Design and control of stochastic manufacturing systems

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Wirtschaftswissenschaften
der Universität Mannheim

vorgelegt von

Johannes Diefenbach

Ilvesheim

Dekan: *Joachim Lutz*

Referent: *Prof. Dr. Raik Stolletz*

Korreferent: *Prof. Dr. Moritz Fleischmann*

Tag der mündlichen Prüfung: *15.07.2022*

Acknowledgements

This dissertation is the result of more than six years of academic work. As with any project, it benefited tremendously from the support of and the discussions with many people. I appreciate this continuous support and all the feedback I received.

First, I would like to thank Raik Stolletz for giving me the opportunity to deepen my knowledge in the field of Operations Research. I learned a lot during our many discussions and especially appreciate his feedback in the preparation of the different presentations I held over the years. Furthermore, I would like to thank Moritz Fleischmann for writing the second review to my thesis and the open exchange we have had related to my research. I extend my gratitude to Christoph Bode and Justus Arne Schwarz for participating in the examination board. In addition, I appreciate everything I was allowed to learn from Justus Arne Schwarz during our joint research projects.

There were many colleagues who helped me with my research and were always open to grab a coffee with me. Thank you all for the good time we had. I want to say a special thank you to Matteo Biondi and Seyed Mohammad Zenouzzadeh for not only sharing an office with me, but also for making me laugh at times when I felt frustrated. In addition, I want to thank Katharina Senz for all her support and her never ending happy attitude. As I finished this thesis in a new job outside the university, I appreciate all the support I have received from my new colleagues, especially my manager Iris Heckmann.

All of this was only possible, because my family has always taught me the value of education, which allowed me to pursue this dream of mine without second guessing myself. My wife Sabrina deserves my biggest thanks in this long list of wonderful people. She supported me through all the highs and lows of the last six years. Especially when my workload was high, she stepped in, allowing me to focus on my research.

Finally, I appreciate the financial support from the Julius-Paul Stiegler-Gedächtnis-stiftung, which allowed me share my research on an international conference.

Mannheim, November 2022,
Johannes Diefenbach

Summary

Many manufacturing systems are subject to uncertainty, which can be described using stochastic processes. These processes might be stable or change with the production quantity. This dissertation analyzes the design and control of such stochastic manufacturing systems.

The first article investigates the balancing of an assembly line with stochastic task times and a constraint on the line reliability. We provide a sampling-based model formulation for generally distributed task times. We prove that any lower bound on the number of stations for the related deterministic problem can be transformed into a lower bound for this sampling formulation. We apply these bounds in a reliability-based branch-and-bound algorithm and show that they substantially reduce the required computation time.

The second article analyzes the impact of the used sampling method and the sample size on the resulting performance measures and optimal decision by considering the performance evaluation of an M/D/1 queueing system and the optimization of an M/M/c staffing level numerically. The article suggests that managers should be aware that the distribution of the resulting performance measures or optimal solution derived from a sampling-based approach may not be symmetrical and that the chosen sampling method may have an impact on this behavior.

The third article investigates the ramp-up of a new product or machine with stochastic and non-stationary yield. We formalize the problem as a Newsvendor problem and prove that any positive optimal ramp-up quantity will always be at least the demand. Furthermore, we characterize the optimal ramp-up quantity for the special case of stationary yield by a critical fractile. The optimal ramp-up quantity tends to be decreasing in the expected yield. However, a numerical analysis shows that an increase in the expected yield can lead to a higher optimal production quantity at first, before the production quantity decreases.

There is a gap in the literature for each of the considered optimization problems under the considered assumptions. Future research could integrate the design and control decisions considered in this dissertation into a single optimization model.

Contents

List of Figures

List of Tables

1 Introduction

The optimization of manufacturing systems can be divided into a design and a control phase. In the design phase, a new manufacturing process is set up or an existing process is redesigned. Optimization objectives of the design phase can be manifold, for example the maximization of the throughput or the minimization of the required stations. After the design phase, the control phase optimizes the production planning in manufacturing systems. A classical example is matching supply with demand. Manufacturing systems are often stochastic, for example due to the involvement of human workers (Doerr and Arreola-Risa, 2000; Cortés et al., 2010), or due to yield (e.g. Lee and Yano, 1988; Tang et al., 2012).

This thesis addresses the design and control of such stochastic manufacturing systems. It consists of three articles, each co-authored by a different set of authors. These authors are Raik Stolletz, Justus Arne Schwarz and Fikri Karaesmen. The articles study different optimization problems focused on the design and control of stochastic manufacturing systems. Each article describes the motivation for the research and formalizes the analyzed optimization problem. Analytical and numerical studies generate insights on the analyzed problems.

Assembly lines are common in today's mass production and their design is a key driver of efficiency. Line balancing is an important step in designing a paced assembly system. The first article (Chapter 2) considers the assembly line balancing problem with stochastic task times. Tasks have to be assigned to a minimum number of stations with a constraint on the line reliability, which is the probability of finishing a work piece completely. A sampling approach is developed that ensures the line reliability. We prove that any lower bound on the number of stations for the related deterministic problem can be transformed into a lower bound for this sampling formulation. We exemplify the usefulness of these bounds in a reliability-based branch-and-bound (RB&B) algorithm that explicitly considers the dependence among all stations due to the constrained line reliability. Effective fathoming strategies based on the new transformed lower bounds or based on a direct consideration of the line reliability are proposed.

For any sampling-based approach, the robustness of the decisions for a given sample size and sampling method is always of interest. Therefore, the second article (Chapter 3) analyzes the impact of the sample size and the sampling method on the performance evaluation and optimization numerically. Sampling-based optimization is often used to analyze the performance or optimize the design of complex, stochastic optimization problems. Common approaches to draw the required random numbers are simple random sampling (SRS) and descriptive sampling (DS). We conduct a numerical study and analyze the distribution of the performance measures or of the optimal decision over independent replications. It analyzes the design of an M/M/c system with c parallel servers and the objective of minimizing the required number of servers. Furthermore, the performance evaluation of an M/D/1 queueing system is analyzed.

After the initial design of a system, a ramp-up phase is often observed, in which the system still improves its performance. The third article (Chapter 4) is motivated by the ramp-up in the semiconductor manufacturing industry. During the introduction of a new product or machine, the yield of a production process tends to start low and increases with the production quantity. This is known as the ramp-up phase and the company has to chose the production quantity during the ramp-up phase ex ante. The analysis of real yield data from the company shows that such a stochastic and non-stationary yield behavior occurs both for the introduction of new products as well as for the introduction of new machines. We formalize the company's problem as a Newsvendor problem with stochastic and non-stationary yield. We derive analytical and numerical insights on the optimal ramp-up quantity and the expected profit.

Chapter 5 discusses the conclusions regarding this thesis as a whole as well as the potential for further research. Appendices A and B further specify the lower bounds and the heuristic used in Chapter 2. Additional instances are analyzed in Appendix C and the complete implementation of the RB&B in Python from Chapter 2 can be found in Appendix D. Appendix E and F show further instances for the performance evaluation and optimization analyzed in Chapter 3. Known results for the normal approximation of a special case of the model considered in Chapter 4 are presented in Appendix G. The references for all articles are presented in a joint bibliography.

2 Stochastic assembly line balancing: General bounds and reliability-based branch-and-bound algorithm

Co-authors:

Johannes Diefenbach
Chair of Production Management, Business School, University of Mannheim, Germany

Raik Stolletz
Chair of Production Management, Business School, University of Mannheim, Germany

Published in:

European Journal of Operational Research 302 (2), 589-605
DOI: 10.1016/j.ejor.2022.01.015

Abstract:

We analyze the assembly line balancing problem with stochastic task times. Tasks have to be assigned to a minimum number of stations with a constraint on the line reliability, which is the probability of finishing a work piece completely. A sampling approach is developed that ensures the line reliability. We prove that any lower bound on the number of stations for the related deterministic problem can be transformed into a lower bound for this sampling formulation. This general transformation can be applied to any bound that has already been developed or to any potential new bound. Those bounds can be applied to any MIP model, optimization algorithm or heuristic procedure based on a sampling formulation. We

exemplify the usefulness of these bounds in a reliability-based branch-and-bound (RB&B) algorithm that explicitly considers the dependence among all stations due to the constrained line reliability. A partial assignment of tasks to stations has to consider already constructed stations and potential further assignments to other stations. Hence, a feasible assignment of tasks to this station may allow for exceeding the cycle time with a certain probability but has to consider the overall line reliability with respect to the remaining stations. Effective fathoming strategies based on the new transformed lower bounds or based on a direct consideration of the line reliability are proposed. A numerical study shows that the transformed lower bounds are tight and that they substantially reduce the required computation times of the RB&B algorithm and of the solver CPLEX.

2.1 Introduction

2.1.1 Motivation and optimization problem

Assembly lines are common in today's mass production and their design is a key driver of efficiency. Line balancing is an important step in designing a paced assembly system. It assigns tasks to stations such that the cycle time is not exceeded at each station and technical precedence relations between the tasks are fulfilled.

The simple assembly line balancing problem (SALBP) assumes task times to be deterministic. However, task times in manufacturing are often stochastic, for example due to the involvement of human workers (Doerr and Arreola-Risa, 2000; Cortés et al., 2010).

Because of stochastic task times, staying within the cycle time at a certain station may no longer be guaranteed. Exceeding the cycle time even at a single station results in an incomplete work piece. Different policies have been proposed to deal with incomplete work pieces. Reeve and Thomas (1973) analyze stopping the line, until all work pieces are completed at all stations. Further approaches are to let the work piece continue down the line with as many tasks completed as possible, but with associated cost of rework, or to steer the line with a constraint on the probability of staying within the cycle time (Liu et al., 2005). For a sequential production process as we consider, Graves (1998) defines the rolled throughput yield as the probability of a work piece to be finished at the end of the line. This rolled throughput yield is an important measure in Six Sigma and can be used to control the performance of production processes (Graves, 2002). In the context of assembly line balancing, the rolled throughput yield is equal to the probability that a work piece can be finished within the cycle time at each station of the entire line, which is defined as the *line reliability*. In contrast, a constraint on the station-based reliability requires only an isolated probability of staying within the cycle time at a single station. To ensure a certain probability of finishing a work piece, managers should consider a constraint on the line reliability when assigning tasks to stations.

The problem of minimizing the length of the line is relevant in practice, see for example Lapierre and Ruiz (2004). The alternate optimization problem of maximizing the line reliability for a given number of stations is also an important optimization problem in practice, see for example Silverman and Carter (1986). The related literature considers the minimization of the probability of a line stoppage, which can

been seen as the counter probability of the line reliability (e.g. Reeve and Thomas, 1973; McMullen and Frazier, 1998).

2.1.2 Solution approach and contributions

We present a sampling-based model for line balancing with a given cycle time and the objective to minimize the number of stations (the deterministic version is known as SALBP-1).

Stochastic Programming is a common technique to model such stochastic optimization problems (e.g. Birge and Louveaux, 2011). One class of stochastic optimization problems are chance-constrained programs. We consider such a chance-constrained program, as a minimal probability of finishing a work piece for the entire line is requested. When the distribution of the random variable is known and analytically tractable, the chance-constraint may directly be implemented in the optimization model. Sampling-based approximation methods have been proposed for the cases with known but analytically intractable distributions (e.g. Calafiore and Campi, 2005; Luedtke and Ahmed, 2008). As we consider generally distributed task times, we present and solve a sampling-based model for this line balancing problem with a constraint on the line reliability with respect to the cycle time and the objective of minimizing the number of stations. The presented sampling-based model can be applied to both uncorrelated or correlated distributions of task times.

Lower bounds are crucial in many solution methods for deterministic assembly line balancing (Scholl and Becker, 2006). For the stochastic assembly line balancing problem, only a few lower bounds on the number of stations are known under specific assumptions regarding the distribution of the task times (Betts and Mahmoud, 1989; Urban and Chiang, 2006; Chiang et al., 2016). To close this gap, we develop a general transformation of any lower bound on the number of stations for the deterministic SALBP-1 into a lower bound for the sampling-based model with stochastic task times. The developed bounds are valid with respect to the analyzed sample and therefore can be considered as approximations for the original problem. This transformation can be applied to any bound that has already been developed or to any potential new bound. These bounds may significantly improve several solution methods. We will exemplify this for a branch-and-bound algorithm as well as for the CPLEX solver.

The constraint on the line reliability creates a dependence among the stations as the probabilities of exceeding the cycle time may be different from station to station.

We develop a reliability-based branch-and-bound (RB&B) algorithm that explicitly considers the dependence among all stations due to the constrained line reliability. When constructing a potential station during the branch-and-bound algorithm, a partial assignment of tasks to this station has to consider already constructed stations and potential further assignments to other stations. Hence, a feasible assignment of tasks to this station may allow for exceeding the cycle time with a certain probability but has to consider the overall line reliability with respect to the remaining stations. This significantly increases the number of potential nodes of the branching tree. Therefore, effective fathoming strategies based on the new transformed lower bounds or based on a direct consideration of the line reliability are proposed.

A numerical study shows that the transformed lower bounds are tight and that they substantially reduce the required computation times of the RB&B algorithm and of the solver CPLEX. We analyze the sensitivity of the desired line reliability and of the distribution of task times on the optimal number of stations.

The main contributions of this paper are as follows:

1. We prove a general transformation of any lower bound on the number of stations from the deterministic problem to a sampling-based model for assembly line balancing with a constraint on the line reliability. To improve the computation times, the new bounds can be applied to any MIP formulation, optimization algorithm or heuristic procedure based on a sampling formulation.
2. To examine the usefulness of these bounds, we develop an RB&B algorithm that explicitly considers the dependence among all stations due to the constrained line reliability. Therefore, a partial assignment of tasks to stations has to consider already constructed stations and potential further assignments to other stations. The developed fathoming strategies based on the transformed lower bounds and on the constraint on the line reliability effectively reduce the required number of nodes and the computation time.

2.1.3 Structure of the paper

The remainder of this paper is organized as follows. We provide an overview of the literature on assembly line balancing with a reliability constraint in Section 2.2. In addition, we review the literature on lower bounds for stochastic assembly line balancing. The chance-constraint and the sampling-based model are presented in Section 2.3. In Section 2.4, we develop and prove a general transformation of any lower bound for the deterministic problem into a lower bound for the sampling-

based model. Section 2.5 presents the RB&B algorithm to optimally solve the sampling-based model. The construction of nodes considers the line reliability and we propose several fathoming strategies. The numerical study in Section 2.6 analyzes the fathoming strategies in isolation and shows the value of the combined lower bounds for the RB&B algorithm and for CPLEX. The sensitivities of the desired line reliability and the distribution of task times on the optimal number of stations are analyzed. The findings are summarized in Section 2.7.

2.2 Literature on assembly line balancing with stochastic task times

We first review the literature on the stochastic assembly line balancing problem with a constraint on the reliability in Section 2.2.1. The literature on lower bounds for stochastic assembly line balancing is reviewed in Section 2.2.2.

2.2.1 Reliability-based Line Balancing

This section reviews the literature on stochastic assembly line balancing with a reliability constraint on the stations or on the entire line. Reliability-based line balancing has also been applied to disassembly lines with stochastic task times (e.g. Bentaha et al., 2015). For a recent review on disassembly line balancing see Özceylan et al. (2019).

Table 2.1 gives an overview of optimization models with constraints on the *station-based reliability*. It shows the underlying distribution, the line setup, the assumed product mix, and the considered objective function. The last column states the applied solution methods.

The literature is sorted according to the distribution of the task times: uniform, normal and general. The majority of the literature on the stochastic assembly line balancing problem studies normally distributed task times. Many papers assuming general distributions apply the Normal distribution for the numerical study (Kao, 1976, 1979; Raouf and Tsui, 1982; Henig, 1986; Nkasu and Leung, 1995; Guerriero and Miltenburg, 2003; Chiang and Urban, 2006; Leitold et al., 2019). Most papers assume straight and U-shaped lines while some consider two-sided lines or lines with parallel stations. Only three papers consider line balancing for mixed products. The objective functions consist of the number of stations, the cycle time or the line

				Objective				
	Distr.	Line	Prod.	Stations	Cycle time	Balance	Reliability	Method
McMullen and Frazier (1997)	Uniform	Parallel	Mixed	x				Incremental utilization
McMullen and Tarasewich (2003)	Uniform	Parallel	Mixed	x				Ant colony
Moodie and Young (1965)	Normal	Straight	Single			x		Greedy heuristic
Carraway (1989)	Normal	Straight	Single	x				Dynamic programming
Urban and Chiang (2006)	Normal	U-Line	Single	x				Linearization
Ağpak and Gökçen (2007)	Normal	U-Line	Single	x				Linearization
Baykasoğlu and Özbakır (2007)	Normal	U-Line	Single	x		x	x	Genetic alg.
Özcan (2010)	Normal	2-Sided	Single	x				Simulated annealing
Bagher et al. (2011)	Normal	U-Line	Single	x		x	x	Imperialist competitive alg.
Cakir et al. (2011)	Normal	Parallel	Single	x		x		Simulated annealing
Chiang et al. (2016)	Normal	2-Sided	Single	x				Particle swarm opt.
Delice et al. (2016)	Normal	2-Sided	Single	x				Genetic alg.
Tang et al. (2017)	Normal	2-Sided	Single	x				Teaching-learning-based alg.
Dong et al. (2018)	Normal	Straight	Single		x			Particle swarm opt.
Özcan (2018)	Normal	Parallel	Single	x				Tabu search
Zhang et al. (2018)	Normal	U-Line	Single	x		x	x	Evolutionary alg.
Aydoğan et al. (2019)	Normal	U-Line	Single	x				Particle swarm opt.
Fathi et al. (2019)	Normal	Straight	Single	x		x		CPLEX
Foroughi and Gökçen (2019)	Normal	Straight	Single		x			Genetic alg.
Kao (1976, 1979)	General	Straight	Single	x				Heuristic DP
Sphicas and Silverman (1976)	General	Straight	Single	x				Ignall's method
Sniedovich (1981)	General	Straight	Single	x				Heuristic DP
Raouf and Tsui (1982)	General	Straight	Single			x		Greedy heuristic
Henig (1986)	General	Straight	Single		x			Heuristic DP
Betts and Mahmoud (1989)	General	Straight	Single	x				Branch-and-bound
Nkasu and Leung (1995)	General	Straight	Single	x	x	x		COMSOAL
Merengo et al. (1999)	General	Straight	Mixed	x				Improvement heuristic
Guerriero and Miltenburg (2003)	General	U-Line	Single	x				Recursion alg.
Chiang and Urban (2006)	General	U-Line	Single	x				Improvement heuristic
Boysen and Fliedner (2008)	General	U-Line	Single	x	x			AVALANCHE
Leitold et al. (2019)	General	Straight	Single		x			Dynamic programming

Table 2.1: Optimization models with a constrained station-based reliability

	Distr.	Line	Prod.	Objective: Stations	Objective: Cycle time	Objective: Balance	Objective: Reliability	Method
Liu et al. (2005)	Normal	Straight	Single		x			Bidirectional assignment
Chiang et al. (2016)	Normal	2-Sided	Single	x				Particle swarm opt.
Tang et al. (2017)	Normal	2-Sided	Single	x				Teaching-learning-based alg.

Table 2.2: Optimization models with a constrained line reliability

balance. In addition, some papers account for the average reliability per station. The minimization of the number of stations is the most common objective while some papers consider several components. Both heuristic and optimal solution methods are applied. For the optimal solution methods, dynamic programming, linearization of chance-constrained problems and branch-and-bound algorithms have been used.

The studies mentioned above constrain the reliability of each station in isolation instead of the entire line. Table 2.2 gives an overview of the literature considering a constraint on the *line reliability*.

Stochastic line balancing with a constraint on the line reliability has been studied for normally distributed task times with a single product. Liu et al. (2005) analyze the problem of minimizing the cycle time in a straight line and present a bidirectional construction and trade-and-transfer based improvement heuristic. Chiang et al. (2016) and Tang et al. (2017) analyze the problem of two-sided assembly lines with the objective of minimizing the number of stations. Chiang et al. (2016) present a particle swarm optimization algorithm and Tang et al. (2017) present a hybrid teaching-learning-based heuristic to solve the problem.

To summarize, line balancing with stochastic task times and station-based reliability is studied with generally distributed task times. However, a constrained line reliability is considered for normal distributions only. None of the studies in Tables 2.1 and 2.2 present a solution approach for a sampling-based model.

2.2.2 Lower bounds for stochastic task times

This section reviews lower bounds for the stochastic line balancing problem. Two bounds are derived for the stochastic problem with a constraint on the *station-based reliability*. They are based on a transformation of lower bounds on the number of stations for the deterministic problem. Urban and Chiang (2006) show the transformation from the lower bound LB^1 for the deterministic SALBP-1 (Scholl and

Becker, 2006) to the stochastic problem under the assumption of normally distributed task times. This bound has been applied under the same assumptions in a wide range of methods (Chiang and Urban, 2006; Ağpak and Gökçen, 2007; Özcan, 2010; Bagher et al., 2011; Chiang et al., 2016; Özcan, 2018; Zhang et al., 2018; Aydoğan et al., 2019; Fathi et al., 2019). Betts and Mahmoud (1989) show the transformation of bounds LB^1 and LB^2 of Scholl and Becker (2006) for symmetrically distributed task times using Chebyshev's inequality.

For the stochastic line balancing problem with a *line reliability*, only two papers consider lower bounds. Liu et al. (2005) minimize the cycle time and apply a bound on it, which is valid if all task times have the same coefficient of variation. Chiang et al. (2016) develop a bound on the number of mated stations in a two-sided line based on LB^1 (Scholl and Becker, 2006) under the assumption of normally distributed task times. Every feasible solution of the stochastic line balancing problem with a constraint on the line reliability is also a solution of the problem with a station-based reliability. Therefore, the three bounds related to a station-based reliability mentioned above are additional feasible lower bounds on the number of stations for the problem with a line reliability.

To summarize, out of the seven lower bounds on the number of stations for the deterministic problem from Scholl and Becker (2006), only two bounds have been applied to line balancing with stochastic task times. They are valid under specific assumptions regarding the distribution of the task times. Therefore, there is a need to study the impact of other bounds for the stochastic problem with generally distributed task times.

2.3 Chance-constraint and sampling-based model formulation

In this section, we first present the chance-constraint on the line reliability and then formulate a sampling-based model.

Let t_i be the stochastic task time of task i. There are I tasks and M stations. The cycle time c is given and the set of precedence relations P has to be considered in the assignment. A constraint on the line reliability R (with $0 \leq R \leq 1$) of finishing a work piece within the cycle time at each station of the entire line has to be fulfilled. The chance-constrained formulation uses two types of binary variables. The assignment-variable $X_{i,m}$ is equal to 1 if task i is assigned to station m and

0 otherwise. The station-variable Z_m is equal to 1 if station m is opened. The constraint on the line reliability is given by

$$\prod_{m=1}^{M} \left(\text{Prob} \left\{ \sum_{i=1}^{I} t_i \cdot X_{i,m} \ \leq \ c \cdot Z_m \right\} \right) \geq R \, . \tag{2.1}$$

We approximate the constraint on the line reliability in Eq. (2.1) using a finite number of N samples, where each sample n represents a work piece assembled in the line. The parameter $t_{n,i}$ is the sampled task time for task i and sample n. The sample-variable B_n is equal to 0 if the cycle time c is exceeded for sample n at least at one station. The formulation of the sampling-based model is as follows:

$$\text{min} \quad \sum_{m=1}^{M} Z_m \tag{2.2}$$

$$\text{s.t.} \quad \sum_{i=1}^{I} t_{n,i} \cdot X_{i,m} \leq c \cdot Z_m + (1 - B_n) \cdot \bar{M} \quad m = 1, \ldots, M; \ n = 1, \ldots, N \tag{2.3}$$

$$\frac{\sum_{n=1}^{N} B_n}{N} \geq R \tag{2.4}$$

$$\sum_{m=1}^{M} X_{i,m} = 1 \qquad i = 1, \ldots, I \tag{2.5}$$

$$\sum_{m=1}^{M} m \cdot X_{i,m} \leq \sum_{m=1}^{M} m \cdot X_{j,m} \qquad \forall \, (i,j) \in P \tag{2.6}$$

$$X_{i,m} \in \{0,1\} \qquad i = 1, \ldots, I; \ m = 1, \ldots, M \tag{2.7}$$

$$Z_m \in \{0,1\} \qquad m = 1, \ldots, M \tag{2.8}$$

$$B_n \in \{0,1\} \qquad n = 1, \ldots, N. \tag{2.9}$$

The objective function (2.2) minimizes the number of open stations. Constraint (2.3) ensures that the sum of all task times assigned to station m stays within the cycle time c if the sample-variable B_n is equal to 1. If the cycle time is exceeded for sample n ($B_n = 0$), the constraint is not binding. To guarantee this, we chose $\bar{M} = \max_n \sum_{i=1}^{I} t_{n,i}$. Hence, if the cycle time is exceeded at least at one station, the sample-variable B_n has to equal 0. Constraint (2.4) ensures that

the cycle time is never exceeded for at least a fraction of R samples. Hence, Constraints (2.3) and (2.4) approximate the chance-constraint on the line reliability from Eq. (2.1). Constraint (2.5) ensures that each task is assigned to exactly one station. For two tasks (i, j) with a precedence relation, Constraint (2.6) ensures that task i is not assigned to a station after the station with task j. Constraints (2.7) - (2.9) describe the variable domains.

2.4 General transformation of lower bounds

Tight lower bounds on the number of stations are crucial in the solution of deterministic assembly line balancing problems (Scholl and Becker, 2006; Pape, 2015; Pereira, 2015). This section develops a general transformation of any deterministic lower bound on the number of stations for the deterministic SALBP-1 to a lower bound for the proposed sampling-based model with a constraint on the line reliability ((2.2) - (2.9)) from Section 2.3. This transformation can use any lower bound that has already been developed or might be developed in the future for the deterministic problem.

Let $LB(n)$ be a lower bound on the number of stations for the deterministic SALBP-1 based on the instance with the task times of a single sample $n \in \{1, \ldots, N\}$ in isolation.

Theorem 1. *Let the samples be ordered such that the lower bounds on the number of stations for the deterministic problem for a single sample are non-decreasing, i.e.* $LB(n) \leq LB(n+1)$ *holds. Let* $LB(\bar{n})$ *be the lower bound of sample* $\bar{n}$ *with* $\bar{n} := \lceil R \cdot N \rceil$.

Then, $LB(\bar{n})$ *is a lower bound on the number of stations for the proposed sampling-based model with a constraint on the line reliability (*(2.2) - (2.9)*) with respect to samples* $n \in \{1, \ldots, N\}$ *and the line reliability* R.

Proof. Assume a feasible solution with $LB' < LB(\bar{n})$ stations. Then, samples $\bar{n}, \ldots, N$ can satisfy Equation (2.3) for $B_n = 0$ only because their lower bounds are larger than LB'. Rearranging Constraint (2.4) on the line reliability results in

$$R \cdot N \;\leq\; \sum_{n=1}^{N} B_n \;=\; \sum_{n=1}^{\bar{n}-1} B_n \;\leq\; \bar{n} - 1 \;=\; \lceil R \cdot N \rceil - 1 \;<\; R \cdot N \,. \qquad (2.10)$$

This contradicts the assumption that a feasible solution with $LB' < LB(\bar{n})$ stations exists. □

Using Theorem 1, any lower bound on the number of stations for the deterministic SALBP-1 can be transformed to a lower bound for the sampling-based model proposed in Section 2.3. The transformation requires the calculation of a lower bound for each of the N deterministic problems in isolation.

We apply Theorem 1 to all seven lower bounds on the number of stations proposed by Scholl and Becker (2006). The sampling-based formulations of those lower bounds are summarized in Appendix A.

2.5 Reliability-based branch-and-bound algorithm

We use the general transformation of lower bounds presented in Section 2.4 in a reliability-based branch-and-bound (RB&B) algorithm to solve the sampling-based model proposed in Section 2.3. The algorithm presented in this section has to consider the line reliability directly.

The general idea of the RB&B algorithm is to open stations successively and assign tasks to them. A node is a partial assignment of tasks to stations. The assignment for this subset of tasks fulfills the cycle time constraint (2.3), the line reliability (2.4) and the precedence constraints (2.6). However, Constraint (2.5) is relaxed, as not every task has to be assigned. A leaf of the branching tree is a complete and feasible assignment of tasks to stations. We use a bidirectional branching strategy in which tasks are assigned from the first and last stations of the line simultaneously (Scholl and Klein, 1997). These are referred to as the *forward* and *backward* directions respectively.

In contrast to the deterministic problem, Constraint (2.4) on the line reliability creates a dependence among the stations, as the probabilities of exceeding the cycle time may be different from station to station. Therefore, when constructing a potential station during the RB&B algorithm, a partial assignment of tasks to this station has to consider already constructed stations and potential further assignments to other stations. Similar to Scholl and Klein (1997), task i is *available* at station m if all preceding tasks have been assigned to a station before m when branching in the forward direction. Likewise, task i is *available* at station m if all following tasks have been assigned to a station following m when branching in the backward direction. The sampled task times and the constraint on the line reliability R have

to be taken into account. To decide whether a task can be added to a station, we call a task R-*assignable* to station m if it is available and assigning it to station m does not cause the line reliability of the entire line to drop below R (see Constraints (2.3) and (2.4)). This also means that a task can be assigned to a station even if the station loads of some samples exceed the cycle time. The station load of station m is called R-*maximal* if no further task can be added without violating the precedence constraints or the line reliability R.

The main difference to a branch-and-bound algorithm for deterministic task times is that we need to branch on R-maximal and non-R-maximal station loads. This means that a node has to be opened for every subset of tasks for every R-maximal station load, leading to a significant increase in the size of the branching tree. The RB&B algorithm reduces the size of the branching tree by using the following new fathoming strategies that incorporate the line reliability directly. Local lower bounds based on Theorem 1 fathom nodes that cannot lead to an improvement of the incumbent. A new dominance rule checks whether the branching of a node with a non-R-maximal station load leads to an improvement in the line reliability. A new logical test checks whether a node can lead to a feasible solution for the line reliability R.

Section 2.5.1 describes the adaptation of the branching strategy of the RB&B algorithm to the sampling-based model in detail. In Section 2.5.2, we present different strategies to fathom nodes based on the transformed lower bounds presented in Section 2.4, the new dominance rule and the new logical test. The updating of both the upper and lower bound during the algorithm is described in Section 2.5.3.

2.5.1 Bidirectional branching

We perform a bidirectional branch-and-bound procedure with a station-oriented branching strategy, which is commonly used in the deterministic setting (Scholl and Becker, 2006). For each child node in the station-oriented branching strategy, the next station is opened and sets of tasks are assigned to it. Branching in the *forward* direction refers to starting at the first station and using the original precedence graph. Branching in the *backward* direction refers to starting at the last station and using the reversed precedence graph. We use a bidirectional search (Scholl and Klein, 1997), where the branching tree is developed in the forward and backward directions simultaneously, eventually connecting both directions. Nodes k and k' can be connected if the assigned tasks of node k and the assigned tasks of node k' are disjunct and form the set of all tasks.

Two empty root nodes are created: one for the forward direction and one for the backward direction. The forward node starts at the first station and the backward node starts at the last station. We use a greedy start heuristic to find an initial upper bound (UB); see Appendix B. The number of the last station is set equal to this UB.

In the deterministic problem, only maximal station loads have to be considered. However, for the stochastic problem, the line reliability is measured for the whole line, which creates a dependence among the stations. Therefore, both R-maximal and non-R-maximal station loads have to be considered. If only R-maximal station loads were to be built, then the algorithm would "use up" the entire line reliability on early stations, running into infeasibility later and, more importantly, potentially fathoming optimal solutions. Therefore, each station is loaded with all feasible combinations of tasks with respect to Constraints (2.3), (2.4) and (2.6) and the relaxation of Constraint (2.5). The need to consider non-R-maximal station loads increases the number of nodes significantly.

The forward and backward nodes are sorted separately to find the most promising node in each direction. For each direction, the algorithm chooses the nodes that have the lowest local lower bound (see Section 2.5.2 for the calculation). Out of these, choose those that use the most stations (depth first). The most promising nodes in the forward and backward direction are compared using a modified version of the "average task time" T_k (Scholl and Klein, 1997, p. 327). A large value of T_k implies that tasks with large average task times are available, that there are not many available tasks or that tasks can only be assigned to few stations. In defining T_k, we account for the sampling-based formulation as follows:

$$T_k := \left(\sum_{i \in AV_k} \frac{t_i}{\frac{1}{N} \cdot \sum_{n=1}^{N} L_i(UB, n) - E_i(n) + 1} \right) \cdot \frac{1}{|AV_k|}$$

where AV_k is the set of available tasks for node k and t_i is the average task time of task i. We define E_i and L_i as the earliest and latest stations that task i can be assigned to (see the formal definition in Appendix A.3). Instead of using the set of R-assignable tasks, we employ the set of available tasks, because the available tasks can be calculated independently of the sample size. Scholl and Klein (1997) use the lower bound as an estimate for the required number of stations. Since we use a greedy heuristic to determine an initial solution, we use the incumbent (UB) as an estimator. Whenever a new incumbent is found, T_k is updated for all open nodes. The T_k value is calculated for the most promising nodes in the forward and backward direction. If $T^f > T^b$ or if $T^f = T^b$ and $|AV^f| \leq |AV^b|$, a forward

step is performed. Otherwise, a backward step is performed. Numerical pretests have shown the efficiency of this way of determining the most promising node by comparing it with other criteria such as the line reliability or the average station load.

In contrast to Scholl and Klein (1997), we do not only open nodes with the same local lower bound. Instead, all feasible combinations are opened as child nodes in descending order of their number of tasks. Nodes with the same number of tasks are opened in no particular order. This allows the dominance rule described in Section 2.5.2 to fathom non-R-maximal station loads that do not lead to an improvement of the line reliability. Algorithm 1 summarizes this branching strategy.

Algorithm 1 Branching of a node

$AS \leftarrow$ Set of all R-maximal assignments
$AS^{all} \leftarrow$ Set of all subsets of all sets in AS
Sort set AS^{all} by descending cardinality of set members
for $assignment$ in AS^{all} **do**
  Create child node with $assignment$
  Check if child node can be fathomed
end for

2.5.2 Fathoming Strategies

The RB&B algorithm uses different strategies to fathom nodes. Local lower bounds are used to show that a node cannot lead to a better solution than the current upper bound. A new dominance rule checks if the branching of a node with a non-R-maximal station load leads to an improvement in the line reliability. A new logical test is used to show that a branch following a node cannot lead to a feasible solution.

Local lower bounds

After the bounds are transformed to the sampling-based model (Section 2.4 and Appendix A), the global lower bound (LB) is equal to the largest of all lower bounds: $LB = max\{\ LB^1, LB^2, LB^3, LB^4, LB^5, LB^6, LB^7\ \}$. In addition to the global lower bound, local lower bounds (LLB) on the number of additionally required stations are used in the RB&B algorithm to determine the most promising node and to fathom nodes. For each node, the local lower bound on the additionally required stations is calculated based on the remaining unassigned tasks of the node. Since we

follow a station-oriented branching strategy, the remaining idle time of the last station to which tasks are currently assigned has no impact on the local lower bound. The local lower bounds LLB_k^1 to LLB_k^7 can be derived according to Section 2.4. The LLB_k of node k is the maximum of all of these bounds. Therefore, node k can be fathomed if the used stations and LLB_k of node k are not smaller than the current upper bound:

$$\max\{LLB_k^1, LLB_k^2, LLB_k^3, LLB_k^4, LLB_k^5, LLB_k^6, LLB_k^7\}$$
$$+ \text{ used stations of node k } \geq UB$$

Dominance Rule

Different dominance rules have been proposed for deterministic assembly line balancing. The Labeling Dominance Rule fathoms a node, if an assignment of a set of tasks does not need fewer stations than an earlier assignment of the same set of tasks (Scholl and Klein, 1997). Similarly, the Tree Dominance Rule fathoms equivalent partial solutions that differ only in their sequence of assignment (Scholl and Klein, 1999). In the problem with a minimal line reliability, in addition to the number of stations, the resulting line reliability is also relevant. Therefore, we propose a new dominance rule to check whether a node with a non-R-maximal station load leads to an improvement in the line reliability. Nodes with a non-R-maximal station load may be fathomed if assigning only a subset of tasks to the same or more stations does not lead to an increase in the line reliability. Hence, node k can be fathomed if there is a node k' such that:

1. The assigned tasks of node k are a subset of the assigned tasks of node k' **and**
2. k requires an equal number or more stations than k' **and**
3. Line reliability of node $k \leq$ line reliability of node k'

This means that an assignment of a subset of tasks that does not require fewer stations and does not lead to a higher line reliability cannot lead to a better solution. The line reliability of node k is computed by $LR^k = \left|\mathcal{C}_1^k \cap \mathcal{C}_2^k \cap \ldots \cap \mathcal{C}_M^k\right| \ / \ N$ where $\mathcal{C}_m^k = \left\{ n \;\middle|\; \sum_{i=1}^{I} t_{n,i} \cdot X_{i,m}^k \leq c \right\}$ is the set of all samples for which the assignment of node k does not exceed the cycle time c.

The order of opening new nodes in the branching strategy is important for the efficiency of the dominance rule. Since we open nodes in descending order of the assigned number of tasks to the child nodes, later nodes often contain subsets of tasks of earlier nodes. Therefore, these nodes can be checked against the already

opened nodes and potentially fathomed, thereby reducing the number of non-R-maximal station loads. This dominance rule includes the Labeling Dominance Rule and the Tree Dominance Rule known from the deterministic problem.

Logical Test

We also introduce a new logical test to assess whether the branch following a node can lead to a feasible solution of the sampling-based model. Independent of the assignment, sample n is guaranteed to be incomplete if its largest task time exceeds the cycle time. Let $G = \{\ n \mid \max_i \{t_{n,i}\} > c\}$ be the set of all samples that are guaranteed to be incomplete independent of the assignment.

Consider the assignment of a certain node k with the resulting sampling-based line reliability LR_k and a specific sample n. If task i has not yet been assigned in node k and the task time of i exceeds the cycle time c for sample n, it is guaranteed that sample n cannot be completed. Therefore, node k can be fathomed if the line reliability of node k minus the percentage of additionally guaranteed incomplete samples is less than R.

For any (partial) assignment in node k, the set of samples for which the cycle time is violated can be calculated by

$$IC_k = \left\{ n \;\middle|\; \max_m \left\{ \sum_{i=1}^{I} t_{n,i} \cdot X_{i,m}^k \right\} > c \right\} . \tag{2.11}$$

Node k can be fathomed, if its line reliability minus the percentage of guaranteed incomplete samples not in IC_k is less than R, i.e. if $1 - \mid IC_k - G \backslash IC_k \mid / N < R$ holds.

2.5.3 Updating of bounds

The algorithm terminates if there are no more open nodes or if the upper bound is equal to the global lower bound LB. Whenever a new solution with fewer stations than the current upper bound is found, this bound is updated. Whenever a new upper bound is found, the algorithm checks if any of the remaining open nodes can be fathomed using the local lower bounds and the new upper bound. The global lower bound LB can be increased if it is smaller than the local lower bound LLB_k of node k plus the number of used stations of node k for all open nodes. In this case, LB is set to the smallest of these values.

2.6 Numerical study

The numerical study first analyzes the effects of the transformed lower bounds on fathoming nodes within the RB&B algorithm and on the computation times for the RB&B algorithm and CPLEX, respectively. The effects of the transformed lower bounds, the dominance rule and the logical test as fathoming strategies in isolation are analyzed in Section 2.6.1. We study the value of the combination of the transformed lower bounds for both the RB&B algorithm and the solver CPLEX in Section 2.6.2. Section 2.6.3 analyzes the impact of the line reliability constraint on the optimal solution. We analyze the sensitivity of the optimal solution to the distribution of the task times in Section 2.6.4.

We use descriptive sampling with a sample size of $N = 10,000$ to draw the task times (Saliby, 1990). Many distributions have infinite support and it is common to truncate the distribution at zero to avoid negative task times (Sarin et al., 1999) or to analyze the convoluted cumulative density function at the point of the desired minimal probability R. For the instances with normally distributed task times, all sampled negative task times are replaced by zero. We conduct numerical pretests using the 81 problem instances described in Section 2.6.1 and sample sizes of $N \in \{1,000;\ 2,500;\ 5,000;\ 7,500;\ 10,000;\ 15,000;\ 20,000\}$. The 81 problem instances are solved for those seven different sample sizes with 20 independent runs each (different random seeds). Setting $N \geq 10,000$, no deviation was observed in the number of stations for any of the analyzed problems. In addition, the absolute deviation between sampling-based and analytical line reliability ranges only between 0 and 0.0288. Therefore, we use a sample size of $N = 10,000$ for the numerical studies in Section 2.6.

All calculations are performed on an Intel Core i7-7700K with 3.60GHz and 64GB of memory. The model is implemented in GAMS 25.1.1 and solved using CPLEX 12.8. The RB&B algorithm is implemented in Python 3.6 using the Spyder environment.

2.6.1 Fathoming strategies in isolation

This section analyzes the global lower bounds and the fathoming strategies in isolation. The performance of global lower bounds in isolation is analyzed first. Then, we apply all fathoming strategies for the RB&B algorithm in isolation: the lower

bounds for each transformed bound according to Theorem 1, the dominance rule and the logical test.

There are different benchmark sets for deterministic assembly line balancing, e.g. Otto et al. (2013). In the stochastic assembly line balancing literature (see Section 2.2), it is common to use the deterministic benchmark set of Scholl (1993). We analyze all problem instances of this set with up to 21 tasks. These are six different precedence graphs (Mertens, Bowman, Jaeschke, Mansoor, Jackson, Mitchell), each with different specified cycle times, resulting in 27 deterministic problem instances. For each task i we assume normally distributed task times with mean $\mu_i = \tilde{t}_i$ from the deterministic benchmark set and standard deviation $\sigma_i = cv \cdot \tilde{t}_i$. We apply coefficients of variation $cv \in \{0.1,\ 0.3,\ 0.5\}$ since both Liu et al. (2005) and Tang et al. (2017) use coefficients of variation up to 0.5. The distributions of the task times are assumed to be independent. The desired line reliability is set to $R = 0.95$. The combination of the 27 deterministic problem instances with three different coefficients of variation leads to 81 problem instances in total. We limit the solution to $10,000$ nodes and $10,000$ seconds, whichever is reached first.

In the deterministic literature, lower bounds are analyzed towards their tightness with respect to the optimal solution. Scholl and Becker (2006) state that out of the 7 analyzed bounds, LB^4 provides the tightest bound for the deterministic problem. Pereira (2015) compares the tightness of LB^1 through LB^4 and finds that there is no significant difference for the deterministic problem.

To analyze the performance of the global lower bounds for the sampling-based model, Table 2.3 states the transformed global lower bound LB for each bound in isolation. The last line gives the number of times each bound has the highest value. We observe that LB^7 has the best bound in 38 of the 43 instances. This is closely followed by LB^4 and LB^5 with 37 each and LB^1 with 33. LB^2 and LB^3 only have the best lower bound in 7 and 6 of the instances respectively, but are never the unique best lower bound. In 7 instances, the lower bound provided by LB^2 is 0 stations, even though all other bound arguments provide a positive number of stations. Therefore, in contrast to Pereira (2015), we find a significant difference for the first 4 bounds for the analyzed instances. Similar to Scholl and Becker (2006), we find that LB^4 has a good performance. In addition, LB^1, LB^5 and LB^7 also perform well.

The remaining part of Section 6.1 is devoted to the performance of both the global and local lower bounds within the RB&B algorithm. Table 2.4 shows the required number of nodes of the RB&B algorithm if each fathoming strategy is applied in

	cv	c	$\mathbf{LB^1}$	$\mathbf{LB^2}$	$\mathbf{LB^3}$	$\mathbf{LB^4}$	$\mathbf{LB^5}$	$\mathbf{LB^6}$	$\mathbf{LB^7}$
Mertens	0.1	7	5	5	5	5	5	**6**	5
		8	4	**5**	**5**	4	4	**5**	4
		10	**4**	**4**	3	**4**	**4**	**4**	**4**
		15	**3**	0	2	**3**	**3**	2	**3**
		18	**2**	0	1	**2**	**2**	**2**	**2**
	0.3	10	**4**	**4**	**4**	**4**	**4**	**4**	**4**
		15	**3**	1	2	**3**	**3**	**3**	**3**
		18	**2**	1	**2**	**2**	**2**	**2**	**2**
	0.5	15	**3**	2	**3**	**3**	**3**	**3**	**3**
		18	**3**	1	2	**3**	**3**	2	**3**
Bowman	0.1	20	4	**5**	4	**5**	**5**	**5**	**5**
Jaeschke	0.1	7	6	7	7	7	7	**8**	**8**
		8	5	**7**	6	6	6	**7**	6
		10	4	4	4	**5**	**5**	4	**5**
		18	**3**	0	1	**3**	**3**	2	**3**
	0.3	10	5	5	5	**6**	**6**	5	**6**
		18	**3**	1	2	**3**	**3**	**3**	**3**
	0.5	18	**3**	1	2	**3**	**3**	**3**	**3**
Jackson	0.1	9	6	**7**	6	6	6	**7**	6
		10	5	**6**	5	5	5	**6**	5
		13	**4**	2	**4**	**4**	**4**	**4**	**4**
		14	**4**	1	3	**4**	**4**	**4**	**4**
		21	**3**	0	1	**3**	**3**	2	**3**
	0.3	13	**5**	4	4	**5**	**5**	4	**5**
		14	**4**	3	**4**	**4**	**4**	**4**	**4**
		21	**3**	1	2	**3**	**3**	**3**	**3**
	0.5	14	**5**	4	4	**5**	**5**	4	**5**
		21	**3**	1	2	**3**	**3**	**3**	**3**
Mansoor	0.1	62	**4**	3	3	**4**	**4**	3	**4**
		94	**3**	1	2	**3**	**3**	2	**3**
	0.3	94	**3**	2	2	**3**	**3**	2	**3**
	0.5	94	**3**	2	2	**3**	**3**	**3**	**3**
Mitchell	0.1	21	**6**	2	3	**6**	**6**	4	**6**
		26	**5**	1	2	**5**	**5**	3	**5**
		35	**4**	0	1	**4**	**4**	3	**4**
		39	**3**	0	1	**3**	**3**	2	**3**
	0.3	21	**6**	3	4	**6**	**6**	4	**6**
		26	**5**	2	3	**5**	**5**	3	**5**
		35	**4**	1	1	**4**	**4**	3	**4**
		39	**4**	0	1	**4**	**4**	2	**4**
	0.5	26	5	3	3	**6**	**6**	4	**6**
		35	**4**	1	2	**4**	**4**	3	**4**
		39	**4**	1	2	**4**	**4**	3	**4**
Count best bound:			**33**	**7**	**6**	**37**	**37**	**21**	**38**

Table 2.3: Analysis of global lower bound LB for different bounds in isolation

isolation. The first three columns describe the precedence graph, coefficient of variation cv and cycle time c, respectively. This is followed by a column for each strategy in isolation: no fathoming strategy (None), only the global and corresponding local lower bound 1-7 (LB & LLB), only the dominance rule (DOM) or only the logical test (LOG) is applied. Bold numbers indicate the least number of nodes for each instance.

Out of the 81 problem instances, 38 are infeasible. Infeasibility was proven immediately, therefore we do not report these instances in Table 2.4. For 36 of the 43 feasible instances, the greedy start heuristic (see Appendix B) is able to find the optimal solution. If the optimal number of stations is equal to the global lower bound, only 1 node is reported. Hence in these cases, the transformed lower bounds prove the optimality of the greedy solution with respect to the sampled data. Table 2.4 shows that the RB&B algorithm requires significantly fewer nodes with each lower bound in isolation than the RB&B algorithm without lower bounds. Using LB^7 requires the least nodes among all isolated strategies for 30 instances. The logical test requires the highest number of nodes in all instances and does not reduce the number of nodes in most instances compared to not using any strategy.

Table 2.5 considers the required computation time and has the same structure as Table 2.4. It can be observed that using lower bounds in isolation also reduces the required solution times significantly. Using LB^1 requires the least time in 29 instances. The dominance rule requires the least time in 6 instances. Interestingly, even though LB^7 requires the least nodes in many cases, it is never the fastest strategy if the start heuristic did not find the optimal solution. This is due to the higher computational effort needed to derive this bound, as the required time to compute LLB^7 in these cases is between 64% and 90% of the total computation time.

To summarize, we find that LB^1, LB^4, LB^5 and LB^7 provide the best global lower bounds. During the RB&B algorithm, the lower bounds in isolation and the dominance rule in isolation significantly reduce the required number of nodes or the computation time of the RB&B algorithm. The logical test in isolation does not show significant positive effects.

				LB & LLB								
	cv	**c**	**None**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**DOM**	**LOG**
Mertens	0.1	7	688	15	75	13	15	56	**1**	10	56	653
		8	941	3	**1**	**1**	3	17	**1**	3	78	941
		10	1,546	28	55	67	25	69	**20**	25	54	1,546
		15	3,823	**1**	386	6	**1**	**1**	3	**1**	48	3,823
		18	3,595	**1**	39	13	**1**	**1**	**1**	**1**	33	3,595
	0.3	10	941	6	6	6	6	19	**3**	6	106	941
		15	3,363	**1**	61	3	**1**	**1**	**1**	**1**	151	3,363
		18	4,051	**3**	232	25	**3**	10	**3**	**3**	83	4,051
	0.5	15	1,882	19	66	37	**17**	42	31	35	185	1,879
		18	3,511	1	19	6	**1**	**1**	3	**1**	223	3,511
Bowman	0.1	20	761	3	**1**	4	**1**	**1**	**1**	**1**	32	757
Jaeschke	0.1	7	590	53	37	32	28	29	**1**	10	40	560
		8	1,012	31	**1**	19	11	14	**1**	11	54	1,012
		10	2,140	27	130	43	24	25	28	**19**	35	2,140
		18	9,167	**1**	341	100	**1**	**1**	3	**1**	30	9,167
	0.3	10	1,012	43	39	50	17	19	26	**15**	63	1,012
		18	7,920	**1**	73	9	**1**	**1**	**1**	**1**	69	7,920
	0.5	18	7,336	7	64	19	**4**	**4**	11	**4**	157	7,336
Jackson	0.1	9	$>$ 10,000	1,090	316	2,702	858	891	145	**130**	145	$>$ 10,000
		10	$>$ 10,000	210	1,000	915	165	167	195	**155**	231	$>$ 10,000
		13	$>$ 10,000	11	4,832	66	**10**	**10**	26	**10**	91	$>$ 10,000
		14	$>$ 10,000	**1**	1,403	5	**1**	**1**	**1**	**1**	76	$>$ 10,000
		21	$>$ 10,000	**1**	3,085	639	**1**	**1**	3	**1**	63	$>$ 10,000
	0.3	13	$>$ 10,000	49	775	368	48	49	49	**41**	622	$>$ 10,000
		14	$>$ 10,000	168	5,021	1,766	158	158	728	**151**	499	$>$ 10,000
		21	$>$ 10,000	**12**	555	47	**12**	**12**	15	**12**	192	$>$ 10,000
	0.5	14	$>$ 10,000	1,768	5,403	5,058	1,663	1,411	2,131	1,278	**249**	$>$ 10,000
		21	$>$ 10,000	81	6,357	264	81	**26**	90	**26**	417	$>$ 10,000
Mansoor	0.1	62	$>$ 10,000	**1**	37	37	**1**	**1**	14	**1**	68	$>$ 10,000
		94	$>$ 10,000	**1**	1,045	42	**1**	**1**	3	**1**	30	$>$ 10,000
	0.3	94	$>$ 10,000	**1**	18	5	**1**	**1**	3	**1**	190	$>$ 10,000
	0.5	94	$>$ 10,000	13	47	47	12	49	18	**12**	169	$>$ 10,000
Mitchell	0.1	21	$>$ 10,000	**1**	$>$ 10,000	$>$ 4,001	**1**	**1**	329	**1**	571	$>$ 10,000
		26	$>$ 10,000	**1**	$>$ 10,000	$>$ 2,607	**1**	**1**	285	**1**	259	$>$ 10,000
		35	$>$ 10,000	**1**	$>$ 10,000	$>$ 10,000	**1**	**1**	64	**1**	278	$>$ 10,000
		39	$>$ 10,000	**1**	$>$ 10,000	$>$ 3,114	**1**	**1**	3	**1**	245	$>$ 10,000
	0.3	21	$>$ 10,000	2,180	$>$ 10,000	$>$ 4,072	$>$ 873	$>$ 1,286	$>$ 1,490	$>$ 1,012	**1,402**	$>$ 10,000
		26	$>$ 10,000	27	$>$ 10,000	$>$ 10,000	15	$>$ 120	$>$ 1,073	**9**	1,646	$>$ 10,000
		35	$>$ 10,000	**592**	$>$ 10,000	$>$ 10,000	$>$ 228	$>$ 175	$>$ 1,462	$>$ 162	934	$>$ 10,000
		39	$>$ 10,000	**1**	$>$ 10,000	$>$ 8,608	**1**	**1**	68	**1**	681	$>$ 10,000
	0.5	26	$>$ 10,000	$>$ 2,873	$>$ 10,000	$>$ 10,000	$>$ 525	$>$ 738	$>$ 5,372	$>$ 1,034	**2,346**	$>$ 10,000
		35	$>$ 10,000	400	$>$ 1,993	$>$ 10,000	**247**	483	$>$ 9,388	$>$ 213	2,004	$>$ 10,000
		39	$>$ 10,000	**533**	$>$ 10,000	$>$ 10,000	$>$ 194	$>$ 161	$>$ 1,699	$>$ 145	2,514	$>$ 10,000
Count best strategy:			0	19	3	1	23	21	12	30	3	0

Table 2.4: Number of nodes for a single fathoming strategy in isolation

				LB & LLB								
	cv	**c**	**None**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**DOM**	**LOG**
Mertens	0.1	7	103	11	28	11	36	97	**2**	29	19	104
		8	115	3	**1**	**1**	9	45	**1**	10	21	136
		10	188	**19**	25	32	65	148	21	77	20	223
		15	424	**1**	117	11	**1**	**1**	7	**1**	19	494
		18	423	**1**	40	15	**1**	**1**	**1**	**1**	16	494
	0.3	10	126	6	6	7	19	53	**5**	20	28	149
		15	397	**1**	36	6	**1**	**1**	**1**	**1**	40	464
		18	476	**6**	81	31	19	64	8	19	27	557
	0.5	15	247	**7**	23	22	20	48	12	58	48	293
		18	423	**1**	26	11	**1**	**1**	7	**1**	56	496
Bowman	0.1	20	110	3	**1**	4	**1**	**1**	**1**	**1**	10	134
Jaeschke	0.1	7	111	22	19	19	62	64	**3**	35	15	123
		8	163	17	**1**	16	38	43	**1**	43	19	198
		10	326	18	52	25	72	80	27	61	**13**	389
		18	1,292	**1**	141	77	**1**	**1**	8	**1**	16	1,476
	0.3	10	175	**20**	21	27	55	58	23	54	23	211
		18	1,162	**2**	60	14	**2**	**2**	**2**	**2**	29	1,345
	0.5	18	1,096	**9**	62	25	20	18	21	18	54	1,259
Jackson	0.1	9	$>$ 1,432	485	359	1,149	1,430	2,124	256	959	**105**	$>$ 1,611
		10	$>$ 1,328	255	424	503	892	1,305	303	1,176	**147**	$>$ 1,462
		13	$>$ 1,309	**23**	2,232	185	87	104	106	102	81	$>$ 1,463
		14	$>$ 1,292	**4**	1,244	17	**4**	**4**	**4**	**4**	85	$>$ 1,429
		21	$>$ 1,355	2	1,987	925	2	2	24	**1**	84	$>$ 1,549
	0.3	13	$>$ 1,525	**91**	678	436	359	468	147	381	351	$>$ 1,679
		14	$>$ 1,398	**284**	1,975	1,075	1,049	1,472	752	1,283	325	$>$ 1,550
		21	$>$ 1,430	**14**	710	134	40	49	18	45	213	$>$ 1,623
	0.5	14	$>$ 3,036	1,264	2,722	2,692	3,243	4,250	1,591	3,822	**182**	$>$ 1,870
		21	$>$ 1,527	**92**	1,725	280	342	103	135	94	381	$>$ 1,666
Mansoor	0.1	62	$>$ 1,333	**2**	165	182	**2**	**2**	98	**2**	87	$>$ 1,476
		94	$>$ 1,461	**2**	896	256	**2**	**2**	26	**2**	67	$>$ 1,689
	0.3	94	$>$ 1,532	**2**	88	23	**2**	**2**	20	**2**	238	$>$ 1,752
	0.5	94	$>$ 1,613	**48**	211	233	158	568	126	187	188	$>$ 1,642
Mitchell	0.1	21	$>$ 3,787	**7**	$>$ 4,384	$>$ 10,000	**7**	**7**	8,055	**7**	1,883	$>$ 4,571
		26	$>$ 2,416	**6**	$>$ 5,994	$>$ 10,000	**6**	**6**	6,085	**6**	1,106	$>$ 2,761
		35	$>$ 2,693	**6**	$>$ 2,969	$>$ 2,903	**6**	**6**	3,496	**6**	2,122	$>$ 3,194
		39	$>$ 3,099	**6**	$>$ 3,389	$>$ 10,000	**6**	**6**	125	7	1,607	$>$ 3,659
	0.3	21	$>$ 6,870	4,362	$>$ 5,803	$>$ 10,000	$>$ 10,000	$>$ 10,000	$>$ 10,000	$>$ 10,000	**4,324**	$>$ 8,532
		26	$>$ 5,053	**192**	$>$ 6,763	$>$ 4,566	1,431	$>$ 10,000	$>$ 10,000	886	5,705	$>$ 5,926
		35	$>$ 3,019	**4,156**	$>$ 3,915	$>$ 3,335	$>$ 10,000	$>$ 10,000	$>$ 10,000	$>$ 10,000	4,384	$>$ 3,394
		39	$>$ 3,115	7	$>$ 3,285	$>$ 10,000	7	7	3,624	**6**	3,820	$>$ 3,584
	0.5	26	$>$ 7,300	$>$ 10,000	$>$ 7,638	$>$ 8,301	$>$ 10,000	$>$ 10,000	$>$ 10,000	$>$ 10,000	**7,325**	$>$ 9,497
		35	$>$ 3,620	**969**	$>$ 10,000	$>$ 5,432	2,731	5,413	$>$ 10,000	$>$ 10,000	7,766	$>$ 4,077
		39	$>$ 3,888	**4,195**	$>$ 5,241	$>$ 5,742	$>$ 10,000	$>$ 10,000	$>$ 10,000	$>$ 10,000	9,423	$>$ 4,364
Count best strategy:			0	29	3	1	15	15	10	16	6	0

Table 2.5: Computation time in seconds for a single fathoming strategy in isolation

2.6.2 Combination of lower bounds

This section analyzes the value of applying the transformed lower bounds in combination as a fathoming strategy within the RB&B algorithm and providing them to the solver CPLEX.

Section 2.6.1 demonstrated that some lower bounds significantly reduce the required number of nodes while others reduce the computation time. In numerical pretests we compared different combinations of fathoming strategies. For example, the logical test is the worst performing strategy in isolation. However, using all strategies in combination with and without the logical test shows that the logical test does not require significant additional computation time in most cases, but decreases the required computation time significantly in some cases. As another example, when using all fathoming strategies except for LB^2 and LB^3, the global lower bound LB was not affected. However, excluding them as a local lower bound requires slightly higher computation time in total. Therefore, we use all fathoming strategies, which is in line with the suggestion for the deterministic line balancing problem (Scholl and Becker, 2006). The order they are applied is based on the performance in isolation. LB^1 and the dominance rule performed well regarding the computation time and LB^5 and LB^7 require the calculation of heads and tails made during LB^4. Therefore, when using all fathoming strategies, we apply the strategies in the following order: LB^1, dominance rule, LB^4, LB^2, LB^3, LB^5, LB^6, LB^7 and the logical test.

We consider a desired line reliability of $R = 0.95$. The first three columns of Table 2.6 describe the same problem instances as those in Section 2.6.1. Columns 4-6 report the global lower bound LB (using LB^1 through LB^7), the optimal number of stations ($\sum Z_m$) and the difference between the optimal number of stations and the global lower bound (Δ). In 22 out of 43 feasible problem instances, the global lower bound LB is equal to the optimal number of stations $\sum Z_m$. The difference is only one station in 17 instances and the maximal difference is three stations, which occurs only once. Therefore, the global lower bounds based on the general transformation of Theorem 1 are tight in most of the analyzed benchmark problems.

To show the value of the combination of the seven lower bounds, Table 2.6 compares the computation time (in seconds) of the RB&B algorithm using all fathoming strategies together (column **with LB**) with the performance of the RB&B algorithm using only the dominance rule and the logical test as fathoming strategies (column **w/o LB**). The value in parentheses behind the computation time states the number of opened nodes. The RB&B algorithm with lower bounds requires significantly

shorter computation times than the RB&B algorithm without lower bounds in most instances. The difference is especially large for the Mitchell instances. The RB&B algorithm using lower bounds requires significantly less nodes than the RB&B algorithm without the lower bounds.

Furthermore, we analyze the value of the lower bounds by providing CPLEX with the global lower bound. Similar to the RB&B algorithm, CPLEX was able to prove infeasibility within seconds. For the feasible instances in Table 2.6, the last two columns report the computation times of CPLEX provided with the global lower bound (column **with LB**) and without the new lower bound (column **w/o LB**). If the optimal solution could not be found or proven within $10,000$ seconds, the current lower bound and upper bound are reported ([*lower*, *upper*]). If this upper bound is equal to the optimal solution but without proof of optimality, the interval is marked with an asterisk, i.e., [*lower*, *upper*]*. We report "-" if no feasible solution is found within $10,000$ seconds.

With the global lower bound, CPLEX is able to solve all instances, except for the Mitchell instances, within $10,000$ seconds. For these instances, the optimality gap is large, but a feasible solution can be found. Without the lower bound, CPLEX fails to prove optimality in many instances. For some instances, no feasible solution can be found within $10,000$ seconds, even though there is one. For the RB&B algorithm, the lower bounds significantly improve the computation time. However, comparing both approaches with lower bounds reveals that the RB&B algorithm is much faster than CPLEX.

As shown in Table 2.6, the proposed general transformation provides tight lower bounds on the number of stations. Figure 2.1 compares them to the lower bounds proposed by Urban and Chiang (2006) and Betts and Mahmoud (1989) (see Section 2.2.2). For Betts and Mahmoud (1989), we report the maximum of LB^1 and LB^2. In all 81 analyzed instances, the lower bound obtained from the proposed general transformation is better than or equal to the lower bounds from the literature. The analysis of further instances from the deterministic benchmark sets of Scholl (1993) and Otto et al. (2013) reveals that the transformed lower bounds are tight, see Appendix C.

To summarize, the proposed transformation of lower bounds LB^1 to LB^7 based on Theorem 1 provides tight lower bounds on the number of stations. They are often better than the bounds reported in the literature. Using these lower bounds reduces the computation times substantially for both the RB&B algorithm and for the solution with the solver CPLEX. Numerical pretests have shown that the sampling-

	cv	c	LB	$\sum \mathbf{Z_m}$	Δ	RB&B with LB	RB&B w/o LB	CPLEX with LB	CPLEX w/o LB
Mertens	0.1	7	6	6	0	2 (1)	19 (52)	2 (0)	200 (0)
		8	5	5	0	1 (1)	24 (78)	2 (0)	5,828 (398)
		10	4	5	1	43 (17)	22 (54)	214 (19)	[2.8, 5]* (2,330)
		15	3	3	0	1 (1)	20 (48)	2 (0)	4,453 (124)
		18	2	2	0	1 (1)	16 (33)	3 (0)	1,414 (12)
	0.3	7	7	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		8	6	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		10	4	5	1	9 (3)	31 (106)	578 (13)	2,409 (99)
		15	3	3	0	1 (1)	44 (151)	8 (0)	512 (15)
		18	2	3	1	6 (3)	29 (83)	158 (697)	158 (697)
	0.5	7	7	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		8	7	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		10	6	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		15	3	4	1	29 (14)	53 (184)	6,718 (632)	1,218 (43)
		18	3	3	0	1 (1)	62 (223)	11 (0)	634 (26)
Bowman	0.1	20	5	5	0	1 (1)	11 (32)	27 (0)	1,337 (50)
	0.3	20	8	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
	0.5	20	8	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
Jaeschke	0.1	7	8	8	0	3 (1)	15 (33)	21 (0)	157 (0)
		8	7	7	0	1 (1)	22 (54)	3 (0)	1,898 (107)
		10	5	6	1	29 (8)	15 (35)	426 (25)	7,443 (288)
		18	3	3	0	1 (1)	17 (30)	5 (0)	7,015 (99)
	0.3	7	9	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		8	9	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		10	6	7	1	69 (15)	27 (63)	501 (26)	1,215 (51)
		18	3	3	0	2 (1)	32 (69)	374 (280)	374 (280)
	0.5	7	9	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		8	9	Inf.		0 (0)	0 (0)	1 (0)	1 (0)
		10	8	Inf.		0 (0)	0 (0)	2 (0)	2 (0)
		18	3	4	1	12 (4)	60 (157)	703 (30)	704 (30)
Jackson	0.1	9	7	8	1	277 (49)	115 (145)	621 (208)	[3.5, 8]* (132)
		10	6	7	1	134 (26)	163 (231)	853 (91)	[2.5, 7]* (829)
		13	4	5	1	46 (10)	88 (91)	2,354 (226)	[1.6, 5]* (70)
		14	4	4	0	4 (1)	91 (76)	144 (0)	[1.5, 4]* (73)
		21	3	3	0	2 (1)	88 (63)	5 (0)	[1.2, 3]* (149)
	0.3	9	8	Inf.		0 (0)	0 (0)	2 (0)	2 (0)
		10	7	Inf.		0 (0)	0 (0)	2 (0)	2 (0)
		13	5	6	1	183 (32)	387 (622)	1,742 (70)	[4, 6]* (372)
		14	4	6	2	321 (63)	351 (499)	7,841 (1,571)	[4, 6]* (843)
		21	3	3	0	69 (21)	224 (192)	46 (0)	46 (0)
	0.5	9	10	Inf.		0 (0)	0 (0)	2 (0)	2 (0)
		10	9	Inf.		0 (0)	0 (0)	2 (0)	2 (0)
		13	6	Inf.		0 (0)	0 (0)	2 (0)	2 (0)
		14	5	8	3	403 (82)	132 (187)	7,125 (409)	[5.4, 8]* (655)
		21	3	4	1	84 (25)	408 (417)	4,372 (337)	4,369 (337)
Mansoor	0.1	62	4	4	0	2 (1)	92 (68)	29 (0)	[1.5, 4]* (152)
		94	3	3	0	2 (1)	69 (30)	30 (0)	[0.9, 3]* (73)
	0.3	62	5	Inf.		0 (0)	0 (0)	2 (0)	2 (0)
		94	3	3	0	1 (1)	247 (190)	213 (15)	214 (15)
	0.5	62	7	Inf.		0 (0)	0 (0)	2 (0)	2 (0)
		94	3	4	1	62 (8)	189 (166)	927 (21)	925 (21)
Mitchell	0.1	21	6	6	0	7 (1)	1,928 (571)	5,799 (185)	- (0)
		26	5	5	0	6 (1)	1,128 (259)	417 (10)	- (0)
		35	4	4	0	6 (1)	2,157 (278)	119 (0)	[0, 11] (0)
		39	3	3	0	6 (1)	1,658 (245)	939 (0)	[0, 11] (0)
	0.3	21	6	8	2	2,027 (106)	3,981 (1,320)	[6, 16] (8)	- (0)
		26	5	6	1	324 (9)	5,805 (1,646)	[5, 6]* (51)	- (0)
		35	4	5	1	3,280 (137)	4,425 (934)	[4, 6] (26)	- (0)
		39	4	4	0	7 (1)	3,875 (681)	286 (0)	[3, 4]* (25)
	0.5	21	9	Inf.		0 (0)	0 (0)	5 (0)	5 (0)
		26	6	8	2	4,655 (226)	7,410 (2,432)	[6, 12] (5)	- (1)
		35	4	5	1	1,160 (84)	7,900 (2,004)	[4, 8] (5)	[4, 8] (6)
		39	4	5	1	2,660 (93)	9,612 (2,514)	[4, 10] (10)	[4, 10] (10)

Table 2.6: Impact of combined lower bounds on computations times (number of nodes)

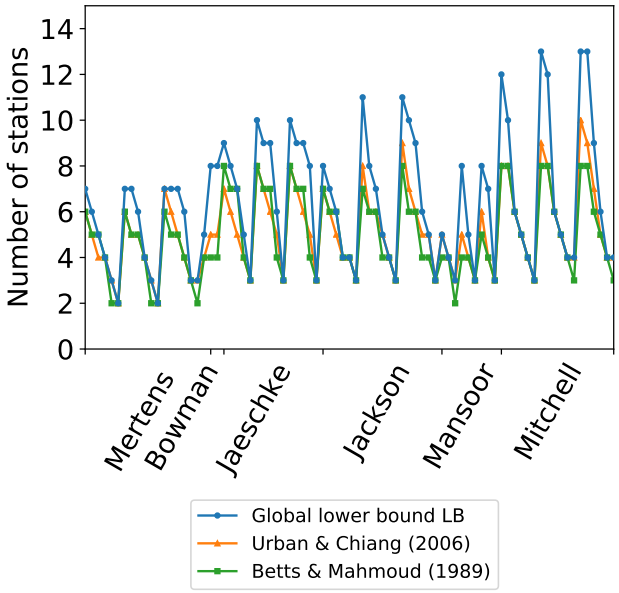


Figure 2.1: Comparison of transformed lower bounds to lower bounds proposed in the literature

based lower bounds also reduce the computation times substantially when solving the model with Gurobi 9.1. The RB&B algorithm with all fathoming strategies requires significantly less computation time than CPLEX.

2.6.3 Sensitivity in line reliability

In this section, we analyze the impact of the constraint on the line reliability R on the solution. Figure 2.2 shows the required computation time, optimal number of stations and global lower bounds on the number of stations for Mertens' and Jackson's precedence graphs for varying R. We assume a cycle time of $c = 10$ for Mertens' precedence graph and $c = 14$ for Jackson's and vary the coefficient of variation. For infeasible line reliabilities, the number of stations is set to zero.

As expected, the optimal number of stations increases with the line reliability. The global lower bound on the number of stations also increases with the line reliability. Whenever the optimal number of stations increases by one, the computation time increases substantially for the higher line reliability. Then, it tends to decrease until the next time the number of stations increases.

The optimal number of stations can be identical for a wide range of the desired line reliability R. For example, for Mertens' precedence graph with a cycle time of $c = 10$ and a coefficient of variation of $cv = 0.1$, the line requires the same number

of stations for $R \in [0.01, 0.23]$ ($R \in [0.24, 0.94]$ and $R \in [0.95, 1.00]$, respectively) but may have different assignments of tasks to stations and hence different realized line reliabilities LR. For example, the optimal solutions for $R = 0.24$ and $R = 0.94$ require four stations. However, the optimal assignment for $R = 0.24$ leads to a line reliability of 50.09%, while the optimal assignment for $R = 0.94$ leads to a line reliability of 94.37%, even though the same number of stations is required. Managers should be aware that even if the desired line reliability is reached with a certain number of stations, it might be possible to reach a substantially higher resulting line reliability with the same number of stations but a different task assignment. Therefore, once the optimal number of stations has been found using the RB&B algorithm, the sampling-based line reliability can be maximized as a secondary objective.

To summarize, the computation times and the structure of the solutions strongly depend on the desired line reliability R. Due to the possibility of multiple optimal solutions, an alternate assignment might exist with the same number of stations and a higher realized line reliability.

2.6.4 Sensitivity in task time distribution

In this section, we analyze the impact of the distribution of the task times on the global lower bounds and on the optimal number of stations. In addition to the results using the normal distribution from Section 2.6.1, we present the results with a gamma distribution. This distribution is able to account for skewed task times, which have been observed in manufacturing. We consider a desired line reliability of $R = 0.95$. A gamma distributed task time for task i with mean t_i and coefficient of variation cv_i has the shape parameters $\alpha_i = \frac{1}{cv_i^2}$ and $\beta_i = \frac{1}{t_i \cdot cv_i^2}$.

Based on the same set of examples, all instances that are infeasible for the normal distribution are also infeasible for the gamma distribution. For the remaining instances, Table 2.7 shows the global lower bound (LB), the optimal number of stations ($\sum Z_m$), the difference between the optimal number of stations and the global lower bound (Δ) and the resulting sampling-based line reliability of the optimal solution (LR) for both distributions.

The global lower bounds for the gamma distribution are always at least as large as those for the normal distribution. For both distributions, the bounds tend to be tighter for a lower coefficient of variation. Three instances with a feasible solution for the normal distribution are infeasible when the gamma distribution is assumed.

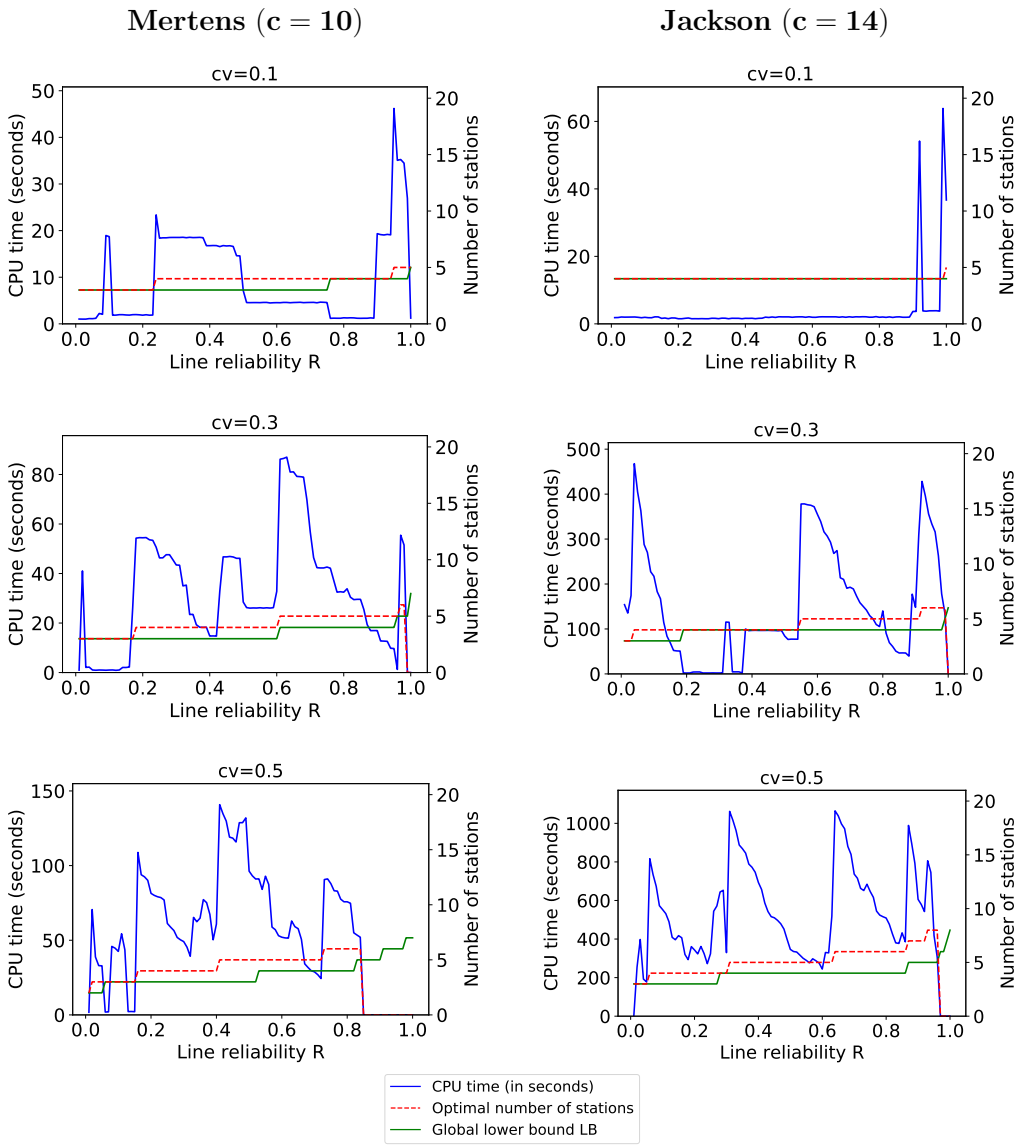


Figure 2.2: Impact of line reliability R on computation time, number of stations and global lower bound

			Normal distribution					Gamma distribution				
	cv	**c**	**LB**	$\sum \mathbf{Z_m}$	Δ	**LR**	**Time (s)**	**LB**	$\sum \mathbf{Z_m}$	Δ	**LR**	**Time (s)**
Mertens	0.1	7	6	6	0	0.9522	2	6	Inf.			0
		8	5	5	0	0.9795	1	5	5	0	0.9764	1
		10	4	5	1	0.9516	43	4	5	1	0.9997	36
		15	3	3	0	0.9947	1	3	3	0	0.9925	1
		18	2	2	0	0.9999	1	2	2	0	0.9998	1
	0.3	10	4	5	1	0.9616	9	4	6	2	0.9552	59
		15	3	3	0	0.9591	1	3	3	0	0.9604	2
		18	2	3	1	0.9556	6	2	3	1	0.9717	5
	0.5	15	3	4	1	0.9601	29	3	5	2	0.9600	78
		18	3	3	0	0.9623	1	3	4	1	0.9557	52
Bowman	0.1	20	5	5	0	0.9550	1	5	6	1	0.9556	29
Jaeschke	0.1	7	8	8	0	0.9522	3	9	Inf.			0
		8	7	7	0	0.9753	1	7	7	0	0.9702	2
		10	5	6	1	0.9989	29	5	6	1	0.9982	33
		18	3	3	0	0.9999	1	3	3	0	0.9999	1
	0.3	10	6	7	1	0.9577	69	6	8	2	0.9558	94
		18	3	3	0	0.9765	2	3	3	0	0.9577	2
	0.5	18	3	4	1	0.9590	12	3	5	2	0.9694	97
Jackson	0.1	9	7	8	1	0.9528	277	7	8	1	0.9941	177
		10	6	7	1	0.9962	135	6	7	1	0.9931	149
		13	4	5	1	0.9858	46	4	5	1	0.9799	43
		14	4	4	0	0.9819	4	4	4	0	0.9764	4
		21	3	3	0	0.9706	2	3	3	0	0.9671	2
	0.3	13	5	6	1	0.9622	183	5	7	2	0.9534	806
		14	4	6	2	0.9716	321	4	6	2	0.9739	418
		21	3	3	0	0.9514	69	3	4	1	0.9644	142
	0.5	14	5	8	3	0.9509	403	6	Inf.			0
		21	3	4	1	0.9500	84	3	5	2	0.9604	488
Mansoor	0.1	62	4	4	0	0.9967	2	4	4	0	0.9936	2
		94	3	3	0	0.9680	2	3	3	0	0.9644	2
	0.3	94	3	3	0	0.9524	1	3	3	0	0.9631	4
	0.5	94	3	4	1	0.9635	62	4	5	1	0.9508	218
Mitchell	0.1	21	6	6	0	0.9561	7	6	6	0	0.9677	7
		26	5	5	0	0.9549	6	5	5	0	0.9519	6
		35	4	4	0	0.9825	6	4	4	0	0.9792	6
		39	3	3	0	0.9800	6	3	3	0	0.9754	7
	0.3	21	6	8	2	0.9524	2,027	7	9	2	0.9517	3,006
		26	5	6	1	0.9502	324	5	[6, 7]	2	0.9534	10,000
		35	4	5	1	0.9506	3,280	4	5	1	0.9561	2,756
		39	4	4	0	0.9551	7	4	4	0	0.9511	7
	0.5	26	6	8	2	0.9500	4,655	7	13	6	0.9501	6,037
		35	4	5	1	0.9512	1,160	4	[5, 6]	2	0.9509	10,000
		39	4	5	1	0.9510	2,660	4	5	1	0.9529	1,693

Table 2.7: Comparison of the optimal solutions for normally and gamma distributed task times

For the remaining 40 instances, the optimal number of stations for the gamma distribution is equal to or larger than the optimal number for the normal distribution. The instance of Mitchell with $cv = 0.5$ and $c = 26$ is especially noteworthy, where the optimal solution with a normal distribution requires 8 stations and the optimal solution with a gamma distribution requires 13 stations.

Computation times are in the same range for many instances, however, sometimes larger for the instances with a Normal distribution and sometimes larger for the instances with a Gamma distribution. The computation times exceed 10,000 seconds for two instances with the Gamma distribution.

To summarize, the sampling approach with the transformation of lower bounds is applicable for general distributions of the task times where no bounds are described in the literature.

2.7 Conclusion

We present the assembly line balancing problem with a chance-constraint on the line reliability and formulate a sampling-based model. We develop a general transformation of any lower bound on the number of stations for the deterministic problem into a lower bound for the sampling-based model. This general transformation can be applied to any bound that has already been developed or to any potential new bound.

We develop a reliability-based branch-and-bound (RB&B) algorithm that explicitly considers the dependence among all stations due to the constrained line reliability. A partial assignment of tasks to stations considers already constructed stations and potential further assignments to other stations which increases the size of the branching tree. However, nodes are effectively fathomed based on the transformed bounds, a new dominance rule and a new logical test. The proposed sampling approach with a transformation of lower bounds is general and can be applied independent of the assumed theoretical or empirical distribution of the task times.

A numerical study shows that the transformed lower bounds are tight and that they substantially reduce the required computation time of the RB&B algorithm and of the solver CPLEX. We analyze the sensitivities of the desired line reliability and of the distribution of task times.

Further research could address algorithmic improvements as well as application-based extensions.

The lower bounds are independent of the specific sampling formulation for the assembly line balancing problem, as long as the sampling-based constraint on the line reliability is included. Furthermore, the lower bounds can be applied to other optimal solution methods or to heuristic approaches. As the general transformation of lower bounds from Theorem 1 is valid for any lower bound to the deterministic problem, newly developed deterministic lower bounds can directly be transformed and applied during the RB&B algorithm. The RB&B algorithm is flexible and therefore any other suitable dominance rule may be integrated. As demonstrated in Section 5.2.2, some dominance rules for the deterministic version of the problem can be integrated into dominance rules for the stochastic version of the problem. In contrast to dominance rules for the deterministic problem, they also need to consider non-R-maximal station loads. Furthermore, adaptions of deterministic dominance rules which compare mean task times need to consider the interdependence among all samples and all realizations within one sample. In addition, fast and reliable heuristics or other exact approaches can be developed to solve the stochastic assembly line balancing for larger instances. If the algorithms are based on a sampling formulation, the transformed lower bounds of Theorem 1 can be used.

Future research could also address more complex lines, such as U-shaped lines, producing multiple products. Alternate concepts of handling incomplete work pieces could be analyzed, for example including rework loops. In addition to the minimization of the number of stations, the optimization problem of maximizing the sampling-based line reliability is an interesting direction for further research. Dedicated solution approaches and fathoming strategies have to be developed and tested numerically.

3 Numerical investigation of sampling-based performance evaluation and optimization

Co-authors:

Johannes Diefenbach
Chair of Production Management, Business School, University of Mannheim, Germany

Justus Arne Schwarz
Chair of Production Management, Faculty of Business, Economics and Management Information Systems, University of Regensburg, Germany

Raik Stolletz
Chair of Production Management, Business School, University of Mannheim, Germany

Working paper.

Abstract:

Sampling-based optimization is often used to analyze the performance or optimize the design of complex, stochastic optimization problems. Common approaches of drawing the required random numbers are simple random sampling (SRS) and descriptive sampling (DS). To gain insights into the impact of the used sampling method and the sample size, we consider the performance evaluation of an M/D/1 queueing system and the optimization of an M/M/c staffing level.

We conduct a numerical study and analyze the distribution of the performance measure or optimal decision over independent replications. The probability to underestimate the desired performance measure of the M/D/1 system is higher for descriptive sampling in the analyzed examples, both for expected values as well as values

based on the standard deviation. This effect can still be observed for large sample sizes. For the optimization of the M/M/c staffing level, the expected value of the optimal decision is similar for both sampling methods. However, the standard deviation for descriptive sampling is lower. The distribution of the optimal decision is not symmetrical and this effect is stronger for descriptive sampling in the analyzed examples.

Therefore, managers should be aware that the distribution of the resulting performance measures or optimal solution derived from a sampling-based approach may not be symmetrical and the chosen sampling method may have an impact on this behavior.

3.1 Introduction

Complex, stochastic systems are often difficult to analyze analytically. Nonetheless, performance evaluation and optimization of such systems is desirable. Therefore, sampling-based optimization is frequently used to solve stochastic optimization problems (Stolletz, 2022). Common approaches of drawing the required random numbers are simple random sampling (SRS) and descriptive sampling (DS). Both have been applied to a broad variety of problems in operations management. Examples for the application of SRS are lot sizing (Kämpf and Köchel, 2006), buffer allocation (Costa et al., 2015), project scheduling (Golenko-Ginzburg and Gonik, 1997) and call-center staffing (Atlason et al., 2008). Similarly, lot sizing (Helber et al., 2013), buffer allocation (Stolletz and Weiss, 2013; Weiss and Stolletz, 2015), project scheduling (Ballestin and Leus, 2009) and production and remanufacturing planning (Hilger et al., 2016) have been analyzed using DS. These are just a few examples of the many applications of both methods.

Simple random sampling, also known as Monte Carlo sampling, draws a random set of values. As the values are drawn at random, they appear in a random sequence. Descriptive sampling was first introduced by Saliby (1990). In contrast to simple random sampling, descriptive sampling draws a deterministic set of values. Figure 3.1 (taken from Saliby (1990)) shows the deterministic set with a sample size of $N = 10$. The sampled values are spread uniformly along the y-axis between 0 and 1 and applied to the cumulative distribution function $F(x)$. This allows descriptive sampling to capture the entire range of the distribution even for small sample sizes. Afterwards, the deterministic set is permutated into a random sequence, thereby inducing stochasticity. Studies have shown that descriptive sampling leads to a lower variability of the sampled values (Saliby and Pacheco, 2002) and the performance measures of a stochastic system (Helber et al., 2013). Furthermore, using descriptive sampling in a sampling-based optimization has been shown to lead to a reduced variability of the sample-optimal solutions for the buffer allocation problem (Stolletz and Weiss, 2013).

The sampling procedure of DS can therefore be divided into two steps: creation and sequencing of the set of values. For the sampling of a single random variable, as is typical for problems such as the Newsvendor, only the creation of the set of values is crucial, as their order of arrival does not play a role. When sampling a random process however, the sequence of the sampled values plays a critical role. Therefore, we analyze the performance evaluation and sampling-based optimization of a queueing system. One approach for the performance evaluation of a system

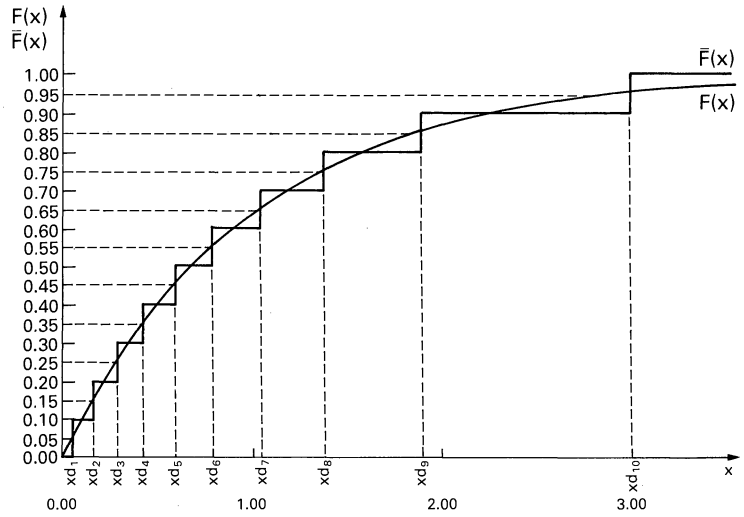


Figure 3.1: Cumulative distribution function [$F(x)$] for a negative exponential random variable with mean E[X] = 1.0 and corresponding descriptive samples xd_n for a sample size $N = 10$ (taken from Figure 2 in Saliby (1990))

is to analyze independent replications and derive their mean for each performance measure. However, this is not a common approach for sampling-based optimization. Rather, often a single replication is considered.

The aim of this paper is to analyze the impact of the sample size and sampling method for a given problem. More specifically, we analyze the distribution of the resulting performance measure or optimal decision for a given sampling-approach over a number of independent replications. Furthermore, we analyze numerically if structural differences can be observed between simple random sampling and descriptive sampling for the analyzed examples.

We analyze the sampling-based performance evaluation for one instance of an M/D/1 queueing system and for three instances of the optimization of an M/M/c staffing level for simple random sampling and descriptive sampling with different sample sizes. We exemplify how the sensitivity of the analyzed performance measures or optimal solutions in the sample size and sampling method can be analyzed based on these four instances. We find that even though descriptive sampling has a lower root mean square error than simple random sampling for all examples of the M/D/1 queueing system, it has a higher probability to underestimate the desired performance measure. This effect can still be observed for large sample sizes. For the optimization of the M/M/c staffing level, the distribution of the optimal decision is not symmetrical and this effect is stronger for descriptive sampling in the analyzed examples.

3.2 Problem description

The aim of this paper is to identify the impact of the chosen sampling method and sample size on the performance evaluation and optimization. Therefore, we analyze the performance evaluation of the analytically tractable M/D/1 queueing system and the optimization of the M/M/c staffing level, which has a single decision on the number of servers.

3.2.1 M/D/1 queueing system

We analyze the performance evaluation of the M/D/1 queue to better understand the effects of the sampling method on the optimization, see Figure 3.2. The arrival process has exponentially distributed inter-arrival times with rate λ. A single server processes all arriving customers with a deterministic processing rate of μ. There is no limit on the queue length.

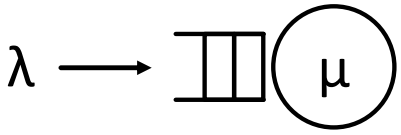


Figure 3.2: Analyzed system for performance evaluation: M/D/1 queue

The M/D/1 queue is chosen, as it is analytically tractable (i.e. exact values are known) and there is only one stochastic process. Therefore, there are no interactions between multiple sampled processes. In the steady state (with $\lambda < \mu$), the utilization ρ of such a system is given by

$$\rho = \frac{\lambda}{\mu}$$

and the processing time τ can be calculated by

$$\tau = \frac{1}{\mu}\,.$$

We analyze the expected waiting time $E[W_q]$, expected queue length $E[L_q]$, standard deviation of waiting times $Std[W_q]$ and standard deviation of queue length $Std[L_q]$. Using the Pollaczek-Khinchine formula and Little's Law, all expected val-

ues can be derived analytically (Shortle et al., 2018):

$$E[W_q] = \frac{\rho \cdot \tau}{2(1-\rho)} \tag{3.1}$$

$$E[L_q] = \lambda \cdot E[W_q] \tag{3.2}$$

The standard deviation of the waiting time $Std[W_q]$ and of the queue length $Std[L_q]$ can be determined by

$$Std[W_q] = \left(\frac{\rho \cdot \tau}{2(1-\rho)}\right)^2 + \frac{\rho \cdot \tau^2}{3(1-\rho)} \tag{3.3}$$

$$Std[L_q] = \frac{\rho^3}{3(1-\rho)} + \left(\frac{\rho^2}{2(1-\rho)}\right)^2 + \frac{\rho^2}{2(1-\rho)}\,. \tag{3.4}$$

3.2.2 M/M/c staffing level

We analyze the M/M/c staffing level with exponentially distributed inter-arrival times and arrival rate λ. The service rate of each server is μ and service times are also exponentially distributed, see Figure 3.3. There is no limit on the queue length.

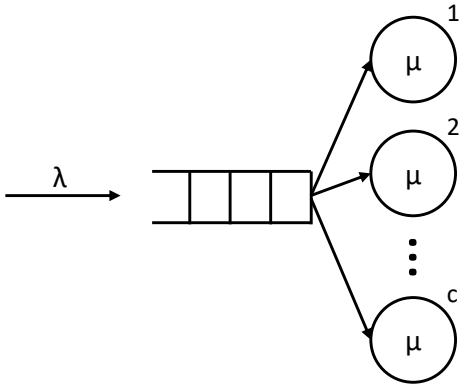


Figure 3.3: Analyzed system for optimization: M/M/c queue

The M/M/c system is chosen as it is a common system for the optimization of the staffing decision and it is analytically tractable. There is a single decision on the number of servers c, with the objective of minimizing the number of servers. Multiple constraints have been applied to this problem. In the sampling-based literature, it is common to apply expected values (e.g. Costa et al., 2015; Weiss and Stolletz, 2015) or an X/Y service level (e.g. Atlason et al., 2008; Ballestin and Leus, 2009).

$$\min c \tag{3.5}$$

$$E[W_q] \leq W^* \tag{3.6}$$

$$Pr\{W_q \leq Y\} \geq X \tag{3.7}$$

The objective is to minimize the required number of servers, see Equation (3.5). Equation (3.6) is a constraint on the expected waiting time, which cannot exceed a predefined value W^*. Equation (3.7) is known as an X/Y service level. The probability of the waiting time to exceed Y cannot exceed X. We consider these two constraints independently of each other in our numerical study. The relevant performance measures used as constraints can be determined analytically by

$$P_0 = \frac{1}{\sum_{n=0}^{c-1} \frac{\rho^n}{n!} + \frac{\rho^c}{c! \cdot (1-\frac{\rho}{c})}} \tag{3.8}$$

$$E[W_q] = \frac{\rho^c}{(c-1)! \cdot (c-\rho)^2 \cdot \mu} \cdot P_0 \tag{3.9}$$

$$Pr\{Wq \leq Y\} = 1 - \frac{\rho^c \cdot P_0}{c!(1-\frac{\rho}{c})} \cdot e^{-(c\mu-\lambda)Y} \ . \tag{3.10}$$

Therefore, the optimal staffing decision c^* can be determined by enumeration. We start with the first c, which fulfills the stability condition for steady state: $\lambda < \mu \cdot c$ and increase c by one in each iteration. The first c which fulfills the respective constraints is the optimal staffing level c^*.

3.3 Numerical study

To understand the impact of the sampling method and sample size on the performance evaluation and optimization of a system, we analyze the two problems from Section 3.2. We use 10000 independent replications and analyze the distribution of the results for each performance measure and constraint, respectively. All computations are performed using Python 3.6. We use the inverse of the cumulative distribution function by utilizing the *stats.expon.ppf* function from Python's *SciPy* library. For simple random sampling, we draw random numbers between 0 and 1 using the *random.uniform* function from Python's *NumPy* library. For descrip-

tive sampling, the deterministic set is permutated into a random sequence using the *random.shuffle* function from Python's *NumPy* library.

3.3.1 Performance evaluation of M/D/1

For the performance evaluation of the M/D/1 queue, we first analyze the impact of the warm-up length on the expected waiting time. Then, we analyze the distribution of the sampling-based performance evaluation.

Analysis of warm-up length

We want to analyze the queueing system in steady-state. However, at the beginning of a sampling replication, the system is empty and therefore, a warm-up length has to be considered. In this section, we analyze the appropriate warm-up length (in number of samples n_0). All performance measures of the M/D/1 queue will later be considered on the N samples following the warm-up length with n_0 samples. Let $W_{n,r}$ be the waiting time of sample n in replication r. Figure 3.4 shows the waiting time for each sample for three independent replications. As can be seen, the waiting time starts at zero for the first sample and tends to increase at first. In this example using a warm-up length of $n_0 = 750$, the performance of the system would be analyzed on the $N = 1000$ samples after the warm-up length.

We conduct a numerical study with arrival rate $\lambda = 0.9$, processing rate $\mu = 1$, and $R = 10000$ independent replications to understand the impact of the warm-up length n_0 using simple random sampling. Fig. 3.5 shows the expected waiting time over all replications $E[W_q](n)$ for sample n, where the expected waiting time is calculated by

$$E[W_q](n) = \frac{\sum_{r=1}^{R} W_{n,r}}{R} .$$

The expected waiting time for the first sample starts at zero. At first, the expected waiting time of each sample increases. Once it reaches the analytical waiting time $E[W_q] = 4.5$ between $n = 500$ and $n = 750$, the expected waiting time of each sample fluctuates around the analytical value. Thus, for the analyzed case, the expected waiting time of a sample arriving into the system is stable for samples after $n = 750$. Therefore, we choose a warm-up length n_0 of 750 for our analysis.

As we do not want the warm-up phase to have an impact on our analysis, we construct the numerical study of the M/D/1 queue in the following way. The samples for the warm-up phase are always drawn using simple random sampling. To be pre-

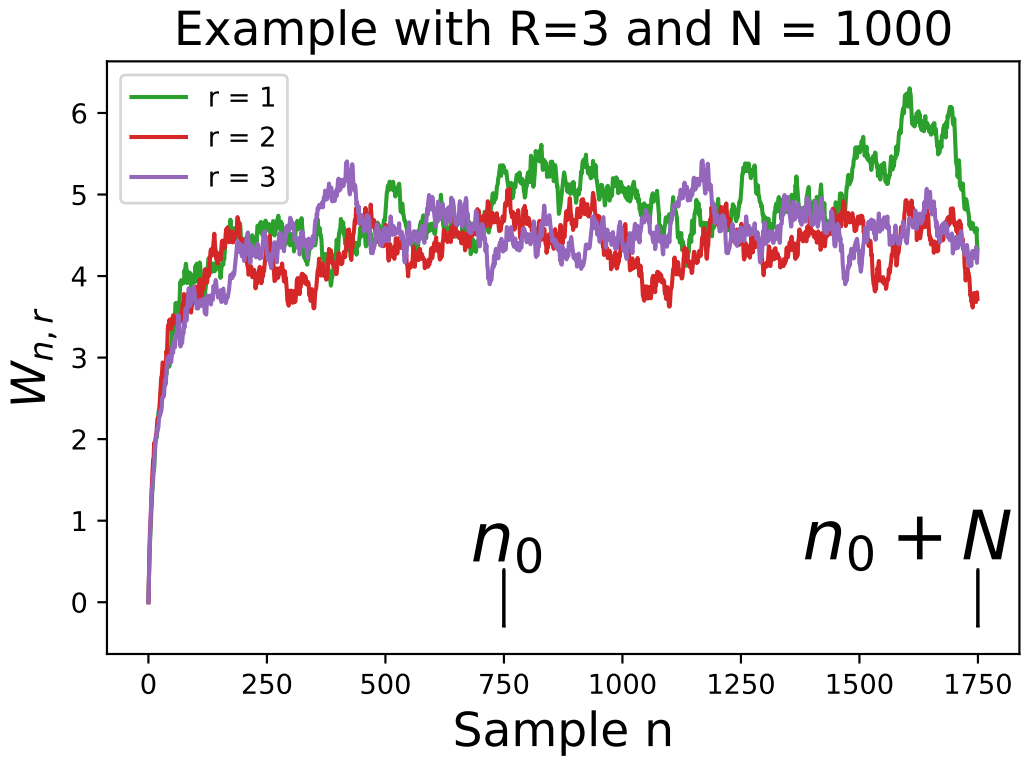


Figure 3.4: Waiting time for each sample for three independent replications

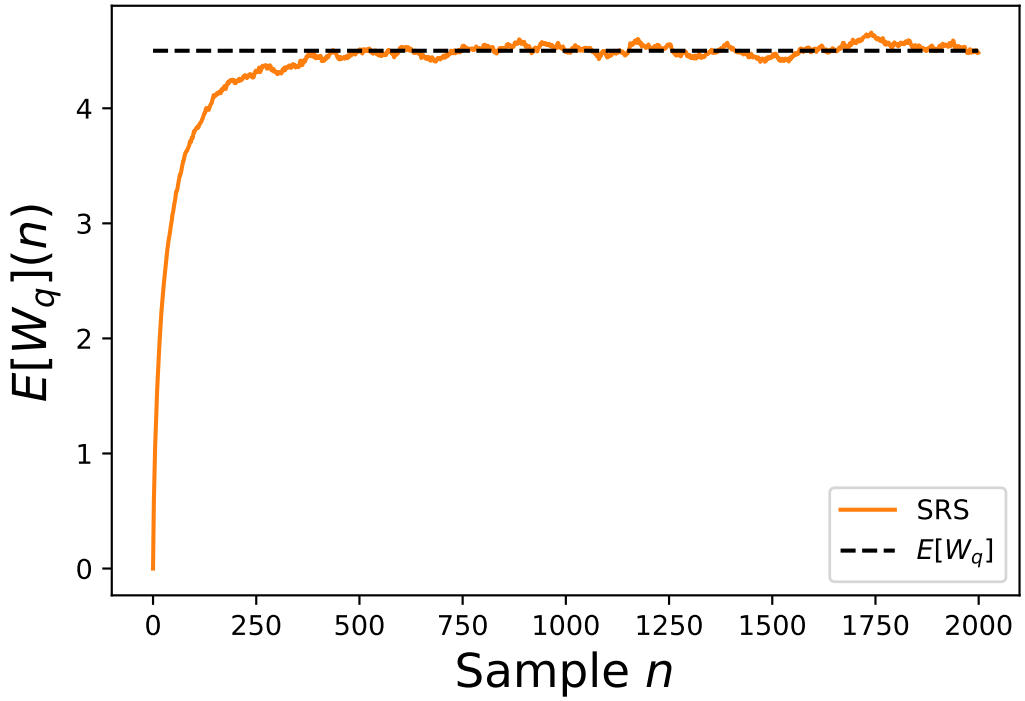


Figure 3.5: Expected waiting time over all replications $E[W_q](n)$ for sample n

cise, we sample R independent warm-up phases using simple random sampling. In the next step, we add the sampled inter-arrival times of N samples for each replication r and each method (SRS; DS) independently. Therefore, both SRS and DS have the same R warm-up phases to start from.

Distribution of sampling-based performance evaluation

We analyze the distribution of the resulting performance evaluation of the M/D/1 queue with exponentially distributed inter-arrival times with rate $\lambda = 0.9$, deterministic processing rate $\mu = 1$, a warm-up length of $n_0 = 750$ and $R = 10000$ independent replications. Different effects can be observed for these examples. The analytical expected waiting time of this system based on Equation (3.1) is $E[W_q] = 4.5$ and the analytical standard deviation based on Equation (3.3) is $Std[W_q] = 4.82$.

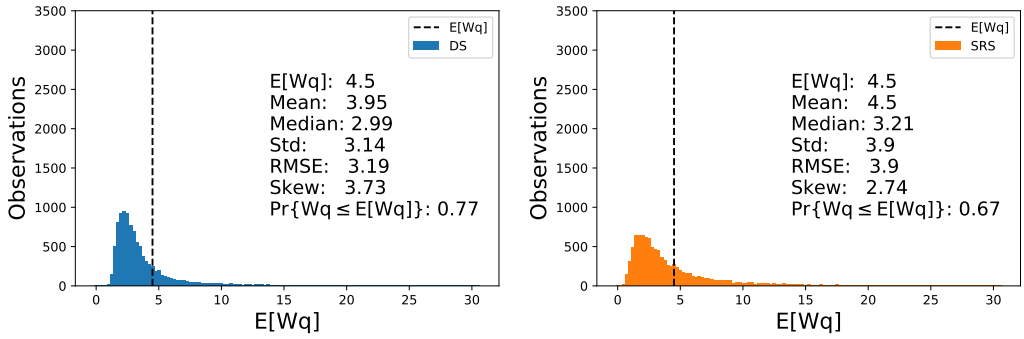
Figures 3.6 and 3.7 shows the resulting histograms of the expected waiting times for different sample sizes N from $N = 100$ to $N = 15000$ for descriptive sampling on the left (in blue) and simple random sampling on the right (in orange). For a sample size $N = 100$ and descriptive sampling, a large part of the distribution is below the analytical expected waiting time. The RMSE is 3.19 and the skew is 3.73. The probability to observe at most the analytical value is 77%. For a sample size $N = 100$ and simple random sampling, again a large part of the distribution is below the analytical value. The RMSE is 3.9 and is higher than for DS. However, the distribution is more symmetrical, which can be seen by the lower skew of 2.74. The probability to observe at most the analytical value is 67% and therefore less than for DS. Figure 3.10 aggregates the results for the (a) RMSE, (b) Skew and (c) Probability that a replication is at most the analytical value. With an increase in the sample size, the spread of the distribution decreases for both sampling methods. As expected, the root mean square error (RMSE) is decreasing in the sample size. In all examples, the RMSE is lower for descriptive sampling for the same sample size N. For all sample sizes, the distributions are not symmetrical, but rather, there is a positive skew. The probability that a single replication is at most the expected waiting time is always higher for descriptive sampling in these examples. All effects can still be observed for sample sizes up to $N = 15000$. The same observations can be made when analyzing the standard deviation of the waiting time for the same examples, see Figures 3.8 and 3.9. In addition, these observations are not limited to the waiting time, but extend to the expected queue length and standard deviation of the queue length, see Appendix E.

To summarize this example with $\mu = 1$ and $\lambda = 0.9$, the root mean square error is decreasing in the sample size for the performance evaluation of the M/D/1 queue for the analyzed examples. However, for the same sample size, descriptive sampling always has a lower root mean square error than simple random sampling in the analyzed examples. For all performance measures, the distribution over all replications is skewed. The probability to underestimate the desired performance measure is higher for descriptive sampling in the analyzed examples.

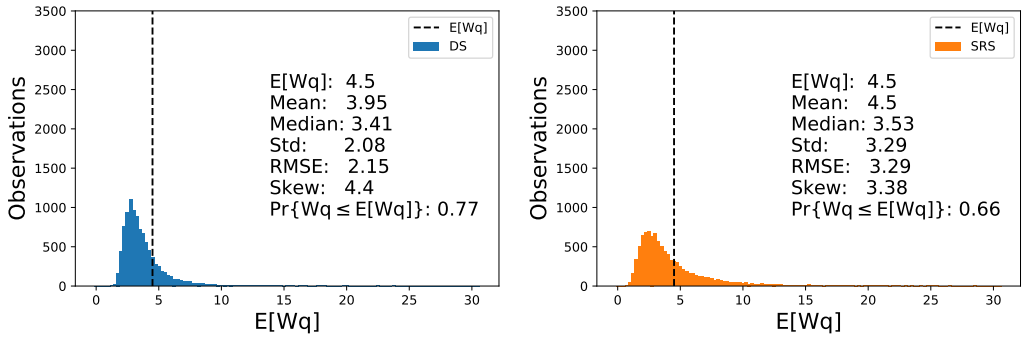
3.3.2 Optimization of M/M/c staffing level

We analyze the distribution of the resulting optimal staffing decision c of the M/M/c staffing level with exponentially distributed inter-arrival times with rate $\lambda = 10$, an exponential processing rate $\mu = 1$ per server c, a warm-up length of $n_0 = 750$ and $R = 10000$ independent replications. As before, we analyze sample sizes from $N = 100$ to $N = 15000$. We consider two examples with a constraint on the expected waiting time ($E[W_q] \leq 0.7$ and $E[W_q] \leq 0.000007$) and one with a constraint on the probability X of waiting at most a specified time Y ($Pr\{W_q \leq 0.7\} \geq 0.8$). The second example of a very small W^* is chosen to analyze a system in which waiting occurs only rarely. The analytical optimal solutions based on Equations (3.9) and (3.10) are $c^* = 11$, $c^* = 25$ and $c^* = 12$, respectively.

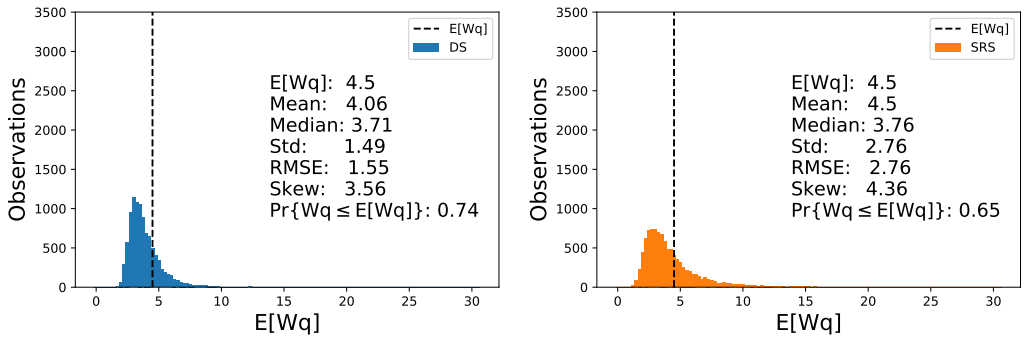
Figure 3.11 and 3.12 show the distribution of the sample-optimal staffing decision c with a constraint on the expected waiting time of $E[W_q] \leq 0.7$ and $E[W_q] \leq 0.000007$, respectively. Figure 3.13 shows the results for an X/Y service level with $Pr\{W_q \leq 0.7\} \geq 0.8$. Again, we report different sample sizes N for descriptive sampling on the left (in blue) and simple random sampling on the right (in orange). The results for sample sizes from 2500 to 15000 can be found in Appendix F. For a sample size $N = 100$, $E[W_q] \leq 0.7$ and descriptive sampling, the analytical optimal solution of $c^* = 11$ is found in 6526 of the 10000 replications. The RMSE is 0.74 and the skew is 1.06. The probability to observe at most the analytical optimal solution is 75%. For a sample size $N = 100$, $E[W_q] \leq 0.7$ and simple random sampling, the analytical optimal solution of $c^* = 11$ is found in 5005 of the 10000 replications and therefore less frequent compared to DS. The RMSE is 0.9 and is higher than for DS. However, the distribution is more symmetrical, which can be seen by the lower skew of 0.66. The probability to observe at most the analytical optimal solution is 70% and therefore less than for DS. For all analyzed examples, the largest observed difference in the mean optimal staffing decision between descriptive sampling and simple random sampling is 0.1. For the problems with



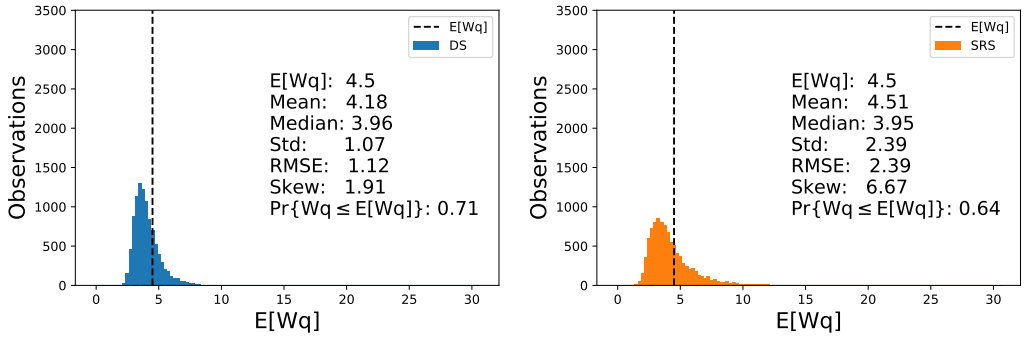
(a) Sample size $N = 100$



(b) Sample size $N = 250$

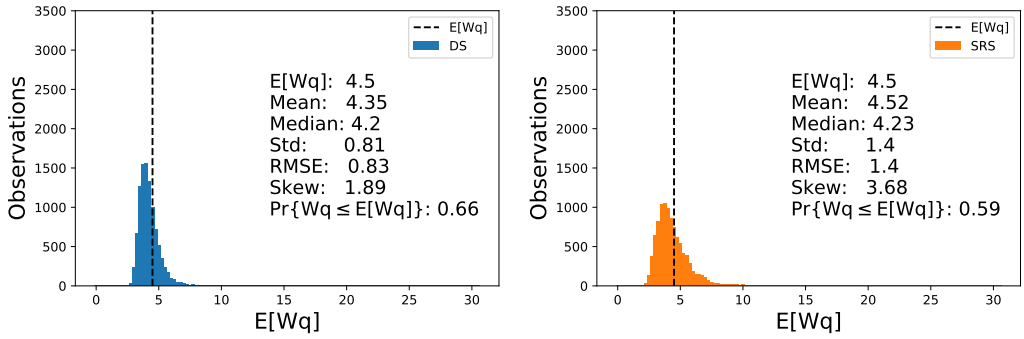


(c) Sample size $N = 500$

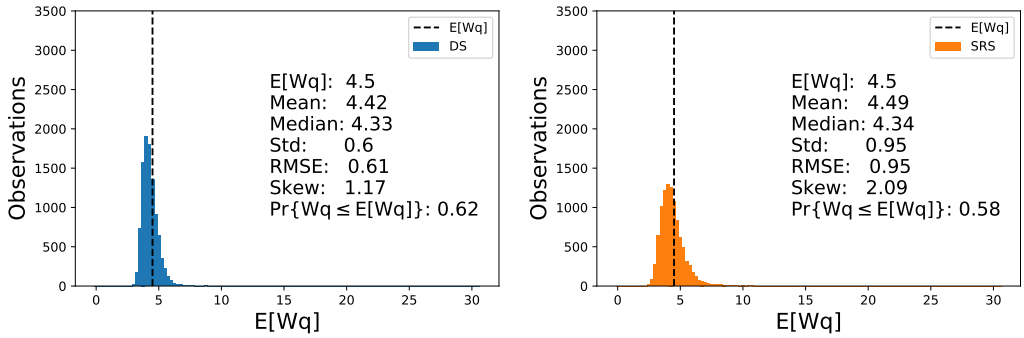


(d) Sample size $N = 1000$

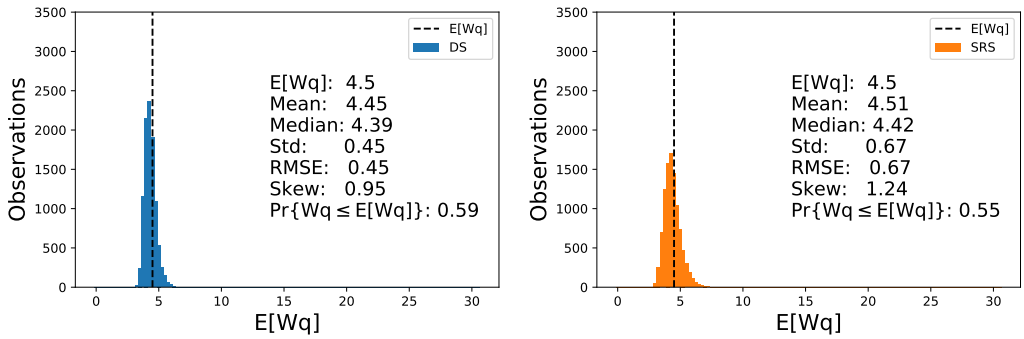
Figure 3.6: Expected waiting time in an M/D/1 system for sample sizes $N = 100$ to $N = 1000$



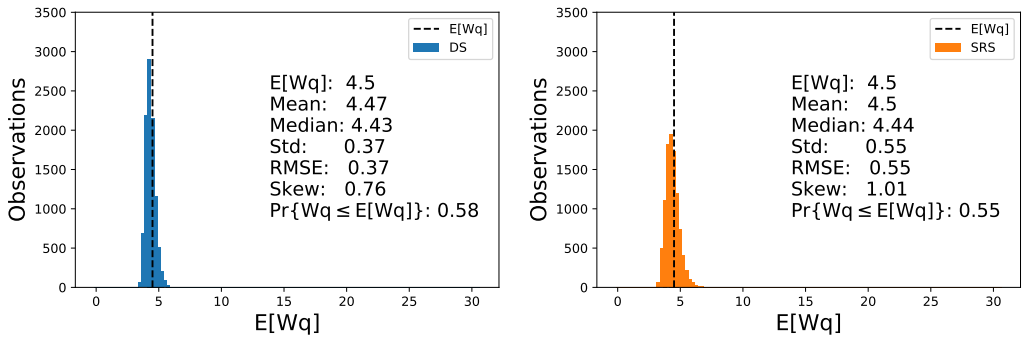
(a) Sample size $N = 2500$



(b) Sample size $N = 5000$

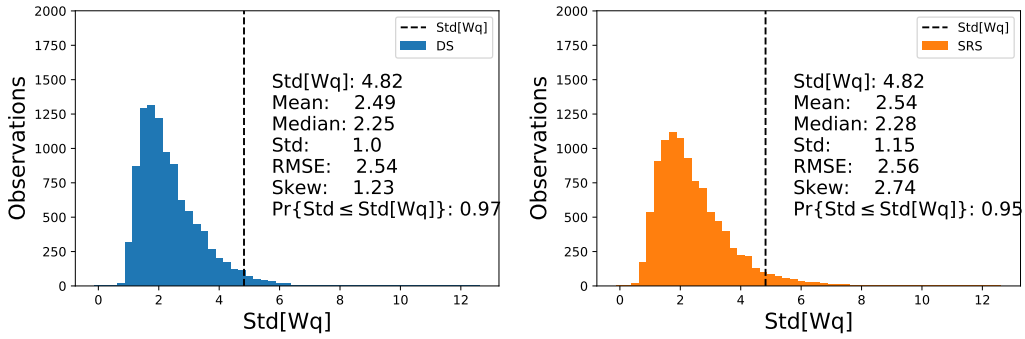


(c) Sample size $N = 10000$

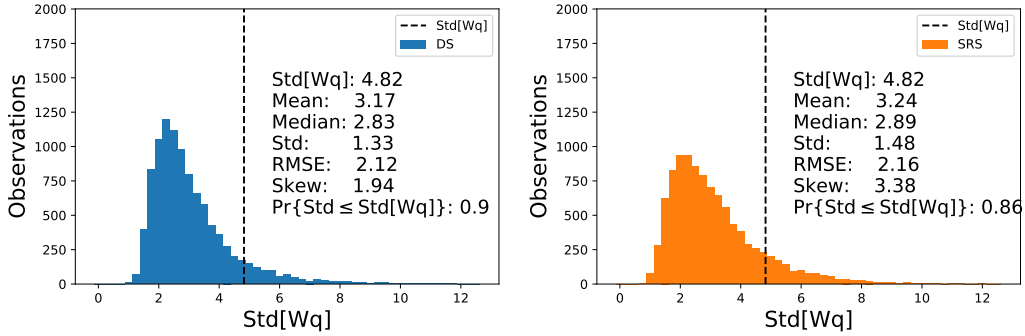


(d) Sample size $N = 15000$

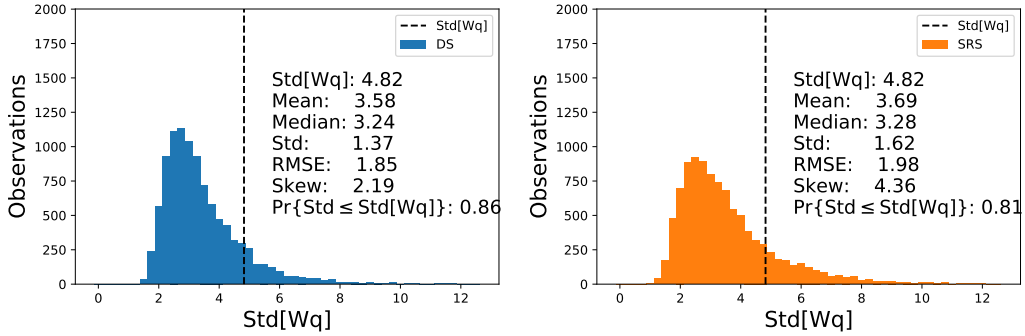
Figure 3.7: Expected waiting time in an M/D/1 system for sample sizes $N = 2500$ to $N = 15000$



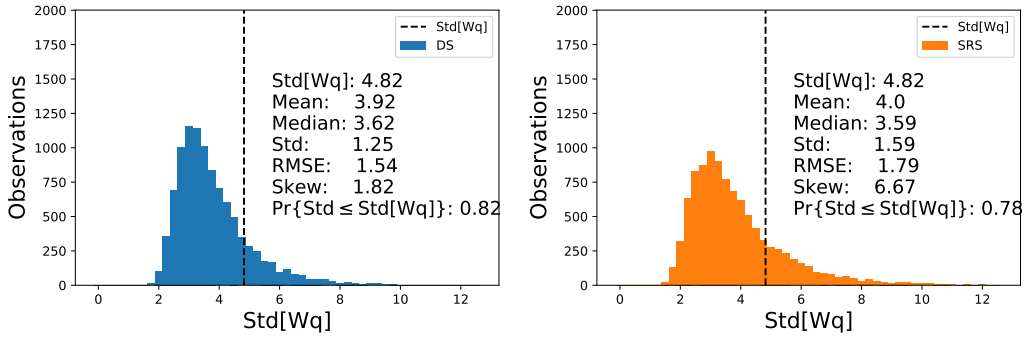
(a) Sample size $N = 100$



(b) Sample size $N = 250$

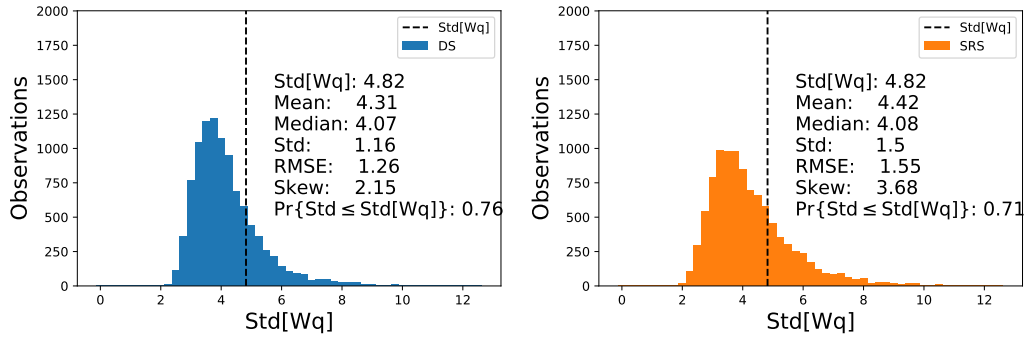


(c) Sample size $N = 500$

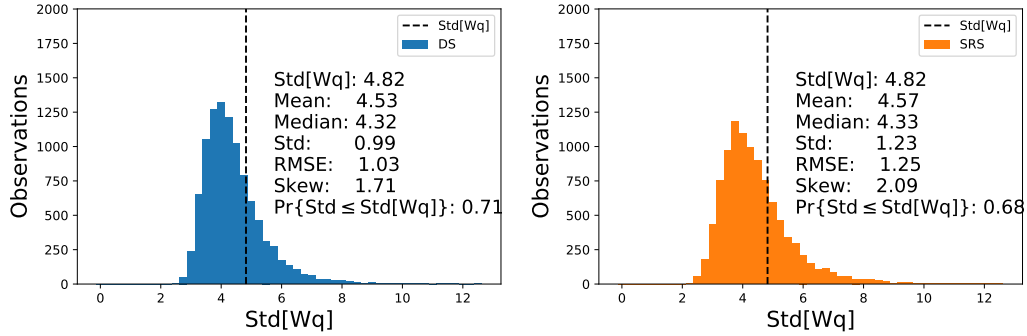


(d) Sample size $N = 1000$

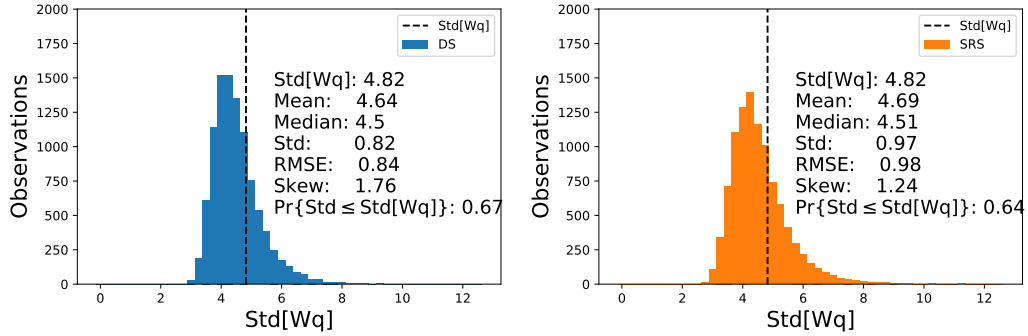
Figure 3.8: Standard deviation of waiting time in an M/D/1 system for sample sizes $N = 100$ to $N = 1000$



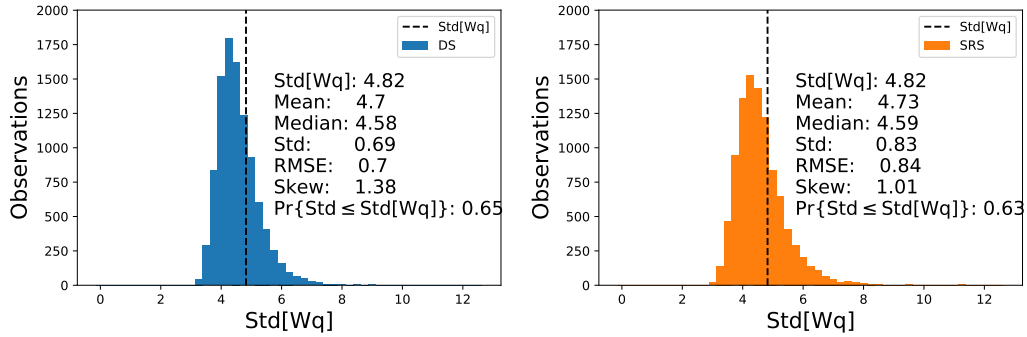
(a) Sample size $N = 2500$



(b) Sample size $N = 5000$

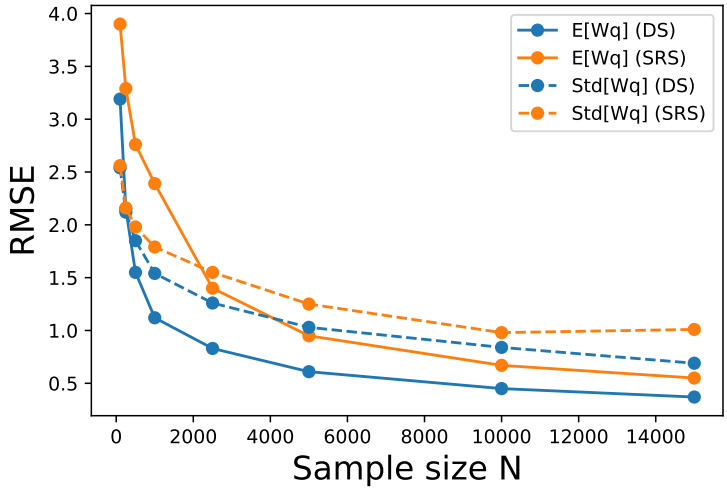


(c) Sample size $N = 10000$

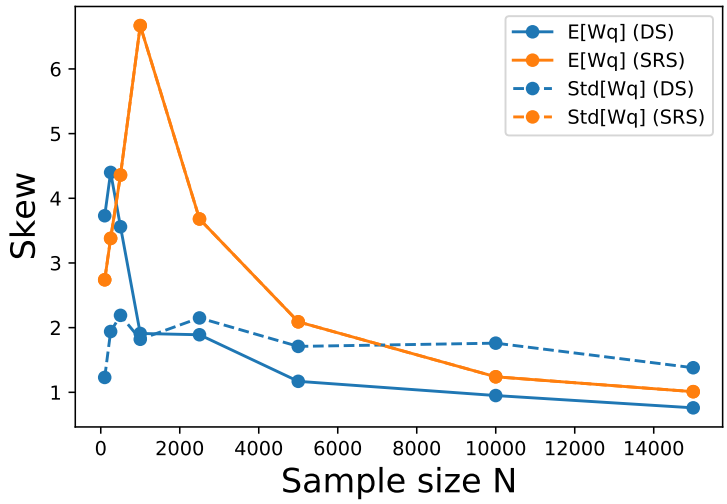


(d) Sample size $N = 15000$

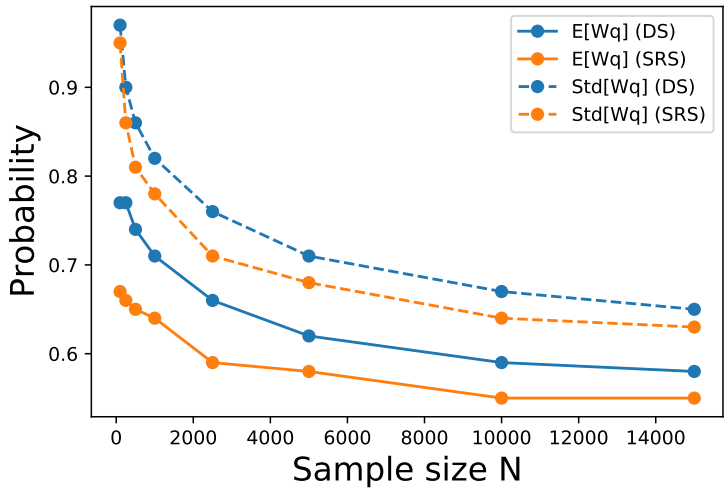
Figure 3.9: Standard deviation of waiting time in an M/D/1 system for sample sizes $N = 2500$ to $N = 15000$



(a) RMSE



(b) Skew



(c) Probability

Figure 3.10: Overview of distribution of results for the performance evaluation of M/D/1

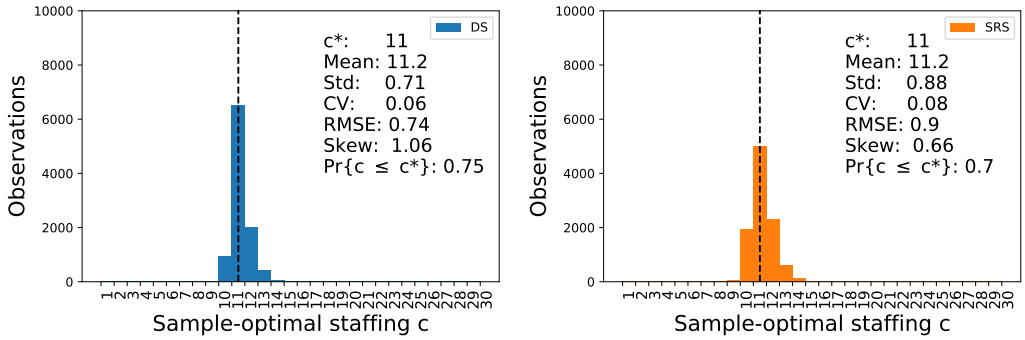
$E[W_q] \leq 0.7$ and $Pr\{W_q \leq 0.7\} \geq 0.8$ (Figures 3.11 and 3.13), the means range from 11.2 to 11.4 and from 11.3 to 11.5, respectively over all sample sizes and both sampling methods. For the problem with $E[W_q] \leq 0.000007$ (Figure 3.12), the mean is increasing in the sample size and ranges from 17.3 to 23.7. This suggests that even larger sample sizes are required for this case. For all analyzed examples, the standard deviation of the optimal staffing decision is larger for simple random sampling. Figure 3.14 aggregates the results for the (a) RMSE, (b) Skew and (c) Probability that a replication is at most the optimal decision derived from the analytical values for the optimization based on the expected waiting time. The largest observed difference in the RMSE between both methods is 0.18. However, the skew is larger for descriptive sampling for the same sample size. Like the results for the performance evaluation, all analyzed examples have a positive skew. Therefore, the distribution of the optimal decision is not symmetrical and this effect is stronger for descriptive sampling in the analyzed examples. The probability to chose at most the analytically optimal solution is equal or larger for descriptive sampling.

To summarize, for the optimization of the M/M/c staffing level, the mean optimal staffing decision does not differ between both methods. However, the distribution of the optimal decision is not symmetrical and this effect is stronger for descriptive sampling in the analyzed examples.

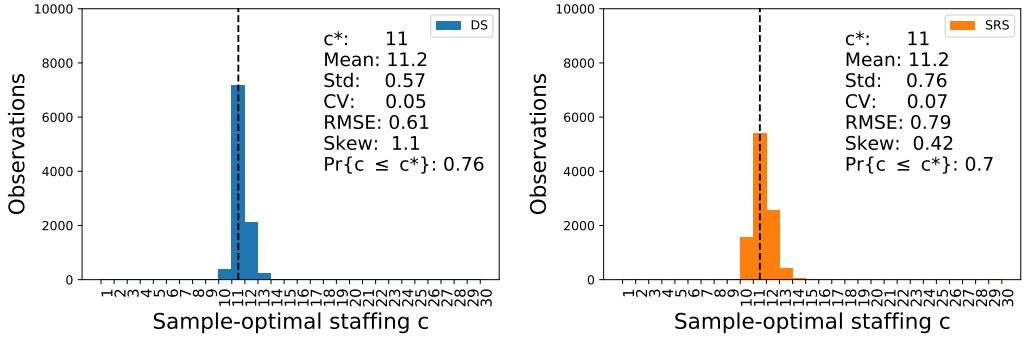
3.4 Conclusion and further research

We analyze the sampling-based performance evaluation of an M/D/1 queueing system and optimization of an M/M/c staffing level. We analyze the impact of different sample sizes and two sampling methods on the distribution of the resulting performance measures and optimal solution. The analyzed sampling methods are simple random sampling and descriptive sampling.

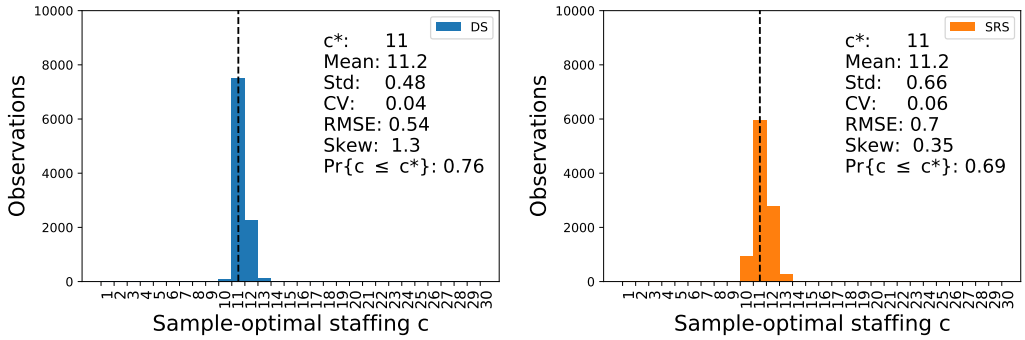
For the analyzed examples, we find that the root mean square error is decreasing in the sample size for the performance evaluation of the M/D/1 queue, both for expected values as well as values based on the standard deviation. This can be observed for both methods. However, for the same sample size, descriptive sampling always has a lower root mean square error than simple random sampling in the analyzed examples. For all performance measures, the distribution over all replications is skewed. The probability to underestimate the desired performance measure is higher for descriptive sampling in the analyzed examples. This effect can still be observed for large sample sizes. For the optimization of the M/M/c staffing level,



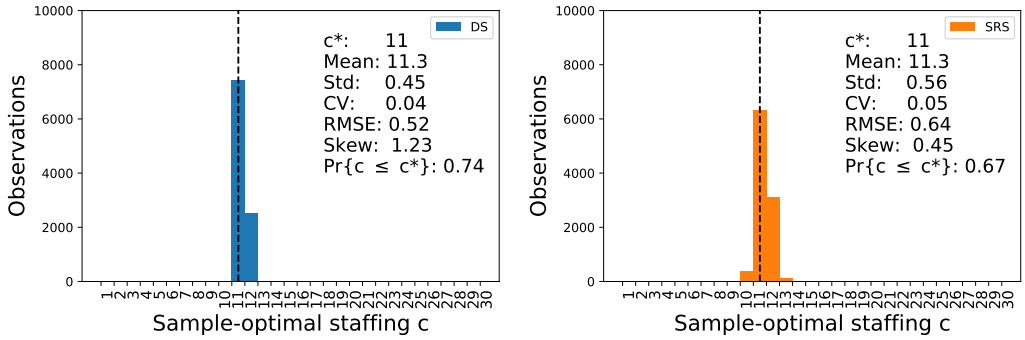
(a) Sample size $N = 100$



(b) Sample size $N = 250$

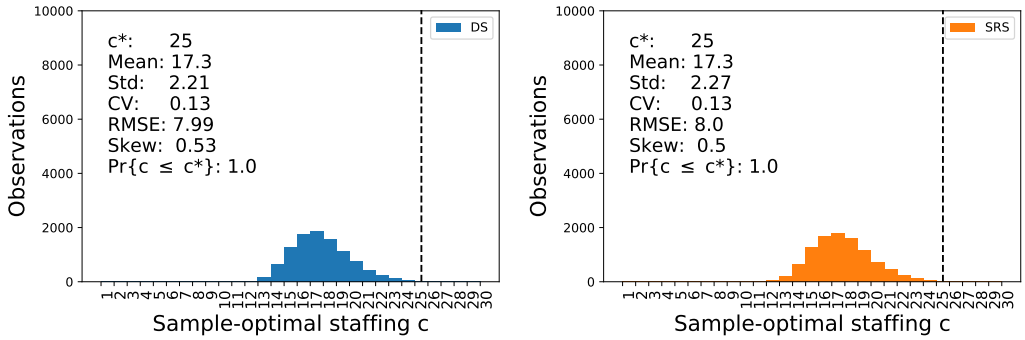


(c) Sample size $N = 500$

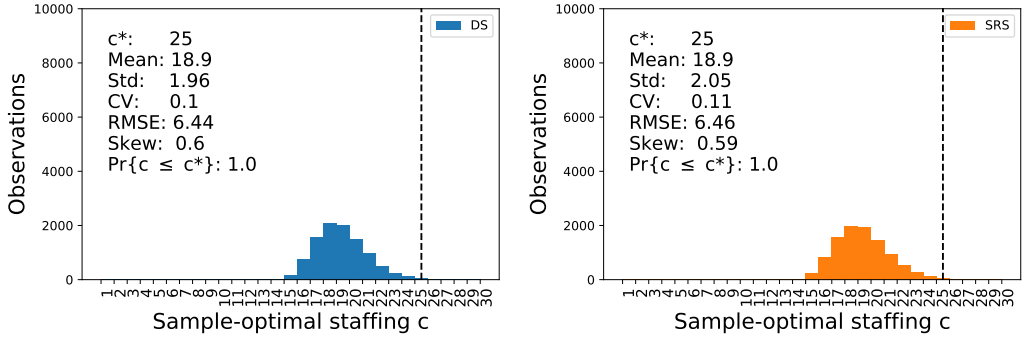


(d) Sample size $N = 1000$

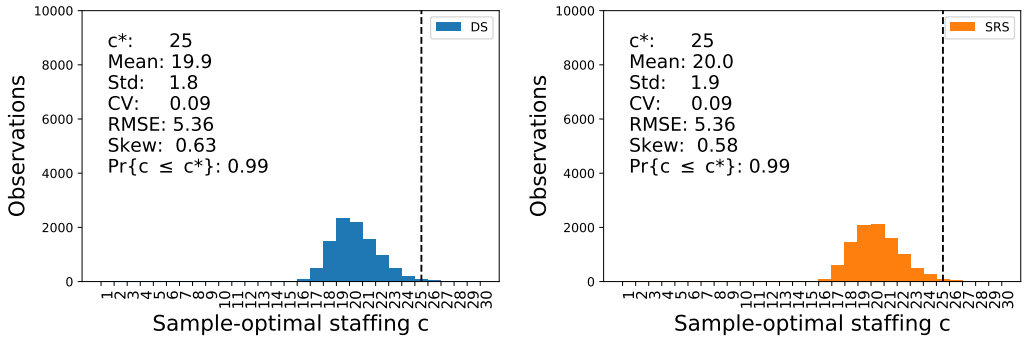
Figure 3.11: Optimal number of servers in an M/M/c system with constraint on expected waiting time with $E[W_q] \leq 0.7$ for sample sizes $N = 100$ to $N = 1000$



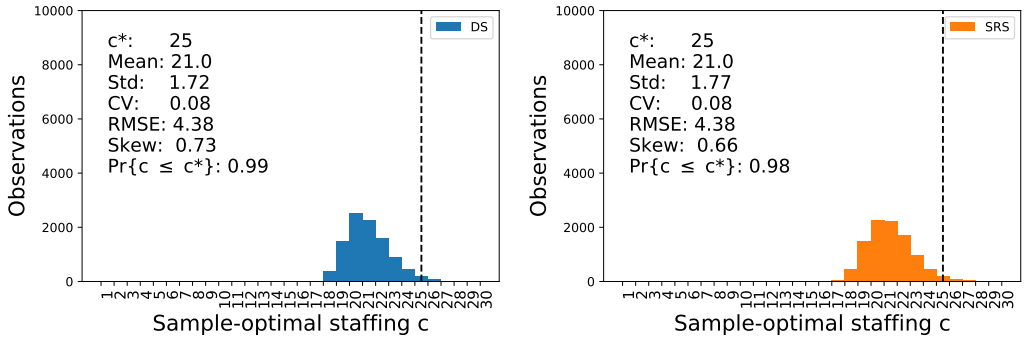
(a) Sample size $N = 100$



(b) Sample size $N = 250$

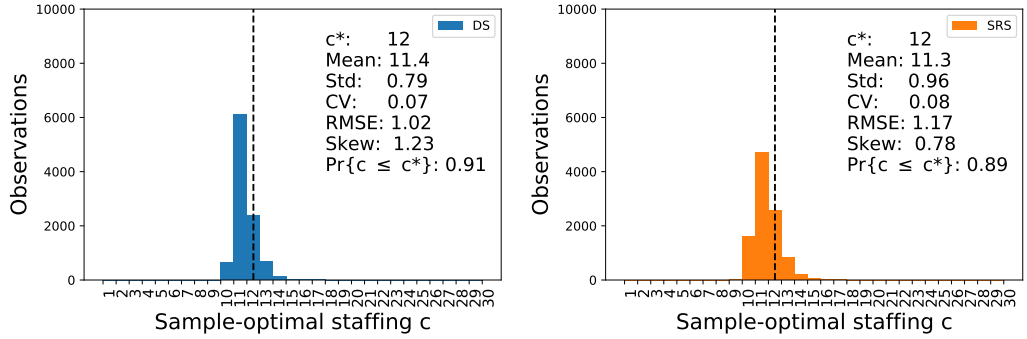


(c) Sample size $N = 500$

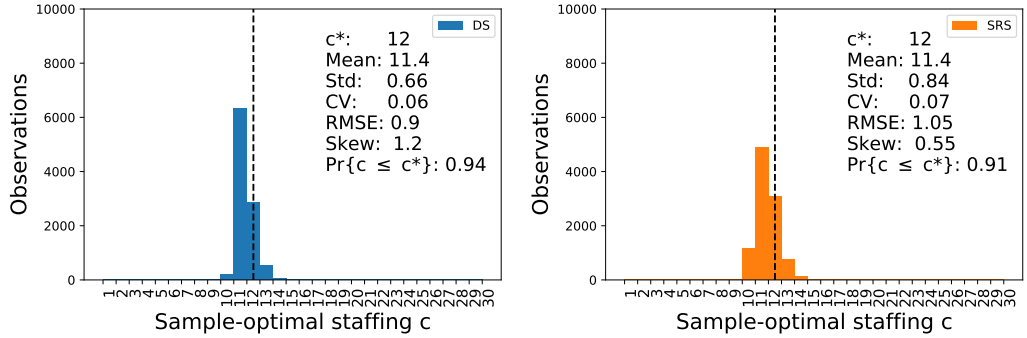


(d) Sample size $N = 1000$

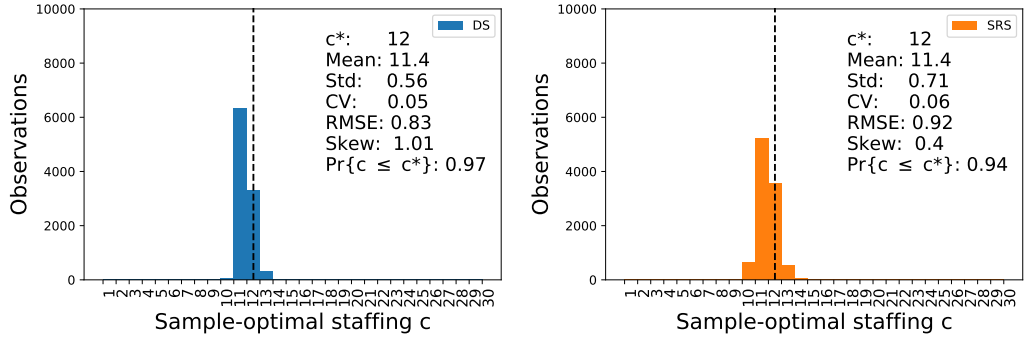
Figure 3.12: Optimal number of servers in an M/M/c system with constraint on expected waiting time with $E[W_q] \leq 0.000007$ for sample sizes $N = 100$ to $N = 1000$



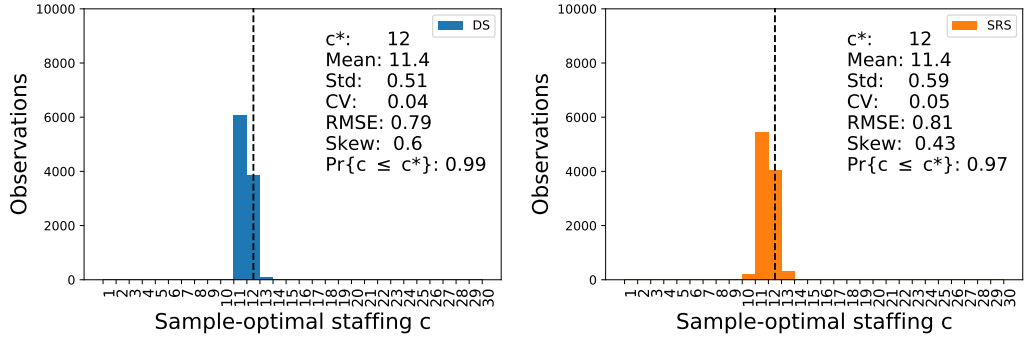
(a) Sample size $N = 100$



(b) Sample size $N = 250$

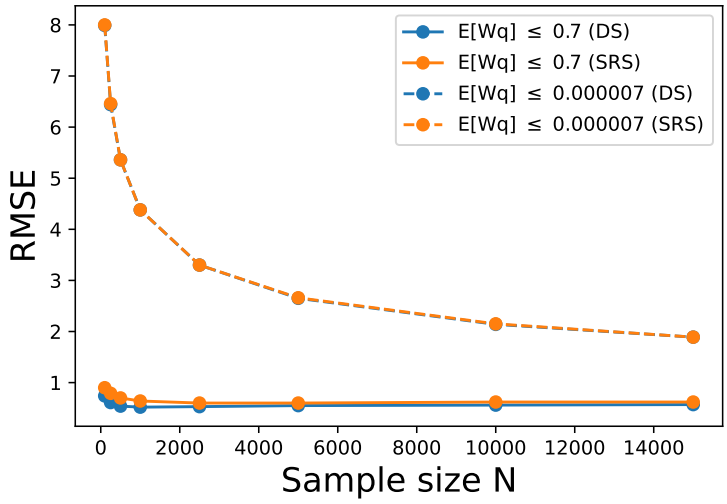


(c) Sample size $N = 500$

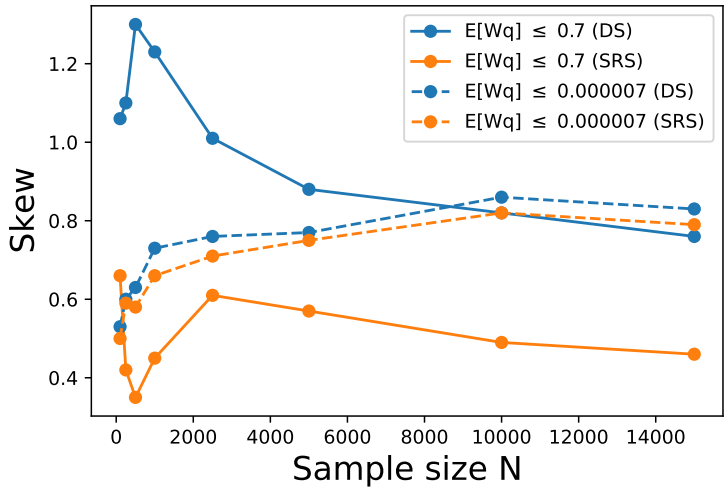


(d) Sample size $N = 1000$

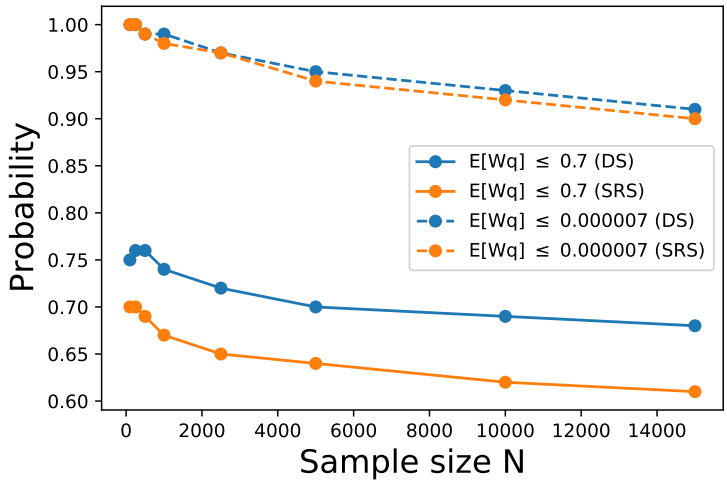
Figure 3.13: Optimal number of servers in an M/M/c system with X/Y service level for sample sizes $N = 100$ to $N = 1000$



(a) RMSE



(b) Skew



(c) Probability

Figure 3.14: Overview of distribution of results for the optimization of M/M/c staffing problem

the distribution of the optimal decision is not symmetrical and this effect is stronger for descriptive sampling in the analyzed examples.

Even though we only analyzed four instances, it can be concluded that managers should be aware that the distribution of the resulting performance measures or optimal solution may not be symmetrical and the chosen sampling method may have an impact on this behavior. In order to avoid a sampling bias, managers could perform a similar analysis as shown in this paper to understand the sensitivity of their problem in the sample size and the sampling method.

Further research could analyze if our findings can be generalized, both for the analyzed systems and for the performance evaluation of other systems or the optimization of different problems. Therefore, further instances of the analyzed systems could be considered. Other assumptions on the distribution and the analyzed systems could be used to provide further insights. The analysis of further sampling methods, such as latin hypercube sampling, could further be investigated. Furthermore, analyzing the impact of the sampling method analytically could be a fruitful direction for research. A deeper statistical analysis on the sampled series could further increase the understanding of these effects, for example by analyzing the auto-correlation of the random numbers in the sequence.

4 Ramp-up with stochastic and non-stationary yield: Insights from a Newsvendor approach

Co-authors:

Johannes Diefenbach
Chair of Production Management, Business School, University of Mannheim, Germany

Justus Arne Schwarz
Chair of Production Management, Faculty of Business, Economics and Management Information Systems, University of Regensburg, Germany

Raik Stolletz
Chair of Production Management, Business School, University of Mannheim, Germany

Fikri Karaesmen
Department of Industrial Engineering, Koç University, Istanbul, Turkey

Working paper.

Abstract:

Our research is motivated by the planning problem of a global manufacturer of semiconductors. During the introduction of a new product or machine, the yield of a production process tends to start low and increases with the production quantity. This is known as the ramp-up phase and the company has to chose the production quantity during the ramp-up phase ex ante. Demand is known, but yield is stochastic and non-stationary, following a known learning curve. The ramp-up quantity has to be chosen such that the expected profit is maximized.

Yield problems occur when some of the produced products do not meet the quality specifications and can therefore not be used to fulfill the demand. The analysis of real yield data from the company shows that such a stochastic and non-stationary yield behavior occurs both for the introduction of new products as well as for the introduction of new machines.

We formalize the company's problem as a Newsvendor problem with stochastic and non-stationary yield. We derive analytical and numerical insights on the optimal ramp-up quantity and the expected profit.

We prove that the expected profit is a discrete concave function of x for stationary yield. However, this property does not hold for the case of increasing yield. In addition, any positive optimal production quantity will always be at least the demand. We characterize the optimal production quantity for stationary yield by a critical fractile.

A numerical study shows that the optimal production quantity using the proposed model is close to the ex-post optimal production quantity from the data. An increase of the learning curve increases the expected yield for each produced unit. The optimal ramp-up quantity tends to be decreasing in the expected yield. However, a numerical analysis shows that an increase in the expected yield can lead to a higher optimal production quantity at first, before the production quantity decreases.

4.1 Introduction

4.1.1 Motivation: Introduction of a new product or machine

During the introduction of a new product or a new machine, the output of a production process tends to start low and is increasing with the production quantity. This is known as the ramp-up phase. With today's frequent technology changes, ramp-ups are becoming ever more important, especially in high-tech industries, such as semiconductor manufacturing.

Ever shortening product life cycles put high pressure on firms' production capacities (Milor, 2013). The performance during the ramp-up phase plays a crucial role for the economic success of a company (Weber, 2004). For example, Intel faced a severe supply shortage in 2018, which severely impacted its financial performance (Kim, 2018). Because of the introduction of its 10 nanometer chips, Intel continued to struggle to raise its supply into the third quarter of 2019 (Hruska, 2019). While ramping up the production for the iPhone X, Apple faced supply shortfalls due to the complexity of producing organic light-emitting diode (OLED) screens (Kubota et al., 2017).

Yield problems during the ramp-up are especially important for the economic situation of a company (Bohn and Terwiesch, 1999). They occur when some of the produced products do not meet the specifications and can therefore not be used to fulfill the demand (Cunningham et al., 1995). In the semiconductor manufacturing industry, as well as in many other industries, stochastic yield is common (e.g. Ehrhardt and Taube, 1987; Lee and Yano, 1988; Tang et al., 2012). Yield problems are typically observed during the ramp-up phase (Bohn and Terwiesch, 1999). It tends to be low at the beginning and increases afterwards (Li and Zheng, 2006; Grasman et al., 2007). This non-stationary yield behavior can be modeled by a learning curve.

Learning curves have been applied in a wide variety of decision problems, see for example the review by Biskup (2008) on scheduling decisions with learning. Other examples include the assignment of tasks to stations in a paced line with learning (Toksarı et al., 2008) or of workers to manufacturing cells with learning and forgetting (Liu et al., 2016). Different optimization problems have considered production planning with a learning curve depending on the past production (e.g. Mazzola et al., 1998). Usually, the production process is assumed to be deterministic, but follows a learning curve. Cavagnini et al. (2020) consider a workforce production

planning problem where the parameters of a linear learning curve are unknown, but the process itself is deterministic. As all these problems apply a learning curve to a deterministic process, they are not suitable to capture an optimization problem with stochastic yield and a learning curve.

4.1.2 Example: Semiconductor manufacturing

Our research is motivated by the ramp-up in the manufacturing of semiconductors. A detailed description and in-depth analysis of the observed data is carried out in Section 4.3. We observe three classes of ramp-ups with respect to the yield behavior: (1) deterministic and stationary yield, (2) stochastic and stationary yield, and (3) stochastic and non-stationary yield.

We analyze the situation, in which the company is planning the start of a new product or a new machine. There is a separation in planning between the ramp-up phase and the steady-state phase. The expected yield during the ramp-up is non-stationary and follows a learning curve. The considered company operates in a business-to-business setting and produces specific products for its customers. The quantities to be delivered by the semiconductor company are contracted. Therefore, the demand can be considered to be known in advance. Lead times are long, sometimes up to several months, forcing the company to decide on the ramp-up quantity before the realization of the stochastic yield is known. Even though the company may produce in lots, lot-sizes can vary and the decision is on the number of products (chips) to be produced. Therefore, there is a single production decision with a stochastic quantity of resulting non-defective end products to meet a known demand. In case of a supply shortage, the company loses revenue and faces contracted shortfall costs. In case of overproduction, the overproduced units can be sold at a lower value in the steady-state phase. This lower value represents both the decreasing prices in the high-tech industry as well as holding costs. The company aims to maximize its expected profit.

4.1.3 Problem description and contributions

We formalize the company's problem as a Newsvendor problem with stochastic and non-stationary yield. The demand is known and the corresponding ramp-up quantity has to be chosen to maximize the expected profit. Yield is stochastic and the probability of each unit to be non-defective follows a learning curve. Therefore, the

quantity of non-defective end products is modelled as a Poisson Binomial distribution. We assume that the learning curve is known but do not make specific assumptions about its shape. Revenue is incurred for the share of the demand that can be fulfilled by non-defective end products. Costs are composed of variable production cost and penalty cost for demand shortfalls. Non-defective end products exceeding the demand can be sold at a salvage value at the end of the ramp-up phase. Choi et al. (2019) have analyzed the special case of stationary yield, i.e. the probability of each unit to be non-defective does not change with the production quantity, and no salvage value.

Our research is motivated by an application in the semiconductor industry, but not limited to it. Rather, it applies to any production setting with stochastic and non-stationary yield and a singular decision on the production quantity. This paper aims to analyze the structure of the optimal ramp-up quantity with non-stationary yield. The main contribution of this paper is summarized as follows.

1. We present a Newsvendor model with stochastic and non-stationary yield dependent on the production quantity.
2. We derive analytical insights on the expected profit and the optimal ramp-up quantity. We prove a bound on the optimal ramp-up quantity and on the expected profit for any learning curve. For increasing yield, we show that a positive production quantity below the demand is never optimal. For stationary yield, we characterize the optimal ramp-up quantity by a critical fractile and show the sensitivity of a change in the model parameters on the optimal ramp-up quantity. We derive the impact of a change in the cost parameters on the expected profit.
3. A numerical study compares our model with the ex-post optimal decision derived from the data. We find that the optimal ramp-up quantity from the model is close to the optimal ex-post quantity. Furthermore, our numerical study suggests that the monotony of the optimal ramp-up quantity in the model parameters extends to the case of increasing yield. Finally, our numerical study shows that an increase in the expected yield can first lead to more production before it leads to less production, i.e. the optimal ramp-up quantity is non-monotone in the expected yield.

4.1.4 Structure of the paper

The remainder of this paper is organized as follows. We provide an overview of the literature on learning curves and Newsvendor problems with stochastic yield in Section 4.2. A detailed description and analysis of the yield data from the semiconductor industry is given in Section 4.3. The Newsvendor model with stochastic and non-stationary yield is presented in Section 4.4. In Section 4.5, we derive analytical insights on the expected profit and the optimal ramp-up quantity. The numerical study in Section 4.6 compares our model to the ex-post solution using real data, analyzes the monotony of the optimal ramp-up quantity and performs a sensitivity analysis of the expected yield on the optimal ramp-up quantity. We summarize our findings in Section 4.7.

4.2 Literature on learning curves and Newsvendor problems with stochastic yield

We analyze a non-stationary behavior of the yield, which is also known as a learning curve. The literature on learning curves with stochastic processes focuses on predicting the learning curve for a set of features. Therefore, many different yield learning curves have been proposed for various specific situations, such as the area in the production process or the type of produced product. In many cases, the literature is concerned with including the right features for a good prediction of the learning curve. However, we assume that the learning curve is known and therefore present the literature on learning curves only briefly in Section 4.2.1. As we formalize the company's optimization problem as a Newsvendor model with stochastic and non-stationary yield, we review the literature on Newsvendor models with stochastic yield in Section 4.2.2.

4.2.1 Learning curves

The literature on learning curves is vast and goes back to Wright (1936), who observed that marginal production cost decrease with the production quantity. An early review on learning curves is given by Yelle (1979), presenting decreasing learning curves with respect to the required production time or cost per unit.

A stream of literature reviews yield learning curves with a focus on predicting the yield learning curve in semiconductor manufacturing. Tirkel (2013) reviews learn-

ing curves based on the production quantity and multivariate learning curves using other inputs in addition to the production quantity. Milor (2013) investigates yield models for the wafer probe data, sources of defects and in-line excursion detection. Kumar et al. (2006) also incorporate other factors, such as spatial defects and radial yield losses, which might improve the prediction quality of the yield learning curve, in their review.

The reviews by Anzanello and Fogliatto (2011) and Grosse et al. (2015) include learning curves both for decreasing costs per unit as well as for an increasing output. Anzanello and Fogliatto (2011) further differentiate between univariate and multivariate learning curves, i.e. predicting the learning curve based on a single or on multiple features. For an increase in productivity of a single product or machine, such as the yield, Grosse et al. (2015) present four different learning curves, which are summarized in Table 4.1.

Two parameter exponential yield (2PE)	$p(i) = k\left(1 - e^{-\left(\frac{i}{R}\right)}\right)$
Three parameter exponential yield (3PE)	$p(i) = k\left(1 - e^{-\left(\frac{i+m}{R}\right)}\right)$
Two parameter hyperbolic yield (2PH)	$p(i) = k\left(\frac{i}{i+R}\right)$
Three parameter hyperbolic yield (3PH)	$p(i) = k\left(\frac{i+m}{i+m+R}\right)$

Table 4.1: Yield learning functions dependent on production quantity (Grosse et al., 2015)

In all functions, $p(i)$ represents the probability of the i^{th} produced unit to be non-defective. With respect to our problem from Section 4.1.2, i refers to each single unit produced within the ramp-up quantity x (i.e. $i = 1, \ldots, x$). The parameter k represents the final yield, i.e. the value to which the function converges. The learning rate R determines how fast the yield is changing, where a low R represents a fast increase. The previous experience, for example from the ramp-up of a similar product or machine, is given by m. There are two classes of functions: exponential yield and hyperbolic yield. Both can be expressed with two parameters (2PE and 2PH) or with three parameters (3PE and 3PH). The three parameter models extend the two parameter models by the experience m, making the two parameter models special cases.

To summarize, learning curves are important to predict the output of many different systems. Furthermore, they are especially important for the planning in the semiconductor manufacturing industry. Learning curves have been applied in a wide variety of optimization problems for deterministic production systems. In order to

analyze the optimal ramp-up quantity, we review the literature with a single production decision and stochastic yield in Section 4.2.2.

4.2.2 Newsvendor models with stochastic yield

For different planning problems with stochastic yield, we refer the reader to Yano and Lee (1995). According to their classification, we consider a single-period model with a single production run. Therefore, this section reviews the literature with models considering a single decision on the production quantity with stochastic yield. This type of problem is also known as Newsvendor models with stochastic yield. For a broader review on Newsvendor problems under various assumptions we refer the reader to Khouja (1999).

Consider a production quantity x. We denote the resulting quantity of non-defective end products by $Q(x)$. The literature can be divided into four different categories with respect to the modelling of stochastic yield.

Additive yield can for example be motivated by agriculture yield dependent on the weather (Keren, 2009) and is defined by:

$$Q(x) = x + \xi \,, \tag{4.1}$$

where ξ is the random error with support $[a, b]$ (with $-x \leq a \leq x \leq b$). Newsvendor models with additive yield has been analyzed in Rekik et al. (2007); Keren (2009); Li et al. (2012).

Applications for *proportional yield*, also known as multiplicative yield, include perishable goods (Shih, 1980), remanufacturing (Inderfurth, 2004) or blood banks (Gerchak et al., 1988), among many others. Proportional yield is defined by:

$$Q(x) = x \cdot Y \,, \tag{4.2}$$

where Y is the random yield and most authors assume Y has support $[0, 1]$. This model has been studied widely, e.g. Shih (1980); Tang and Yin (2007); Okyay et al. (2014).

The model of proportional yield has been extended to incorporate *random capacity*, where equipment might malfunction, resulting in a random capacity K (Okyay et al., 2015). Therefore, the production quantity is the minimum of x and the realized capacity K. The resulting quantity of non-defective end products may further

be reduced by stochastic yield, resulting in:

$$Q(x) = Y \cdot \min\{K, x\} \, . \tag{4.3}$$

Random capacity is studied in Wang and Gerchak (1996); Sayın et al. (2014); Okyay et al. (2015).

The models with additive yield, proportional yield and random supply assume that the distribution of the random yield, random error and random capacity is independent of the production quantity. In contrast, models with *yield as a function of the production quantity* assume that the quantity of non-defective end products follows a distribution, which has parameters dependent on the production quantity. For simplicity, we call this distribution the *yield distribution*. Applications include a Gardener (Abdel-Malek et al., 2008; Abdel-Malek and Otegbeye, 2013) and the production of semiconductors (Noori and Keller, 1986; Choi et al., 2019) among others.

	Yield	**Demand**	**Objective**
Abdel-Malek et al. (2008)	Uniform: $Q(x) \sim U(a, x)$	Uniform	Minimize cost
Abdel-Malek and Otegbeye (2013)	Uniform: $Q(x) \sim U(a, x)$	General continuous	Maximize profit
Noori and Keller (1986)	Normal: $Q(x) \sim N(px, \sigma^2)$ Normal: $Q(x) \sim N\left(px, (sx)^2\right)$ Gamma: $Q(x) \sim G(n = p^2/s^2, \alpha = s^2 x/p)$	Uniform, Exponential	Minimize cost
Gallego and Moon (1993)	Binomial: $Q(x) \sim Bin(p, x)$	General worst case	Maximize profit
Alfares and Elmorra (2005)	Binomial: $Q(x) \sim Bin(p, x)$	General worst case	Maximize profit
Choi et al. (2019)	Binomial: $Q(x) \sim Bin(p, x)$ Normal: $Q(x) \sim N\left(px, \sqrt{xp(1-p)}^2\right)$	Deterministic	Maximize profit

Table 4.2: Literature with yield as a function of production quantity

Table 4.2 reviews the literature, where the quantity of non-defective end products $Q(x)$ follows a distribution whose parameters depend on the production quantity. The table reviews the yield distribution, demand distribution and objective function. For the distribution of yield, Uniform, Normal, Gamma and Binomial distribution of yield have been analyzed. Uniformly distributed yield assumes a fixed lower value a and the upper bound of the interval is x. Even though the mean probability of each unit changes with a change of the production quantity, all units have the same expected probability for a fixed production quantity, i.e. the Uniform distribution does not capture a learning curve for the yield. For the other distributions, a stationary parameter p specifies the expected probability of each unit to be non-defective. In all cases, p is independent of the production quantity, i.e. the presented literature only considers stationary yield and does not consider a learning curve on the probability p. Both stochastic and deterministic demand have been considered.

The objectives feature the minimization of expected cost or the maximization of expected profit.

In conclusion, there is a wide variety of literature analyzing Newsvendor models with stochastic yield, where the distribution of yield is independent of the production decision. For the literature with yield as a function of the production quantity, only stationary yield is considered. Since we want to analyze the optimal ramp-up quantity for stochastic and non-stationary yield, there is a gap in the literature for a model with non-stationary yield. The literature on predicting learning curves is large and many different learning curves have been applied. Therefore, we present a general model in Section 4.4, which does not have a specific assumption on the learning curve. In Section 4.5, we prove properties of the optimal production quantity and optimal objective value for different classes of learning functions.

4.3 Yield analysis for ramp-ups in semiconductor manufacturing

This section describes and analyzes the data we were provided by a global manufacturer of semiconductors. First, we describe the data in general and the applied data cleaning process. Afterwards, we present the data for the introduction of new products in the area wafer probe in Section 4.3.1 and for new machines in Section 4.3.2. The new machines are spread over the entire process of semiconductor manufacturing and their realized yield is measured directly at the machine. In contrast, the yield of the new products is always measured at the end of the area wafer probe. We have two data sets, each spanning approximately three years of production.

The data considers the production process of wafers, which are thin slices used in semiconductor manufacturing, where each wafer consists of a large number of microchips. The manufacturer produces its microchips in lots, where each lot is a batch of wafers. The data consists of the average realized yield for each lot. Each data point represents a single processing step of a single lot in the production process. As the production quantity per lot is not fixed, we consider the realized average yield in relation to the cumulative production quantity.

Data cleaning is the first step in the analysis of yield data in semiconductor manufacturing (Lee et al., 2019). After discussions with process experts from the manufacturing company, we apply the following cleaning rules to the data sets. A data point is removed, if

1. its processing time is negative,
2. its yield is above one,
3. its yield is zero or lower.

For the analyzed data sets, the process experts explained that a yield of zero is most probably caused by a split in the lot, which cannot be traced afterwards. Therefore, we remove these data points. A lot is removed completely, if its first processing step contains less units than its last processing step. Finally, we only consider products or machines with at least 10 lots.

4.3.1 New product introduction

The first data set contains the realized yield of new product introductions in the area wafer probe. For each lot, the functionality of each individual chips is tested. Therefore, the realized average yield is the number of non-defective chips in a lot divided by the total number of chips in that lot.

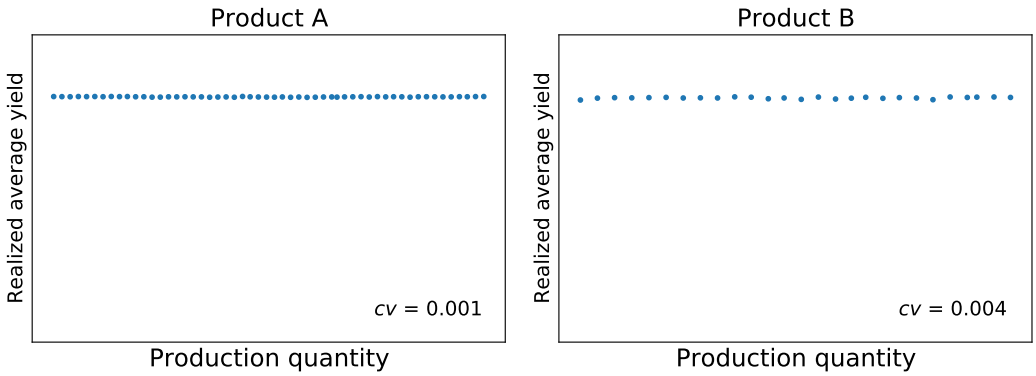


Figure 4.1: New product introductions with deterministic and stationary yield

Figures 4.1 to 4.3 show the realized average yields depending on the production quantity for different products and their corresponding coefficient of variation (cv). We observe that some products have a very low coefficient of variation of $cv \leq 0.004$, which we consider to be deterministic yield (Figure 4.1), while others are stochastic (Figures 4.2 and 4.3). All products with deterministic yield show a stationary yield behavior. For the products with stochastic yield, some have stationary yield (Figure 4.2) while others show a non-stationary yield behavior (Figure 4.3).

To summarize, three different classes of products can be observed. Ramp-ups of new products with stochastic and non-stationary yield are common. As we are interested in the optimal ramp-up quantity for non-stationary yield, we further an-

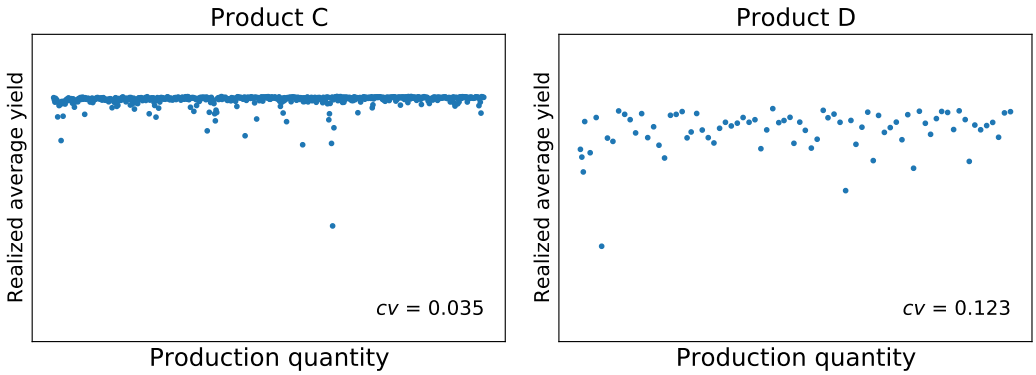


Figure 4.2: New product introductions with stochastic and stationary yield

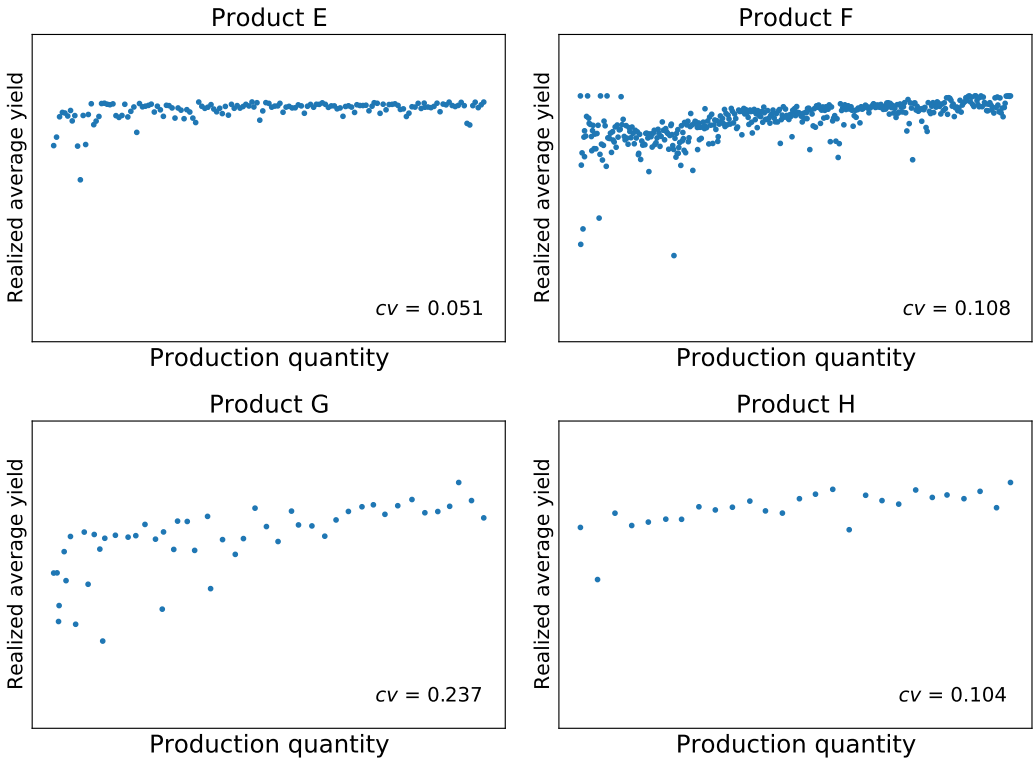


Figure 4.3: New product introductions with stochastic and non-stationary yield

alyze the examples from Figure 4.3 by fitting different learning curves to them in Section 4.3.3.

4.3.2 New machine introduction

The second data set contains newly introduced machines in the entire production process of the semiconductor manufacturer with more than 10 lots. We show some examples of those machines and, again, we report the realized average yield, which is the number of non-defective chips in a lot divided by the total number of chips in that lot.

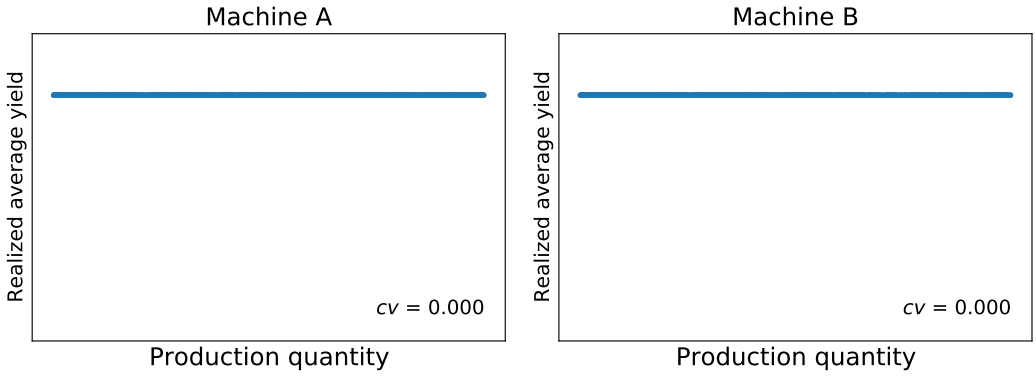


Figure 4.4: New machine introductions with deterministic and stationary yield

Figures 4.4 to 4.6 show the realized average yield depending on the production quantities for different machines and their corresponding coefficients of variation (cv). For the introduction of new machines, we also observe (1) deterministic and stationary yield, (2) stochastic and stationary yield, and (3) stochastic and non-stationary yield. These are the same classes as we have observed for the introduction of new products in Figures 4.1 to 4.3.

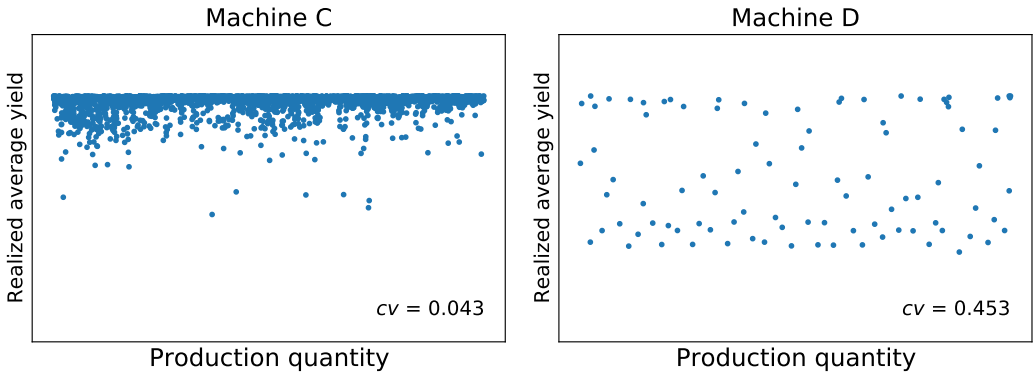


Figure 4.5: New machine introductions with stochastic and stationary yield

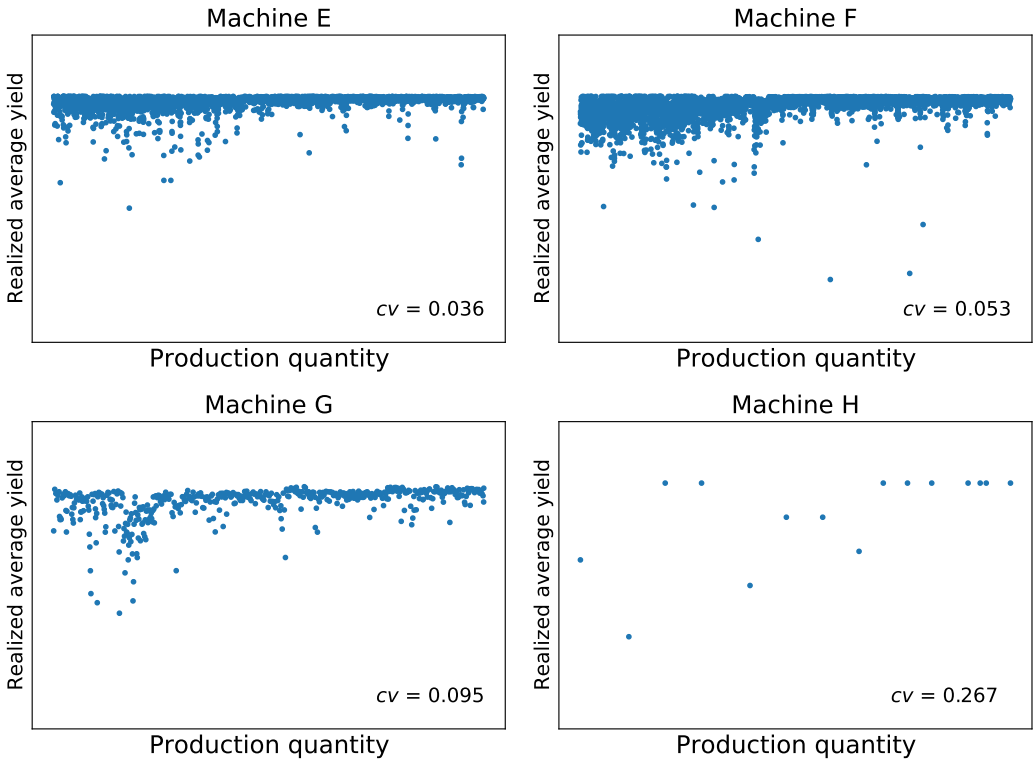


Figure 4.6: New machine introductions with stochastic and non-stationary yield

To summarize, we observe the same three classes for the introduction of a new machine as for the introduction of a new product. In both cases, stochastic and non-stationary yields can be observed. As we are interested in the optimal ramp-up quantity for non-stationary yield, we further analyze the examples from Figure 4.6 by fitting different learning curves to them in Section 4.3.3.

4.3.3 Fitting learning curves to yield data

To further analyze the examples of ramp-ups with stochastic and non-stationary yield from Figures 4.3 and 4.6, we fit the learning curves presented Table 4.1 from the literature review in Section 4.2.1 to the data. We analyze, if the learning curves can be used to model the behavior of the non-stationary yield. For each fitted learning curve, we report the corresponding coefficient of determination R^2. As the two-parameter learning curves from Table 4.1 are special cases of the three parameter learning curves, they can never have a better coefficient of determination R^2 in our analysis. Therefore, we only report the results of the three parameter learning curves.

Tables 4.3 and 4.4 show the original yield data and the different yield learning curve for the introduction of a new product and machine, respectively. Each column corresponds to a learning curve (3 parameter exponential (3PE) and 3 parameter hyperbolic (3PH)). We fit the learning curves using *curve_fit* from Python's *SciPy* library, which uses non-linear least squares to fit a function to data.

The fitted learning curves have values of R^2 between 0.11 and 0.56. In general, the resulting coefficients of determination R^2 are higher for the fitted learning curves of the new product introductions. There is no significant difference between the fit of the exponential and the hyperbolic yield curve. In conclusion, both cases can be modeled using three parameter exponential or hyperbolic yield.

4.3.4 Summary of yield data

To summarize, (1) deterministic and stationary yield, (2) stochastic and stationary yield, and (3) stochastic and non-stationary yield can be observed in the data for new product introduction and new machine introduction. While the literature has focused on models with stochastic and stationary yield, non-stationary yield behavior can be observed in the data. Therefore, there is a practical need for a model to capture the non-stationary behavior of the stochastic yield during the ramp-up of a new product or a new machine. Thus, the model we present in Section 4.4 features stochastic and non-stationary yield. However, stochastic and stationary can also be captured as a special case. Therefore, the model allows to gain insights into both presented classes of stochastic yield.

The three parameter exponential and hyperbolic yield curves can be used to model the non-stationary yield behavior. There is no significant difference in the coefficient of determination between both. However, the model we present in Section 4.4 does not rely on a specific assumption on the learning curve.

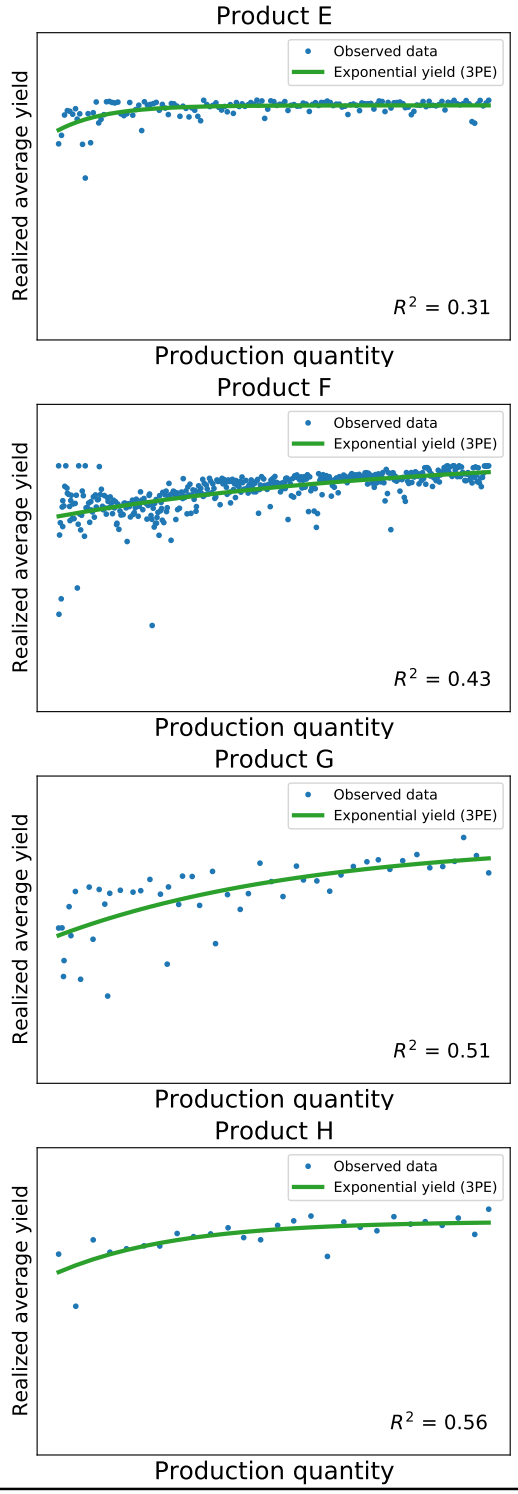
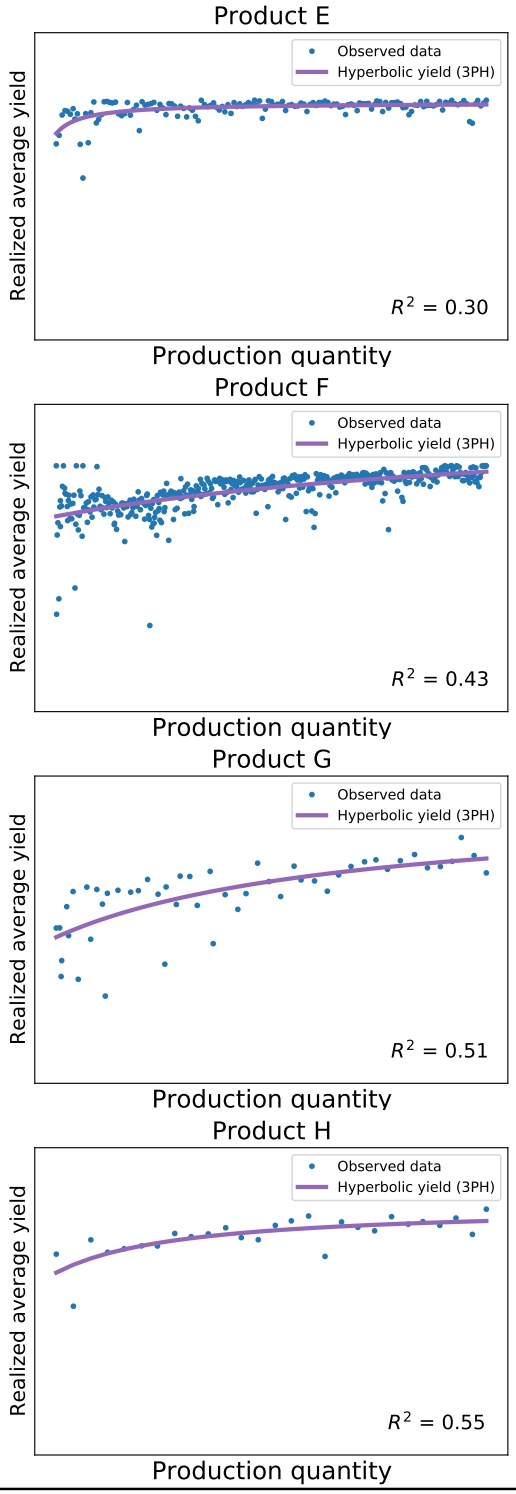
Exponential yield (3PE)	Hyperbolic yield (3PH)
	

Table 4.3: Fitted exponential and hyperbolic yield for realized average lot yield of different products in the production area wafer probe

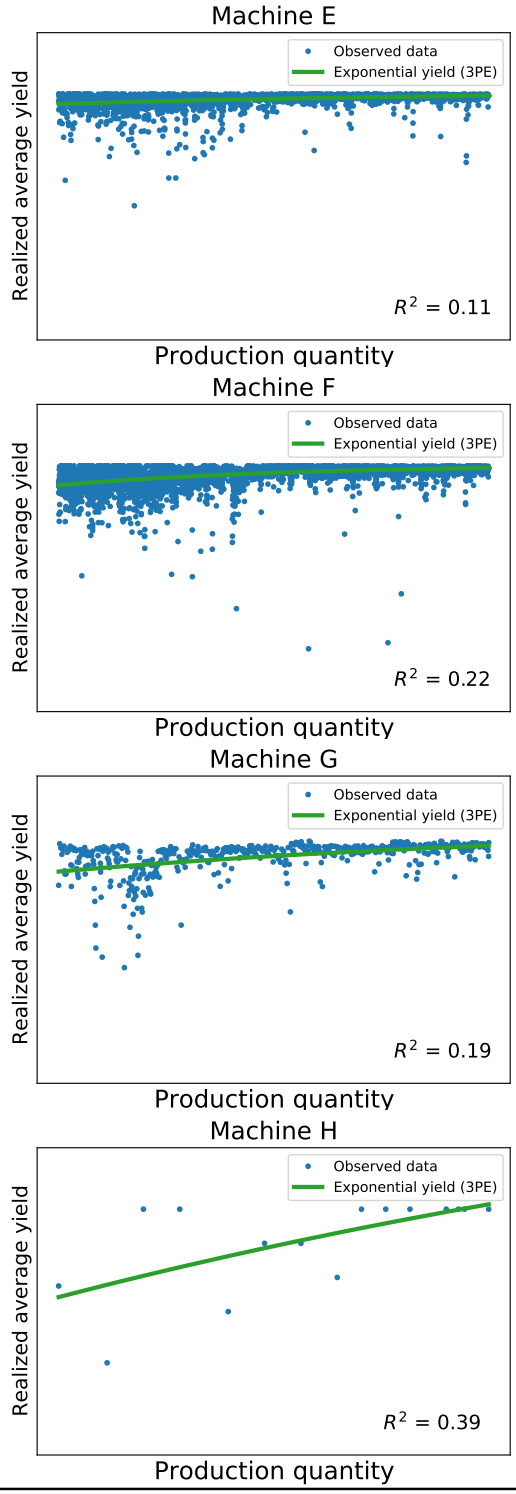
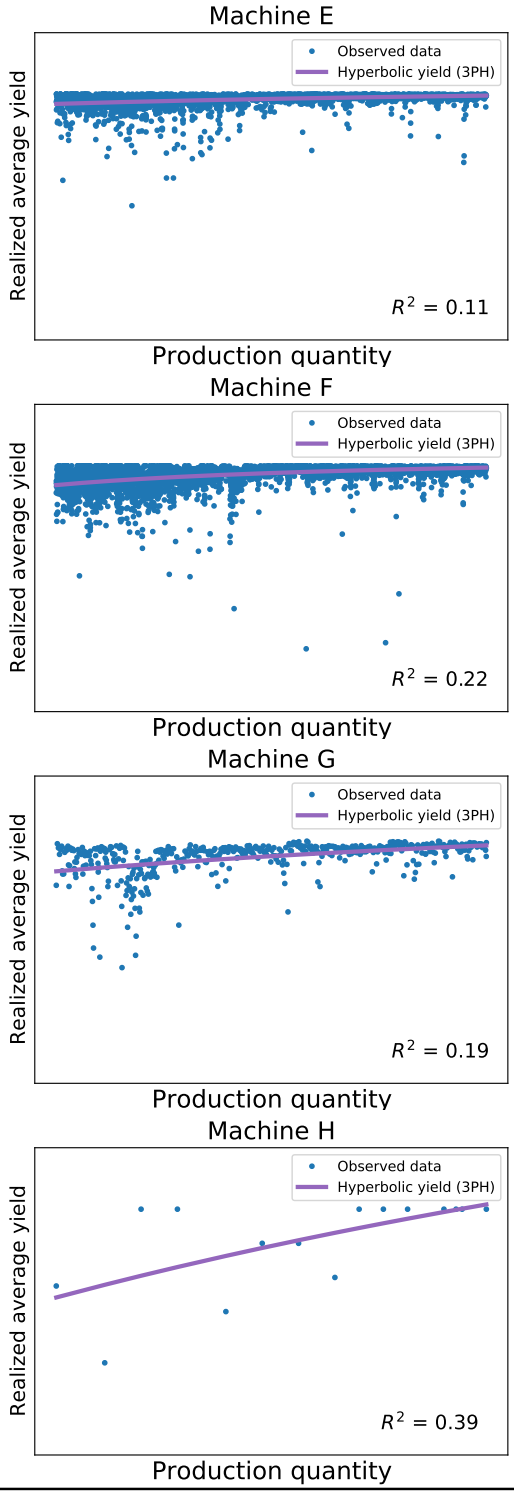
Exponential yield (3PE)	Hyperbolic yield (3PH)
	

Table 4.4: Fitted exponential and hyperbolic yield for realized average lot yield of different new machine introductions

4.4 Newsvendor model with stochastic and non-stationary yield

Index:	
i	i^{th} produced product ($i = 1, \ldots, x$)
Parameters:	
n	Demand for non-defective end product
c	Unit cost per produced end product
r	Unit revenue per non-defective end product
b	Unit penalty cost for unsatisfied demand
s	Unit salvage value per overproduced non-defective end product
$p(i)$	Probability of i^{th} product to be non-defective (non-stationary yield)
Decision variable:	
x	Production quantity ($x \in \mathbb{N}$)
Random variable:	
$Q(x)$	Non-defective end products if x products are produced (depends on $p(i)$)

Table 4.5: List of notation

We consider the decision on the production quantity x to maximize the expected profit $\mathbb{E}[\Pi(x, Q(x))]$ for a known demand n. Each product is either defective or non-defective. The profit is driven by the unit cost per produced end product c (with $c > 0$), unit revenue per sold non-defective end product r (with $r \geq c$) and penalty cost for each unit of unsatisfied demand b (with $b \geq 0$). All non-defective end products, which exceed the demand, can be sold at a salvage value s (with $s < c$). This leads to the following expected profit:

$$\begin{aligned} &\max_{x \in \mathbb{N}} \ \mathbb{E}[\Pi(x, Q(x))] \\ &= \mathbb{E}[r \cdot \min(Q(x), n) + s \cdot (Q(x) - n)^+ - b \cdot (n - Q(x))^+ - cx] \,. \end{aligned} \tag{4.4}$$

To account for non-stationary yield, we introduce non-stationary probabilities for each product i to be non-defective. We assume the yield follows a Poisson Binomial distribution with the probability $p(i)$ of the i^{th} product to be non-defective (with $i = 1, ..., x$). The probability of receiving q non-defective products for a production quantity x is defined by

$$Pr\{Q(x) = q\} = f(x, q) = \sum_{A \in F_q} \prod_{i \in A} p(i) \prod_{j \in A^c} (1 - p(j)) \tag{4.5}$$

where F_q is the set of all subsets A of q integers that can be selected from $\{1, 2, 3, \ldots, x\}$. Equation (4.5) is the probability mass function of the Poisson Binomial distribution. Using Equation (4.5), we can express the expected profit in Equation (4.4) by

$$\max_{x \in \mathbb{N}} \mathbb{E}[\Pi(x, Q(x))] = r \sum_{q=0}^{x} q\, f(x, q) - (r - s) \sum_{q=n}^{x} (q - n)\, f(x, q) - b \sum_{q=0}^{n} (n - q)\, f(x, q) - cx\,. \tag{4.6}$$

The non-stationary yield is reflected by the function $p(i)$. Note that the model of Choi et al. (2019) is included as a special case for stationary yield $p(i) = p(i+1) = p$ and a salvage value of $s = 0$. The analytical results of Choi et al. (2019) for a Normal approximation of this special case are summarized in Appendix G.

4.5 Analytical insights on expected profit and optimal ramp-up quantity

In this section, we present analytical results for the problem defined in Section 4.4. We analyze the properties for stationary and for increasing yield. First, we analyze properties of the objective function, such as concavity and an upper bound on the expected profit. This is followed by insights on the optimal ramp-up quantity.

This section shows how the structure of the objective value changes when considering increasing yield instead of stationary yield. We derive bounds on the optimal ramp-up quantity for increasing yield. Therefore, in order to find the optimal solution, searching a finite number of discrete values in this range is sufficient. Furthermore, the optimal ramp-up quantity is characterized for stationary yield in the form of a critical fractile. In addition, this section shows the impact of the cost parameters and the demand both on the expected profit as well as the optimal ramp-up quantity for stationary yield.

4.5.1 Insights on expected profit

From Equation (4.4), it follows that the expected profit is non-increasing in production cost c if $x = 0$ and decreasing in c for a given $x > 0$. Furthermore,

$\mathbb{E}[\Pi(x, Q(x))]$ is non-increasing in the backlog cost b and non-decreasing in the unit revenue r and the unit salvage value s.

To characterize the shape of the expected profit, we first define the condition on discrete concavity. Afterwards, this condition is applied to stationary yield functions.

Lemma 1 (Condition for discrete concavity)**.** $\mathbb{E}[\Pi(x, Q(x))]$ *is a discrete concave function of* x*, if and only if*

$$\frac{p(x+2)}{p(x+1)} \leq \frac{(r+b-s)\ Pr\{Q(x) < n\} + s}{(r+b-s)\ Pr\{Q(x+1) < n\} + s} \tag{4.7}$$

holds for all x*.*

Proof. Let

$$\Delta(x) = \mathbb{E}[\Pi(x+1, Q(x+1))] - \mathbb{E}[\Pi(x, Q(x))] \tag{4.8}$$

be the difference in expected profit between producing $x+1$ and producing x units. Note that $\mathbb{E}[\Pi(x, Q(x))]$ is a discrete concave function of x if and only if $\Delta(x)$ is a non-increasing function of x (see Yüceer (2002, Theorem 1) for the definition of discrete convex function of a single variable). Note that when the production quantity increases from x to $x+1$, the revenue increases by r and the backlog decreases by b if item $x+1$ is non-defective and $Q(x) < n$. If $Q(x) \geq n$, the revenue and backlog stay the same. The salvage value increases by s if item $x+1$ is non-defective and $Q(x) \geq n$. The salvage value is unchanged if $Q(x) < n$. The production cost increase by c independently of $Q(x)$. Therefore, the difference in Equation (4.8) can be reformulated as

$$\begin{aligned}\Delta(x) &= r\ p(x+1)\ Pr\{Q(x) < n\} + b\ p(x+1)\ Pr\{Q(x) < n\} \\ &\quad + s\ p(x+1)\ Pr\{Q(x) \geq n\} - c \\ &= (r+b-s)\ p(x+1)\ Pr\{Q(x) < n\} + s\ p(x+1) - c\,. \end{aligned} \tag{4.9}$$

Equation (4.9) is a non-increasing function of x for all x, if and only if

$$\begin{aligned} &\Delta(x+1) \leq \Delta(x) \\ \Leftrightarrow\ &(r+b-s)\ p(x+2)\ Pr\{Q(x+1) < n\} + s\ p(x+2) - c \\ &\quad \leq (r+b-s)\ p(x+1)\ Pr\{Q(x) < n\} + s\ p(x+1) - c \\ \Leftrightarrow\ &\frac{p(x+2)}{p(x+1)} \leq \frac{(r+b-s)\ Pr\{Q(x) < n\} + s}{(r+b-s)\ Pr\{Q(x+1) < n\} + s}\,. \end{aligned}$$

□

Lemma 1 defines the condition for discrete concavity of the expected profit. This condition is independent of the production cost c, but depends on all other parameters. Next, this condition is applied to stationary yield functions.

Theorem 2 (Concavity for stationary yield)**.** *For $p(i) = p$, $\mathbb{E}[\Pi(x, Q(x))]$ is a discrete concave function of x.*

Proof. For $p(i) = p$, the left hand side of the condition for discrete concavity in Equation (4.7) reduces to

$$\frac{p(x+2)}{p(x+1)} = \frac{p}{p} = 1\,. \tag{4.10}$$

Therefore, Inequality (4.7) can be rewritten as

$$\begin{aligned} 1 &\leq \frac{(r+b-s)\ Pr\{Q(x) < n\} + s}{(r+b-s)\ Pr\{Q(x+1) < n\} + s} \\ \Leftrightarrow\ & Pr\{Q(x+1) < n\} \leq Pr\{Q(x) < n\} \end{aligned} \tag{4.11}$$

$Pr\{Q(x) < n\}$ is a discrete decreasing function of x for any yield $p(i)$. Therefore, $Pr\{Q(x) < n\} \geq Pr\{Q(x+1) < n\}$ holds and Inequality (4.11) is fulfilled. □

Therefore, stationary yield always leads to a discrete concave function of x. However, this property is lost when considering increasing yield, as counterexamples can be constructed. For an intuition for this change, consider an increasing yield function with low starting yield. For the first produced units, the full production cost c have to be incurred while the probability of receiving a non-defective end product is low. Therefore, the expected revenue of producing the first units can be below its production cost and the expected profit declines with each produced unit. As the yield increases, the expected profit of producing further units becomes positive and the expected profit increases. Therefore, the expected profit is not a discrete concave function of x for increasing yield.

In addition to analyzing the shape of the expected profit, an upper bound on the expected profit can be derived.

Lemma 2 (Upper bound on expected profit)**.** *The expected profit* $\mathbb{E}[\Pi(x, Q(x))]$ *in Equation* (4.6) *is bounded above by*

$$\Pi(x, Q(x) = x) = \begin{cases} (r + b - c)x - bn & x < n \\ rn + s(x - n) - cx & x \geq n \, . \end{cases} \tag{4.12}$$

Proof. We use the special case of a (deterministic) yield of 100%, i.e. $p(i) = 1$ for all i. The corresponding quantity of non-defective end products $Q(x)$ is equal to the production quantity x, $Q(x) = x$. The profit can be expressed by Equation (4.12).

Consider two quantities of non-defective end products $Q(x)$ and $\bar{Q}(x)$ with different underlying non-stationary yields. Assume that the non-stationary yield $p(i)$ is such that $E[Q(x)] = E[\bar{Q}(x)] + \epsilon$, i.e. the expected quantity of non-defective end products $Q(x)$ is always ϵ units larger than that of $\bar{Q}(x)$.

For $\bar{Q}(x) < n$, the difference in expected profit is an increase of ϵ units of revenue and a decrease of ϵ units of backlog. For $\bar{Q}(x) \geq n$, the difference in expected profit is ϵ additional salvage units. Therefore, the expected profit of $Q(x)$ is always larger than that of $\bar{Q}(x)$. Since a deterministic yield of 100% results in the largest possible quantity of non-defective end products, it poses an upper bound on the expected profit. □

The optimum of the deterministic profit with perfect yield is attained when producing exactly the demand and the resulting profit is $\Pi(n, Q(n) = rn - cn$, see Figure 4.7. Since the expected profit is bounded above by the deterministic profit with perfect yield, the optimal expected profit is bounded above by $\mathbb{E}[\Pi(x^*, Q(x^*))] \leq rn - cn$.

4.5.2 Insights on optimal ramp-up quantity

In this section, we use the insights on the expected profit to derive insights on the optimal ramp-up quantity. From the upper bound on the expected profit in Lemma 2, an upper bound on the optimal ramp-up quantity can be derived for any yield function.

Theorem 3 (Bound on optimal production quantity)**.** *The optimal production quantity* x^* *is bounded by*

$$0 \leq x^* \leq n \frac{r - s + b}{c - s} \, . \tag{4.13}$$

Proof. The expected profit for the case of no production is incurring the full backlog cost, i.e. $\mathbb{E}[\Pi(0, Q(0))] = -bn$. The expected profit $\mathbb{E}[\Pi(x, Q(x))]$ in Equation (4.6) is bounded above by Equation (4.12), see Lemma 2. For $x \geq n$, the deterministic profit is a decreasing function in x since $s < c$. For any x larger than the intersection of the upper bound and $-bn$, the upper bound on the expected profit is less than $-bn$, and therefore the expected profit is less than $-bn$, see Figure 4.7. The upper bound equals $-bn$ for $x \geq n$ for

$$\begin{aligned} rn + s(x - n) - cx &= -bn \\ \Leftrightarrow \quad x &= n\frac{r - s + b}{c - s} \end{aligned} \tag{4.14}$$

□

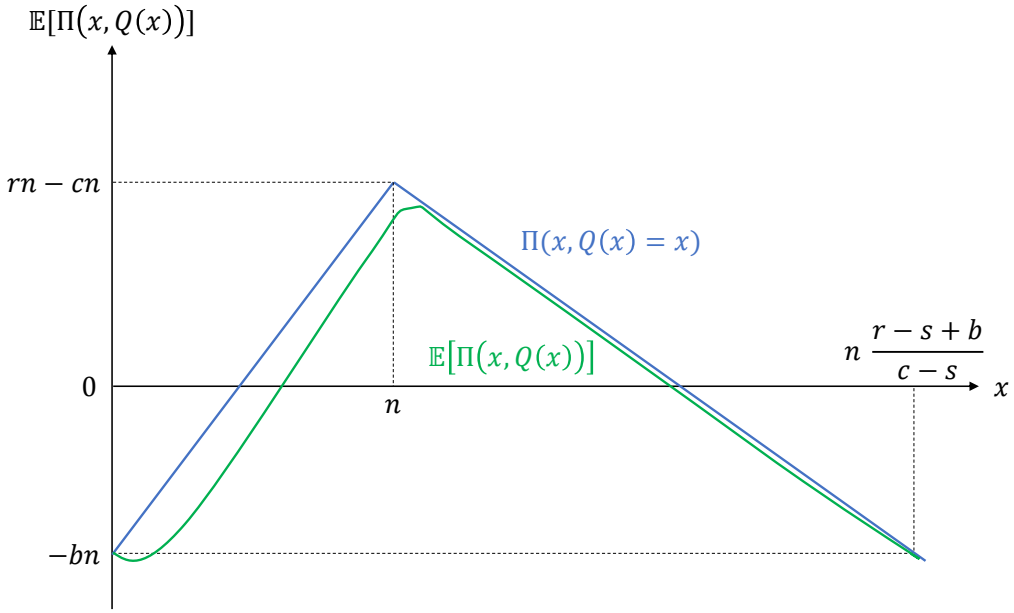


Figure 4.7: Upper bounds on the expected profit and optimal production quantity

Therefore, for any yield function $p(i)$, the optimal production quantity is bounded above. This bound depends on the demand, revenue and the cost parameters. Since we consider a discrete problem, enumerating all integer values between the bounds defined in Equation (4.13) is guaranteed to find the optimal solution. We make use of this in the Numerical Study in Section 4.6.

Theorem 4 (Unique local maximum)**.** *For $p(i) \leq p(i+1)$, there is no unique local maximum in $0 < x < n$.*

Proof. If a local optimum at x^* exists in $0 < x < n$, then

$$\mathbb{E}[\Pi(x^* - 1, Q(x^* - 1))] \leq \mathbb{E}[\Pi(x^*, Q(x^*))] \tag{4.15}$$

$$\mathbb{E}[\Pi(x^* + 1, Q(x^* + 1))] \leq \mathbb{E}[\Pi(x^*, Q(x^*))] \tag{4.16}$$

has to hold. Using the definition of the marginal profit in Equation (4.8), Equations (4.15) and (4.16) can be reformulated to

$$\Delta(x^* - 1) \geq 0 \tag{4.17}$$

$$\Delta(x^*) \leq 0 \tag{4.18}$$

For $0 < x^* < n$, i.e. $Pr\{Q(x^*) < n\} = 1$, and $p(i) \leq p(i+1)$, the marginal profit in Equation (4.9) leads to

$$0 \geq \Delta(x^*) = (r + b)\, p(x^* + 1) - c \geq (r + b)\, p(x^*) - c = \Delta(x^* - 1) \geq 0\,. \tag{4.19}$$

Hence, $0 \geq \Delta(x^*) \geq 0$. The only case, in which Equation (4.19) holds, is for $\Delta(x^*) = \Delta(x^* - 1) = 0$. Hence, if there is a local maximum in $0 < x < n$, then $\Delta(x^*) = 0$ holds for $x^* \leq x \leq n - 1$ and $x = n$ has the same objective value and there is no unique maximum in $0 < x < n$. □

Therefore, $0 < x < n$ can be excluded from the search for the global maximum for $p(i) \leq p(i + 1)$. This implies that the optimal ramp-up quantity is either 0 (no production) or to produce at least the demand n.

Lemma 2 shows that the expected profit is always a discrete concave function of x for stationary yield. The optimal production quantity is characterized in Theorem 5.

Theorem 5 (Optimal production quantity)**.** *For $p(i) = p$, the optimal production quantity is given by*

$$x^* = \min\left\{x :\ Pr\{Q(x) < n\} \leq \frac{c - sp}{p\ (r + b - s)}\right\}\ . \tag{4.20}$$

Proof. As $\mathbb{E}[\Pi(x, Q(x))]$ is a discrete concave function of x, the optimal production quantity x^* is given by the first value where the difference in expected profit between producing $x+1$ and producing x units $\Delta(x)$ (Equation (4.9)) switches from positive

to negative.

$$\begin{aligned} x^* &= \min\{x:\ \Delta(x) \leq 0\} \\ &= \min\{x:\ p\ (r+b-s)\ Pr\{Q(x) < n\} + sp - c \leq 0\} \\ &= \min\left\{x:\ Pr\{Q(x) < n\} \leq \frac{c-sp}{p\ (r+b-s)}\right\} \end{aligned} \tag{4.21}$$

As the considered production quantity is bounded below by zero, the optimal production quantity x^* is given by Equation (4.21). □

The optimal solution is the first x, for which the probability of having less non-defective end products than the demand, is smaller or equal a fraction of the cost parameters, which can be considered a critical fractile. In addition, the monotony of the optimal production quantity x^* in the parameters can be analyzed due to the discrete concavity of the expected profit.

Corollary 1 (Monotony of x^* in parameters)**.** *For $p(i) = p$, the optimal production quantity x^* is monotonically non-decreasing in r, s, b and n and monotonically decreasing in c.*

Proof. To analyze the impact of the cost parameters on the optimal production quantity, we consider the partial derivative of the marginal profit between producing $x+1$ and producing x units $\Delta(x)$ as defined in Equation (4.9) with respect to r, s, b and c.

$$\frac{\delta}{\delta\ r}\Delta(x) = p\ Pr\{Q(x) < n\} \geq 0 \tag{4.22}$$

$$\frac{\delta}{\delta\ s}\Delta(x) = p\ (1 - Pr\{Q(x) < n\}) \geq 0 \tag{4.23}$$

$$\frac{\delta}{\delta\ b}\Delta(x) = p\ Pr\{Q(x) < n\} \geq 0 \tag{4.24}$$

$$\frac{\delta}{\delta\ c}\Delta(x) = -1 < 0 \tag{4.25}$$

Therefore, the entire marginal profit changes with a change in each parameter. This also means a change in the first value where the difference in expected profit between producing $x+1$ and producing x units switches from positive to negative. Thus, the monotony properties of $\Delta(x)$ also hold for the optimal production quantity x^*. The probability of receiving less non-defective end products $Q(x)$ than the demand n on the left hand side of the inequality in (4.20) is non-decreasing in n. Therefore, the optimal production quantity x^* is increasing in n. □

Therefore, the optimal production quantity in the case of stationary yield is characterized by Equation (4.20) from Theorem 5. Furthermore, the impact of cost parameters and demand is characterized in Corollary 1. In addition, the case of stationary yield ($p(i) = p$) also fulfills the conditions of increasing yield, i.e. all results for $p(i) \geq p(i+1)$ extend to the case of stationary yield. Therefore, the optimal ramp-up quantity for stationary yield is also either 0 or at least the demand n. Interestingly, all results presented for stationary yield $p(i) = p$ also hold for decreasing yield ($p(i) \geq p(i+1)$).

4.5.3 Summary of analytical insights

To summarize, the expected profit is a discrete concave function of x for stationary yield. Choi et al. (2019) show that the normal approximation of $\mathbb{E}[\Pi(x, Q(x))]$ with stationary yield converges to a concave function in x when the demand n is sufficiently large. Our results hold for the exact model formulation and are further strengthened as they do not depend on the demand n. However, we show that this property does not extend to the case of increasing yield.

In addition, the expected profit is bounded above by the deterministic profit with perfect yield and the optimal expected profit is bounded above by $\mathbb{E}[\Pi(x^*, Q(x^*))] \leq rn - cn$.

The optimal production quantity is bounded by Equation (4.13). In addition, any positive optimal production quantity will always be at least the demand n. In addition, the optimal production quantity for stationary yield is characterized by Equation (4.20).

For stationary yield, the optimal production quantity is decreasing in the production cost. It is non-decreasing in the backlog cost, revenue, salvage value and demand. Choi et al. (2019) obtain the same direction of the monotony but for the normal approximation of $\mathbb{E}[\Pi(x, Q(x))]$.

4.6 Numerical study: Impact of yield learning

In addition to our analytical insights from Section 4.5, we analyze the presented Newsvendor model with non-stationary yield numerically. In Section 4.6.1, we compare the optimal solution of our model to the optimal ex-post solution using real data and to the approximation with stationary yield. Furthermore, we analyze

the monotony of the optimal solution in Section 4.6.2. Section 4.6.3 analyzes the impact of the expected yield on the optimal production quantity and shows that an increase in learning does not always lead to a decrease in the optimal production quantity.

4.6.1 Comparison to ex-post optimal solution

In this section, we compare the performance of our Newsvendor model with non-stationary yield from Section 4.4 to the ex-post optimal solution using scaled real data. As no other model exists with non-stationary yield, we compare the performance of our model to the state of the art in the current literature, which is using stationary yield as an approximation. We use the cost and demand parameters from an example given in Choi et al. (2019) ($n = 40, c = 20, b = 100, r = 70, s = 0$). The yield function $p(i)$ is fitted to the data as described in Section 4.3. One parameter of this fitting is the final yield k. For stationary yield, we assume the final yield k ($p(i) = k$) or the expected yield at the demand n ($p(i) = p(n)$).

We compare the expected profit and optimal production quantity x^* using these fitted parameters with the ex-post objective value from the real data by enumeration of the production quantity. Figure 4.8 shows the production quantity on the x-axis and the expected profit on the y-axis for the ramp-up of a machine (Machine E) and of a product (Product E). The exponential yield learning model fits well to the profit resulting from the observed data. Table 4.6 shows the resulting optimal production quantity x^*, the expected profit predicted by the models and the resulting real profit from the data. In both cases, the non-stationary model overestimates the optimal production quantity, while the stationary model using the final yield underestimates it. Using the stationary model with the expected yield at the demand results in the same optimal production quantity as the non-stationary model in case of Machine E, and the same optimal production quantity as the stationary model with the final yield in case of Product E. For the introduction of a new machine, the stationary approximation using the final yield underestimates the production quantity by 2.2%. However, the resulting real profit is 15.3% less than the optimal ex-post profit, as the profit function is steep for quantities below the optimal quantity. In contrast to this, the resulting real profit using the non-stationary model is only 2.0% below the optimal ex-post profit. For the introduction of a new product, using the stationary approximation with the final yield is 2.2% below the optimal ex-post quantity, the resulting real profit at this point results in a decrease of 7.9%. Again, the non-stationary model is close to the real data, resulting in a resulting real profit which

is only 1.1% below the optimal ex-post profit. The underestimation of the optimal production quantity by the stationary approximation using the final yield is to be expected for an increasing learning curve. By ignoring the ramp-up of the yield function and assuming the final yield already for the first produced unit, the stationary approximation overestimates the probability to be non-defective for the first units. Therefore, also the expected revenue from the first units is overestimated, which in turn leads to an overestimation of the expected profit. Therefore, the stationary approximation leads to an underestimation of the optimal production quantity. The difference is large, if the difference between the actual ex-post probability and the final yield probability is large. This effect could be reduced by choosing a different stationary yield. In our examples, we analyze the expected yield at the demand quantity. In one case, this leads to the same results as the non-stationary model. In the other case however, it leads to the same results as the stationary model with the final yield.

Using the optimal solution of the non-stationary model results in significantly higher profit (ex-post) than using the stationary yield model with the final yield. Using the stationary model with the expected yield at the demand quantity may or may not lead to better results than the stationary model with the final yield. Therefore, managers should consider the non-stationary yield model for planning the optimal ramp-up quantity.

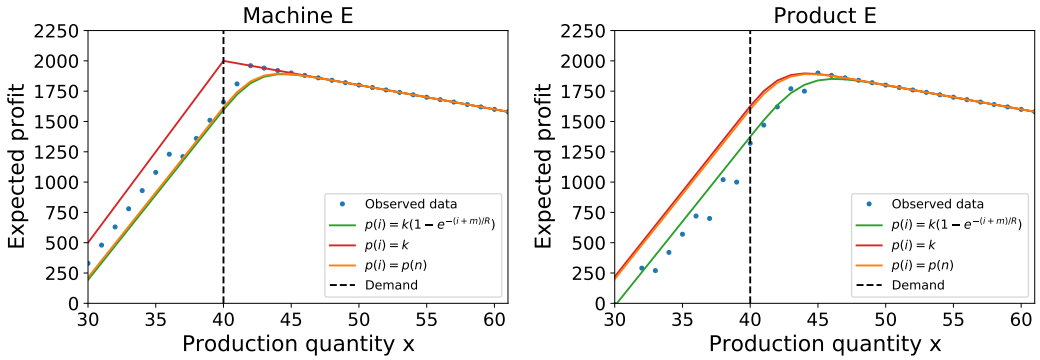


Figure 4.8: Comparison of expected profit from stationary yield model, non-stationary yield model and realizations from real data

4.6.2 Monotony in parameters

Corollary 1 in Section 4.5 shows the monotony of the optimal production quantity x^* in the parameters for a stationary learning function. In this section, we analyze the monotony numerically for an increasing learning function. We use the same

Yield model	$\mathbf{x}^*$	Expected profit	Resulting real profit (data)
Observed data: Machine E	42		1960.00
$\mathbf{p(i) = k\left(1 - e^{-\left(\frac{i+m}{R}\right)}\right)}$	44 (+4.5%)	1889.44	1920.00 (-2.0%)
$\mathbf{p(i) = k}$	40 (-5.0%)	2000.00	1660.00 (-15.3%)
$\mathbf{p(i) = p(n)}$	44 (+4.5%)	1894.45	1920.00 (-2.0%)
Observed data: Product E	45		1900.00
$\mathbf{p(i) = k\left(1 - e^{-\left(\frac{i+m}{R}\right)}\right)}$	46 (+2.2%)	1851.07	1880.00 (-1.1%)
$\mathbf{p(i) = k}$	44 (-2.2%)	1896.70.00	1750.00 (-7.9%)
$\mathbf{p(i) = p(n)}$	44 (-2.2%)	1896.70.00	1750.00 (-7.9%)

Table 4.6: Resulting optimal ramp-up quantities and profits

parameters as in Section 4.6.1 for the fitted non-stationary yield function of Machine G and vary the unit cost c, shortfall cost b, unit revenue r, salvage value s and demand n.

Figure 4.9 shows the optimal production quantity x^* in relation to a change in parameters for an increasing learning function. The optimal production quantity is non-increasing in the unit cost. Furthermore, it is non-decreasing in the shortfall cost, unit revenue, salvage value and demand. This is in line with the results from Corollary 1, which analyzes the monotony for a stationary learning function.

Next, we analyze if monotonic properties can be observed when changing a combination of parameters regarding the learning curve. One of the common questions regarding yield learning is whether a process with a high starting yield and slow increase (*high start*) or a process with low starting yield but a fast increase (*fast increase*) is more profitable. Figure 4.10 shows two such examples in comparison. The parameters are chosen such that the final yield k is the same and the learning speed parameter R of the *high start* is three times that of the *fast increase* (higher R corresponds to slower learning). We set the experience m such that both curves intersect at the demand n. We use a final yield of $k = 0.9$ in all examples. In the upper case of Figure 4.10, we use $R = 30$ and $m = 43$ for the *high start*, and accordingly $R = 10$ and $m = 1$ for the *fast increase*. In the lower case of Figure 4.10, we use $R = 150$ and $m = 43$ for the *high start*, and accordingly $R = 50$ and $m = 1$ for the *fast increase*.

We analyze two different combinations of parameters. In the upper case of Figure 4.10, the *high start* leads to a lower optimal production quantity. This is an expected behavior, as the high yield probability at the beginning of the production

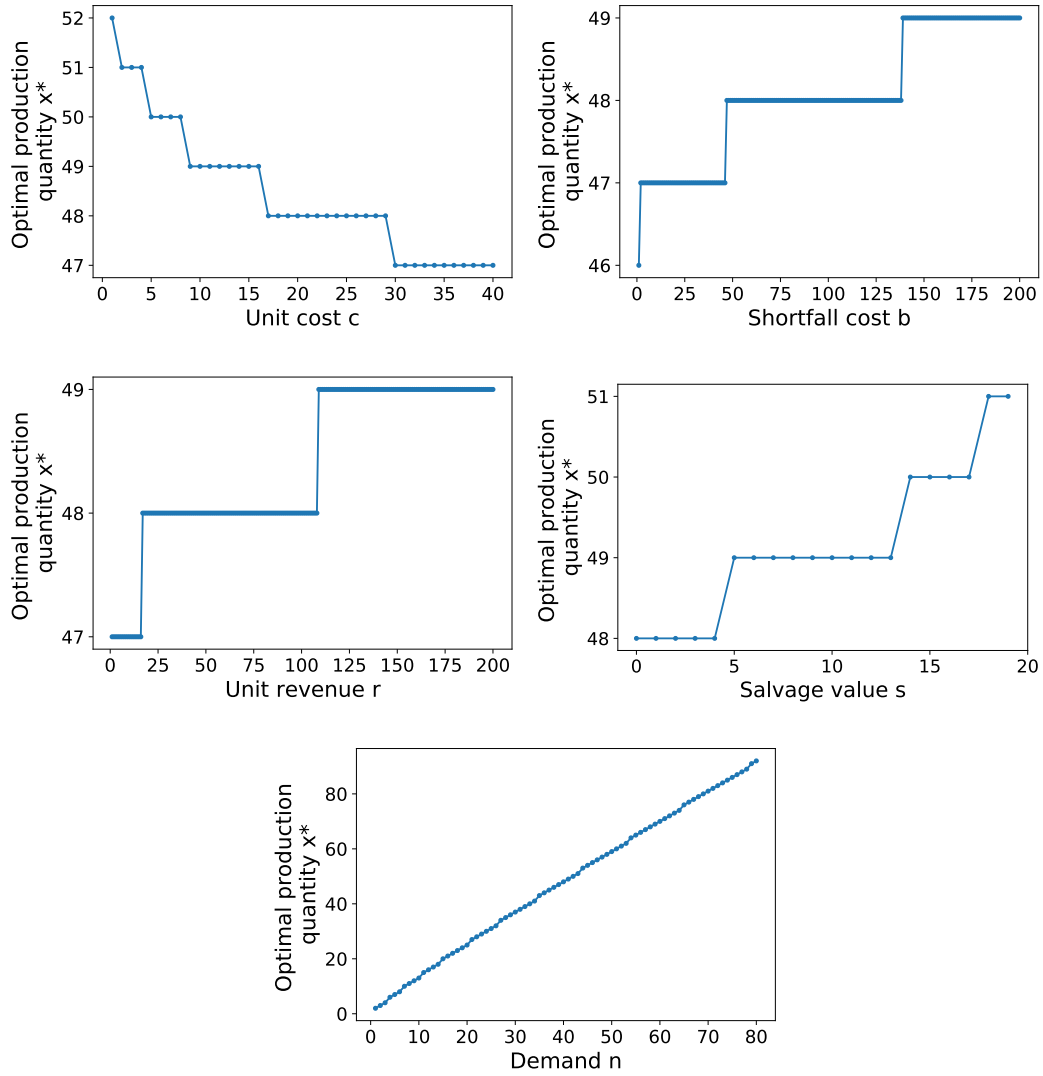
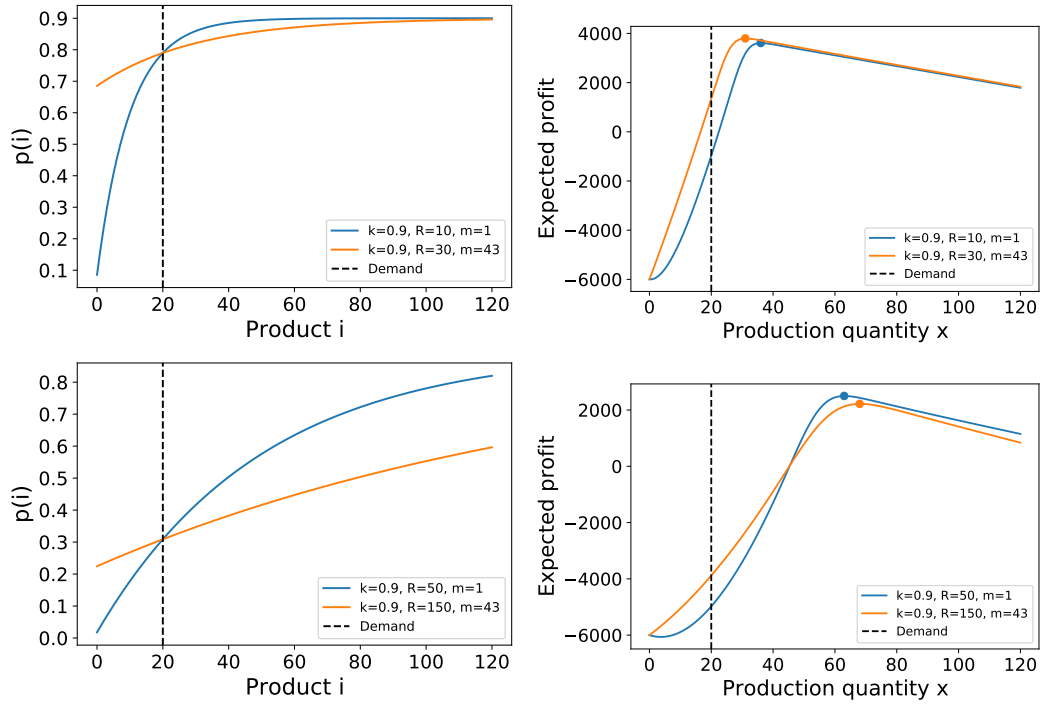


Figure 4.9: Monotony of optimal production quantity in cost parameters and demand

process leads to few defectives. Therefore, a smaller production quantity is needed to maximize the expected profit. In addition, the optimal expected profit of the *high start* (3796.42) exceeds that of the *fast increase* (3610.68). In the lower case of Figure 4.10 however, the *high start* actually results in a larger optimal production quantity. Even though the *high start* has a significantly higher probability of non-defectiveness at the beginning than the *fast increase*, it is still quite low. Therefore, the expected profit of the *fast increase* exceeds that of the *high start* before both attain their maximum. This results in a higher optimal production quantity and a lower optimal expected profit for the *high start*. Therefore, intersecting yield probabilities can lead to a higher or a lower production quantity.

In the next section, we analyze the effect of strictly larger yield probabilities on the optimal production quantity.



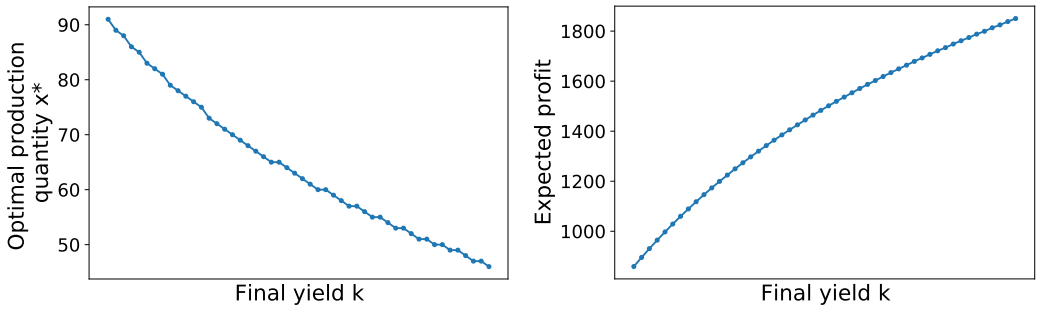
$(n = 20, c = 40, b = 300, r = 250, s = 20)$

Figure 4.10: Analysis of fast versus slow learning

4.6.3 Impact of expected yield

In this section, we analyze the impact of a change in the expected yield on the optimal solution. The expected yield for each produced unit i is $p(i)$. An increase in the

final yield k leads to a parallel shift of the entire non-stationary yield function $p(i)$ upwards, therefore increasing the expected yield of each unit i. We use the same parameter setting as in Section 4.6.1 for the fitted non-stationary yield of Machine G, but vary the final yield k. Figure 4.11 shows the optimal production quantity and expected profit in dependence on the final yield k. For this setting, the optimal production quantity is decreasing and the expected profit is increasing in the final yield.

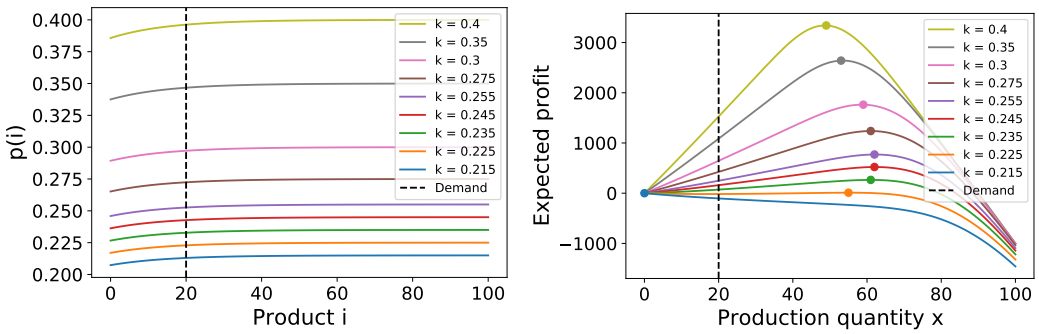


($n = 40, c = 20, b = 100, r = 70, s = 0$)

Figure 4.11: Impact of final yield on optimal production quantity and expected profit

One might think this is intuitive, as a larger final yield k increases the expected yield $p(i)$ for all i and consequently a smaller production quantity x is required to achieve the same quantity of non-defective end products. However, the optimal production quantity is not always monotonically decreasing in the expected yield. Figure 4.12 shows another analysis of the impact of the final yield on the optimal production quantity. In this analysis, we use an example with salvage value $s = 0$ and backlog cost $b = 0$. However, structurally equivalent examples can also be constructed for positive salvage value and backlog cost, as well as for the stationary case $p(i) = p$. The left graph shows the non-stationary yields $p(i)$ for different values of the final yield and right graph shows the resulting expected profits. The colored dots mark the optimal production quantity in each case. We consider final yields from $k = 0.215$ to $k = 0.4$. For $k = 0.215$, the optimal production quantity is zero, as each produced unit incurs a negative expected profit. Therefore, no production is the optimal decision in this case. For $k = 0.225$, the optimal production quantity is positive and is $x^* = 55$. The optimal production quantity is then further increasing until it is 62 for $k = 0.245$ and $k = 0.255$. For $k = 0.275$, it decreases to 61. For further increases in k, the optimal production quantity is further decreasing until it reaches $x^* = 49$ for $k = 0.4$. In contrast to the behavior shown in Figure 4.11, this example shows that a higher expected yield now first leads to an increase in the optimal production quantity before it leads to a decrease. There-

fore, a higher probability of finishing any product i can result in a higher optimal production quantity. While the optimal production quantity shows a non-monotone behavior in this example, the expected optimal profit is increasing in the final yield k.



$(n = 20,\ c = 100,\ b = 0,\ r = 450,\ s = 0,\ k = 0.215 \to 0.4,\ R = 15,\ m = 50)$

Figure 4.12: Counterexample for monotonicity of optimal production quantity x^* in final yield k

To provide a formal reason and an intuition why the optimal production quantity first increases before it decreases, we analyze the derivative of the expected revenue (marginal revenue) and of the production cost (marginal production cost), which are part of the marginal profit $\Delta(x)$ defined in Equation (4.9). Therefore, the marginal revenue is expressed by $r \cdot p(x+1) \cdot Pr\{Q(x) < n\}$ for $b = s = 0$ and the marginal production cost is c.

The derivative of the revenue consists of two parts influenced by a change in the expected yield for each produced unit. The first part, $p(x+1)$, is the probability that the next produced unit is non-defective. For $i \geq 0$, the partial derivative of $p(i)$ with respect to the final yield k is

$$\frac{\delta}{\delta k} p(i) = 1 - e^{-\left(\frac{i+m}{R}\right)} \geq 0\,. \tag{4.26}$$

Therefore, from Equation (4.26) it follows that the probability of the next produced unit to be non-defective is increasing in k. The second part, $Pr\{Q(x) < n\}$, is the probability that the number of non-defective end products is less than the demand and is decreasing in k. Thus, the marginal profit can be increasing or decreasing, depending on which effect is dominating. This in turn changes the point, for which the marginal profit changes from positive to negative, defining the optimal production quantity. The two opposing effects of an increase in the final yield k explain

the non-monotone behavior of the optimal production quantity for an increase in the expected yield. From a managerial point of view, an increase in the expected yield may reduce the optimal ramp-up quantity, as the same quantity of non-defective end products can be obtained from a smaller ramp-up quantity. However, a larger expected yield may also increase the optimal ramp-up quantity, as the expected revenue of an additionally produced unit increases.

We observed this non-monotone effect in different examples where the expected revenue per unit for the final yield $k \cdot r$ is close to the production cost per unit c. This parameter setting can be found in competitive markets, where marginal cost are close to marginal revenue.

To summarize, for values close to the case fitted to real production data, an increase in the expected yield leads to a decrease in the optimal production quantity. This observation can however not be generalized. For settings where $k \cdot r$ is close to the production cost per unit c, an increase in the expected yield may actually lead to an increase in the optimal production quantity at first.

From Section 4.6, we conclude that the model from Section 4.4 fits well to the ex-post analysis using real data from the ramp-up of a new machine in the semiconductor manufacturing industry. Therefore, managers should not use a stationary approximation for their decision on the optimal ramp-up quantity, but should consider the non-stationary yield for planning the ramp-up.

When comparing two different learning curves with a high start and a fast increase, these intersecting yield probabilities can lead to a higher or a lower production quantity. Therefore, there is no clear monotony of the optimal production quantity.

The optimal ramp-up quantity tends to be decreasing in the expected yield. However, a numerical analysis shows that an increase in yield learning can lead to a higher optimal production quantity at first, before the production quantity decreases.

Our numerical analysis suggests that the monotony properties of the optimal production quantity for stationary yield functions also extend to the case of increasing yield.

4.7 Conclusion

We present the problem of planning the optimal ramp-up quantity with stochastic and non-stationary yield for a known demand. The analysis of real yield data from

a global semiconductor manufacturing company shows that such a non-stationary yield behavior occurs both for the introduction of a new product as well as for the introduction of a new machine. We formalize the company's problem as a Newsvendor problem with stochastic and non-stationary yield. The demand is known and the corresponding ramp-up quantity has to be chosen to maximize the expected profit.

The expected profit is a discrete concave function of x for stationary yield. However, this property does not extend to the case of increasing yield. It is bounded above by the expected profit of a (deterministic) yield of 100%.

The optimal production quantity is bounded by Equation (4.13). In addition, any positive optimal production quantity will always be at least the demand for increasing yield, which also extends to stationary yield. Furthermore, we characterize the optimal production quantity for stationary yield by a critical fractile.

For stationary yield, the optimal production quantity is monotonically non-decreasing in the revenue, salvage value, backlog cost and demand. It is monotonically decreasing in the production cost. Our numerical analysis suggests that this insight also extends to the case of increasing yield.

The optimal ramp-up quantity tends to be decreasing in the expected yield. However, a numerical analysis shows that an increase in the expected yield can lead to a higher optimal production quantity at first, before the production quantity decreases. This behavior can be observed where marginal cost are close to marginal revenue.

As our research is motivated by a company operating in the B2B sector, we assume deterministic demand and a fixed revenue per unit. Future research could analyze the impact of stochastic demand or of allowing the company to set the revenue per unit after the realization of the yield is known. In addition, we assume that each chip is either defective or non-defective and that the probability of each is independent of the realization of previous chips. In some production situations, an error in the production process might lead to an entire wafer to be destroyed. Therefore, future research could analyze the impact of different assumptions on the distribution of the non-stationary yield. Furthermore, we assume that the learning curve is known from the previous ramp-up of a similar machine or product. Future research could develop algorithms to predict the learning curve of a new machine or product.

5 Conclusions and outlook

5.1 Conclusions

In this dissertation, three problems on the design and control in stochastic manufacturing systems were discussed. All three considered stochastic variability in the production process, either regarding the time required to finish a process or the yield of the process. In addition, the optimization of a staffing problem considered uncertainty in the demand process. All problems were formalized as mathematical optimization problems and solved using different approaches.

The first article investigated the balancing of an assembly line with stochastic task times and a constraint on the line reliability. We provided a sampling-based model formulation for generally distributed task times. We proved that any lower bound on the number of stations for the related deterministic problem can be transformed into a lower bound for this sampling formulation. We showed the value of these bounds by applying them in a reliability-based branch-and-bound, which directly considered the interdependence between stations due to the constrained line reliability. A numerical study showed that the transformed lower bounds are tight and that they substantially reduce the required computation times of the algorithm and of the solver CPLEX.

The second article investigated two commonly used sampling methods: simple random sampling and descriptive sampling. The article analyzed the impact of the used sampling method and the sample size numerically by considering the performance evaluation of an M/D/1 queueing system and the optimization of the M/M/c staffing level. The article suggests that managers should be aware that the distribution of the resulting performance measures or of the optimal solution derived from a sampling-based approach may not be symmetrical and that the chosen sampling method may have an impact on this behavior.

The third article investigated the ramp-up of a new product or machine with stochastic and non-stationary yield. Using data from a semiconductor manufacturer, we

observed three classes of yield during the ramp-up: (1) deterministic and stationary yield, (2) stochastic and stationary yield, and (3) stochastic and non-stationary yield. We formalized the problem as a Newsvendor problem and proved that any positive optimal ramp-up quantity will always be at least the demand. Furthermore, we characterized the optimal ramp-up quantity for the special case of stationary yield by a critical fractile. One might expect that the optimal ramp-up quantity is decreasing in the expected yield. However, this behavior cannot be generalized and an increase in the expected yield can lead to a higher optimal ramp-up quantity at first, before the ramp-up quantity decreases.

While the third article analyzed the control of ramping-up a new product or machine, the first article analyzed the design of a new assembly line. The stochastic task times in combination with the fixed cycle time lead to a stochastic yield of the line as a whole. However, the expected yield was considered in a constraint. Since the distribution of the task times did not change, there was a stationary yield. In contrast to this, the yield of the third article changed with each produced unit and the decision on the ramp-up quantity had an effect on the resulting quantity of non-defective end products. In a process with non-stationary task times following a learning curve, our approach from the first article could be used to redesign the assembly line once the task times are stationary or do not change significantly anymore. The empirical distribution of observed task times from the ramp-up phase could be used directly in the sampling-based model formulation and solved using the RB&B.

In our first article, we used descriptive sampling and a fixed sample size for the stochastic assembly line balancing problem. The second article analyzed the impact of the chosen sampling method and the sample size on the distribution of the performance measures in an M/D/1 system and of the optimal staffing decision in an M/M/c system. During numerical pretests for our first article, we conducted a similar analysis using the distribution of the optimal decision for independent replications to derive the chosen sample size.

5.2 Further research directions

In addition to suggestions for further research given by each article independently, this dissertation as a whole gives several suggestions for fruitful directions of future research.

The first article assumes stationary task times. However, the third article shows that non-stationary effects can be observed in industry and have an impact on the optimal decision. Therefore, future research could analyze the impact of non-stationary task times on the balancing of an assembly line. In addition, the first article considers the expected value of the line reliability and ignores the ramping-up of the line. From this, two fruitful directions for future research could be considered. First, an empty line at the start of the horizon and explicitly considering the ramp-up of this line could be assumed. This analysis could be further enriched by analyzing different policies on how to treat incomplete work pieces, such as taking out a work piece or letting it continue down the line with rework at the end of the line. Second, a stationary line reliability might not be reasonable for non-stationary task times. Therefore, future research might consider a non-stationary constraint on the line reliability depending on the production quantity.

The third article analyzes the optimal ramp-up under the assumption that each produced product is either defective or non-defective with a changing probability of defectiveness. This leads to the assumption, that the yield follows a Poisson Binomial distribution. However, this assumes that the production of each product is independent of all other products. This might not always be the case. For example, during the production of semiconductors the chips produced on the same wafer might not be independent. Therefore, other assumptions on the distribution of the yield could be analyzed. Sampling-based methods could be used to formulate these problems. However, the impact of the chosen method should be considered and could be analyzed in a similar way as proposed in our second article.

In this dissertation, we consider the design and control of stochastic manufacturing systems independently of each other. As discussed, the design of such a system could impact the non-stationary behavior of the yield during the ramp-up phase. Therefore, an integrated approach could further improve the overall efficiency of such systems. For example, the integrated decision on the design of an assembly line and the produced ramp-up quantity on that line could be considered.

While this dissertation considers stochasticity in some parts of the system, other parts are considered to be deterministic and known. Therefore, future research could analyze the considered systems with further stochastic processes. Alternatively, the parameters might be deterministic but unknown. This could be applied to parameters such as the cycle time for the assembly line balancing or the learning curve and demand for the optimization of the ramp-up quantity.

Finally, all articles assume a risk-neutral decision maker. Therefore, future research could analyze the impact of the risk attitude of the decision maker on the considered optimization problems.

Appendix A: Sampling-based, deterministic lower bounds (Chapter 2)

We use all seven lower bounds on the number of stations proposed by Scholl and Becker (2006). In the following, we present the respective bounds for each sample n.

A.1 Bin packing bounds (Lower bounds 1-3)

The bin packing bounds disregard any precedence relation between the tasks and determine the minimal number of stations solely based on task times.

$$\mathbf{LB^1(n)} = \left\lceil \frac{\sum_{i=1}^{I} t_{n,i}}{c} \right\rceil$$

$$\mathbf{LB^2(n)} = \lceil |N_1(n)| + 1/2 \cdot |N_2(n)| \rceil$$
$$N_1(n) = \{i \mid t_{n,i} > 1/2 \cdot c\}$$
$$N_2(n) = \{i \mid t_{n,i} = 1/2 \cdot c\}$$

$$\mathbf{LB^3(n)} = \lceil\ |N_3(n)| + 1/2 \cdot |N_4(n)| + 2/3 \cdot |N_5(n)| + 1/3 \cdot |N_6(n)|\ \rceil$$
$$N_3(n) = \{i \mid t_{n,i} > 2/3 \cdot c\}$$
$$N_4(n) = \{i \mid 2/3 \cdot c > t_{n,i} > 1/3 \cdot c\}$$
$$N_5(n) = \{i \mid t_{n,i} = 2/3 \cdot c\}$$
$$N_6(n) = \{i \mid t_{n,i} = 1/3 \cdot c\}$$

A.2 One-machine scheduling bound (Lower bound 4)

Lower bound 4 relies on the relaxation of the assembly line problem to the one-machine scheduling problem (Hoffmann, 1990). First, a fictitious source node $i = 0$ is added to the precedence graph with $t_{n,0} = 0$ and precedence relations to all original source nodes. Tasks are viewed as jobs with processing time $p_{n,i} = t_{n,i}/c$ and have to be scheduled on a single machine with the objective of minimizing the makespan. The tail $s_{n,i}$ of task i for sample n is the time required after task i is completed (not necessarily an integer). An optimal solution is sequencing the jobs in order of non-increasing tails. For such an ordering $\{h_1, \ldots, h_I\}$, the minimum makespan is given by (Scholl and Becker, 2006):

$$s_{n,i} = MS(n) = \max\{p_{n,h_1}+s_{n,h_1},\ p_{n,h_1}+p_{n,h_2}+s_{n,h_2}, \ldots,\ p_{n,h_1}+\ldots+p_{n,h_I}+s_{n,h_I}\}$$

The calculation of tails is performed from task I to task 0. The tail of task i is defined by the minimum makespan of all tasks j following task i ($j \in F_i$). For the assembly line problem, the tail is the minimum number of stations required after task i. Note that this number does not have to be an integer, as task i may be placed on the first station of the tail. However, the bound is tightened by a rounding process, where $s_{n,i}$ is rounded up to $\lceil s_{n,i} \rceil$ if $s_{n,i} < \lceil s_{n,i} \rceil$ and $p_{n,i} + s_{n,i} > \lceil s_{n,i} \rceil$. The head $a_{n,i}$ of task i for sample n can be determined by applying the same logic in forward direction with a fictitious sink node $i = I + 1$.

$$\mathbf{LB^4(n)} = \max\left\{\lceil a_{n,i} + p_{n,i} + s_{n,i} \rceil \;\middle|\; i = 0, \ldots, I+1\right\}$$

A.3 Destructive improvement bounds (Lower bounds 5-7)

The destructive improvement bounds try to contradict a valid lower bound (trial value) by showing that no feasible solution exists for this number of stations.

$$\mathbf{LB^5}(\mathbf{n}) = \min\{m \mid L_i(m,n) \geq E_i(n)\ \forall i\}$$

$$E_i(n) = \left\lceil \max\left\{ a_{n,i} + p_{n,i},\ \frac{t_{n,i} + \sum_{j \in P_i} t_{n,j}}{c} \right\} \right\rceil$$

$$L_i(m,n) = m + 1 - \left\lceil \max\left\{ p_{n,i} + n_{n,i},\ \frac{t_{n,i} + \sum_{j \in F_i} t_{n,j}}{c} \right\} \right\rceil$$

$$\mathbf{LB^6}(\mathbf{n}) = \min\left\{ m \;\middle|\; \max\left\{ \sum_{j=0}^{h} t_{n,(h \cdot m + 1 - j)} \;\middle|\; h = 1, \ldots, \lfloor (I-1)/m \rfloor \right\} \leq c \right\}$$

$$\mathbf{LB^7}(\mathbf{n}) = \min\left\{ m \;\middle|\; \left\lceil \frac{\sum_{i \in N_7(m,n)} t_{n,i}}{c} \right\rceil \leq m_2 - m_1 + 1 \right.$$
$$\left. \forall\ m_1, m_2 \text{ with } 1 \leq m_1 \leq m_2 \leq m \right\}$$

$$N_7(m,n) = \{i \mid E_i(n) \geq m_1 \text{ and } L_i(m,n) \leq m_2\}$$

Appendix B: Greedy start heuristic (Chapter 2)

We perform a greedy start heuristic to find an initial upper bound (UB). The heuristic starts at the first station and iteratively determines the R-assignable tasks and assigns the one with the highest mean task time until an R-maximal station load is reached. Then, the next station is loaded R-maximally and the procedure is continued until all tasks have been assigned.

Afterwards, the same procedure is carried out in the backward direction (starting at the last station). The upper bound is set to the better of both solutions. The heuristic may not find a feasible solution if it assigns too many tasks too early. In this case, the upper bound is set to the number of tasks I.

Appendix C: Lower bounds for additional instances (Chapter 2)

This appendix analyzes the transformed lower bounds for additional instances from the deterministic benchmark sets of Otto et al. (2013) and Scholl (1993). We analyze all 525 instances from the benchmark set of Otto et al. (2013) with $I = 20$ tasks. From the benchmark set of Scholl (1993), we analyze the instances of Roszieg ($I = 25$) and Sawyer ($I = 30$), each for its smallest, median and largest cycle time. As described in Section 2.6.1, we transform the deterministic problem instances into stochastic instances by assuming normally distributed task times with a coefficient of variation $cv \in \{0.1,\ 0.3,\ 0.5\}$. The desired line reliability is set to $R = 0.95$.

Table C.1 presents detailed results for a subset of those instances. Besides the instances based on Scholl (1993), we select the first two instances from all categories which have a known deterministic solution from the deterministic benchmark sets of Otto et al. (2013): "less tricky" (instance_n=20_1, instance_n=20_2), "tricky" (instance_n=20_9, instance_n=20_12), "very tricky" (instance_n=20_8, instance_n=20_14) and "extremely tricky" (instance_n=20_17, instance_n=20_39), respectively. We solve the described instances for varying sample sizes $N \in \{100;$ 1,000; 10,000$\}$, which results in 120 instances in total. The first three columns of Table C.1 describe the analyzed instance. Next, we report the global lower bound LB with respect to Theorem 1 and the upper bound UB as a result of the greedy heuristic (the UB is set to the number of tasks in case the greedy solution is infeasible). The optimal solution ($\sum Z_m$) and the computation time in seconds (Time (s)) are given next. In case the optimal solution is not found or proven within a time limit of $10,000$ seconds, we report the final lower and upper bound [LB, UB] derived by the RB&B algorithm. We do not report the instances, which are infeasible for all analyzed sample sizes. Table C.1 demonstrates that the transformed lower bounds are tight for many of the analyzed instances, also for longer lines. The transformed

	cv	c	N = 100 [LB,UB]	N = 100 $\sum Z_m$	N = 100 Time (s)	N = 1,000 [LB,UB]	N = 1,000 $\sum Z_m$	N = 1,000 Time (s)	N = 10,000 [LB,UB]	N = 10,000 $\sum Z_m$	N = 10,000 Time (s)
instance_n=20_1	0.1	1,000	[4, 4]	4	0	[4, 4]	4	1	[3, 4]	4	369
	0.3	1,000	[4, 4]	4	0	[4, 4]	4	1	[4, 4]	4	6
	0.5	1,000	[4, 5]	5	122	[4, 5]	5	1,290	[4, 6]	[4, 5]	10,000
instance_n=20_2	0.1	1,000	[3, 4]	4	5	[3, 4]	4	49	[3, 4]	4	651
	0.3	1,000	[4, 4]	4	0	[4, 4]	4	1	[4, 5]	4	530
	0.5	1,000	[4, 5]	5	128	[4, 5]	5	1,824	[4, 5]	[4, 5]	10,000
instance_n=20_8	0.1	1,000	[4, 4]	4	0	[4, 4]	4	1	[4, 4]	4	10
	0.3	1,000	[4, 4]	4	0	[4, 5]	4	1,299	[4, 5]	[4, 5]	10,000
	0.5	1,000	[4, 5]	5	157	[4, 6]	5	4,268	[4, 7]	[4, 7]	10,000
instance_n=20_9	0.1	1,000	[4, 4]	4	0	[4, 4]	4	1	[4, 4]	4	9
	0.3	1,000	[4, 4]	4	0	[4, 5]	4	875	[4, 4]	4	19
	0.5	1,000	[4, 5]	5	436	[4, 5]	5	5,403	[4, 5]	[4, 5]	10,000
instance_n=20_12	0.1	1,000	[4, 4]	4	0	[4, 4]	4	1	[4, 4]	4	7
	0.3	1,000	[4, 4]	4	0	[4, 5]	4	99	[4, 4]	4	8
	0.5	1,000	[4, 6]	5	107	[4, 6]	5	1,906	[4, 6]	[4, 5]	10,000
instance_n=20_14	0.1	1,000	[4, 4]	4	0	[4, 4]	4	1	[4, 4]	4	8
	0.3	1,000	[4, 4]	4	0	[4, 5]	4	656	[4, 5]	4	7,917
	0.5	1,000	[4, 5]	5	149	[4, 6]	5	4,025	[4, 6]	[4, 6]	10,000
instance_n=20_17	0.1	1,000	[10, 12]	12	476	[10, 12]	12	5,195	[10, 12]	[10, 12]	10,000
instance_n=20_39	0.1	1,000	[14, 17]	16	445	[14, 17]	[16, 17]	10,000	[14, 17]	[14, 17]	10,000
Roszieg	0.1	18	[8, 9]	9	13	[8, 9]	9	193	[8, 9]	9	2,638
		32	[5, 5]	5	0	[5, 5]	5	1	[5, 5]	5	8
	0.3	32	[5, 6]	6	25	[5, 6]	6	315	[5, 6]	6	5,156
	0.5	32	[5, 7]	7	31	[5, 7]	7	870	[5, 8]	[6, 7]	10,000
Sawyer	0.1	36	[10, 12]	11	714	[10, 12]	[10, 12]	10,000	[10, 13]	[10, 13]	10,000
		75	[5, 5]	5	0	[5, 5]	5	1	[5, 5]	5	14
	0.3	75	[5, 7]	6	230	[5, 6]	6	1,421	[5, 7]	[5, 7]	10,000
	0.5	75	[6, 9]	7	962	[6, 9]	[6, 9]	10,000	[6, 30]	[6, 30]	10,000

Table C.1: Additional instances with different sample sizes N

lower bound LB proves the optimality of the greedy heuristic in 37% of the feasible instances from Table C.1.

Out of the $1,575$ instances from the benchmark set of Otto et al. (2013) with 20 tasks and a sample size of $N = 1,000$, 474 instances are infeasible and the RB&B did not find a proven optimal solution within $10,000$ seconds for 154 instances, see Table C.2. The global lower bound equals the optimal solution in 348 instances, and there is a difference of only one station in 366 instances. A difference above four stations is observed in only 11 instances. Table C.2 summarizes the results, aggregated for each problem characteristic of the instances, see Otto et al. (2013). For the feasible instances that are solved within the time limit, the last columns give the average computation time and the percentage of instances with a difference between the optimal solution and the global lower bound $\Delta^{LB} = \sum Z_m - LB$ of only zero or one. For many specifications, the lower bounds are tight. However, for the trickiness category "open", Δ^{LB} was above one station for all four feasible instances. In these instances, the optimal solution ranges between 15 and 17 stations, while the resulting Δ^{LB} is only between 2 and 3 stations. For a task times distribution with a peak in the middle, Δ^{LB} is less or equal one in 7% of the instances. We observe $\Delta^{LB} \leq 2$ for 33% of the instances and $\Delta^{LB} \leq 3$ for 78% of the instances, while the average line length is 15.4 stations.

Characteristics	Specification	Instances	Infeasible	Solved in 10,000 seconds	Average solution time	$\Delta^{LB} \leq 1$
All		1,575	30%	86%	1,037	75%
Graph structure	BN	450	30%	90%	1,598	76%
	CH	450	30%	85%	1,164	76%
	Mixed	675	31%	84%	545	75%
Desired OS	0.2	675	30%	67%	2,071	89%
	0.6	675	30%	100%	660	69%
	0.9	225	30%	100%	70	68%
Task times distribution	peak at the bottom	525	0%	93%	598	99%
	peak in the middle	525	66%	77%	2,325	7%
	bimodal	525	25%	81%	1,152	68%
Trickiness category	less tricky	777	16%	86%	885	85%
	tricky	492	38%	89%	1,757	54%
	very tricky	213	52%	82%	1,044	68%
	extremely tricky	81	54%	76%	1,861	32%
	open (not known yet)	12	67%	100%	951	0%
Coefficient of variation	0.1	525	1%	91%	947	74%
	0.3	525	32%	86%	816	74%
	0.5	525	58%	76%	1,702	82%

Table C.2: Tightness of lower bounds aggregated for different graph characteristics

We performed a similar analysis on two instances of each category with $I = 50$ tasks from the benchmark set of Otto et al. (2013), see Table C.3. Again, the transformed lower bounds are close to the upper bounds for many instances. As

expected, the RB&B algorithm does not terminate within the time limit and a clear judgment on the tightness of the lower bounds is not possible in case of a large gap between LB and UB.

	cv	c	N = 100 [LB,UB]	N = 100 $\sum Z_m$	N = 100 Time (s)	N = 1,000 [LB,UB]	N = 1,000 $\sum Z_m$	N = 1,000 Time (s)	N = 10,000 [LB,UB]	N = 10,000 $\sum Z_m$	N = 10,000 Time (s)
instance_n=50_1	0.1	1,000	[8, 9]	[8, 9]	10,000	[8, 9]	[8, 9]	10,000	[8, 9]	[8, 9]	10,000
	0.3	1,000	[8, 10]	[8, 10]	10,000	[8, 11]	[8, 11]	10,000	[8, 9]	[8, 9]	10,000
	0.5	1,000	[9, 12]	[9, 12]	10,000	[9, 14]	[9, 14]	10,000	[9, 15]	[9, 15]	10,000
instance_n=50_2	0.1	1,000	[6, 7]	[6, 7]	10,000	[6, 7]	[6, 7]	10,000	[6, 7]	[6, 7]	10,000
	0.3	1,000	[7, 8]	[7, 8]	10,000	[7, 8]	[7, 8]	10,000	[7, 9]	[7, 9]	10,000
	0.5	1,000	[7, 8]	[7, 8]	10,000	[7, 10]	[7, 10]	10,000	[7, 11]	[7, 11]	10,000
instance_n=50_7	0.1	1,000	[8, 8]	8	1	[8, 8]	8	5	[8, 8]	8	56
	0.3	1,000	[8, 10]	[8, 10]	10,000	[8, 10]	[8, 10]	10,000	[8, 11]	[8, 11]	10,000
	0.5	1,000	[8, 12]	[8, 12]	10,000	[8, 13]	[8, 13]	10,000	[8, 14]	[8, 14]	10,000
instance_n=50_12	0.1	1,000	[7, 7]	7	1	[7, 7]	7	5	[7, 7]	7	72
	0.3	1,000	[7, 8]	[7, 8]	10,000	[7, 9]	[7, 9]	10,000	[7, 9]	[7, 9]	10,000
	0.5	1,000	[7, 9]	[7, 9]	10,000	[7, 11]	[7, 11]	10,000	[7, 12]	[7, 12]	10,000
instance_n=50_22	0.1	1,000	[8, 8]	8	1	[8, 8]	8	5	[8, 8]	8	57
	0.3	1,000	[8, 10]	[8, 10]	10,000	[8, 10]	[8, 10]	10,000	[8, 11]	[8, 11]	10,000
	0.5	1,000	[8, 11]	[8, 11]	10,000	[8, 13]	[8, 13]	10,000	[8, 14]	[8, 14]	10,000
instance_n=50_26	0.1	1,000	[28, 35]	[28, 35]	10,000	[28, 37]	[28, 37]	10,000	[28, 39]	[28, 39]	10,000
instance_n=50_27	0.1	1,000	[31, 40]	[31, 40]	10,000	[31, 42]	[31, 42]	10,000	[31, 43]	[31, 43]	10,000
instance_n=50_54	0.1	1,000	[11, 13]	[11, 13]	10,000	[11, 13]	[11, 13]	10,000	[11, 13]	[11, 13]	10,000
	0.3	1,000	[12, 16]	[12, 16]	10,000	[12, 50]	-	10,000	[12, 50]	-	10,000

Table C.3: Additional instances with 50 tasks from Otto et al. (2013) with different sample size N

Appendix D: Implementation of RB&B in Python (Chapter 2)

This Appendix features the full code implementation of the reliability-based branch-and-bound (RB&B) from Chapter 2. The RB&B algorithm is implemented in Python 3.6 using the Spyder environment.

```
#=============================================================================
#Functions

pseudonode = -1

def f_pseudo(list_of_tasks, node, direction='forward'):
    """ Create a Pseudo node the check R assignability at the next station """

    global nodelist

    local_node = nodelist[node][:]
    local_node[p_number] = 'X'
    local_node[p_state] = 'Pseudo'

    if direction == 'forward':
        local_node[p_station] += 1
    if direction == 'backward':
        local_node[p_station] -= 1

    for i in list_of_tasks:
        local_node[nodechar+i-1] = local_node[p_station]


    nodelist.append(local_node)

    return nodelist[pseudonode]

def f_open(list_of_tasks, node, direction='forward'):
    """ Create a new Open node with list_of_tasks assigned tasks at next station """

    global nodelist
    global assignedlist
    global unassignedlist
    global currentnode
    global currentstation


    currentnode = len(nodelist)                     # calculate number of current node
    nodelist.append(nodelist[node][:])              # copy parent node


    if direction == 'forward':
        currentstation = nodelist[node][p_station] + 1
    if direction == 'backward':
        currentstation = nodelist[node][p_station] - 1

    nodelist[currentnode][p_number] = currentnode                       # update numbering of the current node
    nodelist[currentnode][p_station] = currentstation                 # update current station for new node
    nodelist[currentnode][p_incumbent] = incumbent
    nodelist[currentnode][p_state] = 'Open'


    for i in list_of_tasks:
        nodelist[currentnode][nodechar+i-1] = nodelist[currentnode][p_station]

    localstations = [m for m in nodelist[currentnode][nodechar:] if m != '_']   # update number of used stations
    if localstations != []:
        nodelist[currentnode][p_usedstation] = max(localstations) - min(localstations) + 1
    else:
        nodelist[currentnode][p_usedstation] = 0

    nodelist[currentnode][p_completion] = f_completionrate(currentnode)  # update current completion rate for new node

    nodelist[currentnode][p_time] = round(time.time() - starttime, 2)

    f_assigned(currentnode)
    assignedlist.append(assigned)
    unassignedlist.append(unassigned)
    Tklist.append(f_Tk(currentnode))

def f_assigned(node):
    """ Determine the assigned and unassigned tasks of a node """
    global unassigned
    global assigned

    unassigned = [i-nodechar+1 for i,x in enumerate(nodelist[node]) if x=='_']    # unassigned tasks of current node
    assigned = [i+1 for i,x in enumerate(nodelist[node][nodechar:tasks+nodechar+1]) if x != '_']
```

```
    # assigned tasks of current node


def f_available(node, mode=0):
    """
    Calculates all available task with 'node' as the parent node
    Mode:   0 -> Data is already available in assignedlist
            1 -> f_assigned has to be done first
    """
    global assigned
    global available

    if mode == 1:
        f_assigned(node)
    else:
        assigned = assignedlist[node]

    if nodelist[node][p_direction] == 'f':
        pairs = [ (i,j) for i in assigned for j in tasklist if (i,j) in P]
        # pairs in P of which i has already been assigned
        Pclear = P[:]
        # matrix of still existing precedence relations (i has not been assigned yet)

    if nodelist[node][p_direction] == 'b':
        pairs = [ (i,j) for i in assigned for j in tasklist if (i,j) in P_rev]
        # pairs in P of which i has already been assigned
        Pclear = P_rev[:]
        # matrix of still existing precedence relations (i has not been assigned yet)


    for i in range(0,len(pairs)) :
        del Pclear[Pclear.index(pairs[i])]

    notavailable = [ Pclear[i][1] for i in range(0,len(Pclear)) ]   # all task NOT available
    available = [i for i in tasklist if i not in notavailable if i not in assigned]     # all tasks available


def f_assignable(node, mode=0):
    """
    Calculates all assignable task with 'node' as the parent node
    Mode:   0 -> Available has already been calculated
            1 -> f_available has to be done first
    """

    global assignable

    if mode == 1:
        f_available(node, 1)

    assignable = [i for i in available if f_LR(node,i) >= R]

    return assignable


constructiontime = 0

def f_branch(node):
    """ Branch the chosen node with all feasible combinations of assignments, check all created child nodes for
    LLB/Logical/Dominance """
    startconstruct = time.time()
    global feasible_assignment
    global assignable
    global nodelist
    global r
    global countconstruct
    global fathomed
    global endnode
    global constructiontime

    if nodelist[node][p_direction] == 'f':
        direction = 'forward'
    else:
        direction = 'backward'

    feasible_assignment = []   # final list of assignments for opening new nodes

    nodelist[node][p_state] = 'Evaluated'
```

```
f_pseudo([], node, direction)
f_assignable(pseudonode, 1)
del nodelist[pseudonode]


r = [[i] for i in assignable]   # intermediate list of possible assignments


while r != []:

    f_pseudo(r[0], node, direction)
    f_assignable(pseudonode, 1)


    if assignable == []:
        if r[0] not in feasible_assignment:
            feasible_assignment.append(r[0])

    else:
        intermediateresults = [ r[0] + [assignable[i]] for i in range(len(assignable))]

        #only add new results
        for entry in range(len(intermediateresults)):
            intermediateresults[entry].sort()
            if intermediateresults[entry] not in r:
                r.append(intermediateresults[entry])

        if r[0] not in feasible_assignment:
            feasible_assignment.append(r[0])    # add this to enforce opening of "idle" time


    del r[0], nodelist[pseudonode]

feasible_assignment = [x for _,x in sorted(zip([len(entry)
                       for entry in feasible_assignment],feasible_assignment),reverse=True)]
# sort results: assignments with more tasks come first

for entry in range(len(feasible_assignment)):
    endnode = False
    fathomed = False

    f_open(feasible_assignment[entry], node, direction)

    f_increase(currentnode)

    f_assigned(currentnode)
    if unassigned == []:
        f_endnode(currentnode)

    if endnode == False and endloop == False:
        f_LLB1(currentnode)
        if fathomed == False:
            f_dominance(currentnode)
        if fathomed == False:
            f_LLB4(currentnode)
        if fathomed == False:
            f_LLB2(currentnode)
        if fathomed == False:
            f_LLB3(currentnode)
        if fathomed == False:
            f_LLB5(currentnode)
        if fathomed == False:
            f_LLB6(currentnode)
        if fathomed == False:
            f_LLB7(currentnode)
        if fathomed == False:
            nodelist[currentnode][p_LLB] = max(LLB1, LLB2, LLB3, LLB4, LLB5, LLB6, LLB7) \
                                           + nodelist[currentnode][p_usedstation]
            f_logical(currentnode)
        if fathomed == False:
            f_connect(currentnode)



    if endloop == True:
        break
```

```
    endconstruct = time.time()
    constructiontime += endconstruct - startconstruct


def f_workload(node):
    """ Calculates the workload of a node """
    global workload    # workload[n][m]


    local_node = nodelist[node][nodechar:]

    workload = [ [0 for m in range(tasks)] for n in range(N) ]

    for n in range(N):
        for i in range(len(local_node)) :
            if local_node[i] != '_' :
                workload[n][local_node[i]-1] += t_n[n][i]

    return workload


def f_utilization(node):
    """ Calculates the utilization of a node """
    global utilization

    f_workload(node)
    utilization  = [ [workload[n][m] / c for m in range(len(workload[n]))] for n in range(N) ]

    return utilization


def f_complete(node):
    """ Calculates the resulting complete samples for a node """
    global complete

    local_utilization = f_utilization(node)
    complete = [ 1 if max(local_utilization[n]) <= 1 else 0 for n in range(N)  ]

    return complete


def f_completionrate(node):
    """ Calculates the resulting completion rate of a node """
    global LR
    LR = sum(f_complete(node)) / N * 100

    return LR

def f_evaluate(node):
    f_workload(node)
    f_utilization(node)
    f_complete(node)
    f_completionrate(node)


def f_endnode(node):
    global nodelist
    global incumbent
    global endloop
    global endnode

    nodelist[node][p_state] = 'End node'
    nodelist[node][p_station] = max(nodelist[node][nodechar:]) - min(nodelist[node][nodechar:]) + 1

    endnode = True

    if nodelist[node][p_usedstation] < incumbent:
        incumbent = nodelist[node][p_usedstation]
        nodelist[node][p_incumbent] = incumbent
        print('\nNew incumbent found:', incumbent, '\n')

        if incumbent == LB :                            # exit if incumbent = global lower bound (global bounding)
            print('Global lower bound reached:', LB)
            endloop = True
            return
```

```
        else:
            endloop = False

            #check all open nodes for LLBs and dominance
            open_list = [ nodelist[k][p_number] for k in range(0,len(nodelist)) if nodelist[k][p_state] == 'Open' ]

            for k in open_list:
                Tklist[k] = f_Tk(k)
                if nodelist[k][p_LLB] >= incumbent:
                    nodelist[k][p_state] = 'Fathomed'


def f_increase(node):
    """ For backward nodes, increase assignment to station by 1 for every station, if current station is 0 """
    global nodelist

    if min([ nodelist[node][nodechar+i] for i in range(tasks) if nodelist[node][nodechar+i] != '_' ]) < 1:
        for i in [ i for i in range(tasks) if nodelist[node][nodechar+i] != '_' ]:
            nodelist[node][nodechar+i] += 1

        nodelist[node][p_station] = 1

    return nodelist[node]


def f_LR(node, task):
    """ Calculates the completion rate of a node if an additional task is assigned """
    global LR

    local_utilization = f_utilization(node)

    for n in range(N):
        local_utilization[n][nodelist[node][p_station]-1] += (t_n[n][task-1] / c)
        # minus one because list starts with station 1 not 0

    local_complete = [ 1 if max(local_utilization[n]) <= 1 else 0 for n in range(N) ]
    LR = sum(local_complete) / N * 100

    return LR


def f_findnode():
    """
    Find the most promising forward and backward node
    """

    global endloop
    global parentnode_f
    global parentnode_b

    outlist = []

    for direc in ['f','b']:
        statelist_0 = [ nodelist[k][:nodechar] for k in range(len(nodelist)) if nodelist[k][p_state] == 'Open'
        and nodelist[k][p_direction] == direc ]
        # List of all open nodes
        if statelist_0 == []:    # If there are no open nodes in current direction
            outlist.append([])

        else:
            statelist_1 = [ statelist_0[k] for k in range(len(statelist_0))
            if statelist_0[k][p_LLB] == min([ statelist_0[k][p_LLB] for k in range(len(statelist_0)) ]) ]
            # Nodes with lowest LLB
            statelist_2 = [ statelist_1[k] for k in range(len(statelist_1))
            if statelist_1[k][p_usedstation] == max([ statelist_1[k][p_usedstation] for k in range(len(statelist_1)) ]) ]
            # Nodes with highest used stations
            local_Tklist = [Tklist[statelist_2[k][p_number]] for k in range(len(statelist_2))]
            statelist_3 = [statelist_2[np.argmax(local_Tklist)]]
            outlist.append(statelist_3[len(statelist_3)-1]) # Newest nodes

    if outlist == [[],[]]:
        print('\nThere are no more open nodes.')
        endloop = True
    else:
        try:
```

```
            parentnode_f = outlist[0][p_number]
        except:
            parentnode_f = 'X'
        try:
            parentnode_b = outlist[1][p_number]
        except:
            parentnode_b = 'X'


def f_Tk(node):
    """ Calculate the Tk value for a node """
    global Tklist
    global Tk

    denominator = [ sum([(latest_n[n][incumbent][i] - earliest_n[n][i] + 1 + 0.0000001)
                    for n in range(N)])/N for i in range(tasks) ]

    f_available(node)
    try:
        Tk = sum([ t[i-1]/(denominator[i-1]) for i in available]) / len(available)
    except:
        Tk = -1

    return Tk


def f_direction():
    """ Find the direction of the next branching """
    global nodelist
    global available
    global parentnode
    global direction
    global assignedlist
    global parentnode
    global steps_f
    global steps_b
    global parentnode_f
    global parentnode_b
    global T_f
    global T_b
    global endloop
    global denominator

    if endloop == True:
        return

    denominator = [ sum([(latest_n[n][incumbent][i] - earliest_n[n][i] + 1 + 0.0000001)
                    for n in range(N)])/N for i in range(tasks) ]

    if parentnode_f != 'X':
        f_available(parentnode_f)
        available_f = available[:]
        T_f = sum([ t[i-1]/(denominator[i-1]) for i in available_f]) / len(available_f)

    else:
        steps_b += 1
        f_available(parentnode_b)
        available_b = available[:]

        direction = 'backward'
        available = available_b
        parentnode = parentnode_b
        return

    if parentnode_b != 'X':
        f_available(parentnode_b)
        available_b = available[:]
        T_b = sum([ t[i-1]/(denominator[i-1]) for i in available_b]) / len(available_b)

    else:
        steps_f += 1
        f_available(parentnode_f)
        available_f = available[:]

        direction = 'forward'
        available = available_f
        parentnode = parentnode_f
```

```
            return

        if T_f > T_b or (T_f == T_b and len(available_f) <= len(available_b)):
            steps_f += 1

            direction = 'forward'
            available = available_f
            parentnode = parentnode_f

        else:
            steps_b += 1

            direction = 'backward'
            available = available_b
            parentnode = parentnode_b


    def f_connect(node):
        """ Connect the backward nodes with the forward nodes """
        global local_node
        global nodelist
        global incumbent
        global endloop
        global fathomed
        global local_max
        global local_min
        global local_dif
        global connection

        connection = False

        for k in range(len(nodelist)):
            if k != node:
                local_list = assignedlist[k] + assignedlist[node]
                local_list.sort()
                if local_list == tasklist:
                    local_node = [ len(nodelist), 'e', 0, 0, round(time.time()-starttime,2), 100.0, nodelist[node][p_LLB],
                                   incumbent, 'End node']
                    if nodelist[node][p_direction] == 'f':
                        local_node += nodelist[node][nodechar:]
                        local_max = nodelist[node][p_station]
                        local_min = nodelist[k][p_station]
                        local_dif = local_min - local_max - 1

                        while '_' in local_node:
                            local_node[local_node.index('_')] = nodelist[k][local_node.index('_')] - local_dif
                    else:
                        local_node += nodelist[k][nodechar:]
                        local_max = nodelist[k][p_station]
                        local_min = nodelist[node][p_station]
                        local_dif = local_min - local_max - 1

                        while '_' in local_node:
                            local_node[local_node.index('_')] = nodelist[node][local_node.index('_')] - local_dif


                    local_node[p_station]       = max(local_node[nodechar:])
                    local_node[p_usedstation]   = local_node[p_station]

                    nodelist.append(local_node)

                    nodelist[-1][p_completion]       = f_completionrate(-1)

                    if nodelist[-1][p_completion] >= R:
                        connection = True
                        assignedlist.append(tasklist)
                        unassignedlist.append([])
                        Tklist.append(-1)

                        f_endnode(len(nodelist)-1)

                    else:
                        del nodelist[-1]

    def f_greedy_heuristic():
        """ Greedy heuristic """
```

```
global nodelist
global endloop
global currentnode
global assigned
global unassigned
global assignedlist
global unassignedlist
global inf
global incumbent
global heu_sol ###

currentnode = 0
currentstation = 1

nodelist = []
nodelist.append([0, 'f', currentstation, 0, 0, 0.0, LB, incumbent, 'GH'] + ['_' for task in tasklist])
# open new node
inf = 0

unassigned = tasklist[:]
while unassigned != []:

    f_assigned(currentnode)
    f_assignable(currentnode,1)

    stationlist = [nodelist[currentnode][nodechar+i-1] for i in assigned]
    if assignable == [] and nodelist[currentnode][p_station] not in stationlist:
        nodelist[currentnode][nodechar:] = [tasks for i in range(tasks)]
        nodelist[currentnode][nodechar] = 1
        break

    if unassigned == []:
        break

    if assignable == []:
        nodelist[currentnode][p_station] += 1

    else: #always assign the task with the highest average task time t
        nodelist[currentnode][nodechar+assignable[np.argmax([t[i-1] for i in assignable])] -1] \
            = nodelist[currentnode][p_station]


heu_sol = [max(nodelist[0][nodechar:])-min(nodelist[0][nodechar:]) + 1, 0]

if max(nodelist[0][nodechar:])-min(nodelist[0][nodechar:]) + 1 > LB:
    currentnode = 1
    inf = 0

    nodelist.append([currentnode, 'b', currentstation, 0, 0, 0.0, LB, incumbent, 'GH'])
    # backward node
    nodelist[currentnode] += ['_' for task in tasklist]

    unassigned = tasklist[:]
    while unassigned != []:

        f_assigned(currentnode)
        f_assignable(currentnode,1)

        stationlist = [nodelist[currentnode][nodechar+i-1] for i in assigned]
        if assignable == [] and nodelist[currentnode][p_station] not in stationlist:

            nodelist[currentnode][nodechar:] = [tasks for i in range(tasks)]
            nodelist[currentnode][nodechar] = 1
            break

        if unassigned == []:
            break

        if assignable == []:
            nodelist[currentnode][p_station] -= 1

        else: #always assign the task with the highest average task time t
            nodelist[currentnode][nodechar+assignable[np.argmax([t[i-1] for i in assignable])] -1] \
                = nodelist[currentnode][p_station]

    while min([ nodelist[currentnode][nodechar+i] for i in range(tasks) \
        if nodelist[currentnode][nodechar+i] != '_' ]) < 1:
```

```
            for i in [ i for i in range(tasks) if nodelist[currentnode][nodechar+i] != '_' ]:
                nodelist[currentnode][nodechar+i] += 1

        print('Forward:', max(nodelist[0][nodechar:])-min(nodelist[0][nodechar:]) + 1)
        print('Backward:', max(nodelist[1][nodechar:])-min(nodelist[1][nodechar:]) + 1)
        heu_sol[1] = max(nodelist[1][nodechar:])-min(nodelist[1][nodechar:]) + 1 ###

        if max(nodelist[0][nodechar:])-min(nodelist[0][nodechar:]) + 1 \
            <= max(nodelist[1][nodechar:])-min(nodelist[1][nodechar:]) + 1:
            del nodelist[1]
        else:
            del nodelist[0]
            nodelist[0][p_number] = 0


    try:
        localstations = [m for m in nodelist[0][nodechar:] if m != '_']
        nodelist[0][p_direction]    = 'GH'
        nodelist[0][p_usedstation]  = max(localstations) - min(localstations) + 1
        nodelist[0][p_completion]   = f_completionrate(0)
    except:
        pass

    if inf != 2:
        f_endnode(0)

    nodelist[0][p_time]         = round(time.time() - starttime, 2)

    f_assigned(0)
    assignedlist.append(assigned)
    unassignedlist.append(unassigned)
    Tklist.append(-1)


def f_LLB1(node, delete=1):
    global nodelist
    global assignedlist
    global unassignedlist
    global LLB1
    global fathomed
    global count_LLB1
    global LLB1_n

    fathomed = False

    LLB1_n = [math.ceil(sum([ t_n[n][i-1] for i in unassignedlist[node] ])/c) for n in range(N)]
    LLB1_n.sort()
    LLB1 = LLB1_n[math.ceil(R/100 * N - 1)]

    if assignedlist[node] != [] and nodelist[node][p_usedstation] + LLB1 >= incumbent:
        if delete == 1:
            del nodelist[node]
            del assignedlist[node]
            del unassignedlist[node]
            del Tklist[node]
        else:
            nodelist[node][p_state] = 'Fathomed (LLB1)'

        fathomed = True
        count_LLB1 += 1

    return LLB1


def f_LLB2(node, delete=1):
    global nodelist
    global assignedlist
    global unassignedlist
    global LLB2
    global fathomed
    global count_LLB2

    fathomed = False

    N_bar_1 = [[i for i in unassignedlist[node] if t_n[n][i-1]>c/2] for n in range(N)]
    N_bar_2 = [[i for i in unassignedlist[node] if t_n[n][i-1]==c/2] for n in range(N)]
    LLB2_n = [math.ceil(len(N_bar_1[n]) + 0.5*len(N_bar_2[n])) for n in range(N)]
```

```
    LLB2_n.sort()
    LLB2 = LLB2_n[math.ceil(R/100 * N - 1)]

    if assignedlist[node] != [] and nodelist[node][p_usedstation] + LLB2 >= incumbent:
        if delete == 1:
            del nodelist[node]
            del assignedlist[node]
            del unassignedlist[node]
            del Tklist[node]
        else:
            nodelist[node][p_state] = 'Fathomed (LLB2)'

        fathomed = True
        count_LLB2 += 1

    return LLB2


def f_LLB3(node, delete=1):
    global nodelist
    global assignedlist
    global unassignedlist
    global LLB3
    global fathomed
    global count_LLB3

    fathomed = False

    N_bar_3 = [[i for i in unassignedlist[node] if t_n[n][i-1]>2*c/3] for n in range(N)]
    N_bar_4 = [[i for i in unassignedlist[node] if 2*c/3>t_n[n][i-1]>1*c/3] for n in range(N)]
    N_bar_5 = [[i for i in unassignedlist[node] if t_n[n][i-1] == 2*c/3] for n in range(N)]
    N_bar_6 = [[i for i in unassignedlist[node] if t_n[n][i-1] == 1*c/3] for n in range(N)]
    LLB3_n = [math.ceil(len(N_bar_3[n]) + 0.5*len(N_bar_4[n]) + 2/3*len(N_bar_5[n]) + 1/3*len(N_bar_6[n]))
                for n in range(N)]

    LLB3_n.sort()
    LLB3 = LLB3_n[math.ceil(R/100 * N - 1)]

    if assignedlist[node] != [] and nodelist[node][p_usedstation] + LLB3 >= incumbent:
        if delete == 1:
            del nodelist[node]
            del assignedlist[node]
            del unassignedlist[node]
            del Tklist[node]
        else:
            nodelist[node][p_state] = 'Fathomed (LLB3)'

        fathomed = True
        count_LLB3 += 1

    return LLB3


def f_LLB4(node, delete=1):
    global nodelist
    global assignedlist
    global unassignedlist
    global LLB4
    global fathomed
    global signaltoLLBs
    global count_LLB4
    global local_n_j
    global local_a_j

    fathomed = False

    signaltoLLBs = node

    # add dummy tasks, root tasks for forward nodes, sink task for backward nodes
    if nodelist[node][p_direction] == 'f':
        add_dummy = [tasks+1]
    else:
        add_dummy = [0]

    local_preceder = [[i for i in LB4_preceder[j] if i in unassignedlist[node]+add_dummy
                        and j in unassignedlist[node]+add_dummy] for j in LB4_tasklist ]
```

```
    local_follower = [[i for i in LB4_follower[j] if i in unassignedlist[node]+add_dummy
                       and j in unassignedlist[node]+add_dummy] for j in LB4_tasklist ]

    local_n_j  = [[0 for i in range(LB4_tasks)] for n in range(N)]

    for n in range(N):
        for i in LB4_tasklist_rev:
            local_listoftails = [[j,local_n_j[n][j]] for j in local_follower[i]]
            local_listoftails.sort(key = lambda l: (l[1]), reverse=True) #sort in non-increasing order of tails
            local_sortedtails = [[local_listoftails[j][0] for j in range(len(local_listoftails))][:j+1]
                                 for j in range(len(local_listoftails))]
            local_n_j[n][i] = max([sum([p_j[n][j] for j in local_sortedtails[jj]] +
                                   [local_n_j[n][local_sortedtails[jj][-1]]])
                                   for jj in range(len(local_sortedtails))] + [0])
            if local_n_j[n][i] < math.ceil(local_n_j[n][i]) and p_j[n][i] + local_n_j[n][i] > math.ceil(local_n_j[n][i]):
                local_n_j[n][i] = math.ceil(local_n_j[n][i])



    local_a_j  = [[0 for i in range(LB4_tasks)] for n in range(N)]

    for n in range(N):
        for i in LB4_tasklist:
            local_listofheads = [[j,local_a_j[n][j]] for j in local_preceder[i]]
            local_listofheads.sort(key = lambda l: (l[1]), reverse=True) #sort in non-increasing order of heads
            local_sortedheads = [[local_listofheads[j][0] for j in range(len(local_listofheads))][:j+1]
                                 for j in range(len(local_listofheads))]
            local_a_j[n][i] = max([sum([p_j[n][j] for j in local_sortedheads[jj]] +
                                   [local_a_j[n][local_sortedheads[jj][-1]]])
                                   for jj in range(len(local_sortedheads))] + [0])
            if local_a_j[n][i] < math.ceil(local_a_j[n][i]) and p_j[n][i] + local_a_j[n][i] > math.ceil(local_a_j[n][i]):
                local_a_j[n][i] = math.ceil(local_a_j[n][i])

    LLB4_n = [max([math.ceil(local_a_j[n][i] + p_j[n][i] + local_n_j[n][i]) for i in LB4_tasklist]) for n in range(N)]


    LLB4_n.sort()
    LLB4 = LLB4_n[math.ceil(R/100 * N - 1)]

    if assignedlist[node] != [] and nodelist[node][p_usedstation] + LLB4 >= incumbent:
        if delete == 1:
            del nodelist[node]
            del assignedlist[node]
            del unassignedlist[node]
            del Tklist[node]
        else:
            nodelist[node][p_state] = 'Fathomed (LLB4)'

        fathomed = True
        count_LLB4 += 1

    return LLB4


def f_LLB5(node, delete=1):
    """
    Requires computation of LLB4 for stronger bound
    """

    global nodelist
    global assignedlist
    global unassignedlist
    global LLB5
    global fathomed
    global count_LLB5

    global local_earliest_n1
    global local_earliest_n
    global local_latest_n1
    global local_latest_n
    global local_a_j
    global local_n_j
    global LLB5_n

    fathomed = False

    if signaltoLLBs != node:
```

```
        local_n_j = [[0 for i in LB4_tasklist] for n in range(N)]
        local_a_j = [[0 for i in LB4_tasklist] for n in range(N)]

    local_preceder = [[i for i in preceder[j-1] if i in unassignedlist[node] and j in unassignedlist[node]]
                        for j in tasklist ]
    local_follower = [[i for i in follower[j-1] if i in unassignedlist[node] and j in unassignedlist[node]]
                        for j in tasklist ]

    local_earliest_n1 = [[ math.ceil((t_n[n][i-1] + sum([ t_n[n][j-1] for j in local_preceder[i-1] ]))/c)
                            for i in tasklist ] for n in range(N)]
    local_earliest_n  = [[max(local_earliest_n1[n][i-1], math.ceil(local_a_j[n][i] + p_j[n][i])) for i in tasklist]
                            for n in range(N)]
    local_latest_n1   = [[math.ceil((t_n[n][i-1] + sum([t_n[n][j-1] for j in local_follower[i-1]]))/c)
                            for i in tasklist  ]
                            for n in range(N)]
    local_latest_n    = [[[m + 1 - max(local_latest_n1[n][i-1], math.ceil(p_j[n][i] + local_n_j[n][i]))
                            for i in tasklist]
                            for m in range(tasks+1)] for n in range(N)]

    LLB5_n = [min([m for m in range(tasks+1) if False not in [local_latest_n[n][m][i] >= local_earliest_n[n][i]
                for i in range(tasks)]])
                for n in range(N) ]
    LLB5_n.sort()
    LLB5 = LLB5_n[math.ceil(R/100 * N - 1)]

    if assignedlist[node] != [] and nodelist[node][p_usedstation] + LLB5 >= incumbent:
        if delete == 1:
            del nodelist[node]
            del assignedlist[node]
            del unassignedlist[node]
            del Tklist[node]
        else:
            nodelist[node][p_state] = 'Fathomed (LLB5)'

        fathomed = True
        count_LLB5 += 1

    return LLB5


def f_LLB6(node, delete=1):
    global nodelist
    global assignedlist
    global unassignedlist
    global LLB6
    global fathomed
    global count_LLB6
    global LLB6_n
    global currentstation

    fathomed = False

    LLB6_n = []

    for n in range(N):
        local_sorted_t_n = [t_n[n][i-1] for i in unassignedlist[node]]
        local_sorted_t_n.sort(reverse=True)

        LLB6_n.append(min([m for m in range(1,tasks-nodelist[node][p_usedstation]+1)
                        if max([sum([local_sorted_t_n[h*m+1-i-1] for i in range(0,h+1)])
                        for h in range(1,math.floor((len(local_sorted_t_n)-1)/m)+1)]+[0]) <= c]))

    LLB6_n.sort()
    LLB6 = LLB6_n[math.ceil(R/100 * N - 1)]

    if assignedlist[node] != [] and nodelist[node][p_usedstation] + LLB6 >= incumbent:
        if delete == 1:
            del nodelist[node]
            del assignedlist[node]
            del unassignedlist[node]
            del Tklist[node]
        else:
            nodelist[node][p_state] = 'Fathomed (LLB6)'

        fathomed = True
        count_LLB6 += 1
```

```
    return LLB6


def f_LLB7(node, delete=1):
    global nodelist
    global assignedlist
    global unassignedlist
    global LLB7
    global fathomed
    global count_LLB7
    global LLB7_n

    global local_n_j
    global local_a_j
    global local_earliest_n
    global local_latest_n

    fathomed = False

    if signaltoLLBs != node:
        local_n_j = [[0 for i in LB4_tasklist] for n in range(N)]
        local_a_j = [[0 for i in LB4_tasklist] for n in range(N)]

        local_preceder = [[i for i in preceder[j-1] if i in unassignedlist[node] and j in unassignedlist[node]]
                          for j in tasklist ]
        local_follower = [[i for i in follower[j-1] if i in unassignedlist[node] and j in unassignedlist[node]]
                          for j in tasklist ]

        local_earliest_n1 = [[ math.ceil((t_n[n][i-1] + sum([ t_n[n][j-1] for j in local_preceder[i-1] ]))/c)
                               for i in tasklist ]
                               for n in range(N)]
        local_earliest_n  = [[max(local_earliest_n1[n][i-1], math.ceil(local_a_j[n][i] + p_j[n][i])) for i in tasklist]
                               for n in range(N)]
        local_latest_n1   = [[math.ceil((t_n[n][i-1] + sum([t_n[n][j-1] for j in local_follower[i-1]]))/c)
                               for i in tasklist  ]
                               for n in range(N)]
        local_latest_n    = [[[m + 1 - max(local_latest_n1[n][i-1], math.ceil(p_j[n][i] + local_n_j[n][i]))
                               for i in tasklist]
                               for m in range(tasks+1)] for n in range(N)]

    LLB7_n = []

    for n in range(N):
        #local_trial_m = max(LLB1, LLB2, LLB3, LLB4, LLB5, LLB6)
        local_trial_m = 1
        local_combinations = [[m_1, m_2, m_2-m_1] for m_1 in range(1,local_trial_m+1) for m_2 in range(1,local_trial_m+1)
                              if m_1 <= m_2]
        local_combinations.sort(key = lambda l: (l[2])) #sort combinations in increasing order of interval length
        for combination in range(len(local_combinations)):
            del local_combinations[combination][2]
        local_found_contr = True

        while local_found_contr == True:
            for combination in range(len(local_combinations)):
                m_1 = local_combinations[combination][0]
                m_2 = local_combinations[combination][1]

                local_N7_n = [j for j in [i for i in unassignedlist[node] if local_earliest_n[n][i-1] >= m_1
                              and local_latest_n[n][local_trial_m][i-1] <= m_2]]

                #Apply LB1
                if math.ceil(sum([t_n[n][i-1] for i in local_N7_n])/c) > m_2 - m_1 + 1:
                    local_trial_m += 1
                    if local_trial_m == tasks:
                        local_found_contr = False
                        break

                    local_combinations = [[m_1, m_2, m_2-m_1] for m_1 in range(1,local_trial_m+1)
                                          for m_2 in range(1,local_trial_m+1) if m_1 <= m_2]
                    local_combinations.sort(key = lambda l: (l[2]))
                    #sort combinations in increasing order of interval length
                    for combination in range(len(local_combinations)):
                        del local_combinations[combination][2]
                    break

            else:
                local_found_contr = False
```

```
            LLB7_n.append(local_trial_m)

        LLB7_n.sort()
        LLB7 = LLB7_n[math.ceil(R/100 * N - 1)]

        if assignedlist[node] != [] and nodelist[node][p_usedstation] + LLB7 >= incumbent:
            if delete == 1:
                del nodelist[node]
                del assignedlist[node]
                del unassignedlist[node]
                del Tklist[node]
            else:
                nodelist[node][p_state] = 'Fathomed (LLB7)'

            fathomed = True
            count_LLB7 += 1

        return LLB7


    def f_logical(node, delete=1):
        global nodelist
        global assignedlist
        global unassignedlist
        global fathomed
        global count_INF

        fathomed = False

        if assignedlist[node] == [] or unassignedlist[node] == []:
            pass

        else:
            f_completionrate(node)

            x_n = [ [  1 if t_n[n][i-1] > c and complete[n] == 1 else 0  for i in unassignedlist[node] ]
                    for n in range(N) ]
            # only count if the sample "helps" the current SL
            x = sum([ max(x_n[n]) for n in range(N) ])

            if (sum(complete) - x ) / N * 100 < R:  # if the maximum achievable SL is already below R
                if delete == 1:
                    del nodelist[node]
                    del assignedlist[node]
                    del unassignedlist[node]
                    del Tklist[node]
                else:
                    nodelist[node][p_state] = 'Fathomed (INF)'

                fathomed = True
                count_INF += 1


    def f_dominance(node, delete=1):
        """ Check if the just opened node is dominated """
        global nodelist
        global assignedlist
        global unassignedlist
        global count_DOM
        global fathomed

        fathomed = False

        if node == 1 or node == 2:
            return

        f_assigned(node)
        for k in range(len(nodelist)):
            if node != k and set(assigned) <= set(assignedlist[k]) \
                and nodelist[node][p_usedstation] >= nodelist[k][p_usedstation] \
                and nodelist[node][p_completion] <= nodelist[k][p_completion]:
                #print('Node ', node , ' is dominated by node ', k)

                if delete == 1:
                    del nodelist[node]
                    del assignedlist[node]
```

```
                del unassignedlist[node]
                del Tklist[node]
            else:
                nodelist[node][p_state] = 'Dominated'

            fathomed = True
            count_DOM += 1
            break

def f_update():
    global endloop
    global LB

    endloop = False

    local_LLBlist = [ nodelist[k][p_LLB] for k in range(len(nodelist)) if nodelist[k][p_state] == 'Open' ]
    if local_LLBlist == []:
        return

    if min(local_LLBlist) > LB:
        LB = min(local_LLBlist)
        print('\nLB has been increased to', LB)

        if incumbent == LB:
            endloop = True


#=============================================================================
#Import packages

import numpy as np                  # for sampling
import time                         # to time the calculation
import math                         # for ceiling function (LB)
import copy                         # for set of follower/preceder
import pprint                       # display nodelist
from scipy.stats import norm
from scipy.stats import gamma

#=============================================================================
#Initial Parameters (change)

model = open('Jackson11.txt', 'r')

c =             13                  # cycle time
R =             95

max_nodes   =   10000                 # maximum number of nodes
time_limit  =   10000
N           =   10000

distribution =  'Normal'
cv_gen =        0.1                   # same cv for all tasks
cv_ex =         []                    # except for [task, cv]

it = 1
np.random.seed(it)


#############################################################################
#Import Problem Data
#############################################################################

lines = (len(model.readlines()))
model.seek(0)

# load number of tasks

tasks = int(next(model))
print('Tasks:  ', tasks)

# load task times

t = []
for i in range(0,tasks) :
    t += [next(model)]

while '\n' in t :
```

```
    del t[t.index('\n')]

for i in range(0,len(t)) :
    t[i] = int(t[i])

# load precedence relations

P = []
P_ = []     # intermediate list

add = [0,0]
while add != ['-1,-1'] :
    add = [next(model)]
    P_ += (add)

    if add == ['-1,-1\n'] :
        break

for i in range(0,len(P_)) :
    P_[i] = P_[i].replace('\n','')
    P_[i] = P_[i].split(sep=',')


P = [ (int(P_[line][0]), int(P_[line][1])) for line in range(len(P_)-1)]


model.close
del P_, add      # data clean up

P_rev = [ (P[pair][1], P[pair][0]) for pair in range(len(P)) ] # reversed precedence relations for backward step
tasklist = [i+1 for i in range(tasks)]  # Determine the list of tasks based on 'tasks'


################################################################################
#Sampling
################################################################################

cv = [cv_gen for i in range(tasks)]

for x in range(len(cv_ex)):
    cv[cv_ex[x][0]-1] = cv_ex[x][1]


startsampling = time.time()

#Create deterministic set
di = [(q - 0.5)/N for q in range(1,N+1)]

t_nT = []
for i in range(tasks):
    if cv[i] == 0:      #in the deterministic case
        t_nT.append([t[i] for n in range(N)])

    else:               #in the stochastic case
        if distribution == 'Normal':
            t_nT.append([norm.ppf(q, loc = t[i], scale = cv[i]*t[i]) for q in di])
        elif distribution == 'Gamma':
            t_nT.append([gamma.ppf(q, a = 1/cv[i]**2, scale = t[i]*cv[i]**2) for q in di])


#Shuffle into random sequence
for i in range(tasks):
    np.random.shuffle(t_nT[i])


t_n = [[row[i] for row in t_nT] for i in range(N)]  # matrix of sampled task times t_n[n][i]


#truncation of normal distribution for negative values
truncations = 0     # counter for truncations
for n in range(N):
    while min(t_n[n]) < 0:
        t_n[n][np.argmin(t_n[n])] = 0
        truncations += 1


endsampling = time.time()
```

```
###############################################################################
#Data check
###############################################################################

#Is the problem feasible?
guaranteed_infeasible = [ n for n in range(N) if max(t_n[n]) > c ]

if (( N - len(guaranteed_infeasible) ) / N)*100 < R:
    print('\n\n!!! R is too high. Maximal R =', (( N - len(guaranteed_infeasible) ) / N)*100, '!!!' )
#    sys.exit('System is now exiting.\n\n')
    max_nodes = 0


del guaranteed_infeasible



###############################################################################
#End of pre-processing
###############################################################################


#==============================================================================
#System Parameters (do not change)

currentnode     = 0     # starting node
currenttask     = 1     # starting task
currentstation  = 0     # starting station
incumbent       = tasks # starting incumbent value = upper bound

parentnode      = 0     #starting parent node

nodechar        = 9     # number of entries to characterize the node before task list starts

p_number        = 0     # position of node number
p_direction     = 1     # position of direction
p_station       = 2     # position of current station
p_usedstation   = 3     # position of the number of used stations
p_time          = 4     # position of time
p_completion    = 5     # position of completion rate
p_LLB           = 6     # position of local lower bound
p_incumbent     = 7     # position of current incumbent
p_state         = 8     # position of node state


constructiontime = 0

LLB1            = 0
LLB2            = 0
LLB3            = 0
LLB4            = 0
LLB5            = 0
LLB6            = 0
LLB7            = 0


count_LLB1      = 0
count_LLB2      = 0
count_LLB3      = 0
count_LLB4      = 0
count_LLB5      = 0
count_LLB6      = 0
count_LLB7      = 0
count_DOM       = 0
count_INF       = 0

steps_f         = 0
steps_b         = 0
signaltoLLB5    = ' '

nodelist        = []
assignedlist    = []
unassignedlist  = []
Tklist          = []
assignable      = []
```

```
results = [['Model','LB1','LB2','LB3','LB5','Starting LB','Ending LB','Opt. sol.','Opt. assignment','Found at node',
'Found at time','c','N','cv','R','LR','CPUs','Opened nodes','Open nodes','Evaluated nodes','End nodes',
'Fathomed nodes','Dominated nodes','Infeasible nodes','Forward','Backward']]

endnode = False
endloop = False


#############################################################################
#Calculate LB
#############################################################################
#Initialize

nodelist.append([0, 'f', 1, 0, 0, 0.0, 0, incumbent, 'Open'])
nodelist[0] += ['_' for task in tasklist]
f_assigned(0)
assignedlist.append(assigned)
unassignedlist.append(unassigned)

#Precalculation for LB4-LB7
directfollower = [[P[pair][1] for pair in range(0,len(P)) if i==P[pair][0]-1] for i in range(tasks)]
directpreceder = [[P[pair][0] for pair in range(0,len(P)) if i==P[pair][1]-1] for i in range(tasks)]

follower = copy.deepcopy(directfollower)
for j in range(tasks):
    for iterate in range(tasks):   # do this so all appended tasks are also checked!
        for i in range(len(follower[j])):
            for ii in directfollower[follower[j][i]-1]:
                if ii not in follower[j]:
                    follower[j].append(ii)

preceder = copy.deepcopy(directpreceder)
for j in range(len(preceder)):
    for iterate in range(tasks):   # do this so all appended tasks are also checked!
        for i in range(len(preceder[j])):
            for ii in directpreceder[preceder[j][i]-1]:
                if ii not in preceder[j]:
                    preceder[j].append(ii)

LB4_t_n = [[0] + t_n[n] + [0] for n in range(N)]
LB4_tasks = len(LB4_t_n[0])
LB4_tasklist = [i for i in range(len(LB4_t_n[0]))]
LB4_tasklist_rev = LB4_tasklist[::-1]
p_j = [[LB4_t_n[n][i]/c for i in range(LB4_tasks)] for n in range(N)]
LB4_P = [(0,j) for j in [i+1 for i in range(tasks) if directpreceder[i]==[]]] + P + [(j,tasks+1)
            for j in [i+1 for i in range(tasks) if directfollower[i]==[]]]

LB4_directfollower = [[LB4_P[pair][1] for pair in range(0,len(LB4_P)) if i==LB4_P[pair][0]] for i in range(LB4_tasks)]
LB4_directpreceder = [[LB4_P[pair][0] for pair in range(0,len(LB4_P)) if i==LB4_P[pair][1]] for i in range(LB4_tasks)]

LB4_follower = copy.deepcopy(LB4_directfollower)
for j in range(tasks):
    for iterate in range(tasks):   # do this so all appended tasks are also checked!
        for i in range(len(LB4_follower[j])):
            for ii in LB4_directfollower[LB4_follower[j][i]]:
                if ii not in LB4_follower[j]:
                    LB4_follower[j].append(ii)


LB4_preceder = copy.deepcopy(LB4_directpreceder)
for j in range(len(LB4_preceder)):
    for iterate in range(tasks):   # do this so all appended tasks are also checked!
        for i in range(len(LB4_preceder[j])):
            for ii in LB4_directpreceder[LB4_preceder[j][i]]:
                if ii not in LB4_preceder[j]:
                    LB4_preceder[j].append(ii)


LB1 = f_LLB1(0,0)
LB2 = f_LLB2(0,0)
LB3 = f_LLB3(0,0)
LB4 = f_LLB4(0,0)
LB5 = f_LLB5(0,0)
LB6 = f_LLB6(0,0)
LB7 = f_LLB7(0,0)

earliest_n = local_earliest_n
```

```
latest_n = local_latest_n

LB = max([i for i in [LB1, LB2, LB3, LB4, LB5, LB6, LB7, 1] if type(i)==int])
LB_start = LB

#LB_start = 1
#LB = 1

print('LB: ', LB)

del nodelist[0], assignedlist[0], unassignedlist[0]
signaltoLLB5    = ' '

###############################################################################
#Greedy heuristic
###############################################################################
starttime = time.time() # starting time

if max_nodes > 0:
    f_greedy_heuristic()

if incumbent == LB:
    endtime = time.time()    # ending time

else:
    ###############################################################################
    #Begin of BnB
    ###############################################################################
    if max_nodes > 0:
        #create forwards root node
        nodelist.append([currentnode, 'f', currentstation, 0, round(time.time() - starttime, 2), 0.0, LB,
            incumbent, 'Open'])              # open new node
        nodelist[currentnode] += ['_' for task in tasklist]     # add list of tasks to this new node
        f_assigned(currentnode)
        assignedlist.append(assigned)
        unassignedlist.append(unassigned)
        Tklist.append(f_Tk(currentnode))
        currentnode += 1


        #create backwards root node
        nodelist.append([currentnode, 'b', incumbent+1, 0, round(time.time() - starttime, 2), 0.0, LB,
            incumbent, 'Open'])
        # open new node
        nodelist[currentnode] += ['_' for task in tasklist]     # add list of tasks to this new node
        f_assigned(currentnode)
        assignedlist.append(assigned)
        unassignedlist.append(unassigned)
        Tklist.append(f_Tk(currentnode))

        parentnode = currentnode

    while currentnode <= max_nodes and max_nodes > 0:
        if time.time() - starttime > time_limit:
            print('Time limit of', time_limit, 'seconds exceeded.')
            break
        # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        # Choose direction of next step

        f_findnode()
        f_direction()
        if endloop == True:
            break

        # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        # Perform next step

        f_branch(parentnode)
        if endloop == True:
            break


        # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        # Update LB

        f_update()
        if endloop == True:
```

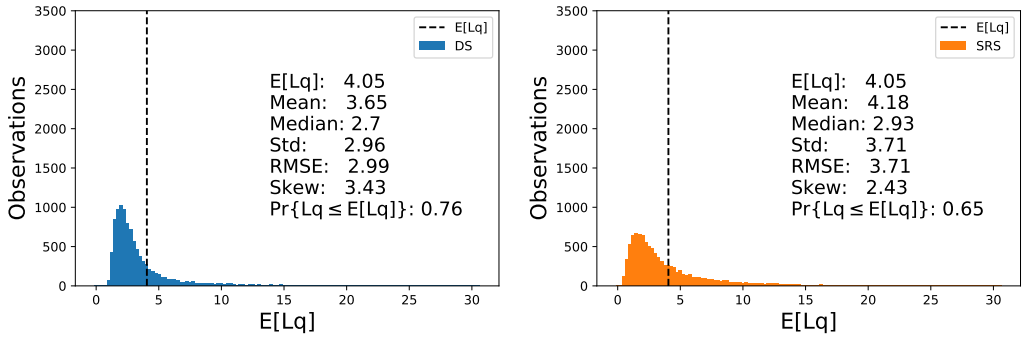
```
        break

else:
    print('\nMaximum number of nodes exceeded!')

endtime = time.time()    # ending time
```

Appendix E: Performance evaluation for an M/D/1 system based on queue length (Chapter 3)

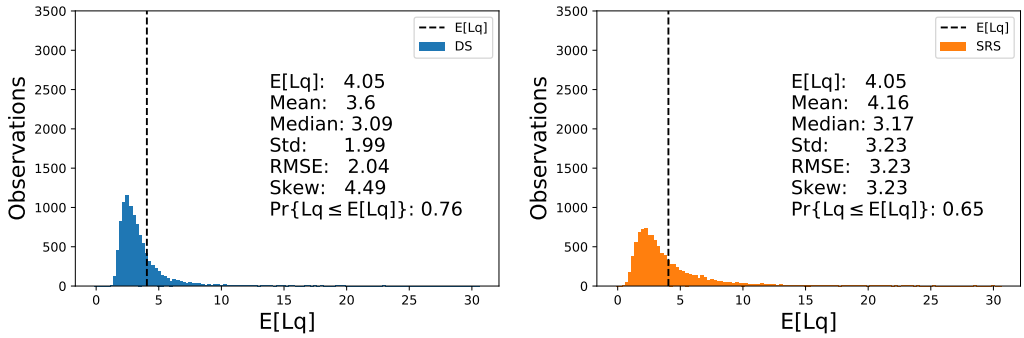
This Appendix shows the results for the sampling-based performance evaluation of an M/D/1 system as described in Chapter 3 with respect to the expected queue length and the standard deviation of the queue length. We use the same parameters as in Chapter 3: exponentially distributed inter-arrival times with rate $\lambda = 0.9$, deterministic processing rate $\mu = 1$, a warm-up length of $n_0 = 750$ and $R = 10000$ independent replications. The analytical expected queue length of this system based on Equation (3.2) is $E[L_q] = 4.05$ and the analytical standard deviation based on Equation (3.4) is $Std[L_q] = 4.78$.

Figures E.1 and E.2 show the resulting histograms of the expected queue length and Figures E.3 and E.4 the resulting histograms of the standard deviation of the queue length. For a sample size $N = 100$ and descriptive sampling, a large part of the distribution is below the analytical expected queue length. The RMSE is 2.99 and the skew is 3.43. The probability to observe at most the analytical value is 76%. For a sample size $N = 100$ and simple random sampling, again a large part of the distribution is below the analytical value. The RMSE is 3.71 and is higher than for DS. However, the distribution is more symmetrical, which can be seen by the lower skew of 2.43. The probability to observe at most the analytical value is 65% and therefore less than for DS. The same effects can be observed on the expected queue length and on the standard deviation of the queue length, as have been observed on the waiting time described in detail in Chapter 3. The root mean square error is decreasing in the sample size for the analyzed examples. However, for the same sample size, descriptive sampling always has a lower root mean square error than simple random sampling in the analyzed examples. For all performance measures, the distribution over all replications is skewed. The probability to underestimate

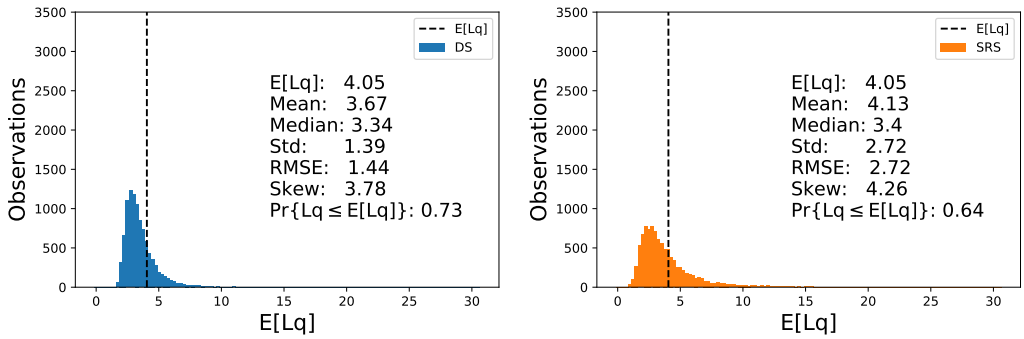
the desired performance measure is higher for descriptive sampling in the analyzed examples.



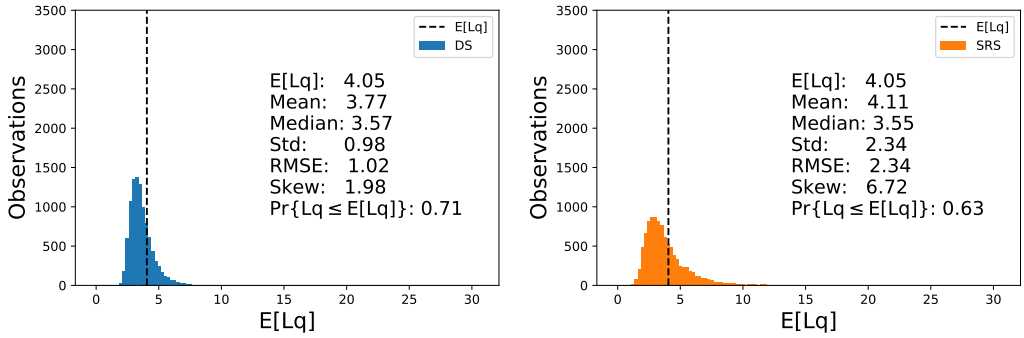
(a) Sample size $N = 100$



(b) Sample size $N = 250$

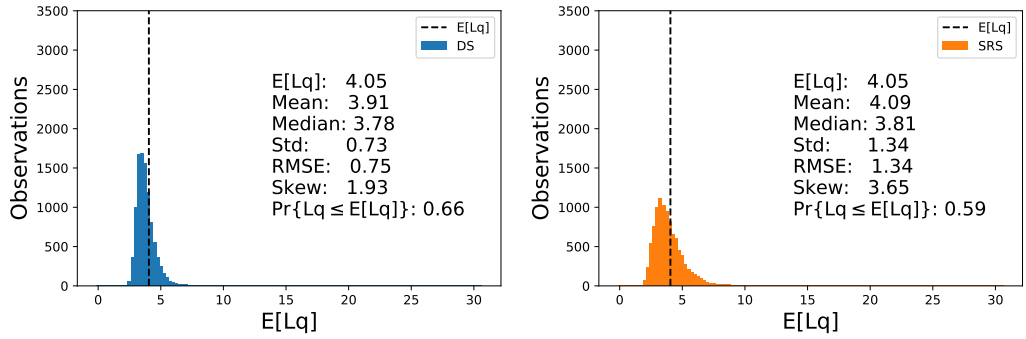


(c) Sample size $N = 500$

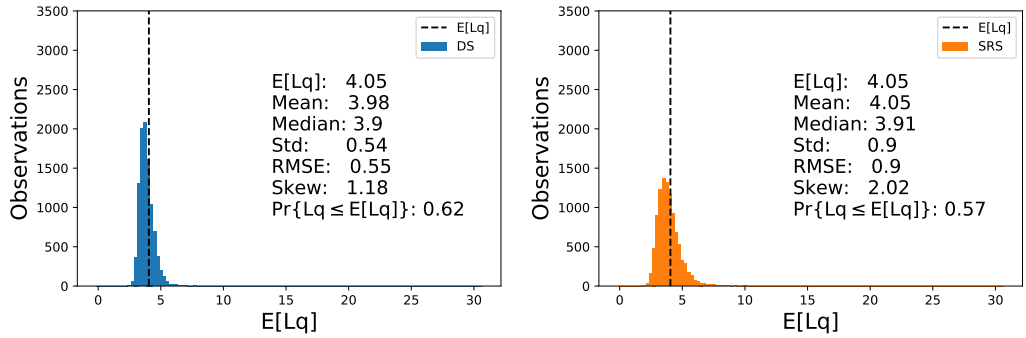


(d) Sample size $N = 1000$

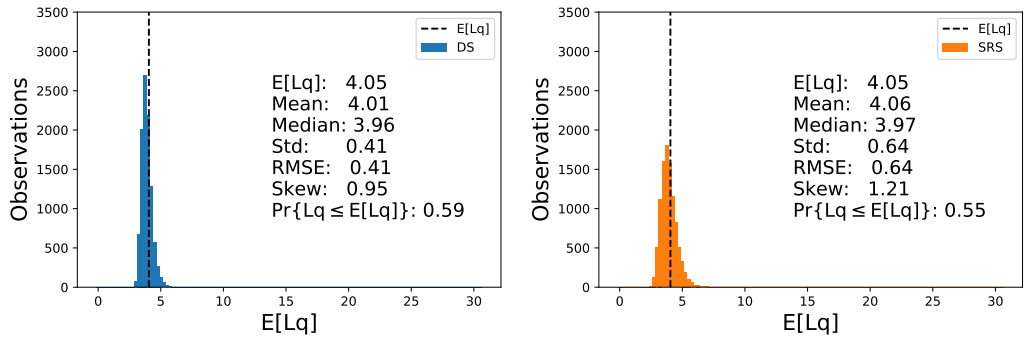
Figure E.1: Expected queue length in an M/D/1 system for sample sizes $N = 100$ to $N = 1000$



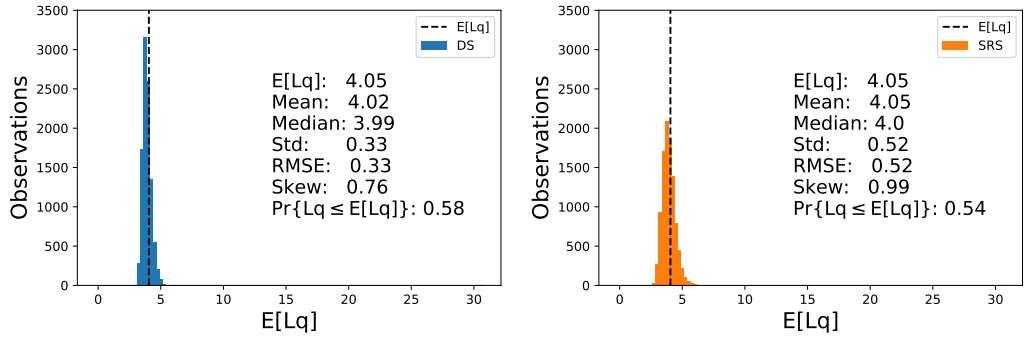
(a) Sample size $N = 2500$



(b) Sample size $N = 5000$

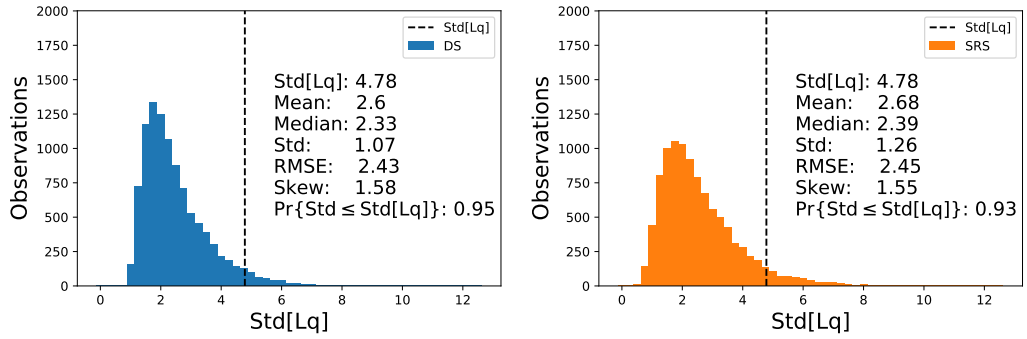


(c) Sample size $N = 10000$

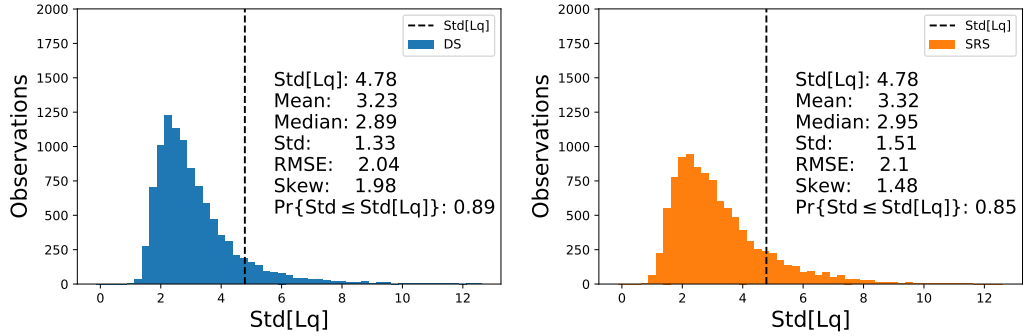


(d) Sample size $N = 15000$

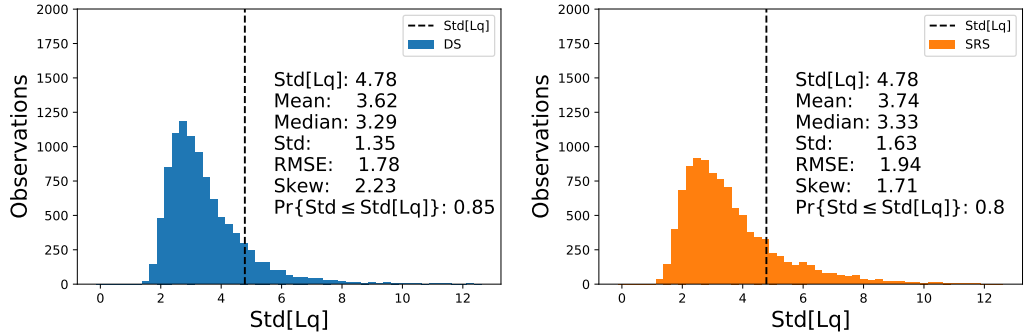
Figure E.2: Expected queue length in an M/D/1 system for sample sizes $N = 2500$ to $N = 15000$



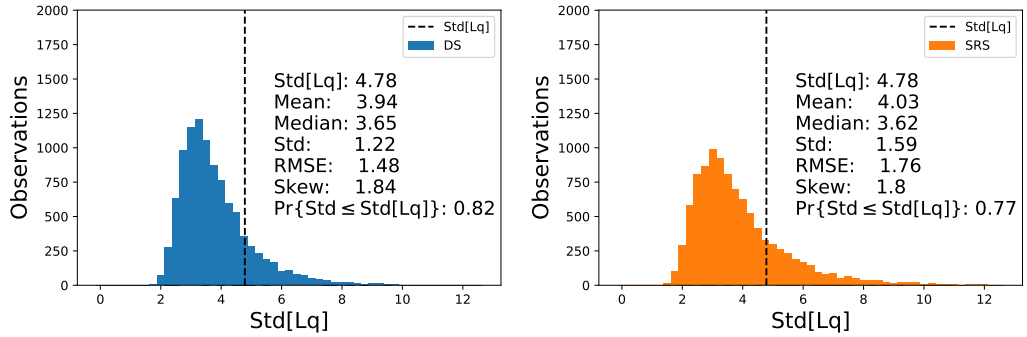
(a) Sample size $N = 100$



(b) Sample size $N = 250$

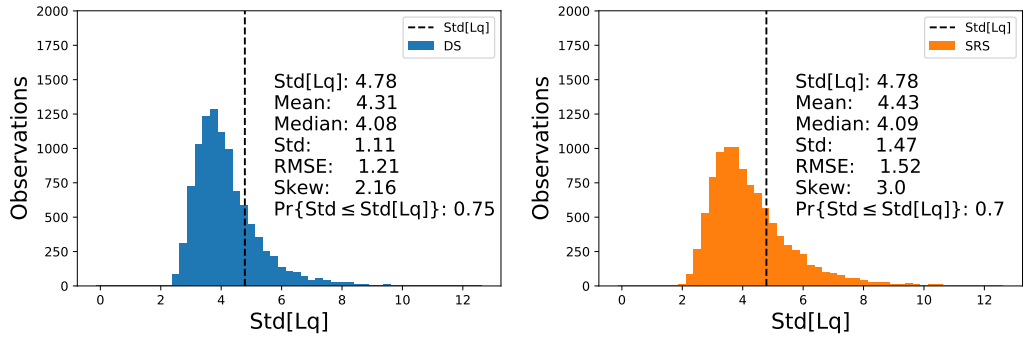


(c) Sample size $N = 500$

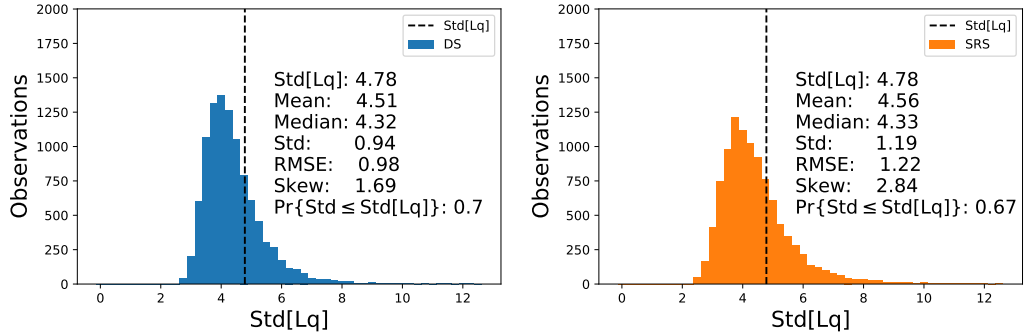


(d) Sample size $N = 1000$

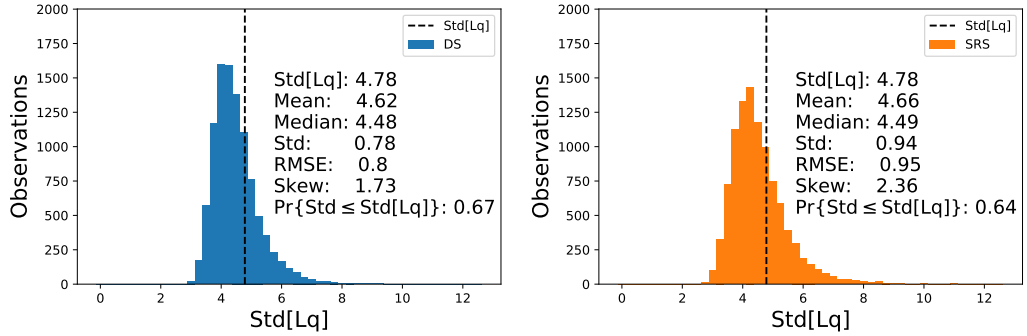
Figure E.3: Standard deviation of queue length in an M/D/1 system for sample sizes $N = 100$ to $N = 1000$



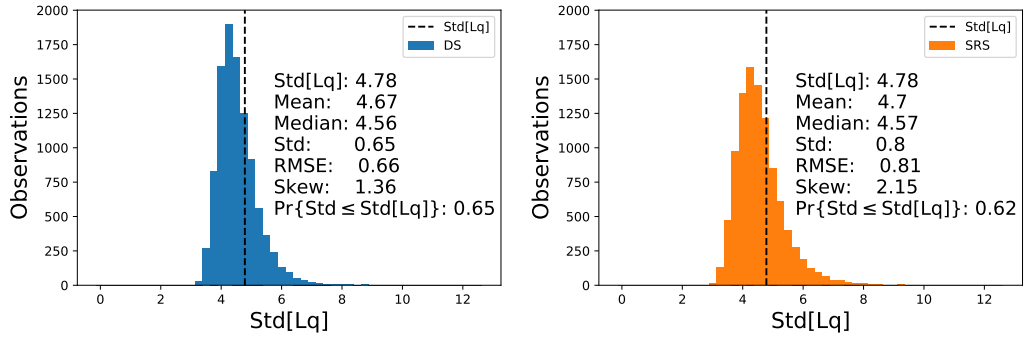
(a) Sample size $N = 2500$



(b) Sample size $N = 5000$



(c) Sample size $N = 10000$



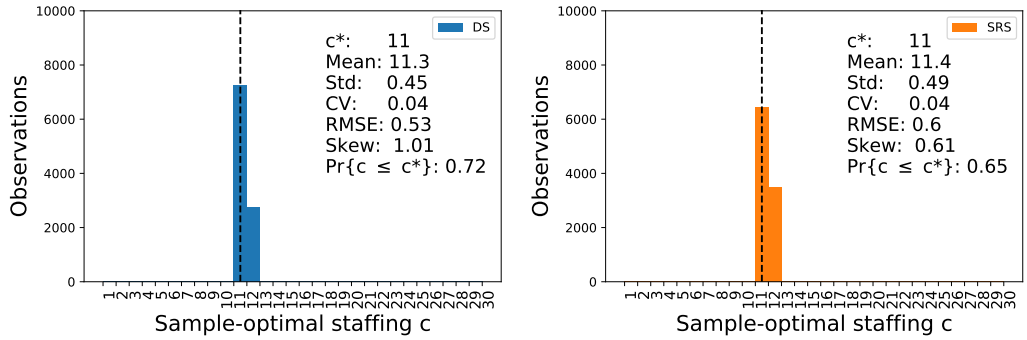
(d) Sample size $N = 15000$

Figure E.4: Standard deviation of queue length in an M/D/1 system for sample sizes $N = 2500$ to $N = 15000$

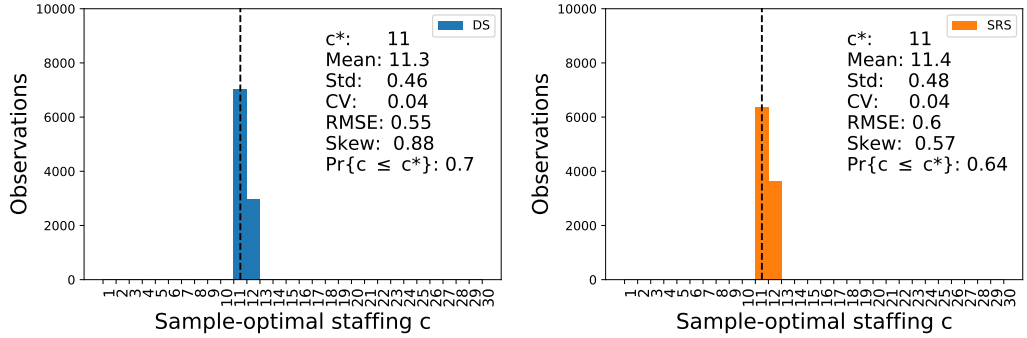
Appendix F: Optimization of an M/M/c staffing level for further sample sizes (Chapter 3)

This Appendix shows the results for the sampling-based optimization of an M/M/c system as described in Chapter 3 for larger sample sizes. We use the same parameters as in Chapter 3: exponentially distributed inter-arrival times with rate $\lambda = 10$, an exponential processing rate $\mu = 1$ per server c, a warm-up length of $n_0 = 750$ and $R = 10000$ independent replications. This appendix shows the detailed histograms for sample sizes from $N = 2500$ to $N = 15000$. We consider two examples with a constraint on the expected waiting time ($E[W_q] \leq 0.7$ and $E[W_q] \leq 0.000007$) and one with a constraint on the probability X of waiting at most a specified time Y ($Pr\{W_q \leq 0.7\} \geq 0.8$). The analytical optimal solutions based on Equations (3.9) and (3.10) are $c^* = 11$, $c^* = 25$ and $c^* = 12$, respectively.

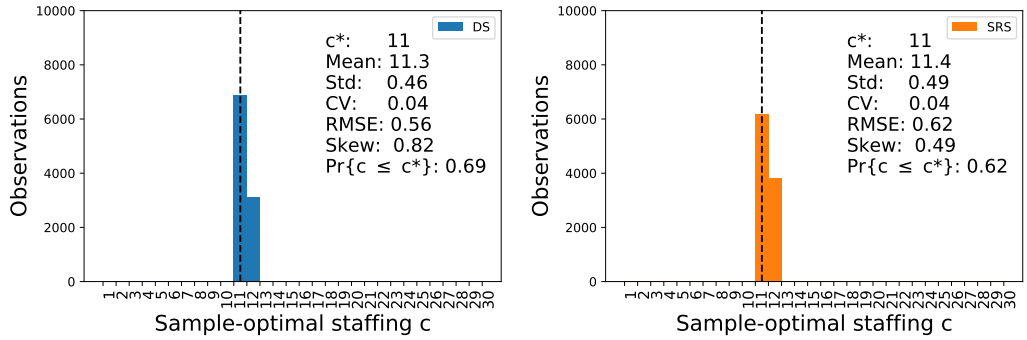
Figures F.1, F.2 and F.3 show the resulting histograms of the optimal staffing decision c^* for the three different considered examples. The analysis in Chapter 3 extends to these graphs. Therefore, the same effects on the mean, skew and root mean square error can be observed for these larger sample sizes.



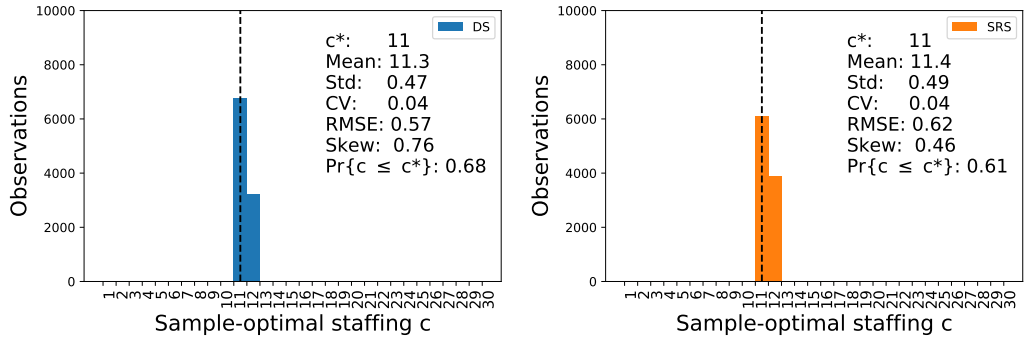
(a) Sample size $N = 2500$



(b) Sample size $N = 500$

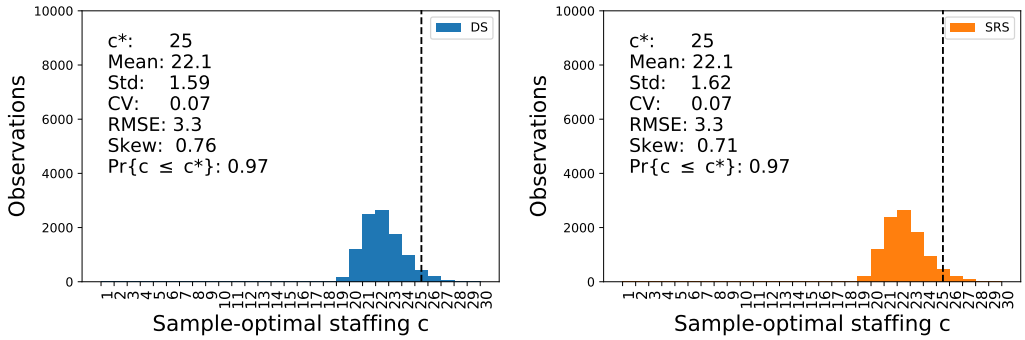


(c) Sample size $N = 10000$

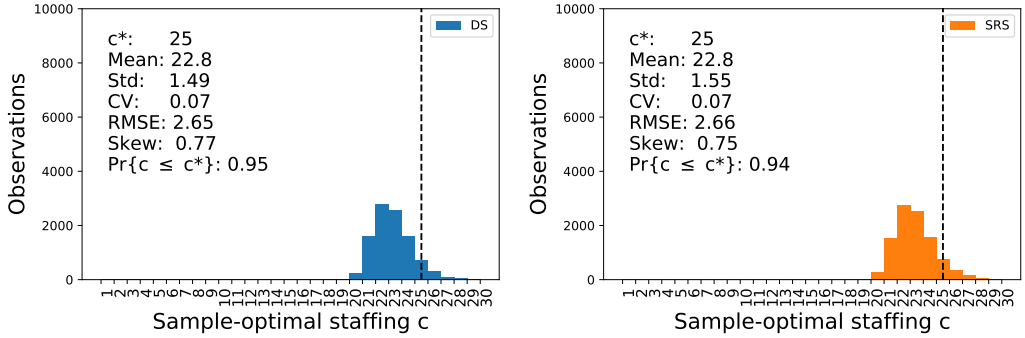


(d) Sample size $N = 15000$

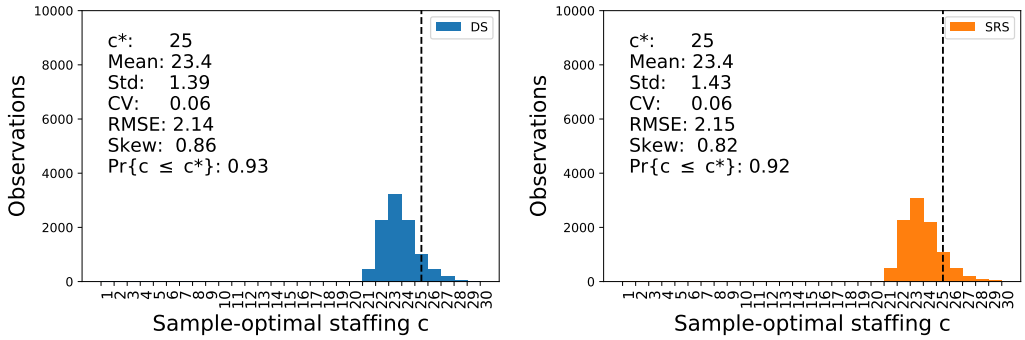
Figure F.1: Optimal number of servers in an M/M/c system with constraint on expected waiting time with $E[W_q] \leq 0.7$ for sample sizes $N = 2500$ to $N = 15000$



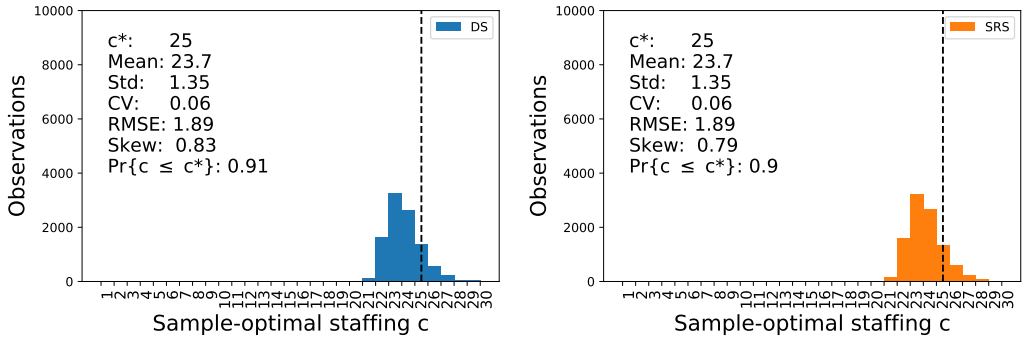
(a) Sample size $N = 2500$



(b) Sample size $N = 5000$

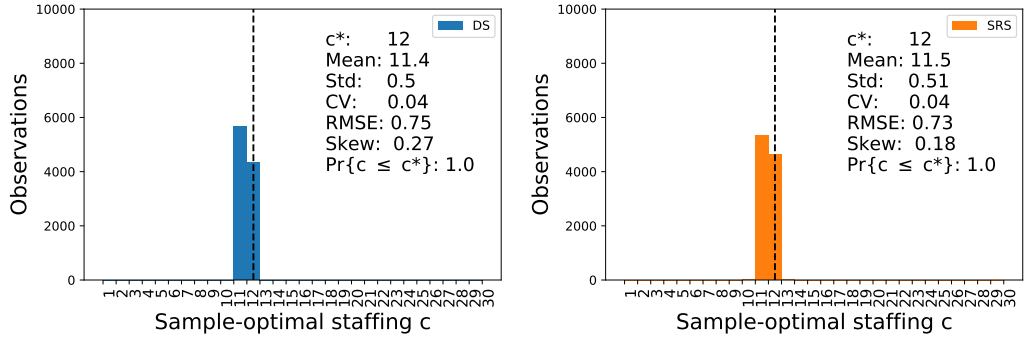


(c) Sample size $N = 10000$

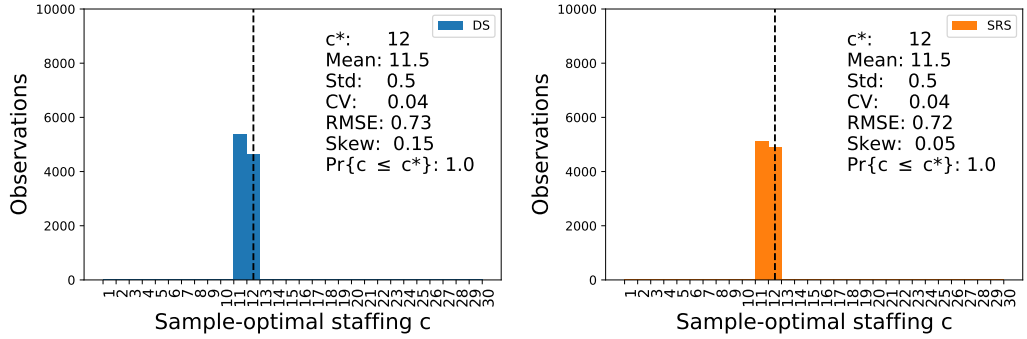


(d) Sample size $N = 15000$

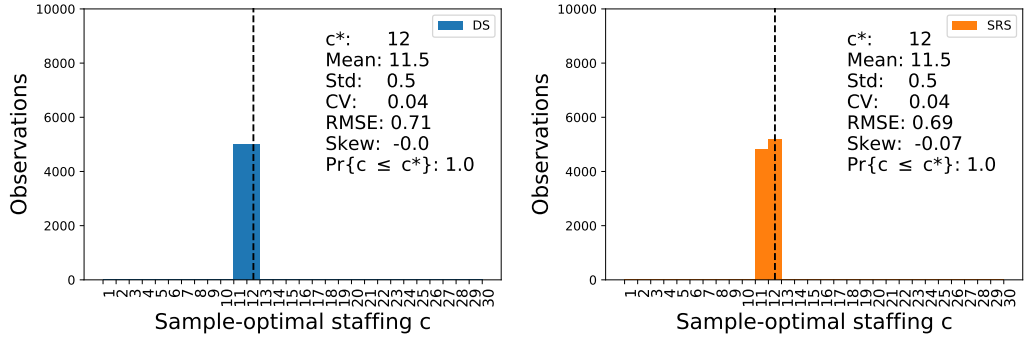
Figure F.2: Optimal number of servers in an M/M/c system with constraint on expected waiting time with $E[W_q] \leq 0.000007$ for sample sizes $N = 2500$ to $N = 15000$



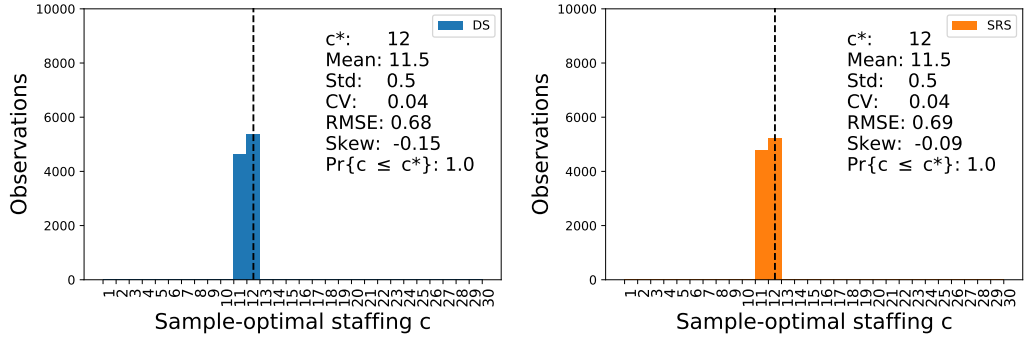
(a) Sample size $N = 2500$



(b) Sample size $N = 5000$



(c) Sample size $N = 10000$



(d) Sample size $N = 15000$

Figure F.3: Optimal number of servers in an M/M/c system with X/Y service level for sample sizes $N = 2500$ to $N = 15000$

Appendix G: Known Results from Choi et al. (2019) (Chapter 4)

For the normal approximation of the non-defective end products with $Q(x) \sim N\left(xp, \sqrt{xp(1-p)}^2\right)$, the following results are known (Choi et al., 2019) for stationary yield $p(i) = p(i+1) = p$ and a salvage value of $s = 0$:

1.	**Concavity:** The normal approximation of $\mathbb{E}[\Pi(x, Q(x))]$ converges to a concave function in x when the demand n is sufficiently large.
2a.	**Parameter sensitivity:** For the normal approximation of $\mathbb{E}[\Pi(x, Q(x))]$, the optimal solution x^* is decreasing in c and increasing in b.
2b.	**Parameter sensitivity for infinite demand:** For the normal approximation of $\mathbb{E}[\Pi(x, Q(x))]$ and $n \to \infty$, the optimal solution x^* is increasing in r and non-decreasing in n.

Table G.1: Known results for the normal approximation of the special case of stationary yield and no salvage value

Bibliography

Abdel-Malek, L., R. Montanari, and D. Meneghetti (2008). The capacitated newsboy problem with random yield: The gardener problem. *International Journal of Production Economics 115*(1), 113–127.

Abdel-Malek, L. L. and M. Otegbeye (2013). Separable programming/duality approach to solving the multi-product newsboy/gardener problem with linear constraints. *Applied Mathematical Modelling 37*(6), 4497–4508.

Ağpak, K. and H. Gökçen (2007). A chance-constrained approach to stochastic line balancing problem. *European Journal of Operational Research 180*(3), 1098–1115.

Alfares, H. K. and H. H. Elmorra (2005). The distribution-free newsboy problem: Extensions to the shortage penalty case. *International Journal of Production Economics 93*, 465–477.

Anzanello, M. J. and F. S. Fogliatto (2011). Learning curve models and applications: Literature review and research directions. *International Journal of Industrial Ergonomics 41*(5), 573–583.

Atlason, J., M. A. Epelman, and S. G. Henderson (2008). Optimizing call center staffing using simulation and analytic center cutting-plane methods. *Management Science 54*(2), 295–309.

Aydoğan, E. K., Y. Delice, U. Özcan, C. Gencer, and Ö. Bali (2019). Balancing stochastic U-lines using particle swarm optimization. *Journal of Intelligent Manufacturing 30*(1), 97–111.

Bagher, M., M. Zandieh, and H. Farsijani (2011). Balancing of stochastic U-type assembly lines: an imperialist competitive algorithm. *International Journal of Advanced Manufacturing Technology 54*(1-4), 271–285.

Ballestin, F. and R. Leus (2009). Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Production and Operations Management 18*(4), 459–474.

Baykasoğlu, A. and L. Özbakır (2007). Stochastic U-line balancing using genetic algorithms. *The International Journal of Advanced Manufacturing Technology 32*, 139–147.

Bentaha, M. L., O. Battaïa, A. Dolgui, and S. J. Hu (2015). Second order conic approximation for disassembly line design with joint probabilistic constraints. *European Journal of Operational Research 247*(3), 957–967.

Betts, J. and K. I. Mahmoud (1989). Identifying multiple solutions for assembly line balancing having stochastic task times. *Computers & Industrial Engineering 16*(3), 427–445.

Birge, J. R. and F. Louveaux (2011). *Introduction to stochastic programming*. Springer Science & Business Media.

Biskup, D. (2008). A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research 188*(2), 315–329.

Bohn, R. E. and C. Terwiesch (1999). The economics of yield-driven processes. *Journal of Operations Management 18*(1), 41–59.

Boysen, N. and M. Fliedner (2008). A versatile algorithm for assembly line balancing. *European Journal of Operational Research 184*(1), 39–56.

Cakir, B., F. Altiparmak, and B. Dengiz (2011). Multi-objective optimization of a stochastic assembly line balancing: A hybrid simulated annealing algorithm. *Computers & Industrial Engineering 60*(3), 376–384.

Calafiore, G. and M. C. Campi (2005). Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming 102*(1), 25–46.

Carraway, R. L. (1989). A Dynamic Programming Approach to Stochastic Assembly Line Balancing. *Management Science 35*(4), 459–471.

Cavagnini, R., M. Hewitt, and F. Maggioni (2020). Workforce production planning under uncertain learning rates. *International Journal of Production Economics 225*, 107590.

Chiang, W.-C. and T. L. Urban (2006). The stochastic U-line balancing problem : A heuristic procedure. *European Journal of Operational Research 175*, 1767–1781.

Chiang, W.-C., T. L. Urban, and C. Luo (2016). Balancing stochastic two-sided assembly lines. *International Journal of Production Research 54*(20), 6232–6250.

Choi, S., S. Jeon, J. Kim, and K. Park (2019). A newsvendor analysis of a binomial yield production process. *European Journal of Operational Research 273*(3), 983–991.

Cortés, P., L. Onieva, and J. Guadix (2010). Optimising and simulating the assembly line balancing problem in a motorcycle manufacturing company: a case study. *International Journal of Production Research 48*(12), 3637–3656.

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. *Computers & Operations Research 64*, 97–112.

Cunningham, S. P., C. J. Spanos, and K. Voros (1995). Semiconductor yield improvement: results and best practices. *IEEE Transactions on Semiconductor Manufacturing 8*(2), 103–109.

Delice, Y., E. K. Aydoğan, and U. Özcan (2016). Stochastic two-sided u-type assembly line balancing: a genetic algorithm approach. *International Journal of Production Research 54*(11), 3429–3451.

Doerr, K. H. and A. Arreola-Risa (2000). A worker-based approach for modeling variability in task completion times. *IIE Transactions 32*(7), 625–636.

Dong, J., L. Zhang, and T. Xiao (2018). A hybrid pso/sa algorithm for bi-criteria stochastic line balancing with flexible task times and zoning constraints. *Journal of Intelligent Manufacturing 29*(4), 737–751.

Ehrhardt, R. and L. Taube (1987). An inventory model with random replenishment quantities. *International Journal of Production Research 25*(12), 1795–1803.

Fathi, M., A. Nourmohammadi, A. H. C. Ng, and A. Syberfeldt (2019). An optimization model for balancing assembly lines with stochastic task times and zoning constraints. *IEEE Access 7*, 32537–32550.

Foroughi, A. and H. Gökçen (2019). A multiple rule-based genetic algorithm for cost-oriented stochastic assembly line balancing problem. *Assembly Automation 39*(1), 124–139.

Gallego, G. and I. Moon (1993). The distribution free newsboy problem: review and extensions. *Journal of the Operational Research Society 44*(8), 825–834.

Gerchak, Y., R. G. Vickson, and M. Parlar (1988). Periodic review production models with variable yield and uncertain demand. *IIE transactions 20*(2), 144–150.

Golenko-Ginzburg, D. and A. Gonik (1997). Stochastic network project scheduling with non-consumable limited resources. *International Journal of Production Economics 48*(1), 29–37.

Grasman, S. E., Z. Sari, and T. Sari (2007). Newsvendor solutions with general random yield distributions. *RAIRO Operations Research 41*(4), 455–464.

Graves, S. B. (1998). Statistical quality control of a multistep production process using total process yield. *Quality Engineering 11*(2), 187–195.

Graves, S. B. (2002). Six sigma rolled throughput yield. *Quality Engineering 14*(2), 257–266.

Grosse, E. H., C. H. Glock, and S. Müller (2015). Production economics and the learning curve: A meta-analysis. *International Journal of Production Economics 170*, 401–412.

Guerriero, F. and J. Miltenburg (2003). The stochastic U-line balancing problem. *Naval Research Logistics 50*(1), 31–57.

Helber, S., F. Sahling, and K. Schimmelpfeng (2013). Dynamic capacitated lot sizing with random demand and dynamic safety stocks. *OR spectrum 35*(1), 75–105.

Henig, M. I. (1986). Extensions of the dynamic programming method in the deterministic and stochastic assembly-line balancing problems. *Computers & Operations Research 13*(4), 443–449.

Hilger, T., F. Sahling, and H. Tempelmeier (2016). Capacitated dynamic production and remanufacturing planning under demand and return uncertainty. *OR spectrum 38*(4), 849–876.

Hoffmann, T. R. (1990). Assembly line balancing: a set of challenging problems. *International Journal of Production Research 28*(10), 1807–1815.

Hruska, J. (2019). Intel: Cpu shortage will extend into q3 2019. https://www.extremetech.com/computing/290374-intel-cpu-shortage-will-extend-into-q3-2019.

Inderfurth, K. (2004). Analytical solution for a single-period production-inventory problem with uniformly distributed yield and demand. *Central European Journal of Operations Research 12*(2), 117.

Kämpf, M. and P. Köchel (2006). Simulation-based sequencing and lot size optimisation for a production-and-inventory system with multiple items. *International Journal of Production Economics 104*(1), 191–200.

Kao, E. P. C. (1976). A Preference Order Dynamic Program for Stochastic Assembly Line Balancing. *Management Science 22*(10), 1051–1173.

Kao, E. P. C. (1979). Computational experience with a stochastic assembly line balancing algorithm. *Computers & Operations Research 6*(2), 79–86.

Keren, B. (2009). The single-period inventory problem: Extension to random yield from the perspective of the supply chain. *Omega 37*(4), 801–810.

Khouja, M. (1999). The single-period (news-vendor) problem: literature review and suggestions for future research. *Omega 27*(5), 537–553.

Kim, T. (2018). Jp morgan says intel's chip shortage is 'worsening' and will hurt fourth-quarter pc sales by up to 7%. https://www.cnbc.com/2018/09/14/jp-morgan-says-intels-chip-shortage-is-worsening-and-will-hurt-pc-sales.html.

Kubota, Y., T. Mickle, and T. Mochizuki (2017). iphone's summer production glitches create holiday jitters. https://www.wsj.com/articles/iphones-summer-production-glitches-create-holiday-jitters-1504801636.

Kumar, N., K. Kennedy, K. Gildersleeve, R. Abelson, C. Mastrangelo, and D. Montgomery (2006). A review of yield modelling techniques for semiconductor manufacturing. *International Journal of Production Research 44*(23), 5019–5036.

Lapierre, S. D. and A. B. Ruiz (2004). Balancing assembly lines: an industrial case study. *Journal of the Operational Research Society 55*(6), 589–597.

Lee, D.-H., J.-K. Yang, C.-H. Lee, and K.-J. Kim (2019). A data-driven approach to selection of critical process steps in the semiconductor manufacturing process considering missing and imbalanced data. *Journal of Manufacturing Systems 52*, 146–156.

Lee, H. L. and C. A. Yano (1988). Production control in multistage systems with variable yield losses. *Operations Research 36*(2), 269–278.

Leitold, D., A. Vathy-Fogarassy, and J. Abonyi (2019). Empirical working time distribution-based line balancing with integrated simulated annealing and dynamic programming. *Central European Journal of Operations Research 27*, 455–473.

Li, Q. and S. Zheng (2006). Joint inventory replenishment and pricing control for systems with uncertain yield and demand. *Operations Research 54*(4), 696–705.

Li, X., Y. Li, and X. Cai (2012). A note on the random yield from the perspective of the supply chain. *Omega 40*(5), 601–610.

Liu, C., J. Wang, and J. Y.-T. Leung (2016). Worker assignment and production planning with learning and forgetting in manufacturing cells by hybrid bacteria foraging algorithm. *Computers & Industrial Engineering 96*, 162–179.

Liu, S. B., H. L. Ong, and H. C. Huang (2005). A bidirectional heuristic for stochastic assembly line balancing Type II problem. *The International Journal of Advanced Manufacturing Technology 25*(1-2), 71–77.

Luedtke, J. and S. Ahmed (2008). A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization 19*(2), 674–699.

Mazzola, J. B., A. W. Neebe, and C. M. Rump (1998). Multiproduct production planning in the presence of work-force learning. *European Journal of Operational Research 106*(2-3), 336–356.

McMullen, P. R. and G. Frazier (1998). Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations. *International Journal of Production Research 36*(10), 2717–2741.

McMullen, P. R. and G. V. Frazier (1997). A heuristic for solving mixed-model line balancing problems with stochastic task durations and parallel stations. *International Journal of Production Economics 51*, 177 – 190.

McMullen, P. R. and P. Tarasewich (2003). Using ant techniques to solve the assembly line balancing problem. *IIE Transactions 35*, 605–617.

Merengo, C., F. Nava, and A. Pozzetti (1999). Balancing and sequencing manual mixed-model assembly lines. *International Journal of Production Research 37*(12), 2835–2860.

Milor, L. (2013). A survey of yield modeling and yield enhancement methods. *IEEE transactions on semiconductor manufacturing 26*(2), 196–213.

Moodie, C. and H. Young (1965). A heuristic method of assembly line balancing for assumptions of constant or variable work element times. *Journal of Industrial Engineering 16*(1), 23–29.

Nkasu, M. M. and K. H. Leung (1995). A stochastic approach to assembly line balancing. *International Journal of Production Research 33*(4), 975–991.

Noori, H. and G. Keller (1986). One-period order quantity strategy with uncertain match between the amount received and quantity requisitioned. *INFOR: Information Systems and Operational Research 24*(1), 1–11.

Okyay, H., F. Karaesmen, and S. Özekici (2014). Newsvendor models with dependent random supply and demand. *Optimization Letters 8*(3), 983–999.

Okyay, H., F. Karaesmen, and S. Özekici (2015). Hedging demand and supply risks in the newsvendor model. *OR Spectrum 37*(2), 475–501.

Otto, A., C. Otto, and A. Scholl (2013). Systematic data generation and test design for solution algorithms on the example of salbpgen for assembly line balancing. *European Journal of Operational Research 228*(1), 33–45.

Özcan, U. (2010). Balancing stochastic two-sided assembly lines: A chance-constrained, piecewise-linear, mixed integer program and a simulated annealing algorithm. *European Journal of Operational Research 205*(1), 81–97.

Özcan, U. (2018). Balancing stochastic parallel assembly lines. *Computers & Operations Research 99*, 109–122.

Özceylan, E., C. B. Kalayci, A. Güngör, and S. M. Gupta (2019). Disassembly line balancing problem: a review of the state of the art and future directions. *International Journal of Production Research 57*(15-16), 4805–4827.

Pape, T. (2015). Heuristics and lower bounds for the simple assembly line balancing problem type 1: Overview, computational tests and improvements. *European Journal of Operational Research 240*(1), 32–42.

Pereira, J. (2015). Empirical evaluation of lower bounding methods for the simple assembly line balancing problem. *International Journal of Production Research 53*(11), 3327–3340.

Raouf, A. and C. L. Tsui (1982). A new Method for Assembly Line Balancing having Stochastic Work Elements. *Computers & Industrial Engineering 6*(2), 131–148.

Reeve, N. R. and W. H. Thomas (1973). Balancing stochastic assembly lines. *AIIE Transactions 5*(3), 223–229.

Rekik, Y., E. Sahin, and Y. Dallery (2007). A comprehensive analysis of the newsvendor model with unreliable supply. *OR Spectrum 29*(2), 207–233.

Saliby, E. (1990). Descriptive Sampling: A Better Approach to Monte Carlo Simulation. *The Journal of the Operational Research Society 41*(12), 1133–1142.

Saliby, E. and F. Pacheco (2002). An empirical evaluation of sampling methods in risk analysis simulation: quasi-monte carlo, descriptive sampling, and latin hypercube sampling. In *Proceedings of the Winter Simulation Conference, 2002*, Volume 2, pp. 1606–1610. IEEE.

Sarin, S. C., E. Erel, and E. M. Dar-El (1999). A methodology for solving single-model, stochastic assembly line balancing problem. *Omega 27*(5), 525–535.

Sayın, F., F. Karaesmen, and S. Özekici (2014). Newsvendor model with random supply and financial hedging: Utility-based approach. *International Journal of Production Economics 154*, 178–189.

Scholl, A. (1993). Data of assembly line balancing problems. *Schriften zur Quantitativen Betriebswirtschaftslehre, TU Darmstadt 16/93*.

Scholl, A. and C. Becker (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research 168*(3), 666–693.

Scholl, A. and R. Klein (1997). Salome - a bidirectional branch-and-bound procedure for assembly line balancing. *INFORMS Journal on Computing 9*(4), 319–451.

Scholl, A. and R. Klein (1999). ULINO: Optimally balancing U-shaped JIT assembly lines. *International Journal of Production Research 37*(4), 721–736.

Shih, W. (1980). Optimal inventory policies when stockouts result from defective products. *International Journal of Production Research 18*(6), 677–686.

Shortle, J. F., J. M. Thompson, D. Gross, and C. M. Harris (2018). *Fundamentals of queueing theory*, Volume 399. John Wiley & Sons.

Silverman, F. N. and J. C. Carter (1986). A cost-based methodology for stochastic line balancing with intermittent line stoppages. *Management Science 32*(4), 455–463.

Sniedovich, M. (1981). Analysis of a Preference Order Assembly Line Problem. *Management Science 27*(9), 1067–1080.

Sphicas, G. P. and F. N. Silverman (1976). Deterministic Equivalents for Stochastic Assembly Line Balancing. *AIIE Transactions 8*(2), 280–282.

Stolletz, R. (2022). Optimization of time-dependent queueing systems. *Queueing Systems*, 1–3.

Stolletz, R. and S. Weiss (2013). Buffer allocation using exact linear programming formulations and sampling approaches. *IFAC Proceedings Volumes 46*(9), 1435–1440.

Tang, C. S. and R. Yin (2007). Responsive pricing under supply uncertainty. *European Journal of Operational Research 182*(1), 239–255.

Tang, O., S. N. Musa, and J. Li (2012). Dynamic pricing in the newsvendor problem with yield risks. *International Journal of Production Economics 139*(1), 127–134.

Tang, Q., Z. Li, L. Zhang, and C. Zhang (2017). Balancing stochastic two-sided assembly line with multiple constraints using hybrid teaching-learning-based optimization algorithm. *Computers & Operations Research 82*, 102–113.

Tirkel, I. (2013). Yield learning curve models in semiconductor manufacturing. *IEEE transactions on semiconductor manufacturing 26*(4), 564–571.

Toksarı, M. D., S. K. İşleyen, E. Güner, and Ö. F. Baykoç (2008). Simple and u-type assembly line balancing problems with a learning effect. *Applied Mathematical Modelling 32*(12), 2954–2961.

Urban, T. L. and W.-C. Chiang (2006). An optimal piecewise-linear program for the U-line balancing problem with stochastic task times. *European Journal of Operational Research 168*(3), 771–782.

Wang, Y. and Y. Gerchak (1996). Periodic review production models with variable capacity, random yield, and uncertain demand. *Management Science 42*(1), 130–137.

Weber, C. (2004). Yield learning and the sources of profitability in semiconductor manufacturing and process development. *IEEE Transactions on Semiconductor Manufacturing 17*(4), 590–596.

Weiss, S. and R. Stolletz (2015). Buffer allocation in stochastic flow lines via sample-based optimization with initial bounds. *OR Spectrum 37*(4), 869–902.

Wright, T. P. (1936). Factors affecting the cost of airplanes. *Journal of the aeronautical sciences 3*(4), 122–128.

Yano, C. A. and H. L. Lee (1995). Lot sizing with random yields: A review. *Operations research 43*(2), 311–334.

Yelle, L. E. (1979). The learning curve: Historical review and comprehensive survey. *Decision Sciences 10*(2), 302–328.

Yüceer, Ü. (2002). Discrete convexity: convexity for functions defined on discrete spaces. *Discrete Applied Mathematics 119*(3), 297–304.

Zhang, H., C. Zhang, Y. Peng, D. Wang, G. Tian, X. Liu, and Y. Peng (2018). Balancing problem of stochastic large-scale u-type assembly lines using a modified evolutionary algorithm. *IEEE Access 6*, 78414–78424.

Curriculum vitae

Personal Information

Name	Johannes Diefenbach (né Schnitzler)

Education

2016 – 2021	Doctoral Student Graduate School of Economic and Social Sciences, University of Mannheim
2014 – 2016	Mannheim Master in Management (M. Sc.) Business School, University of Mannheim
2011 – 2014	Studies in Business Administration (B. Sc.) Business School, University of Mannheim
2010	Abitur, Carl-Benz-Gymnasium Ladenburg

Professional Experience

2021 – 2022	Senior Consultant for Supply Chain Risk and Resilience Camelot ITLab GmbH, Mannheim
2016 – 2021	Research Assistant, Chair of Production Management Business School, University of Mannheim