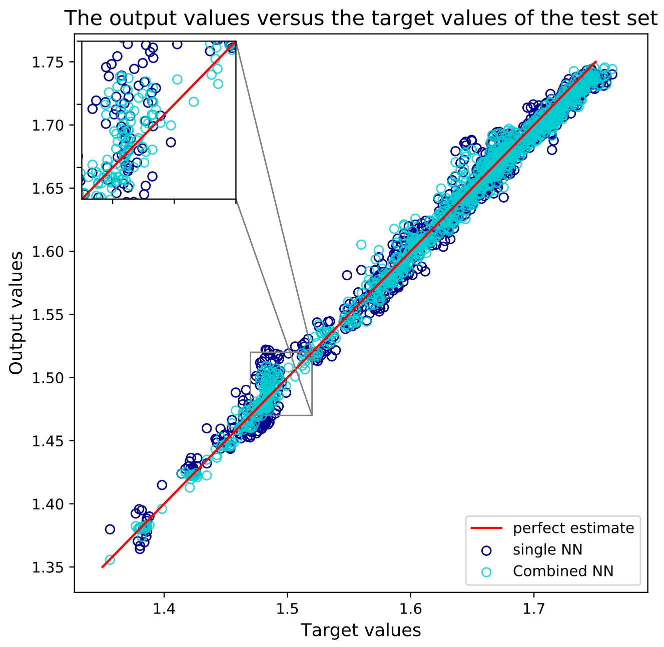


Figure 57: The target versus the predictions of a single neural network and fourteen neural networks combined.

7.2.5 The Fifth Experiment: Data Set Influence

In Section 6.2.9 the size of the solution space was calculated. According to this calculation, the total data space for the considered single-server buffer allocation problem is equal to 2.45×10^{16}. As also the total number of servers is a parameter of MSBAP, the total data set space is even larger. A large part of the computation is similar to that of Section 6.2.9, but now also the varying number of servers has to be taken into account.

Table 19 presents the total number of combinations one can make when the number of stations that can be allocated ranges between I and ten (see the assumptions in Section 4.3.1 and Section 7.1). These values were calculated by the formula:

$$\sum_{j=I}^{10} \binom{j-1}{I-1}. \tag{83}$$

The data space size can then be calculated by multiplying the values in Table 6 by the values in Table 19 which results in Table 20. The total data space size is then 6.28×10^{17}. This means that a data set of size 25000 only comprises $3.98 \times 10^{-12}\%$ of the total data space.

Table 19: The total number of server allocations.

Number of Stations (I)	Total Number of Server Allocations
2	45
3	120
4	210
5	252

Table 20: The data space size for each number of stations.

Number of Stations	Size Data Space
2	6.56×10^{09}
3	1.24×10^{13}
4	1.18×10^{16}
5	6.17×10^{17}

A large data set is generated in a similar way as the data of experiment three. The data set thus also contains theoretical features. One hundred single neural networks are trained on 10000, 20000, 30000, etc. data rows of this data set. The MAE values on the test set are saved and these are depicted in Figure 58. Also in this case both the mean and spread of the errors reduces for a larger data set size.

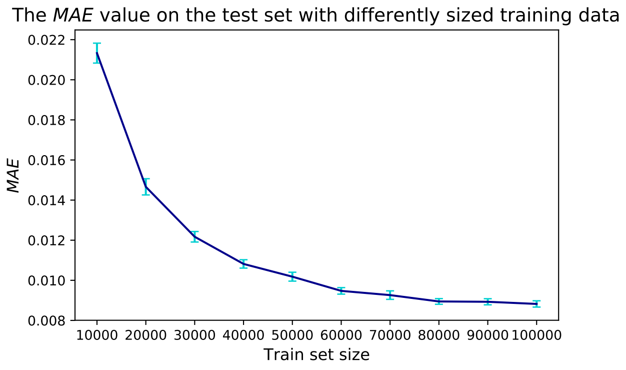


Figure 58: The MAE values together with 95% confidence intervals on the test set for differently sized train sets.

It is also interesting to consider the ranking measures of Section 4.3.3 in combination with the data set size influence. Two types of single neural networks are trained: one on a data set of size 25000 and one on a data set of size 100000. The mean MAE, MAD, MSD and the maximum MD of 100 independent runs on five test set instances are reported in Tables 21, 22, 23, 24, respectively. The tables for the remaining five test set instances can be found in Appendix D. The neural network with the lowest value of the respective performance measure has a green cell colour. In all cases except for one, the neural network trained on a data set of size 100000 is better, which was also as expected.

Table 21: The mean MAE on the individual test set instances.

Instance	2_15	3_15	3_30	4_20	5_20
Mean MAE (25000)	0.0094	0.0164	0.0180	0.0117	0.0160
Mean MAE (100000)	0.0081	0.0123	0.0137	0.0095	0.0105

Table 22: The mean MAD on the individual test set instances.

Instance	2_15	3_15	3_30	4_20	5_20
Mean MAD (25000)	0.899	8.245	11.216	19.987	122.264
Mean MAD (100000)	0.907	6.688	9.044	14.256	77.224

Table 23: The mean MSD on the individual test set instances.

Instance	2_15	3_15	3_30	4_20	5_20
Mean MSD (25000)	2.842	228.295	410.919	1446.136	55293.979
Mean MSD (100000)	2.718	145.225	260.676	761.111	22307.462

Table 24: The maximum MD on the individual test set instances.

Instance	2_15	3_15	3_30	4_20	5_20
Maximum MD (25000)	8	79	144	363	2009
Maximum MD (100000)	7	57	105	243	1139

7.2.6 The Final Metamodeller

To conclude, the final metamodeller is trained on a train set with randomly selected tandem lines and one random buffer configuration per tandem line (according to the second experiment). The theoretical features are also part of the data set, as these have an extensive positive influence. Using multiple neural networks is also better, and the best metamodeller is obtained by taking the mean estimates of fourteen independent neural networks. To obtain insights into the performance of the final developed metamodeller, two training data sets are considered: one of 25000 data rows, and one of 100000 data rows. A validation set is generated that contains 11 total enumerations. Appendix C contains the parameters of these total enumerations.

The parameters that were selected in experiment three (see Appendix D) are used. The mean RAE values on the validation set with a model trained with 25000 and 100000 rows are equal to 1.105% and 0.687%, respectively. These values are low enough for optimisation purposes and much better than the errors of the "classical" machine learning approach of the first experiment.

Figure 59 presents a plot of the true throughputs versus the estimated throughputs of a queuing system with 3 stations and 22 buffers for the final metamodeller trained on a data set of 100000 rows. The plot follows the line $x = y$, which indicates that the metamodeller creates good estimates. This can also be seen in Figure 60, which is a plot of the true throughputs versus the estimated throughputs. The RAE value of this instance is equal to 0.655%.

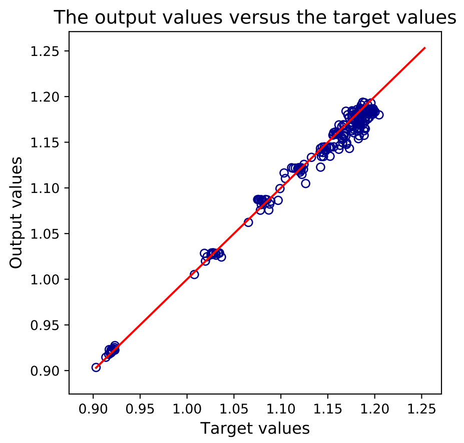


Figure 59: The output values versus the target values of the final metamodeller.

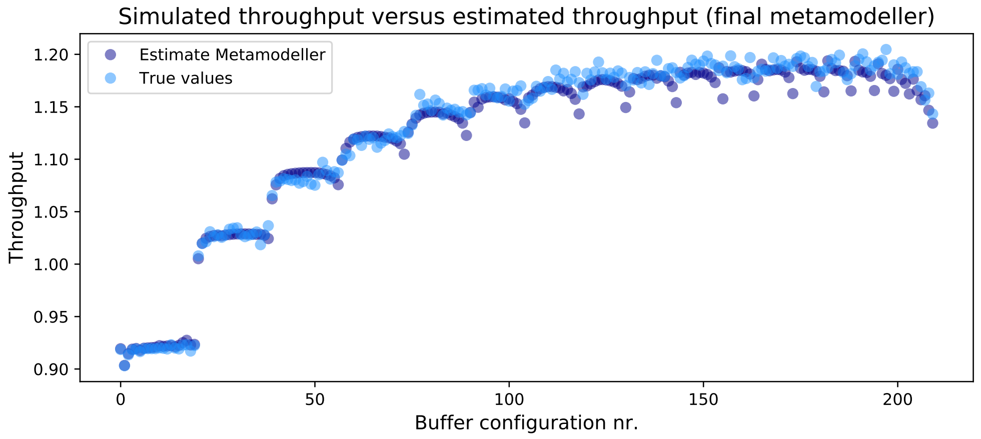


Figure 60: The enumerated values versus the estimated values of the final metamodeller.

7.2.7 The Benchmark Models

The performance of the final metamodeller is compared to the benchmark models that were introduced in Section 4.9. The train set of 100000 data rows of the previous section is considered and regression models are build on both a theoretical and non-theoretical train set. The validation set introduced in the previous section is used to assess the performance.

The *RAE* value on the validation set are 14.444% and 11.548% by using the non-theoretical train set for the exponential and linear models, respectively. The mean *RAE* values on the validation set are 5.172% and 2.470% by using the theoretical train set for the exponential and linear models, respectively. The exponential model performs worse than the linear model, and the models trained on the non-theoretical data set also perform worse than the models trained on the theoretical data set. A mean *RAE* value of 2.470% is still quite high, though, which is also reflected in Figures 61 and 62. Figure 61 plots the target versus the predictions of the linear model on the same validation instance as in the previous section. Figure 62 presents the estimated values versus the true values of the throughput. The estimates are not close to the true values and the *RAE* value of this instance is equal to 2.099%. These estimates are not accurate enough to be used in combination with the evolutionary algorithm. Experiments were also performed with a smaller data set of 25000 rows, but these results were similar and therefore not presented here.

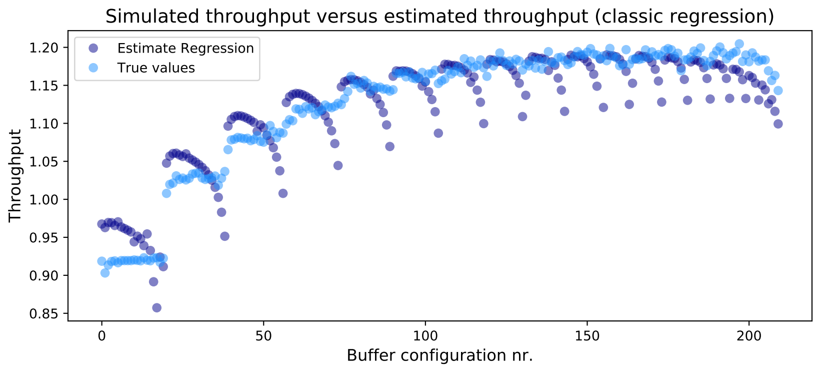


Figure 61: The output values versus the target values of a linear regression model trained on a theoretical data set.

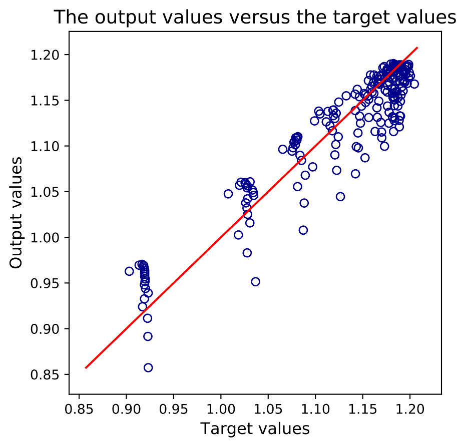


Figure 62: The estimated values versus the true values of a linear regression model trained on a theoretical data set.

Table 25 presents the mean RAE values of the regression models and the final metamodellers on the validation set. The final metamodellers perform much better than the regression models. Even a metamodeller trained on a much smaller data set (25000 data rows) performs better than the regression models build on a much larger data set (100000 rows). Thus, the final MSBAP metamodeller outperforms the MSBAP benchmark regression models.

Table 25: The RAE values of the final metamodellers and the benchmark models.

	Linear Model	Exp. Model	Metamodeller (25000)	Metamodeller (100000)
RAE	2.470%	5.172%	1.105%	0.655%

7.2.8 Comparison SSBAP and MSBAP Final Metamodellers

It is interesting to directly compare the performance of the final SSBAP and MSBAP metamodellers to determine whether one of the two outperforms the other. This can be done by estimating the error on the same data set. Therefore, the SSBAP validation set is transformed into a MSBAP validation set. Thus, these validation sets are identical, except for the fact that the MSBAP validation set contains extra features that represent the number of servers at each station (that are all equal to one). This will make it possible to compare the two models directly. The difference between the models is the training data on which they are trained: the MSBAP final metamodeller is trained on a data set that contains data on multi-server tandem lines, whereas the SSBAP final metamodeller is trained on a data set that only contains data on single server tandem lines.

Table 26 presents the mean RAE values on the SSBAP validation set for the SSBAP and MSBAP metamodellers trained on a train set of 25000 and 100000 data rows. Figure 63 shows the MAE values on the SSBAP validation set for the four models. The performance of the models is quite similar, although the SSBAP model seems to slightly outperform the MSBAP model. A significant Mann-Whitney U test also confirms this. This was also expected, as MSBAP is a more difficult problem than SSBAP.

Table 26: The mean RAE values of the final metamodellers on the same validation set and a train set of 25000/100000 data rows.

Model	Mean RAE (25000)	Mean RAE (100000)
SSBAP	0.811%	0.614%
MSBAP	0.838%	0.669%

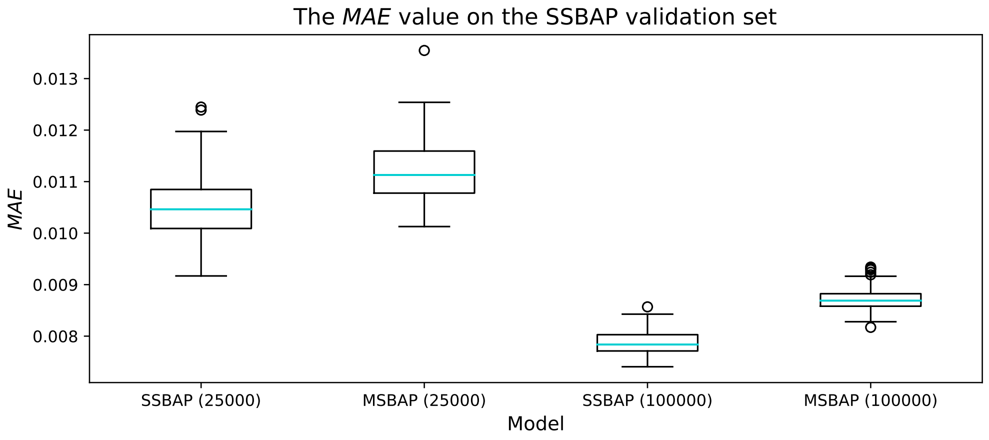


Figure 63: MAE values on the validation set for the four final metamodellers.

7.3 Combining the Metamodeller and the EA

In this section, the final metamodeller is combined with the evolutionary algorithm.

Both representations that are discussed in Section 5.3.1 can be used for this problem. In Section 6.3 it was found that the integer representation outperformed the binary representation. Moreover, the uniform crossover and *inc_dec* mutation operator were selected as the best parameters. The same settings are used here.

The parameters *pop_size* and *nr_generations* are set to 50 and 25, respectively. These are the same as the settings used for SSBAP. The evolutionary algorithm is combined with the final metamodeller trained on a data set of 100000 data rows. Table 27 presents the results of 100 independent runs. The mean deviation to the true maximum is quite small and the running times are again low. Instance 5_25 is apparently a difficult instance, as the EA only finds the NN maximum five times. The mean deviation to the true maximum is quite low though, which indicates that the EA still finds a good solution.

Table 27: The results of the EA combined with a metamodeller trained on a data set of 100000 rows.

Instance	3_30	4_25	4_29	5_25	5_27
Nr. times NN max found	100	100	98	5	100
Mean deviation to true max	0.0103	0.0117	0.0305	0.0043	0.0137
Considered solutions (%)	39.93	21.37	52.01	7.02	4.41
Found solution	[14,9,7]	[12,6,5,2]	[21,1,1,4]	[9,9,3,3,1]	[8,5,6,4,4]
True maximum	[12,10,8]	[12,7,4,2]	[15,7,1,6]	[7,9,2,6,1]	[10,6,4,4,3]
Running time one run (min)	00:17	00:20	00:25	00:38	00:45

Figure 64 shows a plot of the true throughputs versus the estimates of the metamodeller together with the true and found maximum on instance 4_25. The metamodeller estimates are quite close to the true throughputs, and the two maxima also lie close to each other.

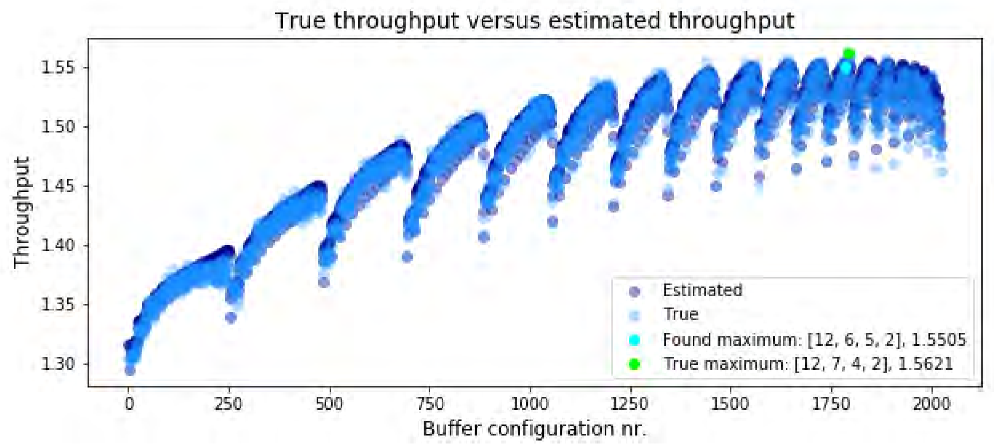


Figure 64: The true throughputs versus the estimates of the metamodeller together with the true and found maximum.

Table 28 presents the results of the final metamodeller trained with a data set of 25000 rows combined with the evolutionary algorithm. The combination works also well in this case. Instance 5_25 is also a difficult case for this metamodeller.

Table 28: The results of the EA combined with a metamodeller trained on a data set of 25000 rows.

Instance	3_30	4_25	4_29	5_25	5_27
Nr. times NN max found	100	100	100	43	100
Mean deviation to true max	0.0026	0.0117	0.0009	0.0151	0.0285
Considered solutions (%)	40.21	20.57	19.22	7.99	4.52
Found solution	[13,11,6]	[12,6,5,2]	[21,1,1,6]	[19,1,2,2,1]	[8,6,6,4,3]
True maximum	[12,10,8]	[12,7,4,2]	[15,7,1,6]	[7,9,2,6,1]	[10,6,4,4,3]

7.4 Simulation versus the Metamodeller

In this section, the performance of the final metamodeller combined with the EA is compared to the performance of a simulation-based EA in a similar way as in Section 6.5.

Tables 29, 30 and 31 present the results of the 10 independent runs. The running time of the simulation-based EA quickly increases for larger and more difficult instances, whereas the running time of the metamodeller-based EA only increases slightly. One should also note that the reported running times of the metamodeller-based EA include training of the neural network. The average training time for a small data set (25000 rows) is equal to 01:12 minutes and for a large data set (100000 rows) to 02:17 minutes. This training has to be performed only once for all instances, which means that the actual running time of the metamodeller-based EA is even shorter when it is applied on an instance.

The mean deviation to the true maximum are for all EAs quite small (Table 30). However, the metamodeller-based EA also outperforms the simulation-based EA in this area. Recall that “Mean deviation to true max” represents the mean deviation of the *true* throughput of the found solution and the throughput of the true best solution. The cells with a green colour have the smallest mean deviation, whereas the cells with a red colour have the largest mean deviation. All mean deviations of the metamodeller-based EA are smaller than those of the simulation-based EA, except for two cases. Moreover, the standard deviations are low and similar (Table 31), and again usually highest in case of the simulation.

In theory, when the accuracy of the simulation is increased, the simulation-based EA will outperform the metamodeller-based EA in terms of accuracy. However, this will also increase the running times even more. This makes the metamodeller-based EA an attractive solution method when a good solution is desired in a short amount of time.

Table 29: The running times (in minutes) of the EA combined with the metamodeller and the simulations.

Instance	3_30	4_25	4_29	5_25	5_27
Mean running time one run simulation (min)	02:13	06:34	05:04	05:35	14:44
Mean running time one run NN (inc. training large) (min)	02:33	02:35	02:37	02:50	02:57
Mean running time one run NN (inc. training small) (min)	01:29	01:30	01:34	01:46	01:52

Table 30: The mean deviation of the EA combined with the metamodeller and the simulations.

Instance	3_30	4_25	4_29	5_25	5_27
Mean deviation to true max simulation	0.0185	0.0245	0.0203	0.0133	0.0230
Mean deviation to true max NN (large)	0.0120	0.0142	0.0209	0.0108	0.0148
Mean deviation to true max NN (small)	0.0071	0.0136	0.0103	0.0128	0.0248

Table 31: The standard deviation of the deviation of the EA combined with the metamodeller and the simulations.

Instance	3_30	4_25	4_29	5_25	5_27
Std simulation	0.0073	0.0099	0.0062	0.0050	0.0084
Std NN (large)	0.0018	0.0050	0.0127	0.0045	0.0078
Std NN (small)	0.0055	0.0088	0.0073	0.0041	0.0078

8 Application: Dashboard

A part of the deliverables for Avanade was a real-time dashboard. This dashboard provides real-time insights into a production line and has the functionality to run the developed evolutionary algorithm combined with the final metamodeller. This dashboard was created to further demonstrate the value of this thesis. The dashboard is programmed in Python with the *Dash* package (Parmer, 2021).

Figure 65 contains an illustration of the pipeline of the dashboard. Data is generated by a real-time simulation programmed in Python. The data is send to a SQL Database, after which it is further processed and send to the dashboard for visualisation. Note that in principle any other manufacturing process could be connected to the dashboard.

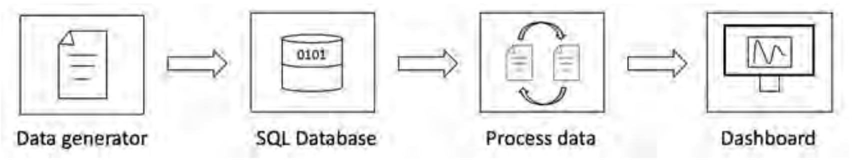
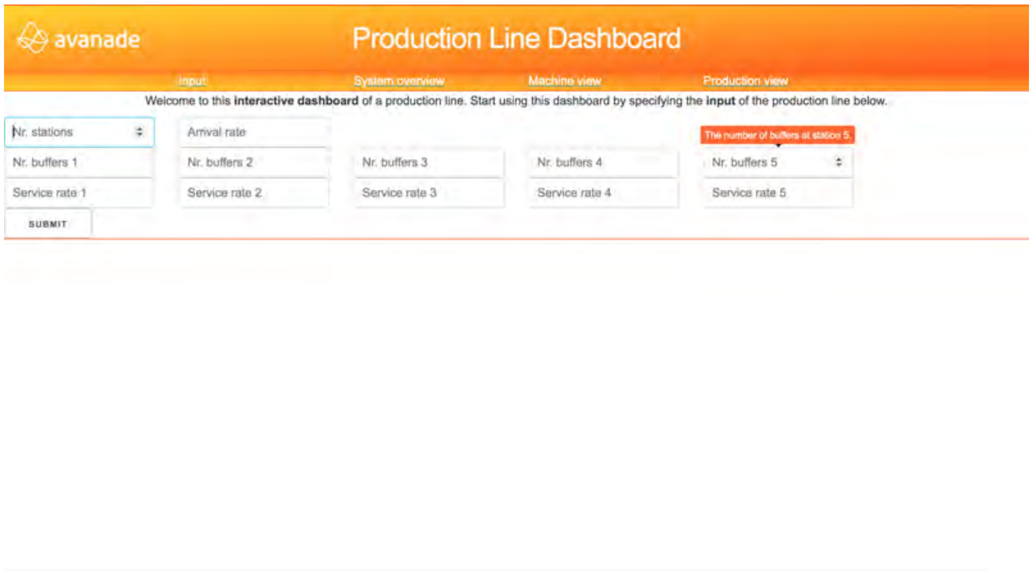


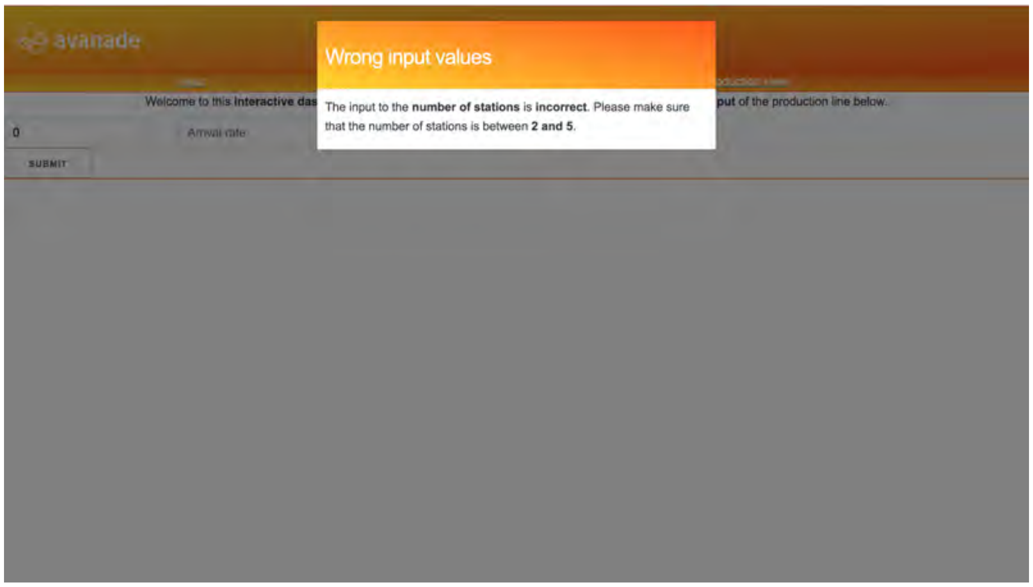
Figure 65: Illustration of the dashboard pipeline.

When the dashboard is launched, the user arrives at the welcome and input screen (Figure 66). The user can provide the parameters of the considered production line. Each input box has a small message box that explains which kind of data should be provided. Various warning messages are provided if the user inputs wrong or incompatible parameters (Figure 67). If the user has provided correct input values, the simulation starts and data is generated. The user is transferred to a new dashboard page (Figure 68) that provides a system overview. It shows (in real-time) how many products are at each station, the mean service rates at each station and the mean arrival rate. It also shows a picture of the tandem line and stations that are currently blocked are highlighted in orange. A button is provided that runs the algorithm. A message is displayed once the algorithm has finished running that contains the current allocation and an improved allocation (Figure 69). The user can also navigate to the machine view (Figure 70) which shows a (real-time) graph of the mean utilisation of each machine, and boxes with the mean utilisation and mean service rate per machine. It also provides the mean total production time of a product and the mean arrival rate. The last page is the production view (Figure 71) which presents the number of waiting products at each station and in total, and a (real-time) graph of the mean waiting time at each station.



Welcome to this **interactive dashboard** of a production line. Start using this dashboard by specifying the **input** of the production line below.

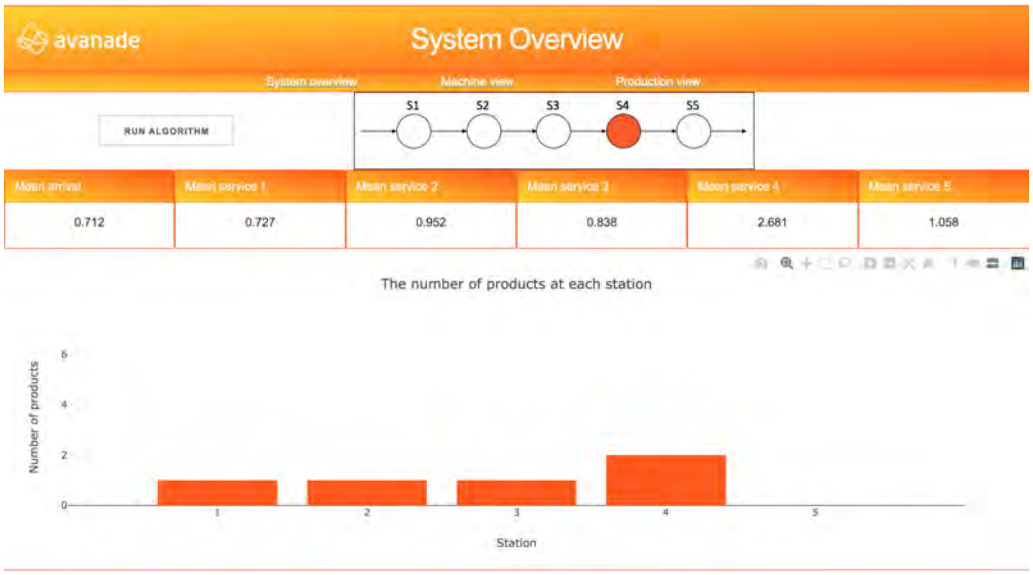
Figure 66: The welcome and input screen of the dashboard.



Wrong input values

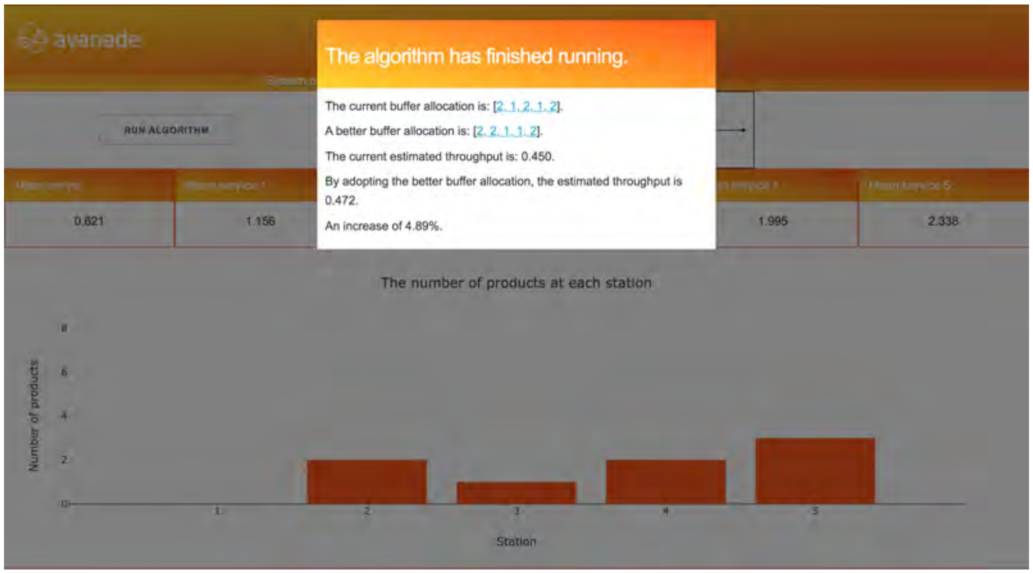
The input to the **number of stations is incorrect**. Please make sure that the number of stations is between **2 and 5**.

Figure 67: One of the warning messages if the provided input is incorrect.



Mean arrival	Mean service 1	Mean service 2	Mean service 3	Mean service 4	Mean service 5
0.712	0.727	0.952	0.838	2.681	1.058

Figure 68: The system overview of the dashboard.



The algorithm has finished running.

The current buffer allocation is: [2, 1, 2, 1, 2].

A better buffer allocation is: [2, 2, 1, 1, 2].

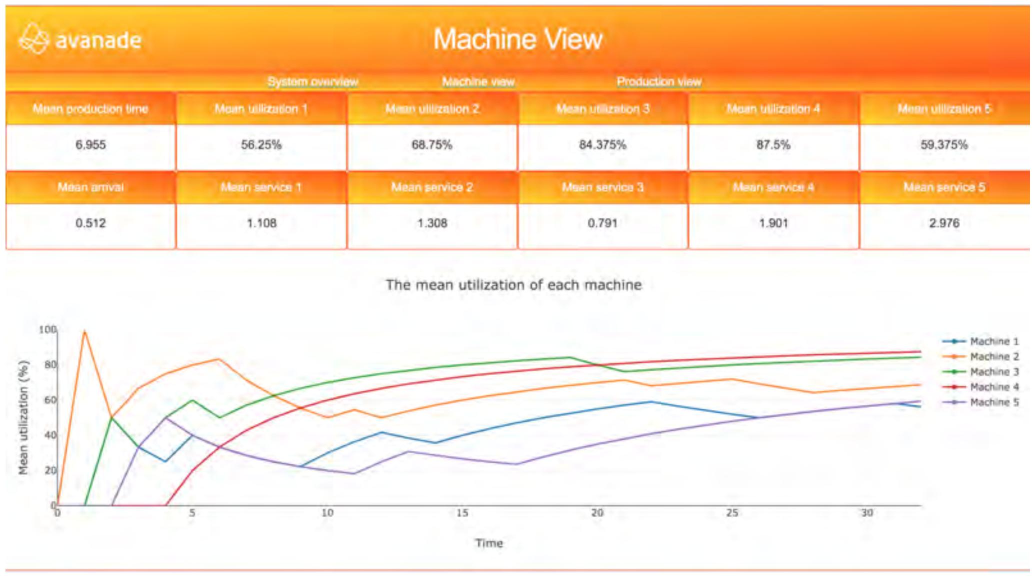
The current estimated throughput is: 0.450.

By adopting the better buffer allocation, the estimated throughput is 0.472.

An increase of 4.89%.

Mean arrival	Mean service 1			Mean service 4	Mean service 5
0.621	1.156			1.995	2.338

Figure 69: The algorithm message at the system overview of the dashboard.

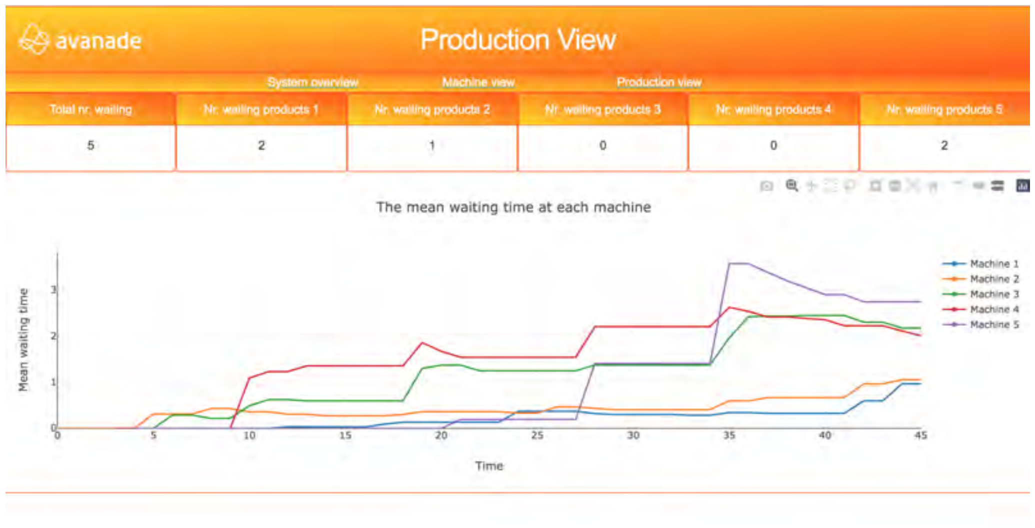

avanade

Machine View

System overview | Machine view | Production view

Mean production time	Mean utilization 1	Mean utilization 2	Mean utilization 3	Mean utilization 4	Mean utilization 5
6.955	56.25%	68.75%	84.375%	87.5%	59.375%
Mean arrival	**Mean service 1**	**Mean service 2**	**Mean service 3**	**Mean service 4**	**Mean service 5**
0.512	1.108	1.308	0.791	1.901	2.976

Figure 70: The machine view of the dashboard.


avanade

Production View

System overview | Machine view | Production view

Total nr. waiting	Nr. waiting products 1	Nr. waiting products 2	Nr. waiting products 3	Nr. waiting products 4	Nr. waiting products 5
5	2	1	0	0	2

Figure 71: The production view of the dashboard.

9 Application: Cloud Networks

Avanade helps customers to migrate to the cloud by using Microsoft Azure. This is a complex process, and typically a cloud solution consists of a network of Microsoft solutions (such as databases, virtual machines, etc.). These networks can also be modelled as queues and a lot of optimisation can be performed on these queues.

Traditionally, on-premises data centres were used to store and process data. If the capacity at these on-premises data centres was met, it was difficult to allocate extra resources to handle the high load. However, when the cloud is used to process the data, it is much easier to allocate extra resources in case of a high load. In fact, if the load of a certain service becomes too high, one can even automatically allocate an extra database, virtual machine, etc. The disadvantage is that this is a rather expensive method of coping with a high load. It would be better to allocate the capacity of existing resources carefully. The methods that have been developed in this thesis can be used to do so. This will reduce the operation costs of the cloud solution, and results in Avanade providing extra value to their customers. In the following, an example is provided of a specific use-case of one of the customers of Avanade.

This customer recently moved to the cloud, which resulted in a shift from a single resource (a database) to multiple cloud solutions (data loading factories, databases, etc), each with their own capacities and buffers. These cloud solutions form an Extract, Transform, Load (ETL) process, which is used to process data transactions. Data is first extracted from data sources, then transformed and cleaned into a usable format, and then loaded into a data warehouse or other data storage solution. Figure 72 is an illustration of such an ETL process. The process can be modelled as a tandem line with three "stations". Each station represents a cloud solution, such as a data warehouse or database and there are buffer slots available in front of each station.

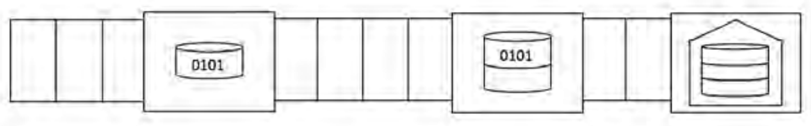


Figure 72: An illustration of a simple cloud network.

Typically, at some points of the day a higher capacity is required at a certain station to be able to process all the load. Currently, this is handled by allocating additional resources to the process (for example by creating more data warehouses), which is very costly. The capacity at the various parts of the queue remains fixed, which also means that some parts of the queue have a high utilisation, whereas other parts of the queue do not. A much better solution would be to distribute the *capacity* itself more efficiently over the queue. Thus, instead of adding more data warehouses to the process, the current capacity is used more effectively. This will reduce the costs considerably. Moreover, optimising the capacity will also increase the throughput of the process, which will reduce the costs even further.

The capacity can be allocated in a more efficient way by using the proposed methods of this thesis. Given a certain cloud network (such as Figure 72), data is available within Microsoft Azure on the throughput of the queue given a certain queue configuration. A metamodeller can be trained on this data, which can then be used in combination with the evolutionary algorithm to optimise the queue configuration. At the time of writing this thesis no data was available to test this, but there is a huge opportunity to reduce the costs by leveraging the proposed methods.

10 Methodology: A Binary Neural Network

An experiment was conducted in which a binary neural network was created instead of a regression neural network. Instead of estimating the throughput, the binary neural network is given data on a tandem line and two buffer allocations, and the binary neural network determines which of the two performs better. This is a natural way of looking at optimisation: one compares two different options and each time decides which one of the two options is best, until, hopefully, the maximum has been found.

The train set contains 17 features: *arrival_rate*, *nr_stations*, *Service_i* (service rate at station i), $Buffer\_i\_1$ (number of buffers at station i in buffer allocation 1), $Buffer\_i\_2$ (number of buffers at station i in buffer allocation 2). The train set is generated by randomly generating five different buffer allocations for a specific tandem line (if there are less than five buffer allocations in total, then all allocations are considered). All possible pairs (at most ten per tandem line) of these five buffer allocations are created and added to the data set with the target 1 if the throughput of the first allocation is lower than the throughput of the second allocation, and 0 else. A train set of size 25000 is generated in this way.

As test set, the test instances introduced in Section 6.2.3 are used. These are transformed into a "binary" test set by creating all possible pairs of buffer allocations and by assigning the 1/0 target depending on which allocation is better.

The balanced accuracy score measure is used as performance measure of the binary neural network. This measure can cope with imbalanced data sets, and is the average recall of each class. Recall is the number of true positives divided by the number of true positives plus false negatives.

As a binary neural network is also a randomised model, 50 independent neural networks are trained and the mean balanced accuracy on each test set instance is saved, which is provided in Table 32.

Table 32: The mean balanced accuracy on each test set instance.

Test Set Instance	Mean Balanced Accuracy
2_15	0.715
2_29	0.866
3_15	0.678
3_28	0.815
3_30	0.759
4_18	0.663
4_20	0.644
4_27	0.738
5_20	0.712
5_22	0.690

To gain insights into how well this binary neural network actually works, a trained regression neural network is used as a binary classifier. This trained regressor is also trained on a non-theoretical data set of size 25000, which means that the features *arrival_rate*, *nr_stations*, *Service_i* (service rate at station i), $Buffer\_i$ (number of buffers at station i) are part of the data set. The regressor estimates the throughputs of all the allocations of the test set instances. The test set instances can then be transformed to a “binary” test set instance by creating all possible pairs of buffer allocations and assigning the target 1 if the *estimated* throughput of the first allocation is lower than the *estimated* throughput, and else 0. These target predictions can then be compared to the true targets, and the balanced accuracy can be computed. This results in Figure 73, which contains the balanced accuracy for the binary neural network and the regression neural network. The regression neural network outperforms the binary neural network in 9/10 cases. The binary neural network only has a higher balanced accuracy in case of instance 4_20.

This experiment was conducted at the beginning of the internship period. As the regression neural network outperformed the binary neural network, it was decided to focus on the regression neural network for the rest of the internship.

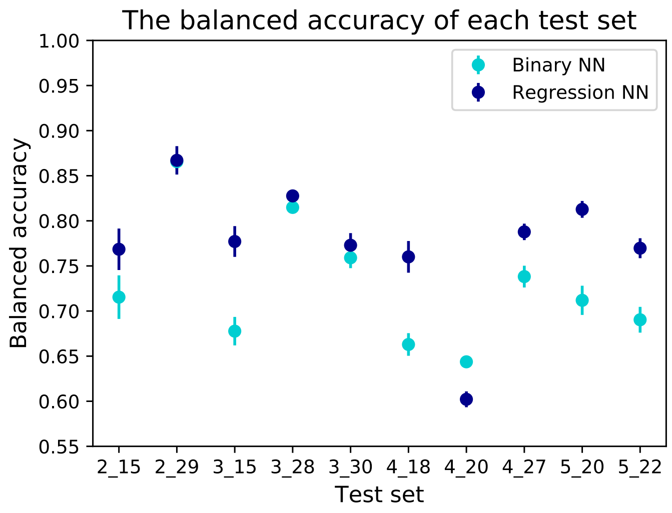


Figure 73: The balanced accuracy on the test set instances of the binary and regression neural networks.

11 Conclusion and Discussion

Three research questions were addressed throughout this thesis. These questions are:

- Main question: Is it possible to use neural networks to estimate the throughput of a variety of tandem lines?
- Sub-question 1: Can the training of the neural network be improved by using theoretical knowledge on queuing systems?
- Sub-question 2: Does the solution of a neural network combined with an evolutionary algorithm outperform the solution of a simulation-based evolutionary algorithm?

The results of this thesis can be used to answer these research questions. By many examples and tests, it has been demonstrated that it is possible to use neural networks to estimate the throughput of a variety of tandem lines. In case of SSBAP and MSBAP, the mean *RAE* of the final metamodellers are equal to 0.6% and 0.7% on validation sets containing total enumerations of multiple tandem lines, respectively. This was achieved by creating training data sets with 100000 rows containing data on different tandem lines, adding theoretical features based on queuing theory and using fourteen independently trained neural networks.

It should be noted that the training data set contains information on many different tandem lines, whereas the test set contains data of total enumerations of a few tandem lines. There is thus a difference between the train and test set, which is not conform "classical" machine learning. However, the mean *RAE* values on a classical train set that also contains a few total enumerations was around 10-20%, which is thus much worse than what is achieved by the developed train set. This is an interesting result, and this type of training data could perhaps also be used to solve other kind of machine learning problems.

The first sub-question can also be answered convincingly. Adding theoretical data to the train and test sets reduces the error tremendously. In case of SSBAP, the mean *RAE* reduces from 1.92% to 1.49% and in case of MSBAP, the error reduces from 2.00% to 0.94%. The regression benchmark models also have a better performance when theoretical features are added to the data.

The second sub-question can be answered by considering the results of the simulation-based EA and the metamodeller-based EA. The simulation-based EA is much slower than the EA combined with the metamodeller. For instance, on one test instance the metamodeller-based EA takes 2 minutes to run (including one and a half minutes of training), whereas the simulation-based EA takes 13 minutes to run. Moreover, the metamodeller-based EA often finds a better allocation than the simulation-based EA. This makes the metamodeller-based EA an attractive solution method when a good solution is desired in a short amount of time.

This research also contributes to the current state-of-the art solution methods that can be found in the literature. First of all, no solution methods based on machine learning techniques have been developed that can estimate the throughput of a variety of tandem lines. Secondly, the solution method works for both SSBAP and MSBAP, and usually only SSBAP is considered in the literature. Thirdly, a detailed study was conducted to determine what kind of training data is most suitable to train a metamodeller. Fourthly, it was demonstrated that adding theoretical features to the data set helped training the neural network. Lastly, an experiment with a binary neural network was performed, which has not been done before.

The results of this thesis can be used to design and improve production lines, as was demonstrated in Section 8, which presents the developed dashboard. Moreover, Avanade can use the results to improve the design of cloud solutions, as was discussed in Section 9, which will reduce the costs of these solutions significantly.

It is not complex to implement the methods developed in this thesis. Data is required on the queuing network that is being studied, which should also contain some sort of throughput measure that depends on the buffer allocation. It is advised to add theoretical features based on queuing theory and to train multiple neural networks (preferably fourteen). This metamodeller can then be used in combination with a generative model (such as an evolutionary algorithm) to perform the optimisation.

This research does have several drawbacks. First of all, no real data was available, so the theories have been developed on data generated by simulations. It is expected that the conclusions will be the same when real data is used, though.

Secondly, in some cases the difference between the throughputs of different buffer allocations estimated by simulations were not substantial. It might be better to estimate the throughputs by considering more and longer simulation runs. This could not be done due to time constraints, but it might nevertheless be worthwhile to do so. However, it is expected that the conclusions of this thesis will remain the same.

Thirdly, the evolutionary algorithm almost always found the maximum as was estimated by the neural network. This, however, was usually not the true maximum, as the maximum corresponded to another buffer allocation according to the neural network estimates. It would therefore be interesting to be able to combine the neural network estimates by for example a simulation (also see Section 12 Future Research). The neural network can be used to quickly move to an area with high throughputs, after which the simulation is used to find the true maximum. This does increase the running time of the solution method, though.

Fourthly, a binary neural network was also developed as an experiment. The target (1/0) was determined by considering the throughputs of two different buffer allocations and deciding which of the throughputs was higher. It might be the case that the difference between the throughputs is very small, but by adopting a 1/0 target this information is lost. This may make it more difficult for the binary neural network to learn when to output which target value.

12 Future Research

This thesis provides multiple opportunities for further research. First of all, it may be interesting to develop neural networks that estimate the throughput of more complicated queuing systems, for example queuing systems that also involve branching.

Secondly, it may also be interesting to allow general service distributions instead of only exponential service distributions. New theoretical features may be added to the data sets to incorporate these general service distributions, although the current theoretical features may be used as approximate features. The non-theoretical features only require little adaptation if general service distributions are allowed.

Thirdly, machine failures often arise in production lines, and it may therefore be worthwhile to develop a neural network that can also cope with machine failures. The used methodology can remain the same, although some features may have to be changed or added that incorporate machine failures.

Fourthly, a hybrid approach can be created that combines the final metamodeller, a simulation and the evolutionary algorithm. During the first part of the optimisation, the final metamodeller may be used to quickly find good solutions. Once X good solutions have been found, the simulation can be used to assess which of the X solutions is the best. Although the results of the final metamodeller combined with the EA are already good, additional accuracy could be achieved by using this hybrid approach. This will increase the running time, though.

Fifthly, as was already discussed in Section 9, the solution method can also be applied to optimise computer network systems. No data was available to directly test the solution method, but in the future it would be worthwhile to apply the method on computer networks as well.

Sixthly, a method was proposed to create confidence intervals by using the variation in the throughput estimates of multiple neural networks. It is interesting to develop this method further. Moreover, these confidence intervals could also be used within the optimisation process, for example by incorporating them in the fitness of the solutions.

Lastly, the proposed methodology to create a neural network as metamodeller can also be used for other combinatorial optimisation problems. The first problem that comes to mind is the Server Allocation Problem (SAP), which is closely related to BAP. Instead of allocating buffers to buffer slots of stations, servers are allocated to stations with SAP. Moreover, SAP and BAP can also be combined into a joint SAP and BAP problem in which both the servers and the buffers are allocated to the tandem line. The proposed methodology can be applied to these two problems, and it is expected that the results are also valuable for these problems.

A Appendix A: SSBAP Data Set Parameters

A.1 Test 1 Data Sets

A.1.1 Test_1_train_1

The first train set contains 25000 data rows with twelve total enumerations. Table 33 contains the parameters of the individual total enumerations that combined make up the total train set of 25000 data rows.

Table 33: Parameters of the individual total enumerations that together make up *Test_1_train_1*.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_1_Train_1_2_20	2	20	2.145	2.347	2.730	0	0	0
Test_1_Train_1_2_24	2	24	1.893	2.759	2.024	0	0	0
Test_1_Train_1_2_27	2	27	1.602	2.645	2.036	0	0	0
Test_1_Train_1_3_24	3	24	2.890	2.335	2.594	2.562	0	0
Test_1_Train_1_3_25	3	25	1.165	1.931	2.211	2.401	0	0
Test_1_Train_1_3_29	3	29	1.174	2.652	2.880	2.856	0	0
Test_1_Train_1_4_25	4	25	2.389	2.747	2.704	2.722	2.505	0
Test_1_Train_1_4_26	4	26	0.513	0.980	0.544	2.672	0.910	0
Test_1_Train_1_4_29	4	29	1.402	1.600	2.647	2.400	1.778	0
Test_1_Train_1_5_14	5	14	1.500	2.579	1.685	2.947	1.605	2.546
Test_1_Train_1_5_19	5	19	1.490	2.130	2.862	1.895	2.834	2.570
Test_1_Train_1_5_26	5	26	1.272	2.684	1.809	1.291	1.979	2.362

A.1.2 Test_1_train_2

The second train set contains 25000 data rows with eighteen total enumerations. Table 34 contains the parameters of the individual total enumerations that combined make up the total train set of 25000 data rows.

Table 34: Parameters of the individual total enumerations that together make up *Test_1_train_2*.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_1_Train_2_2_12	2	12	1.322	2.859	1.821	0	0	0
Test_1_Train_2_2_17	2	17	1.950	2.941	2.841	0	0	0
Test_1_Train_2_2_21	2	21	1.403	1.514	2.629	0	0	0
Test_1_Train_2_2_24	2	24	2.465	2.803	2.560	0	0	0
Test_1_Train_2_2_27	2	27	2.398	2.590	2.934	0	0	0
Test_1_Train_2_3_17	3	17	1.940	2.102	2.700	2.610	0	0
Test_1_Train_2_3_18	3	18	0.865	2.728	2.779	2.628	0	0
Test_1_Train_2_3_24	3	24	1.908	2.573	1.995	2.059	0	0
Test_1_Train_2_3_27	3	27	1.171	2.440	1.214	2.585	0	0
Test_1_Train_2_4_10	4	10	1.729	2.058	2.689	2.744	1.853	0
Test_1_Train_2_4_16	4	16	0.863	2.030	2.848	2.535	2.899	0
Test_1_Train_2_4_21	4	21	2.074	2.637	2.709	2.204	2.614	0
Test_1_Train_2_4_26	4	26	1.829	1.856	2.527	2.817	2.245	0
Test_1_Train_2_4_29	4	29	0.621	2.202	1.591	0.909	1.642	0
Test_1_Train_2_5_14	5	14	1.808	2.305	1.876	2.405	2.977	2.890
Test_1_Train_2_5_18	5	18	0.813	1.256	2.288	2.592	0.947	1.305
Test_1_Train_2_5_21	5	21	1.946	2.147	2.949	2.868	2.035	2.204
Test_1_Train_2_5_24	5	24	1.917	2.448	2.364	2.718	2.092	2.613

A.1.3 Test_1_test

The test contains 15000 data rows with ten total enumerations. Table 35 contains the parameters of the individual total enumerations that combined make up the total test set of 15000 data rows.

Table 35: Parameters of the individual total enumerations that together make up *Test_1_test*.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_1_Test_2_15	2	15	0.884	1.248	0.999	0	0	0
Test_1_Test_2_29	2	29	1.734	1.906	1.816	0	0	0
Test_1_Test_3_15	3	15	0.946	2.463	2.198	2.437	0	0
Test_1_Test_3_28	3	28	0.761	2.842	1.738	2.271	0	0
Test_1_Test_3_30	3	30	2.055	2.688	2.390	2.748	0	0
Test_1_Test_4_18	4	18	1.618	1.658	1.911	1.804	2.402	0
Test_1_Test_4_20	4	20	2.072	2.442	2.946	2.699	2.633	0
Test_1_Test_4_27	4	27	0.980	1.727	1.886	1.358	1.372	0
Test_1_Test_5_20	5	20	1.416	1.740	2.615	1.695	1.545	2.456
Test_1_Test_5_22	5	22	0.944	1.989	2.690	0.987	1.022	2.191

A.2 Test 2 Data Sets

The train sets contain 25000 data rows. The following tables contain the parameters of the individual total enumerations that combined make up the total train sets of 25000 data rows for different values of Δ.

A.2.1 Test_2_train_1_0.025

Table 36: Parameters of the individual total enumerations that together make up $Test\_2\_train\_1\_0.025$.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_2_Train_1_2_20_0.025	2	20	1.975	2.850	2.800	0	0	0
Test_2_Train_1_2_24_0.025	2	24	1.200	1.625	2.675	0	0	0
Test_2_Train_1_2_27_0.025	2	27	2.150	2.700	2.975	0	0	0
Test_2_Train_1_3_24_0.025	3	24	2.000	2.675	2.625	2.600	0	0
Test_2_Train_1_3_25_0.025	3	25	1.175	2.550	1.850	2.700	0	0
Test_2_Train_1_3_29_0.025	3	29	1.475	2.050	2.050	2.650	0	0
Test_2_Train_1_4_25_0.025	4	25	1.425	1.850	2.800	2.075	2.200	0
Test_2_Train_1_4_26_0.025	4	26	2.00	2.775	2.725	2.750	2.125	0
Test_2_Train_1_4_29_0.025	4	29	1.525	2.375	2.350	2.125	2.975	0
Test_2_Train_1_5_14_0.025	5	14	1.700	2.250	1.875	1.975	2.075	2.575
Test_2_Train_1_5_19_0.025	5	19	1.00	2.850	1.425	1.050	1.750	1.550
Test_2_Train_1_5_26_0.025	5	26	1.00	2.850	1.425	1.050	1.750	1.550

A.2.2 Test_2_train_1_0.05

Table 37: Parameters of the individual total enumerations that together make up $Test\_2\_train\_1\_0.05$.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_2_Train_1_2_20_0.05	2	20	2.250	2.550	2.450	0	0	0
Test_2_Train_1_2_24_0.05	2	24	2.050	2.150	2.500	0	0	0
Test_2_Train_1_2_27_0.05	2	27	1.250	2.500	1.500	0	0	0
Test_2_Train_1_3_24_0.05	3	24	0.500	0.750	0.800	0.550	0	0
Test_2_Train_1_3_25_0.05	3	25	1.050	1.400	1.100	2.050	0	0
Test_2_Train_1_3_29_0.05	3	29	1.450	2.950	2.800	2.700	0	0
Test_2_Train_1_4_25_0.05	4	25	2.250	2.550	2.550	2.350	2.900	0
Test_2_Train_1_4_26_0.05	4	26	0.550	0.700	1.700	2.650	1.900	0
Test_2_Train_1_4_29_0.05	4	29	1.050	3.000	2.050	1.250	2.500	0
Test_2_Train_1_5_14_0.05	5	14	0.800	1.100	2.950	2.350	1.350	1.200
Test_2_Train_1_5_19_0.05	5	19	0.750	0.900	1.000	1.450	1.600	1.800
Test_2_Train_1_5_26_0.05	5	26	0.750	0.900	1.00	1.450	1.600	1.800

A.2.3 Test_2_train_1_0.1

Table 38: Parameters of the individual total enumerations that together make up $Test\_2\_train\_1\_0.1$.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_2_Train_1_2_20_0.1	2	20	1.100	1.600	1.900	0	0	0
Test_2_Train_1_2_24_0.1	2	24	1.000	1.800	1.100	0	0	0
Test_2_Train_1_2_27_0.1	2	27	2.000	2.300	2.200	0	0	0
Test_2_Train_1_3_24_0.1	3	24	0.500	1.800	0.600	0.600	0	0
Test_2_Train_1_3_25_0.1	3	25	2.500	3.00	2.600	2.600	0	0
Test_2_Train_1_3_29_0.1	3	29	1.300	1.600	2.300	3.000	0	0
Test_2_Train_1_4_25_0.1	4	25	0.500	2.300	0.800	2.700	1.200	0
Test_2_Train_1_4_26_0.1	4	26	0.500	2.600	0.700	0.900	1.200	0
Test_2_Train_1_4_29_0.1	4	29	2.200	2.900	2.600	2.600	2.700	0
Test_2_Train_1_5_14_0.1	5	14	1.200	3.000	2.300	1.300	2.500	2.700
Test_2_Train_1_5_19_0.1	5	19	0.800	2.400	1.900	1.400	1.700	1.100
Test_2_Train_1_5_26_0.1	5	26	0.800	2.400	1.900	1.400	1.700	1.100

A.2.4 Test_2_train_1_0.25

Table 39: Parameters of the individual total enumerations that together make up $Test\_2\_train\_1\_0.25$.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_2_Train_1_2_20_0.25	2	20	1.500	2.250	3.000	0	0	0
Test_2_Train_1_2_24_0.25	2	24	1.000	1.500	3.00	0	0	0
Test_2_Train_1_2_27_0.25	2	27	1.000	2.250	2.250	0	0	0
Test_2_Train_1_3_24_0.25	3	24	1.750	2.500	3.00	2.00	0	0
Test_2_Train_1_3_25_0.25	3	25	1.750	2.500	3.000	2.000	0	0
Test_2_Train_1_3_29_0.25	3	29	1.500	2.250	2.250	2.500	0	0
Test_2_Train_1_4_25_0.25	4	25	2.000	3.000	2.250	2.250	2.750	0
Test_2_Train_1_4_26_0.25	4	26	1.500	2.750	2.500	1.750	1.750	0
Test_2_Train_1_4_29_0.25	4	29	1.500	2.750	2.250	2.750	2.750	0
Test_2_Train_1_5_14_0.25	5	14	2.500	2.750	3.00	2.750	2.750	3.000
Test_2_Train_1_5_19_0.25	5	19	1.500	2.750	2.500	2.500	2.000	3.000
Test_2_Train_1_5_26_0.25	5	26	1.500	2.750	2.500	2.500	2.000	3.000

A.2.5 Test_2_train_2_0.025

Table 40: Parameters of the individual total enumerations that together make up $Test\_2\_train\_2\_0.025$.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_2_Train_2_2_12_0.025	2	12	1.825	3.00	2.725	0	0	0
Test_2_Train_2_2_17_0.025	2	17	0.500	2.025	0.575	0	0	0
Test_2_Train_2_2_21_0.025	2	21	2.200	2.775	2.225	0	0	0
Test_2_Train_2_2_24_0.025	2	24	2.400	2.825	2.650	0	0	0
Test_2_Train_2_2_27_0.025	2	27	2.225	2.600	2.825	0	0	0
Test_2_Train_2_3_17_0.025	3	17	1.425	1.650	2.300	2.175	0	0
Test_2_Train_2_3_18_0.025	3	18	1.200	2.750	1.375	2.450	0	0
Test_2_Train_2_3_24_0.025	3	24	2.350	2.950	2.950	2.750	0	0
Test_2_Train_2_3_27_0.025	3	27	1.650	1.825	2.900	2.800	0	0
Test_2_Train_2_4_10_0.025	4	10	1.600	2.975	2.800	2.125	2.625	0
Test_2_Train_2_4_16_0.025	4	16	2.275	2.875	2.675	2.375	2.400	0
Test_2_Train_2_4_21_0.025	4	21	2.225	2.550	2.700	2.550	2.625	0
Test_2_Train_2_4_26_0.025	4	26	1.250	1.550	1.550	1.925	2.575	0
Test_2_Train_2_4_29_0.025	4	29	2.200	2.275	2.725	2.775	2.375	0
Test_2_Train_2_5_14_0.025	5	14	1.650	2.575	2.025	2.625	2.025	2.225
Test_2_Train_2_5_18_0.025	5	18	1.525	3.000	1.775	2.900	2.800	1.700
Test_2_Train_2_5_21_0.025	5	21	1.325	1.475	2.400	2.375	2.175	2.150
Test_2_Train_2_5_24_0.025	5	24	2.025	2.525	2.275	2.225	2.725	2.100

A.2.6 Test_2_train_2_0.05

Table 41: Parameters of the individual total enumerations that together make up $Test\_2\_train\_2\_0.05$.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_2_Train_2_2_12_0.05	2	12	1.750	1.900	2.850	0	0	0
Test_2_Train_2_2_17_0.05	2	17	1.00	1.150	2.750	0	0	0
Test_2_Train_2_2_21_0.05	2	21	2.050	2.750	2.200	0	0	0
Test_2_Train_2_2_24_0.05	2	24	1.700	1.7500	2.800	0	0	0
Test_2_Train_2_2_27_0.05	2	27	1.300	1.800	1.900	0	0	0
Test_2_Train_2_3_17_0.05	3	17	1.300	1.400	2.550	2.100	0	0
Test_2_Train_2_3_18_0.05	3	18	1.150	2.200	2.450	2.600	0	0
Test_2_Train_2_3_24_0.05	3	24	2.350	3.000	2.600	2.900	0	0
Test_2_Train_2_3_27_0.05	3	27	2.200	2.500	2.650	2.500	0	0
Test_2_Train_2_4_10_0.05	4	10	0.900	2.300	1.300	1.800	1.350	0
Test_2_Train_2_4_16_0.05	4	16	1.900	2.950	3.000	2.300	2.350	0
Test_2_Train_2_4_21_0.05	4	21	0.550	1.500	1.100	0.800	1.500	0
Test_2_Train_2_4_26_0.05	4	26	1.800	2.050	2.700	2.950	2.550	0
Test_2_Train_2_4_29_0.05	4	29	1.600	2.600	2.250	2.250	2.100	0
Test_2_Train_2_5_14_0.05	5	14	1.900	2.500	2.050	2.550	2.550	2.900
Test_2_Train_2_5_18_0.05	5	18	0.850	1.05	1.300	1.150	1.750	2.250
Test_2_Train_2_5_21_0.05	5	21	2.250	2.650	2.700	2.500	3.000	2.800
Test_2_Train_2_5_24_0.05	5	24	1.800	1.900	2.300	2.550	2.850	2.550

A.2.7 Test_2_train_2_0.1

Table 42: Parameters of the individual total enumerations that together make up *Test_2_train_2_0.1*.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_2_Train_2_2_12_0.1	2	12	2.300	2.400	2.600	0	0	0
Test_2_Train_2_2_17_0.1	2	17	1.800	2.300	2.200	0	0	0
Test_2_Train_2_2_21_0.1	2	21	1.100	1.900	2.100	0	0	0
Test_2_Train_2_2_24_0.1	2	24	1.400	1.800	2.900	0	0	0
Test_2_Train_2_2_27_0.1	2	27	1.600	2.800	2.700	0	0	0
Test_2_Train_2_3_17_0.1	3	17	0.500	0.800	2.000	1.800	0	0
Test_2_Train_2_3_18_0.1	3	18	2.000	2.700	2.900	2.900	0	0
Test_2_Train_2_3_24_0.1	3	24	1.000	2.100	1.100	2.300	0	0
Test_2_Train_2_3_27_0.1	3	27	2.300	2.900	2.600	2.800	0	0
Test_2_Train_2_4_10_0.1	4	10	0.500	1.000	2.000	1.900	1.200	0
Test_2_Train_2_4_16_0.1	4	16	0.700	1.500	2.700	2.500	1.600	0
Test_2_Train_2_4_21_0.1	4	21	2.000	2.900	2.900	2.100	2.200	0
Test_2_Train_2_4_26_0.1	4	26	0.700	1.900	2.200	2.200	2.600	0
Test_2_Train_2_4_29_0.1	4	29	1.300	2.500	2.900	2.600	1.600	0
Test_2_Train_2_5_14_0.1	5	14	2.200	2.800	2.700	2.400	2.600	2.300
Test_2_Train_2_5_18_0.1	5	18	0.800	2.700	1.400	1.100	1.500	2.600
Test_2_Train_2_5_21_0.1	5	21	1.900	2.900	3.000	2.400	2.200	2.500
Test_2_Train_2_5_24_0.1	5	24	0.600	2.600	1.700	1.200	0.800	2.000

A.2.8 Test_2_train_2_0.25

Table 43: Parameters of the individual total enumerations that together make up *Test_2_train_2_0.25*.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_2_Train_2_2_12_0.25	2	12	2.000	2.750	3.000	0	0	0
Test_2_Train_2_2_17_0.25	2	17	1.500	2.000	2.250	0	0	0
Test_2_Train_2_2_21_0.25	2	21	2.250	2.750	2.500	0	0	0
Test_2_Train_2_2_24_0.25	2	24	1.750	3.000	2.250	0	0	0
Test_2_Train_2_2_27_0.25	2	27	2.250	2.500	3.000	0	0	0
Test_2_Train_2_3_17_0.25	3	17	2.500	3.000	2.750	2.750	0	0
Test_2_Train_2_3_18_0.25	3	18	0.750	2.500	2.000	1.250	0	0
Test_2_Train_2_3_24_0.25	3	24	2.000	3.000	2.250	2.250	0	0
Test_2_Train_2_3_27_0.25	3	27	1.000	1.250	2.500	1.500	0	0
Test_2_Train_2_4_10_0.25	4	10	1.000	2.250	2.750	2.750	2.750	0
Test_2_Train_2_4_16_0.25	4	16	2.250	3.000	2.750	2.750	2.750	0
Test_2_Train_2_4_21_0.25	4	21	0.500	3.000	2.750	1.750	1.000	0
Test_2_Train_2_4_26_0.25	4	26	1.750	2.750	2.750	2.000	2.250	0
Test_2_Train_2_4_29_0.25	4	29	1.250	1.750	2.000	2.500	3.000	0
Test_2_Train_2_5_14_0.25	5	14	1.500	2.250	2.250	3.000	3.000	3.000
Test_2_Train_2_5_18_0.25	5	18	2.500	2.750	2.750	2.750	3.000	2.750
Test_2_Train_2_5_21_0.25	5	21	1.000	2.750	2.500	1.500	2.000	1.750
Test_2_Train_2_5_24_0.25	5	24	2.000	2.500	2.500	2.500	2.250	2.250

A.3 Test 3 Data Sets

A.3.1 Test_3_test

The test set contains 15000 data rows with ten total enumerations. Table 44 contains the parameters of the individual total enumerations that combined make up the total test set of 15000 data rows.

Table 44: Parameters of the individual total enumerations that together make up *Test_3_test*.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Test_1_Test_2_15	2	15	1.250	1.350	1.300	0	0	0
Test_1_Test_2_29	2	29	2.400	2.450	2.500	0	0	0
Test_1_Test_3_15	3	15	1.500	2.000	1.600	2.000	0	0
Test_1_Test_3_28	3	28	1.750	2.750	2.000	2.500	0	0
Test_1_Test_3_30	3	30	2.500	2.800	2.700	2.800	0	0
Test_1_Test_4_18	4	18	1.300	1.500	1.600	1.500	1.600	0
Test_1_Test_4_20	4	20	0.650	2.000	1.800	2.500	1.200	0
Test_1_Test_4_27	4	27	2.300	2.500	2.800	2.800	2.500	0
Test_1_Test_5_20	5	20	1.850	2.100	2.400	2.100	2.300	2.450
Test_1_Test_5_22	5	22	1.250	1.500	1.300	1.300	2.000	1.800

A.4 Validation Set

The validation set contains 15000 data rows with eleven total enumerations. Table 45 contains the parameters of the individual total enumerations that combined make up the total validation set of 15000 data rows.

Table 45: Parameters of the individual total enumerations that together make up the validation set.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
Validation_2_19	2	19	2.100	2.250	2.700	0	0	0
Validation_2_27	2	27	0.975	2.100	2.825	0	0	0
Validation_2_29	2	29	1.150	1.200	2.500	0	0	0
Validation_3_15	3	15	2.250	2.775	2.800	3.000	0	0
Validation_3_22	3	22	2.450	2.700	2.500	2.875	0	0
Validation_3_26	3	26	1.625	2.875	2.400	2.300	0	0
Validation_3_30	3	30	0.625	1.000	2.875	0.975	0	0
Validation_4_21	4	21	2.300	2.350	2.550	2.575	2.800	0
Validation_4_28	4	28	1.025	2.525	1.975	2.575	1.200	0
Validation_5_15	5	15	2.500	2.575	2.975	2.825	2.800	3.000
Validation_5_24	5	24	1.850	1.875	2.750	2.500	2.600	1.950

A.5 EA Test Sets

Two total enumerations are used in the grid search of the EA and five total enumerations are used to test the EA combined with the metamodellers and the simulation. Table 46 contains the parameters of these total enumerations.

Table 46: Parameters of the EA test sets.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
EA_4_22	4	22	1.250	1.500	1.750	1.500	2.000	0
EA_5_24	5	24	1.500	2.000	2.250	1.750	2.500	1.750
EA_3_30	3	30	2.425	2.625	2.550	2.925	0	0
EA_4_25	4	25	0.950	1.525	1.475	1.325	2.975	0
EA_4_29	4	29	1.875	2.125	1.925	2.500	2.625	0
EA_5_25	5	25	1.650	2.375	2.775	1.750	2.775	3.000
EA_5_27	5	27	1.550	2.425	1.725	2.500	2.175	2.600

A.6 Large EA Test Set

Table 47 contains the parameters of the large EA test instance.

Table 47: Parameters of the large EA test set.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5
EA_large_5_42	5	42	1.500	1.650	1.700	1.600	1.650	1.700

B Appendix B: SSBAP Results

B.1 Results First Experiment

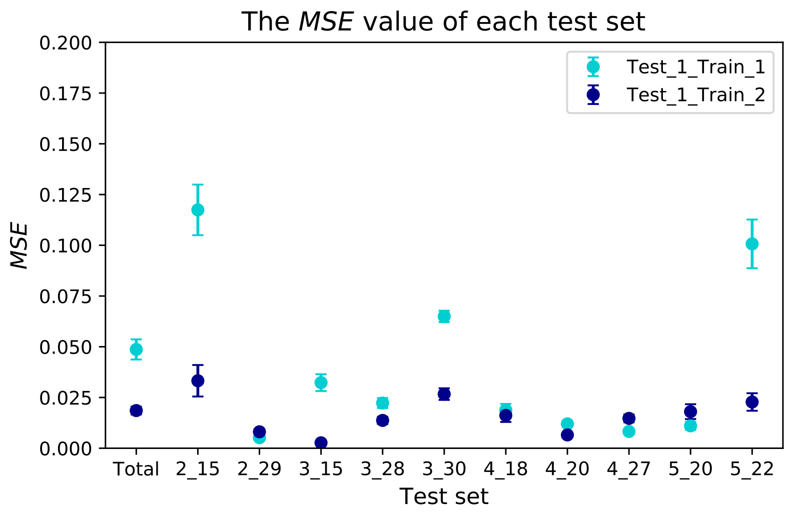


Figure 74: The MSE values together with 95% confidence intervals for each test set.

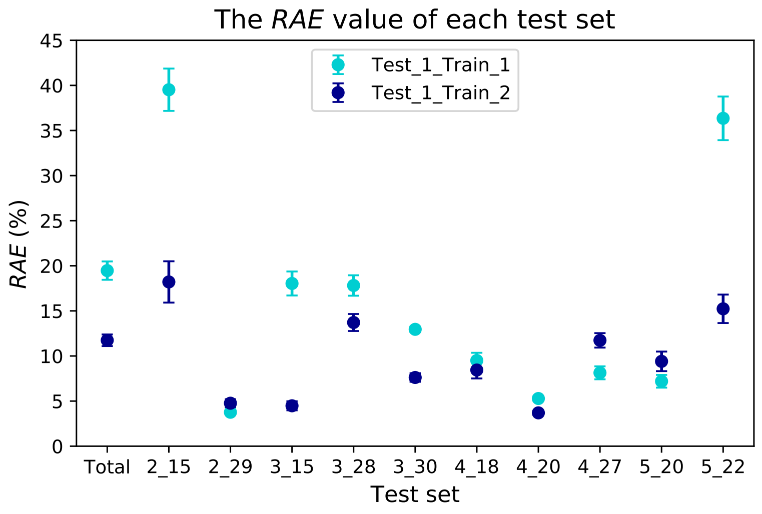


Figure 75: The RAE values together with 95% confidence intervals for each test set.

B.2 Results Second Experiment

Table 48: The selected grid search parameters for each discretisation step Δ.

Parameter	$Test\_2\_train\_1\_0.025$	$Test\_2\_train\_1\_0.05$	$Test\_2\_train\_1\_0.01$	$Test\_2\_train\_1\_0.25$
$max\_iterations$	1000	2000	500	2000
$alpha$	0.01	0.001	0.00001	0.001
$early\_stopping$	True	True	True	True
$hidden\_layer\_sizes$	100	100	100	100

Table 49: The selected grid search parameters for each discretisation step Δ.

Parameter	$Test\_2\_train\_2\_0.025$	$Test\_2\_train\_2\_0.05$	$Test\_2\_train\_2\_0.01$	$Test\_2\_train\_2\_0.25$
$max\_iterations$	500	200	500	2000
$alpha$	0.001	0.001	0.01	0.00001
$early\_stopping$	True	True	True	True
$hidden\_layer\_sizes$	100	100	90	100

B.3 Results Third Experiment

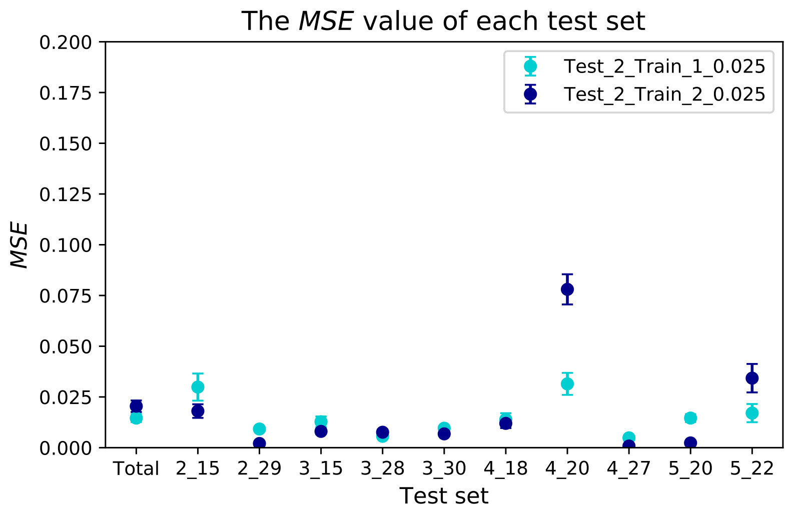


Figure 76: The MSE values together with 95% confidence intervals for each test set.

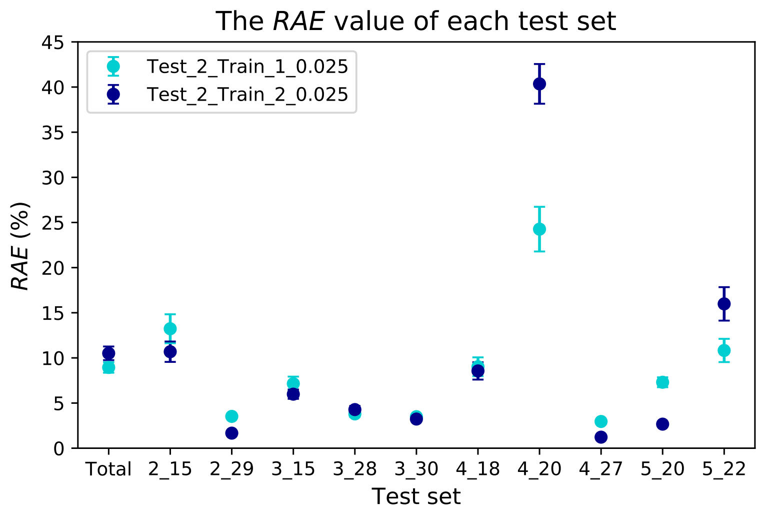


Figure 77: The RAE values together with 95% confidence intervals for each test set.

B.4 Results Fourth Experiment

Table 50: The selected grid search parameters for each training set.

Parameter	$Test\_4\_train\_1$	$Test\_4\_train\_2$
$max\_iterations$	1000	1000
$alpha$	0.01	0.01
$early\_stopping$	True	True
$hidden\_layer\_sizes$	100	90

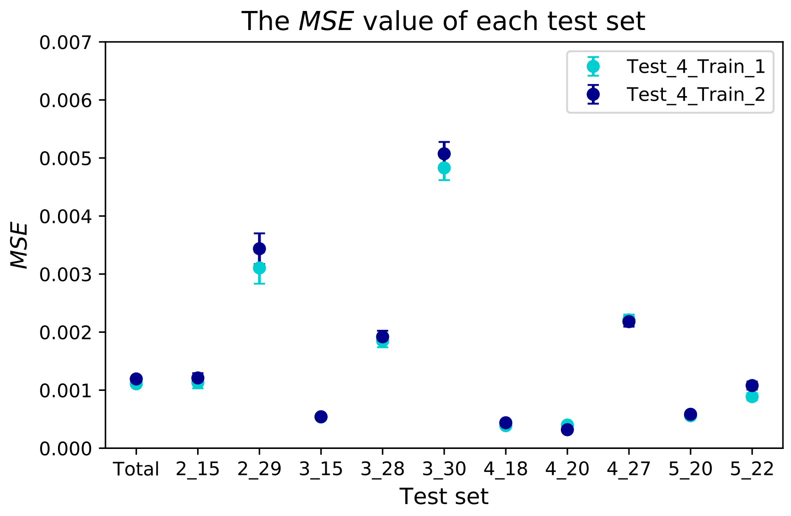


Figure 78: The MSE values together with 95% confidence intervals for each test set.

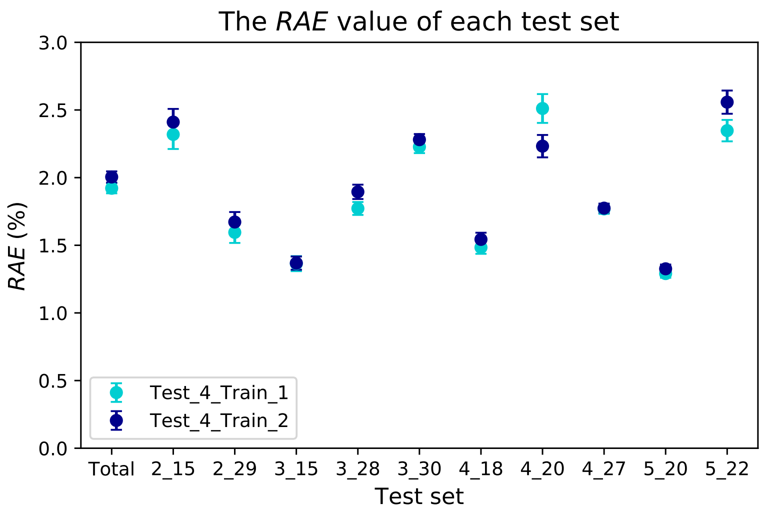


Figure 79: The RAE values together with 95% confidence intervals for each test set.

B.5 Results Fifth Experiment

Table 51: The selected grid search parameters for each training set.

Parameter	$Test\_5\_train\_1$
$max\_iterations$	2000
$alpha$	0.01
$early\_stopping$	True
$hidden\_layer\_sizes$	100

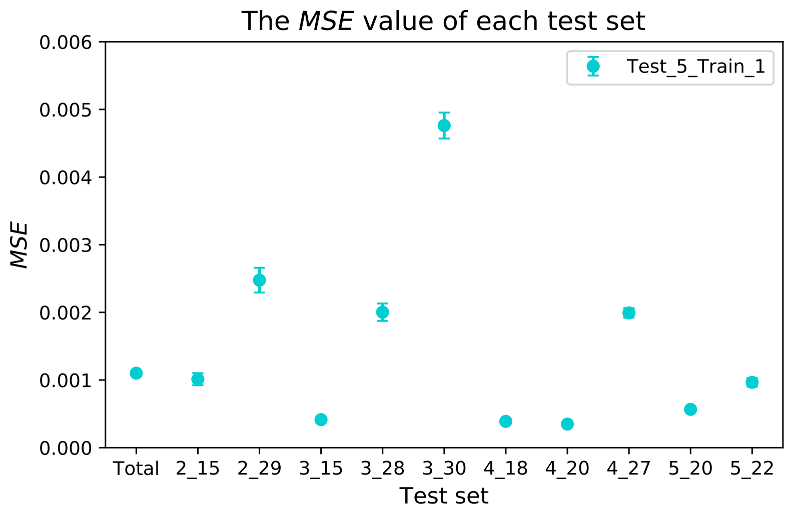


Figure 80: The MSE values together with 95% confidence intervals for each test set.

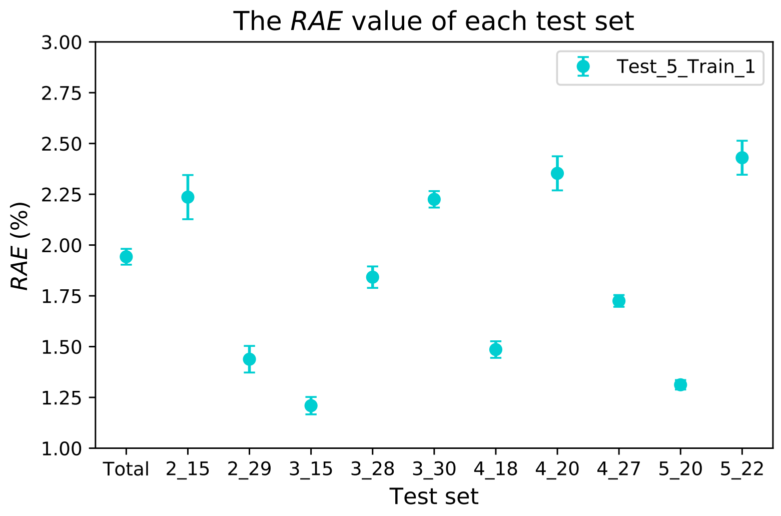


Figure 81: The RAE values together with 95% confidence intervals for each test set.

B.6 Results Sixth Experiment

Table 52: The selected grid search parameters for each training set.

Parameter	$Test\_6\_train\_C1$	$Test\_6\_train\_C2$	$Test\_6\_train\_C3$	$Test\_6\_train\_C4$	$Test\_6\_train\_C5$
$max\_iterations$	1000	1000	1000	200	1000
$alpha$	0.01	0.01	0.01	0.01	0.01
$early\_stopping$	True	True	True	True	True
$hidden\_layer\_sizes$	100	90	100	100	80

B.7 Results Seventh Experiment

Table 53: The selected grid search parameters for each training set.

Parameter	$Test\_7\_no\_theory$	$Test\_7\_theory$
$max\_iterations$	1000	2000
$alpha$	0.01	0.01
$early\_stopping$	True	True
$hidden\_layer\_sizes$	100	70

B.8 Results Ninth Experiment

Table 54: The mean MAE on the individual test set instances.

Instance	2_29	3_28	4_18	4_27	5_22
Mean MAE (25000)	0.0250	0.0144	0.0162	0.0172	0.0200
Mean MAE (100000)	0.0146	0.0112	0.0081	0.0117	0.0110

Table 55: The mean MAD on the individual test set instances.

Instance	2_29	3_28	4_18	4_27	5_22
Mean MAD (25000)	0.958	14.110	25.248	48.737	241.572
Mean MAD (100000)	0.775	11.546	12.740	33.888	133.313

Table 56: The mean MSD on the individual test set instances.

Instance	2_29	3_28	4_18	4_27	5_22
Mean MSD (25000)	3.449	682.916	2268.926	8653.182	210888.266
Mean MSD (100000)	2.485	460.367	580.980	3972.073	63371.527

Table 57: The maximum MD on the individual test set instances.

Instance	2_29	3_28	4_18	4_27	5_22
Maximum MD (25000)	10	162	429	1056	5407
Maximum MD (100000)	7	116	189	527	2930

B.9 The Evolutionary Algorithm

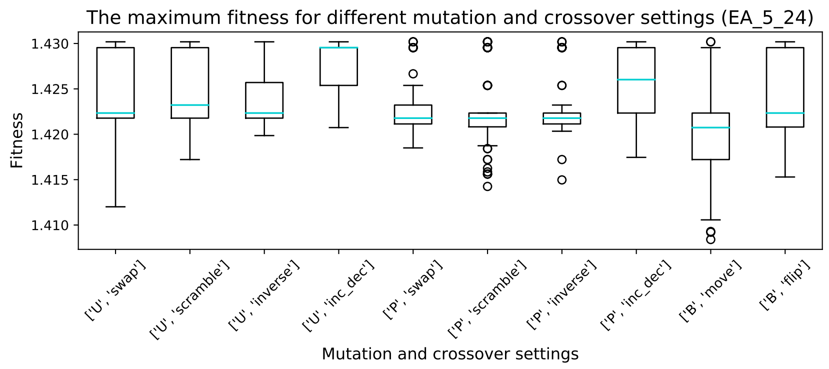


Figure 82: Maximum fitness of all configurations.

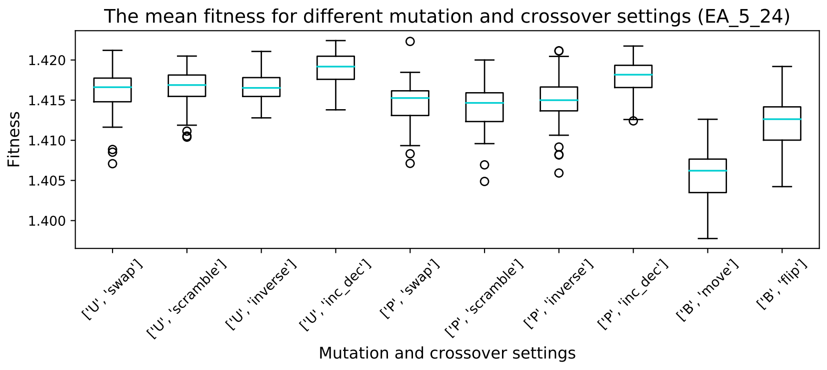


Figure 83: Mean fitness of all configurations.

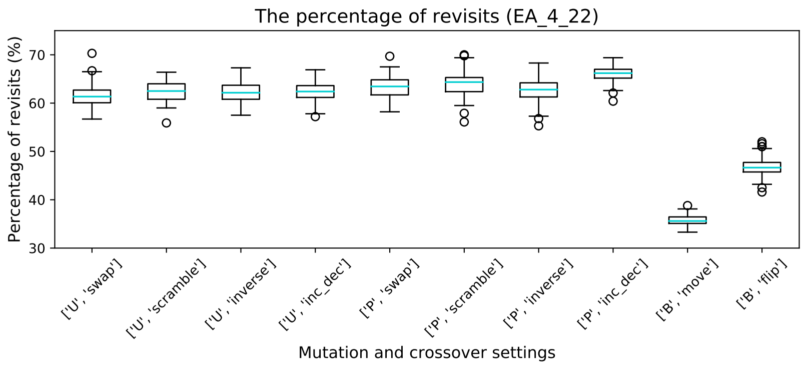


Figure 84: The percentage of revisits for test instance EA_4_22.

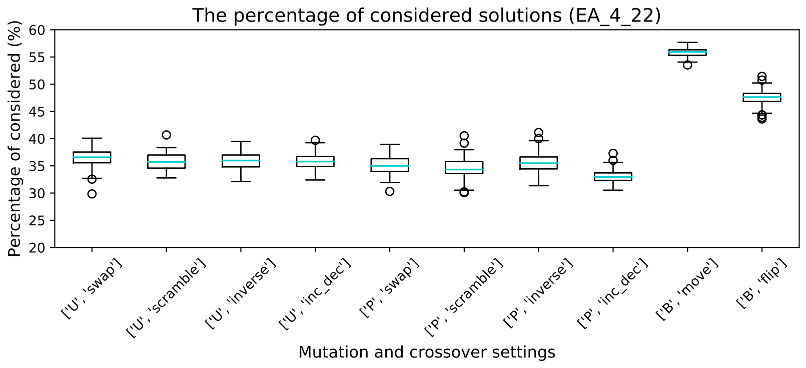


Figure 85: The percentage of considered solutions for test instance EA_4_22.

C Appendix C: MSBAP Data Set Parameters

C.1 MSBAP Test 1 Data Sets

C.1.1 MSBAP_Test_1_train_1

The first train set contains 25000 data rows with twelve total enumerations. Table 58 contains the parameters of the individual total enumerations that combined make up the total train set of 25000 data rows.

Table 58: Parameters of the individual total enumerations that together make up *MSBAP_Test_1_train_1*.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5	S_1	S_2	S_3	S_4	S_5
Te_1_Tr_1_2_20	2	20	1.500	0.550	0.580	0	0	0	5	5	0	0	0
Te_1_Tr_1_2_24	2	24	2.000	2.950	2.650	0	0	0	1	1	0	0	0
Te_1_Tr_1_2_27	2	27	2.200	1.488	0.625	0	0	0	2	4	0	0	0
Te_1_Tr_1_3_24	3	24	1.175	0.625	1.2625	1.975	0	0	2	2	1	0	0
Te_1_Tr_1_3_25	3	25	2.250	2.450	2.275	0.783	0	0	1	1	3	0	0
Te_1_Tr_1_3_29	3	29	1.975	2.900	0.719	2.350	0	0	1	4	1	0	0
Te_1_Tr_1_4_25	4	25	1.200	2.700	0.569	2.800	0.717	0	1	4	1	3	0
Te_1_Tr_1_4_26	4	26	1.925	2.475	2.575	1.088	2.050	0	1	1	2	1	0
Te_1_Tr_1_4_29	4	29	2.050	2.525	1.238	0.570	1.500	0	1	2	5	2	0
Te_1_Tr_1_5_14	5	14	1.175	0.631	2.800	0.888	2.725	0.625	4	1	2	1	2
Te_1_Tr_1_5_19	5	19	0.925	2.175	1.400	1.375	0.650	2.925	1	1	1	2	1
Te_1_Tr_1_5_26	5	26	0.950	0.763	0.875	0.575	1.100	1.463	2	3	2	1	2

C.1.2 MSBAP_Test_1_train_2

The second train set contains 25000 data rows with eighteen total enumerations. Table 59 contains the parameters of the individual total enumerations that combined make up the total train set of 25000 data rows.

Table 59: Parameters of the individual total enumerations that together make up $MSBAP\_Test\_1\_train\_2$.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5	S_1	S_2	S_3	S_4	S_5
Te_1_Tr_2_2_12	2	12	1.550	1.800	2.600	0	0	0	1	1	0	0	0
Te_1_Tr_2_2_17	2	17	2.400	0.992	0.950	0	0	0	3	3	0	0	0
Te_1_Tr_2_2_21	2	21	0.775	1.050	0.375	0	0	0	2	3	0	0	0
Te_1_Tr_2_2_24	2	24	1.300	0.356	0.550	0	0	0	4	4	0	0	0
Te_1_Tr_2_2_27	2	27	0.950	0.208	0.892	0	0	0	6	3	0	0	0
Te_1_Tr_2_3_17	3	17	0.675	1.075	0.267	0.625	0	0	2	3	3	0	0
Te_1_Tr_2_3_18	3	18	1.850	2.200	1.438	2.150	0	0	1	2	1	0	0
Te_1_Tr_2_3_24	3	24	1.650	1.063	2.750	1.325	0	0	2	1	2	0	0
Te_1_Tr_2_3_27	3	27	0.550	2.475	0.700	2.200	0	0	1	1	1	0	0
Te_1_Tr_2_4_10	4	10	1.175	2.850	2.500	1.625	1.775	0	1	1	1	1	0
Te_1_Tr_2_4_16	4	16	1.650	0.608	0.983	2.875	0.767	0	3	3	1	3	0
Te_1_Tr_2_4_21	4	21	1.500	2.825	2.175	1.575	0.550	0	1	1	1	3	0
Te_1_Tr_2_4_26	4	26	2.425	2.725	2.800	2.775	2.525	0	1	1	1	1	0
Te_1_Tr_2_4_29	4	29	1.100	1.600	1.825	1.625	0.913	0	1	1	1	2	0
Te_1_Tr_2_5_14	5	14	2.200	2.700	2.225	2.900	1.188	2.875	1	1	1	2	1
Te_1_Tr_2_5_18	5	18	1.550	2.225	2.175	2.225	2.925	2.750	1	1	1	1	1
Te_1_Tr_2_5_21	5	21	1.675	1.700	1.138	1.300	1.800	2.300	1	2	2	1	1
Te_1_Tr_2_5_24	5	24	1.250	2.250	3.000	2.200	0.767	1.100	1	1	1	3	2

C.1.3 Test_1_test

The test set contains 15000 data rows with ten total enumerations. Table 60 contains the parameters of the individual total enumerations that combined make up the total test set of 15000 data rows.

Table 60: Parameters of the individual total enumerations that together make up $MSBAP\_Test\_1\_test$.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5	S_1	S_2	S_3	S_4	S_5
Te_1_Te_2_15	2	15	1.700	0.404	1.200	0	0	0	6	2	0	0	0
Te_1_Te_2_29	2	29	1.550	0.470	0.992	0	0	0	2	3	0	0	0
Te_1_Te_3_15	3	15	0.925	2.500	1.525	0.475	0	0	1	1	2	0	0
Te_1_Te_3_28	3	28	1.375	2.225	0.350	1.675	0	0	1	4	1	0	0
Te_1_Te_3_30	3	30	2.275	2.750	1.225	0.908	0	0	1	2	3	0	0
Te_1_Te_4_18	4	18	2.125	1.125	2.225	1.375	2.800	0	2	1	2	1	0
Te_1_Te_4_20	4	20	1.850	1.200	2.100	2.650	1.313	0	2	1	1	2	0
Te_1_Te_4_27	4	27	1.125	2.375	2.775	2.500	0.892	0	1	1	1	3	0
Te_1_Te_5_20	5	20	2.000	2.750	2.175	2.650	2.450	2.125	1	1	1	1	1
Te_1_Te_5_22	5	22	1.525	1.000	1.975	2.950	1.038	1.213	2	1	1	2	2

C.2 Validation Set

The validation set contains 15000 data rows with eleven total enumerations. Table 61 contains the parameters of the individual total enumerations that combined make up the total validation set of 15000 data rows.

Table 61: Parameters of the individual total enumerations that together make up *MSBAP_Validation*.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5	S_1	S_2	S_3	S_4	S_5
MSBAP_Validation_2_19	2	19	1.725	0.513	0.430	0	0	0	4	5	0	0	0
MSBAP_Validation_2_27	2	27	1.550	2.100	0.438	0	0	0	1	4	0	0	0
MSBAP_Validation_2_29	2	29	1.925	0.483	0.619	0	0	0	6	4	0	0	0
MSBAP_Validation_3_15	3	15	0.625	2.525	0.900	2.800	0	0	1	2	1	0	0
MSBAP_Validation_3_22	3	22	1.200	1.650	2.300	1.450	0	0	1	1	1	0	0
MSBAP_Validation_3_26	3	26	0.850	0.950	1.650	2.350	0	0	1	1	1	0	0
MSBAP_Validation_3_30	3	30	0.925	0.383	0.867	0.325	0	0	3	3	4	0	0
MSBAP_Validation_4_21	4	21	1.400	1.488	0.713	1.025	1.025	0	2	2	2	2	0
MSBAP_Validation_4_28	4	28	1.175	1.300	0.400	1.213	2.350	0	1	3	2	1	0
MSBAP_Validation_5_15	5	15	0.700	0.392	0.725	1.425	2.450	0.488	3	3	1	1	2
MSBAP_Validation_5_24	5	24	2.325	1.175	2.500	2.575	2.525	2.375	2	1	1	1	1

C.3 EA Test Sets

Five total enumerations are used to test the evolutionary algorithm combined with the metamodellers and the simulation. Table 62 contains the parameters of these total enumerations.

Table 62: Parameters of the MSBAP EA test sets.

Name	Nr. Stations	Nr. Buffers	λ	μ_1	μ_2	μ_3	μ_4	μ_5	S_1	S_2	S_3	S_4	S_5
MSBAP_EA_3_30	3	30	1.875	1.250	2.100	2.425	0	0	2	1	1	0	0
MSBAP_EA_4_25	4	25	1.600	0.405	0.592	2.550	2.600	0	5	3	1	1	0
MSBAP_EA_4_29	4	29	1.100	0.667	0.731	2.075	1.825	0	3	4	1	1	0
MSBAP_EA_5_25	5	25	0.575	1.363	0.313	1.625	0.342	0.900	2	2	1	3	1
MSBAP_EA_5_27	5	27	1.950	2.875	2.575	2.425	2.875	2.650	1	1	1	1	1

D Appendix D: MSBAP Results

D.1 MSBAP Results First Experiment

Table 63: The selected grid search parameters for each training set.

Parameter	*MSBAP_Test_1_train_1*	*MSBAP_Test_1_train_2*
max_iterations	500	2000
alpha	0.001	0.001
early_stopping	True	True
hidden_layer_sizes	(30, 30)	100

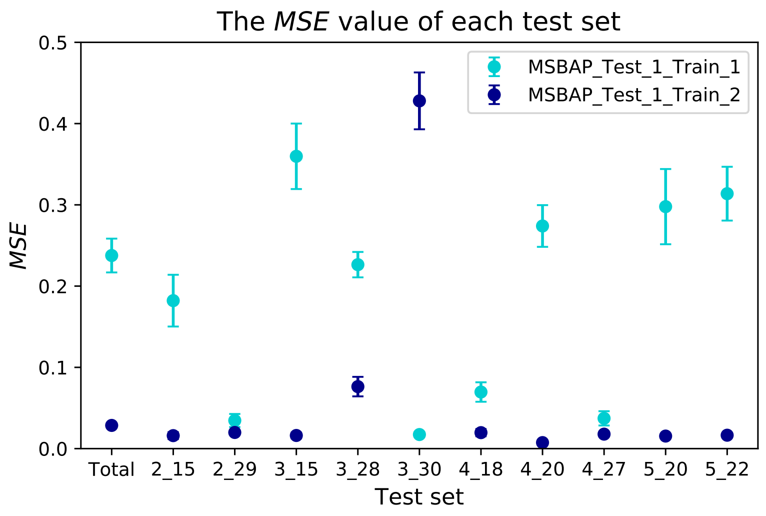


Figure 86: The MSE values together with 95% confidence intervals for each test set.

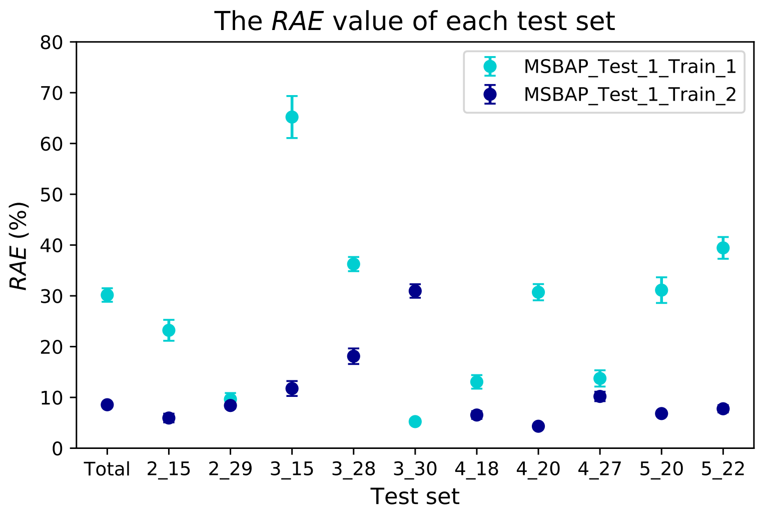


Figure 87: The RAE values together with 95% confidence intervals for each test set.

D.2 MSBAP Results Second Experiment

Table 64: The selected grid search parameters for each training set.

Parameter	$MSBAP\_Test\_2\_train\_1$	$MSBAP\_Test\_2\_train\_2$
$max\_iterations$	1000	200
$alpha$	0.01	0.01
$early\_stopping$	True	True
$hidden\_layer\_sizes$	90	100

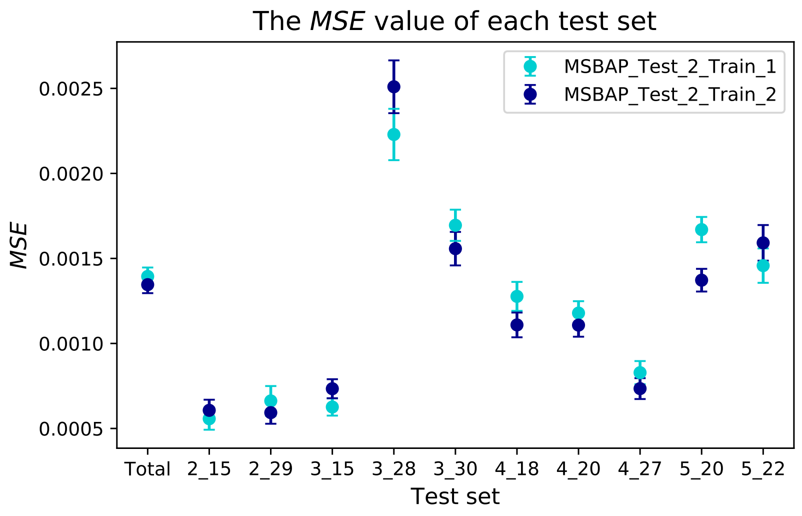


Figure 88: The MSE values together with 95% confidence intervals for each test set.

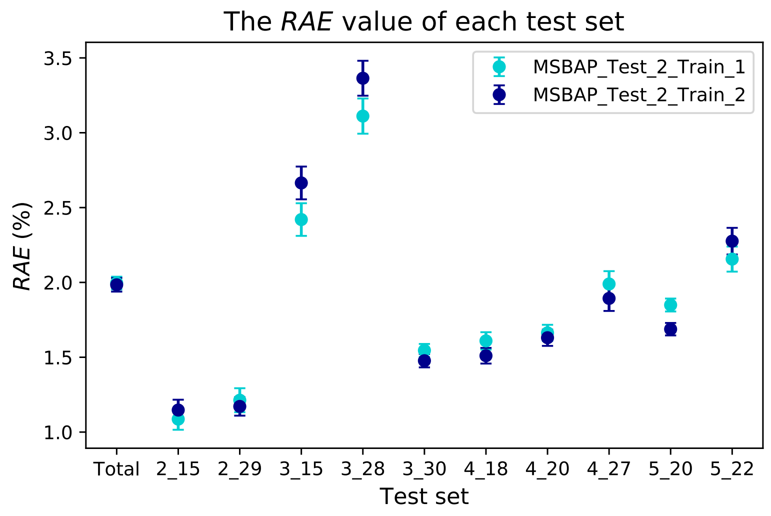


Figure 89: The RAE values together with 95% confidence intervals for each test set.

D.3 MSBAP Results Third Experiment

Table 65: The selected grid search parameters for each training set.

Parameter	$MSBAP\_Test\_3\_no\_theory$	$MSBAP\_Test\_3\_theory$
$max\_iterations$	1000	500
$alpha$	0.01	0.01
$early\_stopping$	True	True
$hidden\_layer\_sizes$	90	100

D.4 MSBAP Results Fifth Experiment

Table 66: The mean MAE on the individual test set instances.

Instance	2_29	3_28	4_18	4_27	5_22
Mean MAE (25000)	0.0091	0.0175	0.0150	0.0117	0.0112
Mean MAE (100000)	0.0075	0.0102	0.0094	0.0085	0.0072

Table 67: The mean MAD on the individual test set instances.

Instance	2_29	3_28	4_18	4_27	5_22
Mean MAD (25000)	2.930	19.248	15.761	249.575	125.105
Mean MAD (100000)	2.706	13.639	9.995	247.117	85.697

Table 68: The mean MSD on the individual test set instances.

Instance	2_29	3_28	4_18	4_27	5_22
Mean MSD (25000)	29.293	1234.734	903.390	203288.746	53995.654
Mean MSD (100000)	25.051	606.593	366.903	199058.497	25536.486

Table 69: The maximum MD on the individual test set instances.

Instance	2_29	3_28	4_18	4_27	5_22
Maximum MD (25000)	22	250	341	1940	2066
Maximum MD (100000)	20	143	167	1817	1510

E Appendix E: Mathematical Derivations

In the following, Little's Law is used to derive certain performance measures. The following holds:

$$E[L] = \lambda E[S], \tag{84}$$

$$E[L^q] = \lambda E[W^q], \tag{85}$$

where $E[L]$ is the expected number of customers in the system, $E[S]$ is the expected system time, $E[L^q]$ is the expected number of customers in the queue and $E[W^q]$ is the expected waiting time.

E.1 $M/M/1$ analysis

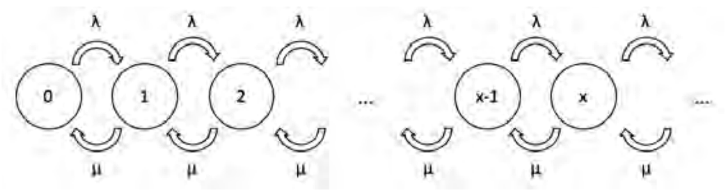


Figure 90: Flow diagram of an $M/M/1$ queue.

Let $\rho = \frac{\lambda}{\mu}$. The balance equations can be formulated by using the flow diagram in Figure 90 and are given by:

$$\pi(x) = \rho^x \pi_0, \tag{86}$$

$$\sum_{x=0}^{\infty} \pi(x) = 1. \tag{87}$$

Substituting (86) into (87) yields:

$$\sum_{x=0}^{\infty} \rho^x \pi_0 = 1. \tag{88}$$

Solving for π_0 gives:

$$\pi_0 = 1 - \rho. \tag{89}$$

The stationary distribution of an $M/M/1$ queue is then:

$$\pi(x) = (1 - \rho)\rho^x, \text{ for } x \geq 0. \tag{90}$$

The expected number of customers in the system can be calculated as follows:

$$\begin{aligned}
E[L] &= \sum_{x=0}^{\infty} x\pi(x) \\
&= \sum_{x=0}^{\infty} x(1-\rho)\rho^x \\
&= (1-\rho)\sum_{x=0}^{\infty} x\rho^x \\
&= (1-\rho)\frac{\rho}{(1-\rho)^2} \\
&= \frac{\rho}{1-\rho}. \qquad (91)
\end{aligned}$$

The expected number of customers in the queue can be calculated by subtracting the mean number of customers in service from $E[L]$:

$$\begin{aligned}
E[L^q] &= E[L] - \rho \\
&= \frac{\rho}{1-\rho} - \rho \\
&= \frac{\rho^2}{1-\rho}. \qquad (92)
\end{aligned}$$

The expected waiting time can be obtained from $E[L^q]$ by using Little's Law:

$$\begin{aligned}
E[W^q] &= \frac{E[L^q]}{\lambda} \\
&= \frac{\frac{\rho^2}{1-\rho}}{\lambda} \\
&= \frac{\rho}{\mu(1-\rho)}. \qquad (93)
\end{aligned}$$

The expected system time can be obtained from $E[W^q]$ by adding the expected service time:

$$\begin{aligned}
E[S] &= \frac{1}{\mu} + E[W^q] \\
&= \frac{1}{\mu} + \frac{\rho}{\mu(1-\rho)}. \qquad (94)
\end{aligned}$$

The probability that more than K customers are present is given by:

$$\begin{aligned}
P(X > K) &= 1 - P(X \le K) \\
&= 1 - \sum_{x=0}^{K}(1-\rho)\rho^x. \qquad (95)
\end{aligned}$$

The probability on an idle station is given by π_0:

$$\pi_0 = 1 - \rho. \tag{96}$$

The probability that there are exactly K customers present is given by $\pi(K)$:

$$\pi(K) = (1 - \rho)\rho^K. \tag{97}$$

E.2 $M/M/1/K$ analysis

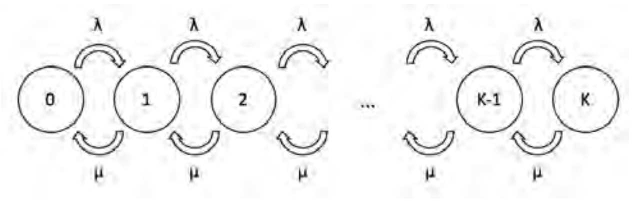


Figure 91: Flow diagram of an $M/M/1/K$ queue.

The balance equations can be formulated by using the flow diagram of Figure 91 and are given by:

$$\pi(x) = \rho^x \pi_0, \text{ for } x = 0, ..., K, \tag{98}$$

$$\sum_{x=0}^{K} \pi(x) = 1. \tag{99}$$

Substituting (98) into (99) yields:

$$\sum_{x=0}^{K} \rho^x \pi_0 = 1. \tag{100}$$

Solving for π_0 gives:

$$\pi_0 = \frac{1 - \rho}{1 - \rho^{K+1}}. \tag{101}$$

Thus, the stationary distribution is given by:

$$\pi(x) = \frac{1 - \rho}{1 - \rho^{K+1}} \rho^x, \text{ for } x = 1, ..., K. \tag{102}$$

The expected number of customers in the system is given by:

$$\begin{aligned}
E[L] &= \sum_{x=0}^{K} x\pi(x) \\
&= \sum_{x=0}^{K} x \frac{1-\rho}{1-\rho^{K+1}} \rho^x \\
&= \frac{1-\rho}{1-\rho^{K+1}} \sum_{x=0}^{K} x\rho^x \\
&= \frac{1-\rho}{1-\rho^{K+1}} \frac{\rho(K\rho^{K+1} - (K+1)\rho K + 1)}{(1-\rho)^2} \\
&= \frac{\rho(K\rho^{K+1} - (K+1)\rho^K + 1)}{(\rho-1)(\rho^{K+1}-1)}. \qquad (103)
\end{aligned}$$

The blocking probability is given by $\pi(K)$:

$$\pi(K) = \frac{1-\rho}{1-\rho^{K+1}} \rho^K. \qquad (104)$$

The probability on an idle server is given by π_0:

$$\pi_0 = \frac{1-\rho}{1-\rho^{K+1}}. \qquad (105)$$

The formula for the effective throughput can be determined by noticing that customers arriving when the queue is in state K are blocked. The following then holds for the throughput:

$$\begin{aligned}
\gamma &= \sum_{x=0}^{K-1} \lambda\pi(x) \\
&= \lambda(1 - \pi(K)) \\
&= \lambda(1 - \frac{1-\rho}{1-\rho^{K+1}} \rho^K). \qquad (106)
\end{aligned}$$

Algorithm 1 then easily follows by recursively applying this formula for the throughput of each station (from left to right).

E.3 $M/M/S$ analysis

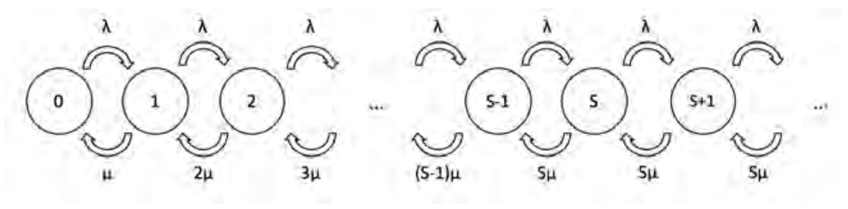


Figure 92: Flow diagram of an $M/M/S$ queue.

Let $\rho = \frac{\lambda}{S\mu}$. The balance equations can be formulated by using the flow diagram in Figure 92 and are given by:

$$\pi(x) = \begin{cases} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!}\pi_0 & \text{for } x = 0,1,\ ...,\ S-1 \\ \left(\frac{\lambda}{\mu}\right)^x \frac{1}{S!}\frac{1}{S^{x-S}}\pi_0 & \text{for } x \geq S, \end{cases} \tag{107}$$

$$\sum_{x=0}^{\infty} \pi(x) = 1. \tag{108}$$

Substituting (107) into (108) gives:

$$\begin{aligned} &\sum_{x=0}^{S-1} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!}\pi_0 + \sum_{x=S}^{\infty} \left(\frac{\lambda}{\mu_1}\right)^x \frac{1}{S!}\frac{1}{S^{x-S}}\pi_0 = 1 \\ &\sum_{x=0}^{S-1} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!}\pi_0 + \frac{1}{S!}\pi_0 \sum_{x=S}^{\infty} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{S^{x-S}} = 1 \\ &\sum_{x=0}^{S-1} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!}\pi_0 + \frac{1}{S!}\pi_0 \frac{S(\frac{\lambda}{\mu})^S}{S - \frac{\lambda}{\mu}} = 1 \\ &\sum_{x=0}^{S-1} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!}\pi_0 + \frac{1}{S!}\pi_0 \frac{(\frac{\lambda}{\mu})^S}{1-\rho} = 1 \\ &\pi_0 = \left(\left(\sum_{x=0}^{S-1} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!} \right) + \left(\frac{\lambda}{\mu}\right)^S \frac{1}{S!}\frac{1}{1-\rho} \right)^{-1}. \end{aligned} \tag{109}$$

To derive the expected number of customers at the station, it is worthwhile to slightly change the notation of the stationary distribution (Adan and Resing, 2002). Define:

$$\pi_x = \frac{\left(\frac{\lambda}{\mu}\right)^x}{x!}\pi_0 \text{ for } x = 0,\ ...,\ S, \tag{110}$$

and

$$\pi_{S+x} = \rho^x \pi_S \text{ for } x = 0,1,... \tag{111}$$

Then define the delay probability Π:

$$\begin{aligned} \Pi &= \pi_S + \pi_{S+1} + ... \\ &= \frac{\pi_S}{1-\rho} \\ &= \left(\frac{\lambda}{\mu_i}\right)^S \frac{1}{S!} \left((1-\rho) \sum_{x=0}^{S-1} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!} + \left(\frac{\lambda}{\mu}\right)^S \frac{1}{S!} \right)^{-1}. \end{aligned} \tag{112}$$

It is now easy to derive the expected number of customers at the queue:

$$
\begin{aligned}
E[L^q] &= \sum_{x=0}^{\infty} x\pi_{S+x} \\
&= \sum_{x=0}^{\infty} x\rho^x \pi_S \\
&= \frac{\pi_S}{1-\rho}\sum_{x=0}^{\infty} x(1-\rho)\rho^x \\
&= \Pi\frac{\rho}{1-\rho}. \qquad (113)
\end{aligned}
$$

The expected number of customers at the station can then be computed by adding the expected number of customers in service to $E[L^q]$. This is given by $\frac{\lambda}{\mu}$, which results in:

$$
E[L] = \frac{\rho}{1-\rho}\Pi + \frac{\lambda}{\mu}. \qquad (114)
$$

The probability that the station is idle is given by π_0.
The probability on $K = S + C$ customers at the station is given by $\pi(K)$:

$$
\pi(K) = \Big(\frac{\lambda}{\mu}\Big)^x \frac{1}{S!}\frac{1}{S^{x-S}}\pi_0. \qquad (115)
$$

The probability of more than $K = S + C$ customers at the station can be calculated as follows:

$$
\begin{aligned}
P(X > K) &= 1 - P(X \leq K) \\
&= 1 - \sum_{x=0}^{S-1}\Big(\frac{\lambda}{\mu}\Big)^x \frac{1}{x!}\pi_0 - \sum_{x=S}^{K}\Big(\frac{\lambda}{\mu_i}\Big)^x \frac{1}{S!}\Big(\frac{1}{S}\Big)^{x-S}\pi_0. \qquad (116)
\end{aligned}
$$

E.4 $M/M/S/K$ analysis

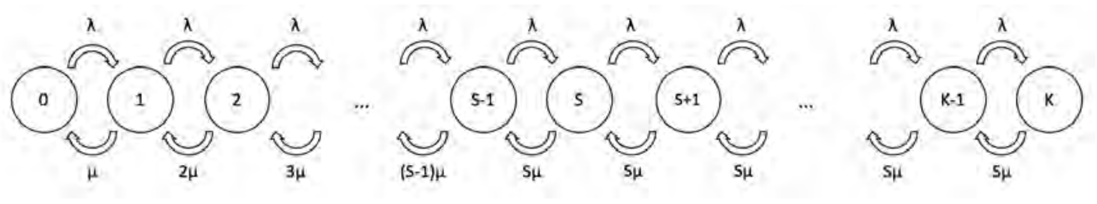


Figure 93: Flow diagram of an $M/M/S/K$ queue.

The balance equations can be formulated by using the flow diagram of Figure 93 and are given by:

$$
\pi(x) = \begin{cases} (\frac{\lambda}{\mu})^x \frac{1}{x!}\pi_0 & \text{for x = 1,2, ..., S-1} \\ (\frac{\lambda}{\mu})^x \frac{1}{S!}\frac{1}{S^{x-S}}\pi_0 & \text{for x = S, ..., K,} \end{cases} \qquad (117)
$$

$$\sum_{x=0}^{K} \pi(x) = 1. \tag{118}$$

Substituting (117) into (118) gives:

$$\sum_{x=0}^{S-1} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!}\pi_0 + \sum_{x=S}^{K} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{S!}\frac{1}{S^{x-S}}\pi_0 = 1. \tag{119}$$

Solving for π_0 gives:

$$\pi_0 = \left(\left(\sum_{x=0}^{S-1} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!}\right) + \frac{1}{S!}\sum_{x=S}^{K} \left(\frac{\lambda}{\mu}\right)^x \frac{1}{S^{x-S}}\right)^{-1}. \tag{120}$$

The probability on an idle station is given by π_0.

The blocking probability is given by $\pi(K)$:

$$\pi(K) = \left(\frac{\lambda}{\mu}\right)^K \frac{1}{S!}\frac{1}{S^{K-S}}\pi_0. \tag{121}$$

The expected number of customers at the station can be calculated as follows:

$$\begin{aligned} E[L] &= \sum_{x=0}^{\infty} x\pi(x) \\ &= \sum_{x=0}^{S-1} x\left(\frac{\lambda}{\mu}\right)^x \frac{1}{x!}\pi_0 + \sum_{x=S}^{K} x\left(\frac{\lambda}{\mu}\right)^x \frac{1}{S!}\frac{1}{S^{x-S}}\pi_0. \end{aligned} \tag{122}$$

The formula for the effective throughput can be determined by noticing that customers arriving when the queue is in state K are blocked. The following then holds for the throughput:

$$\begin{aligned} \gamma &= \sum_{x=0}^{K-1} \lambda\pi(x) \\ &= \lambda(1 - \pi(K)) \\ &= \lambda\Big(1 - \Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^{K_i} \frac{1}{S_i!}\frac{1}{S_i^{K_i-S_i}}\Big(\sum_{x=0}^{S_i-1}\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^x \frac{1}{x!} + \frac{1}{S_i!}\sum_{x=S_i}^{K_i}\frac{1}{S_i^{x-S_i}}\Big(\frac{\gamma_{i-1}}{\mu_i}\Big)^x\Big)^{-1}\Big). \end{aligned} \tag{123}$$

Algorithm 2 then easily follows by recursively applying this formula for the throughput of each station (from left to right)

F Appendix F: Feature Names

F.1 SSBAP Non-theoretical Features

Table 70: The non-theoretical feature names and their definition.

Feature	Definition
$NR\_STATIONS$	The number of stations of the tandem line.
$Arrival\_rate$	The external arrival rate to the tandem line.
$Buffer\_i$	The number of buffers at station i.
$Service\_i$	The service rate at station i.

F.2 SSBAP Theoretical Features

Table 71: The mathematical name and the corresponding feature name.

Mathematical name	Feature name
$E[L_i]$	MM1_EL_i
EL_{sum}	MM1_EL_sum
EL_{σ}	MM1_EL_std
$P(X_i > K_i)$	MM1_prob_i
$P(X_i = 0)$	MM1_idle_i
$P(X_i = 0)_{\sigma}$	MM1_idle_std
$P(X_i = K_i)$	MM1_blocked_i
$P(X_i = K_i)_{\sigma}$	MM1_blocked_std
t_{tp}	MM1_t_tp
$E[L_i]^{MM1K}$	MM1K_EL_i
$E[L_i]^{MM1K}_{\sigma}$	MM1K_EL_std
π^{0_i}	MM1K_idle_i
π^{0}_{σ}	MM1K_idle_std
π^{K_i}	MM1K_blocked_i
π^{K}_{σ}	MM1K_blocked_std
γ_i	gamma_i
$E[L_i]^{MM1K}_{\gamma}$	MM1K_EL_g_i
$E[L_i]^{MM1K}_{\gamma\sigma}$	MM1K_EL_g_std
$\pi^{0_i}_{\gamma}$	MM1K_idle_g_i
$\pi^{0}_{\gamma\sigma}$	MM1K_idle_g_std
$\pi^{K_i}_{\gamma}$	MM1K_blocked_g_i
$\pi^{K}_{\gamma\sigma}$	MM1K_blocked_g_std

F.3 MSBAP Non-theoretical Features

Table 72: The non-theoretical feature names and their definition.

Feature	Definition
NR_STATIONS	The number of stations of the tandem line.
Arrival_rate	The external arrival rate to the tandem line.
Nr_servers_i	The number of servers at station i.
Buffer_i	The number of buffers at station i.
Service_i	The service rate at station i.

F.4 MSBAP Theoretical Features

Table 73: The mathematical name and the corresponding feature name.

Mathematical name	Feature name
$E[L_i]$	MMS_EL_i
EL_{sum}	MMS_EL_sum
EL_σ	MMS_EL_std
$P(X_i > K_i)$	MMS_prob_i
$\pi_{0_i}^{MMS}$	MMS_idle_i
$\pi_0^{MMS_\sigma}$	MMS_idle_std
$\pi_{K_i}^{MMS}$	MMS_blocked_i
$\pi_K^{MMS_\sigma}$	MMS_blocked_std
$E[L_i]^{MMSK}$	MMSK_EL_i
$E[L]_\sigma^{MMSK}$	MMSK_EL_std
$\pi_{0_i}^{MMSK}$	MMSK_idle_i
$\pi_0^{MMSK_\sigma}$	MMSK_idle_std
$\pi_{K_i}^{MMSK}$	MMSK_blocked_i
$\pi_K^{MMSK_\sigma}$	MMSK_blocked_std
γ_i	gamma_i
$E[L_i]_\gamma^{MMSK}$	MMSK_EL_g_i
$E[L]_{\gamma\sigma}^{MMSK}$	MMSK_EL_g_std
$\pi_{\gamma 0_i}^{MMSK}$	MMSK_idle_g_i
$\pi_{\gamma 0}^{MMSK_\sigma}$	MMSK_idle_g_std
$\pi_{\gamma K_i}^{MMSK}$	MMSK_blocked_g_i
$\pi_{\gamma K}^{MMSK_\sigma}$	MMSK_blocked_g_std

References

Adan, I. and J. Resing (2002). *Queuing Theory.*

Allon, G., D.P. Kroese, T. Raviv, and R.Y. Rubinstein (2005). “Application of the Cross-Entropy Method to the Buffer Allocation Problem in a Simulation-Based Environment”. In: *Annals of Operations Research* 134, pp. 137–151.

Altiok, T. (1982). “Approximate analysis of exponential tandem queues with blocking”. In: *European Journal of Operational Research* 11, pp. 390–398.

Altiparmak, F., B. Dengiz, and A.A. Bulgak (2007). “Buffer Allocation and Performance Modeling in Asynchronous Assembly System Operations: An Artificial Neural Network Metamodeling Approach”. In: *Applied Soft Computing* 7, pp. 946–956.

Amaran, S., N.V. Sahinids, B. Sharda, and S.J. Bury (2016). “Simulation Optimization: a Review of Algorithms and Applications”. In: *Annuals of Operations Research* 240.1, pp. 351–380.

Amiri, M. and A. Mohtashami (2012). “Buffer Allocation in Unreliable Production Lines Based on Design of Experiments, Simulation, and Genetic Algorithm”. In: *The International Journal of Advanced Manufacturing Technology* 62, pp. 371–383.

Angelini, E., G. di Tollo, and A. Roli (2008). “A Neural Network Approach for Credit Risk Evaluation”. In: *The Quarterly Review of Economics and Finance* 48, pp. 733–755.

Ankenman, B., B.L. Nelson, and J. Staum (2010). “Stochastic Kriging for Simulation Metamodeling”. In: *Operations Research* 58.2, pp. 371–382.

Aussem, A. and D. Hill (2000). “Neural-Network Metamodelling for the Prediction of Caulerpa Taxifolia Development in the Mediterranean Sea”. In: *Neurocomputing* 30, pp. 71–78.

Badea, L.M. (2014). “Predicting Consumer Behavior with Artificial Neural Networks”. In: *Procedia Economics and Finance* 15, pp. 238–246.

Bagnasco, A., A. Siri, G. Aleo, G. Rocco, and L. Sasso (2015). “Applying Artificial Neural Networks to Predict Communcation Risks in the Emergency Department”. In: *The Journal of Advanced Nursing* 71.10, pp. 2293–2304.

Battini, D., A. Persona, and A. Regattieri (2009). “Buffer Size Design Linked to Reliability Performance: A Simulative Study”. In: *Computers & Industrial Engineering* 56, pp. 1633–1641.

Bierbooms, R., I.J.B.F. Adan, and M. van Vuuren (2013). “Approximate analysis of single-server tandem queues with finite buffers”. In: *Annals of Operations Research* 209, pp. 67–84.

Botchkarev, A. (2019). “Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Topology”. In: *Interdisciplinary Journal of Information, Knowledge, and Management* 14, pp. 45–76.

Bulgak, A.A. (2006). “Analysis and Design of Split and Merge Unpaced Assembly Systems by Metamodelling and Stochastic Search”. In: *International Journal of Production Research* 44.18-19, pp. 4067–4080.

Burke, P.J. (1956). “The Output of a Queuing System”. In: *Operations Research* 4.6, pp. 699–704.

Burman, M., S.B. Gershwin, and C. Suyematsu (1998). “Hewlett-Packard Uses Operations Research to Improve the Design of a Printer Production Line”. In: *Interfaces* 28.1, pp. 24–36.

Can, B. and C. Heavy (2012). “A Comparison of Genetic Programming and Artificial Neural Networks in Metamodeling of Discrete-Event Simulation Models”. In: *Computers and Operations Research* 39, pp. 424–436.

Chaharsooghi, S.K. and N. Nahavandi (2003). “Buffer Allocation Problem, a Heuristic Approach”. In: *Scientia Iranica* 10.4, pp. 401–409.

Chen, C., J. Li, and Y. Wang (2015). “An energy-saving task scheduling strategy based on vacation queuing theory in cloud computing”. In: *Tsinghua Science and Technology* 20.1, pp. 28–39.

Chen, M. and T. Yang (2002). “Design of Manufacturing Systems by a Hybrid Approach with Neural Network Metamodelling and Stochastic Local Search”. In: *International Journal of Production Research* 40.1, pp. 71–92.

Cigizoglu, H.K (2003). “Incorporation of ARMA Models into Flow Forecasting by Artificial Neural Networks”. In: *Environmentrics* 14, pp. 417–427.

Costa, A., A. Alfieri, A. Matta, and S. Fichera (2015). “A Parallel Tabu Search for Solving the Primal Buffer Allocation Problem in Serial Production Systems”. In: *Computers & Operations Research* 64, pp. 97–112.

Cruz, F.R.B., G. Kendall, L. While, A.R. Duarte, and N.L.C. Brito (2012). “Throughput Maximization of Queueing Networks with Simultaneous Minimization of Service Rates and Buffers”. In: *Mathematical Problems in Engineering* 2012, pp. 1–19.

Dallery, Y. and Y. Frein (1993). “On Decomposition Methods for Tandem Queueing Networks with Blocking”. In: *Operations Research* 41.2, pp. 386–399.

Dallery, Y. and S.B. Gershwin (1992). “Manufacturing flow line systems: a review of models and analytical results”. In: *Queueing Systems* 12, pp. 3–94.

DeJong, K.A. (1975). “An Analysis of the Behavior of a Class of Genetic Adaptive Systems”. In: *PhD Dissertation, University of Michigan.*

Demir, L., S. Tunali, and D.T. Eliiyi (2012). “An Adaptive Tabu Search Approach for Buffer Allocation Problem in Unreliable Non-Homogeneous Production Lines”. In: *Computers & Operations Research* 39, pp. 1477–1486.

— (2014). “The State of the Art on Buffer Allocation Problem: a Comprehensive Survey”. In: *Journal of Intelligent Manufacturing* 25.3, pp. 371–392.

Diamantidis, A.C. and C.T. Papadopoulos (2004). “A Dynamic Programming Algorithm for the Buffer Allocation Problem in Homogeneous Asymptotically Reliable Serial Production Lines”. In: *Mathematical Problems in Engineering* 3, pp. 209–223.

Dolgui, A., A. Eremeev, A. Kolokolov, and V. Sigaev (2002). “A Genetic Algorithm for Allocation of Buffer Storage Capacities in Production Line with Unreliable Machines”. In: *Journal of Mathematical Modelling and Algorithms* 1.2, pp. 89–104.

Doncieux, S., N. Bredeche, J. Mouret, and A.E. Eiben (2015). “Evolutionary Robotics: what, why and where to”. In: *Frontlets in Robotics and AI* 2.4.

Eiben, A.E. and J.E. Smith (2015). *Introduction to Evolutionary Computing.* Springer.

Fonseca, D.J., D.O. Navaresse, and G.P. Moynihan (2003). “Simulation Metamodeling through Artificial Neural Networks”. In: *Engineering Applications of Artificial Intelligence* 16, pp. 177–183.

Gal, Y. and Z. Ghahramani (2016). “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”. In: *In proceedings of the 33rd International Conference on Machine Learning.* New York, USA, pp. 1050–1059.

Garson, G.D. (1991). “Interpreting Neural-Network Connection Weights”. In: *Artificial Intelligence Expert* 6, pp. 47–51.

Gedeon, T.D. (1997). "Data Mining of Inputs: Analysing Magnituda and Functional Measures". In: *International Journal of Neural Systems* 8.2, pp. 209–218.

Gershwin, S.B. (1987). "An Efficient Decomposition Method for the Approximate Evaluation of Tandem Queues with Finite Storage Space and Blocking". In: *Operations Research* 35.2, pp. 291–305.

Glorot, X., A. Bordes, and Y. Bengio (2011). "Deep Sparse Rectifier Neural Networks". In: *In proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS).* Fort Lauderdale, FL, USA, pp. 315–323.

Govil, M.K. and M.C. Fu (1999). "Queueing Theory in Manufacturing: A Survey". In: *Journal of Manufacturing Systems* 18.3, pp. 214–240.

Gürkan, G. (2000). "Simulation Optimization of Buffer Allocations in Production Lines with Unreliable Machines". In: *Annals of Operations Research* 93, pp. 177–216.

Hartmann, S. (2001). "Project Scheduling with Multiple Modes: A Genetic Algorithm". In: *Annals of Operationa Research* 102, pp. 111–135.

Helber, S., K. Schimmelpfend, R. Stolletz, and S. Lagershausen (2011). "Using Linear Programming to Analyze and Optimize Stochastic Flow Lines". In: *Annals of Operations Research* 182, pp. 193–211.

Hoef, J.M. Ver (2012). "Who Invented the Delta Method?" In: 66.2, pp. 124–127.

Hoogendoorn, M. and B. Funk (2018). *Machine Learning for the Quantified Self: On the Art of Learning from Sensory Data.* Springer Nature.

Hornby, G.S., J.D. Lohn, and D.S. Linden (2011). "Computer-Automated Evolution of an X-Band Antenna for NASA's Space Technology 5 Mission". In: *Evolutionary Computing* 19.1, pp. 1–23.

Hurrion, R.D. (1997). "An Example of Simulation Optimisation Using a Neural Network Metamodel: Finding the Optimum Number of Kanbans in a Manufacturing System". In: *Journal of the Operational Research Society* 48, pp. 1105–1112.

Jebari, K., M. Madiafi, and A. Elmoujahid (2013). "Parent Selection Operators for Genetic Algorithms". In: *International Journal of Engineering Research & Technology (IJERT)* 2.11, pp. 1141–1145.

Kilmer, R.A. and A.E. Smith (1997). "An Emergency Department Simulation and a Neural Network Metamodel". In: *Journal of the Society for Health Systems* 5.3, pp. 63–79.

Kilmer, R.A., A.E. Smith, and L.J. Shuman (1999). "Computing Confidence Interals for Stochastic Simulation using Neural Network Metamodels". In: *Computers and Industrial Engineering* 36, pp. 391–407.

Kingma, D.P. and J.L. Ba (2017). "Adam: a Method for Stochastic Optimization". In: *Retrieved January 20, from https://arxiv.org/abs/1412.6980.*

Kuo, Y., T. Yang, B.A. Peters, and I. Chang (2007). "Simulation Metamodel Development using Uniform Design and Neural Networks for Automated Material Handling Systems in Semiconductor Wafer Fabrication". In: *Simulation Modelling Practice and Theory* 15, pp. 1002–1015.

Kussul, E., T. Baidyk, and D. C. Wunsch (2010). *Neural Networks and Micromechanics.* Springer.

Lechevalier, D., R. Ak, Y. Lee, S. Hudak, and S. Foufou (2015). "A Neural Network Meta-Model and its Applications for Manufacturing". In: *IEEE International Conference on Big Data.* Santa Clara, United States.

Leshno, M., V.Y. Lin, A. Pinkud, and S. Schocken (1993). "Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximaty Any Function". In: *Neural Networks* 6, pp. 861–867.

Leung, M.T., A. Chen, and H. Daouk (2000). "Forecasting Exchange Rates using General Regression Neural Networks". In: *Computers & Operations Research* 27, pp. 1093–1110.

Li, J. (2005). "Overlapping Decomposition: a System-Theoretic Method for Modeling and Analysis of Complex Manufacturing Systems". In: *IEEE Transactions on Automation Science and Engineering* 2.1, pp. 40–53.

Li, S., Q. Wang, and G. Koole (2021). "Optimal Contact Center Staffing and Scheduling with Machine Learning". In: *Currently submitted*.

Liao, Y., D. Yau, and C. Chen (2012). "Evolutionary Algorithm to Traveling Salesman Problems". In: *Computers and Mathematics with Applications* 64, pp. 788–797.

Liefoogh, A. and L. Paquete (2019). "Proceedings of EvoCOP". In: *19th European Conference, EvoCOP 2019, Held as Part of EvoStar 2019.* Leipzig, Germany.

Lim, J., S.M. Meerkov, and F. Top (1990). "Homogeneous, Asymptotically Reliable Serial Production Lines: Theory and a Case Study". In: *IEEE Transactions on Automatic Control* 35.5, pp. 524–534.

Lünsdorf, O. and S. Scherfke (2020). "SimPy Package". In: *Retrieved December 23, from: https://simpy.readthedocs.io/en/latest/index.html.*

M. Nourelfath, N. Nahas and D. Ait-Kadi (2005). "Optimal Design of Series Production Lines with Unreliable Machines and Finite Buffers". In: *Journal of Quality in Maintenance Engineering* 11.2, pp. 121–138.

Mahfoud, S.W. (1995). "Niching Methods for Genetic Algorithms". In: *PhD Dissertation, University of Illinois.*

Matta, A., M. Pezonni, and Q. Semeraro (2012). "A Kriging-Based Algorithm to Optimize Production Systems Approximated by Analytical Models". In: *Journal of Intelligence Manufacturing* 23, pp. 587–597.

Mesghouni, K., S. Hammadi, and P. Borne (2004). "Evolutionary Algorihms for Job-Shop Scheduling". In: *International Journal of Applied Mathematics and Computer Science* 14.1, pp. 91–103.

Mühlenbein, H., M. Gorges-Schleuter, and O. Krämer (1988). "Evolution Algorithms in Combinatorial Optimization". In: *Parallel Computing* 7, pp. 65–85.

Nahas, N. (2017). "Buffer Allocation and Preventive Maintenance Optimization in Unreliable Production Lines". In: *Journal of Intelligent Manufacturing* 28, pp. 85–93.

Nahas, N., D. Ait-Kadi, and M. Nourelfath (2006). "A New Approach for Buffer Allocation in Unreliable Production Lines". In: *International Journal of Production Economics* 103, pp. 873–881.

Nahas, N., M. Nourelfath, and D. Ait-Kadi (2009). "Selecting Machines and Buffers in Unreliable Series-Parallel Production Lines". In: *International Journal of Production Research* 47.14, pp. 3741–3774.

Nilsen, G.K., A.Z. Munthe-Kaas, H.J. Skaug, and M. Brun (2019). "On the Delta Method for Uncertainty Approximation in Deep Learning". In: *Retrieved January 20, from: https://arxiv.org/abs/1912.00832.*

Nsoesie, E.O., R.J. Beckman, S. Shashaani, K.S. Nagaraj, and M.V. Marathe (2013). "A Simulation Optimization Approach to Epidemic Forecasting". In: *PLoS One* 8.6.

Olden, J.D. and D.A. Jackson (2002). “Illuminating the “Black Box”: a Randomization Approach for Understanding Variable Contributions in Artificial Neural Networks”. In: *Ecological Modelling* 154, pp. 135–150.

Olden, J.D., M.K. Joy, and R.G. Death (2004). “An Accurate Comparison of Methods for Quantifying Variable Importance in Artificial Neural Using Simulated Data”. In: *Ecological Modelling* 178, pp. 389–397.

Onvural, R.O. and H.G. Perros (1986). “On equivalences of blocking mechanisms in queueing networks with blocking”. In: *Operations Research Letters* 5.6, pp. 293–297.

Palvannan, R.K. and K.L. Teow (2012). “Queueing for Healthcare”. In: *Journal of Medical Systems* 36, pp. 541–547.

Papadopoulos, H.T. and M.I. Vidalis (2001). “A Heuristic Algorithm for the Buffer Allocation in Unreliable Unbalanced Production Lines”. In: *Computers and Industrial Engineering* 41, pp. 261–277.

Papadopoulos, H.T. and G.A. Vouros (1997). “A Model Management System (MMS) for the Design and Operation of Production Lines”. In: *International Journal of Production Research* 35.8, pp. 2213–2236.

Parmer, C. (2021). “Dash Package”. In: *Retrieved January 14, from: https://pypi.org/project/dash/*.

Pearce, T., M. Zaki, A. Brintrup, and A. Neely (2018). “High-Quality Prediction Intervals for Deep Learning: A Distribution-Free, Ensembled Approach”. In: *In proceedings of the 35th International Conference on Machine Learning*. Stockholm, Sweden, pp. 4075–4084.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, V. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pierreval, H. (1996). “A Metamodeling Approach Based on Neural Networks”. In: *Internation Journal of Computer Simulation* 6, pp. 365–388.

Powell, S.G. (1994). “Buffer Allocation in Unbalanced Three-Station Serial Lines”. In: *International Journal of Production Research* 32.9, pp. 2201–2217.

Prins, C. (2004). “A simple and effective Evolutionary Algorithm for the Vehicle Routing Problem”. In: *Computers & Operations Research* 31, pp. 1985–2002.

Qaid, G.R.S. and S.N. Talbar (2013). “Encrypting and Decrypting Images by using Genetic Algorithms”. In: *Proceedings of the third international conference on Computational Intelligence and Information Technology CIIT*. Mumbai, India, pp. 50–54.

Qudeiri, J.A., H. Yamamoto, R. Ramli, and A. Jamali (2008). “Genetic Algorithm for Buffer Size and Work Station Capacity in Serial-Parallel Production Lines”. In: *Artificial Life Robotics* 12, pp. 102–106.

Qudeiri, J.A., H. Yamamoto, R. Ramli, and K.R. Al-Momani (2007). “Development of Production Simulator for Buffer Size Decisions in Complex Production Systems Using Genetic Algorithm”. In: *Journal of Advanced Mechanical Design, Systems, and Manufacturing* 1.3, pp. 418–429.

Rosca, J.P. and D.H. Ballard (1995). “Causality in Genetic Programming”. In: *Proceedings of the fifth International Coference ICGA*. Pittsburgh, PA, USA, pp. 256–263.

Sabuncuoglu, I., E. Erel, and Y. Gocgun (2006). “Analysis of Serial Production Lines: Characterisation Study and a New Heuristic Procedure for Optimal Buffer Allocation”. In: 44.13, pp. 2499–2523.

Sabuncuoglu, I. and S. Touhami (2002). “Simulation Metamodelling with Neural Networks: an Experimental Investigation”. In: *International Journal of Production Research* 40.11, pp. 2483–2505.

Sareni, B. and L. Krähenbühl (1998). “Fitness Sharing and Niching Methods Revisited”. In: *IEEE Transactions on Evolutionary Computation* 2.3, pp. 97–106.

Schmidt, M. and H. Lipson (2009). “Distilling Free-Form Natural Laws from Experimental Data”. In: *Science* 324, pp. 81–85.

Schwartz, J.D., W. Wang, and D.E. Rivera (2006). “Simulation-Based Optimization of Process Control Policies for Inventory Management in Supply Chains”. In: *Automatica* 42, pp. 1311–1320.

Seong, D., S.Y. Chang, and Y. Hong (1995). “Heuristics Algorithms for Buffer Allocation in a Production Line with Unreliable Machines”. In: 33.7, pp. 1989–2005.

Shi, L. and S. Men (2003). “Optimal Buffer Allocation in Production Lines”. In: *IIE Transactions* 35, pp. 1–10.

Shin, Y.W. and D.H. Moon (2014). “Approximation of throughput in tandem queues with multiple servers and blocking”. In: *Applied Mathematical Modelling* 38, pp. 6122–6132.

Shourian, M., S.J. Mousavi, M.B. Menhaj, and E. Jabbari (2008). “Neural-Network-Based Simulation-Optimization Model for Water Allocation Planning at Basin Scale”. In: *Journal of Hydroinformatics* 10.4, pp. 331–343.

Singh, A. and J. MacGregor Smith (1997). “Buffer Allocation for an Integer Nonlinear Network Design Problem”. In: *Computers & Operations Research* 24.5, pp. 453–472.

Spinellis, D.D. and C.T. Papadopoulos (2000a). “A Simulated Annealing Approach for Buffer Allocation in Reliable Production Lines”. In: *Annals of Operations Research* 93, pp. 373–384.

— (2000b). “Stochastic Algorithms for Buffer Allocation in Reliable Production Lines”. In: *Mathematical Problems in Engineering* 5, pp. 441–458.

Srivastav, R.K., K.P. Sudheer, and I. Chaubey (2007). “A Simplified Approach to Quantifying Predictiva and Parametric Uncertainty in Artificial Neural Network Hydrologic Models”. In: *Water Resources Research* 43.

Tijms, H.C. (2003). *A First Course in Stochastic Models*. Wiley.

Tsadiras, A.K., C.T. Papadopoulos, and M.E.J. O’Kelly (2013). “An Artificial Neural Network Based Decision Support System for Solving the Buffer Allocation Problem in Reliable Production Lines”. In: *Computers and Industrial Engineering* 66, pp. 1150–1162.

Ursem, R.K. (2002). “Diversity-Guided Evolutionary Algorithms”. In: *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*. Granada, Spain, pp. 462–471.

Vasconcelos, M.J. Perestrello de, S. Silva, M. Tomé, M. Alvim, and J.M.C Pereira (2001). “Spatial Prediction of Fire Ignition Probabilities: Comparing Logistic Regression and Neural Networks”. In: *Photogrammetric Engineering & Remote Sensing* 67.1, pp. 73–81.

Vinterbo, S. and L. Ohno-Machado (1999). “A Genetic Algorithm to Select Variables in Logistic Regression: Example in the Domain of Myocardial Infarction”. In: *Proceedings of AMIA Symposium*. Washington, DC, USA, pp. 984–988.

Weiss, S., J.A. Schwarz, and R. Stolletz (2019). “The Buffer Allocation Problem in Production Lines: Formulations, Solution Methods, and Instances”. In: *IISE Transactions* 51.5, pp. 456–485.

Xu, K., M. Zhang, J. Li, S.S. Du, K. Kawarabayashi, and S. Jegelka (2020). “How Neural Networks Extrapolate: From Feedforwar to Graph Neural Networks”. In: *Retrieved January 16, from https://arxiv.org/abs/2009.11848.*

Yu, L., S. Wang, and K.K. Lai (2007). *Foreign-Exchange-Rate Forecasting with Artificial Neural Networks*. Springer.