8-2023

Modeling and Solution Methodologies for Mixed-Model Sequencing in Automobile Industry

Ibrahim Ozan Yilmazlar
Clemson University, iyilmaz@clemson.edu

MODELING AND SOLUTION METHODOLOGIES FOR MIXED-MODEL SEQUENCING IN AUTOMOBILE INDUSTRY

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Industrial Engineering

---

by
Ibrahim Ozan Yilmazlar
August 2023

Accepted by:
Dr. Mary Elizabeth Kurz, Committee Chair
Dr. Hamed Rahimian, Committee Co-chair
Dr. Scott J. Mason
Dr. Kevin Taaffe

Abstract

The global competitive environment leads companies to consider how to produce high-quality products at a lower cost. Mixed-model assembly lines are often designed such that average station work satisfies the time allocated to each station, but some models with work-intensive options require more than the allocated time. Sequencing varying models in a mixed-model assembly line, mixed-model sequencing (MMS), is a short-term decision problem that has the objective of preventing line stoppage resulting from a station work overload. Accordingly, a good allocation of models is necessary to avoid work overload. The car sequencing problem (CSP) is a specific version of the MMS that minimizes work overload by controlling the sequence of models. In order to do that, CSP restricts the number of work-intensive options by applying capacity rules. Consequently, the objective is to find the sequence with the minimum number of capacity rule violations.

In this dissertation, we provide exact and heuristic solution approaches to solve different variants of MMS and CSP. First, we provide five improved lower bounds for benchmark CSP instances by solving problems optimally with a subset of options. We present four local search metaheuristics adapting efficient transformation operators to solve CSP. The computational experiments show that the Adaptive Local Search provides a significant advantage by not requiring tuning on the operator weights due to its adaptive control mechanism.

Additionally, we propose a two-stage stochastic program for the mixed-model sequencing (MMS) problem with stochastic product failures, and provide improvements to the second-stage problem. To tackle the exponential number of scenarios, we employ the sample average approximation approach and two solution methodologies. On one hand, we develop an L-shaped decomposition-based algorithm, where the computational experiments show its superiority over solving the deterministic equivalent formulation with an off-the-shelf solver. We also provide a tabu search algorithm in addition to a greedy heuristic to tackle case study instances inspired by our car manufacturer part-

ner. Numerical experiments show that the proposed solution methodologies generate high-quality solutions by utilizing a sample of scenarios. Particularly, a robust sequence that is generated by considering car failures can decrease the expected work overload by more than 20% for both small- and large-sized instances. To the best of our knowledge, this is the first study that considers stochastic failures of products in MMS.

Moreover, we propose a two-stage stochastic program and formulation improvements for a mixed-model sequencing problem with stochastic product failures and integrated reinsertion process. We present a bi-objective evolutionary optimization algorithm, a two-stage bi-objective local search algorithm, and a hybrid local search integrated evolutionary optimization algorithm to tackle the proposed problem. Numerical experiments over a case study show that while the hybrid algorithm provides a better exploration of the Pareto front representation and more reliable solutions in terms of waiting time of failed vehicles, the local search algorithm provides more reliable solutions in terms of work overload objective. Finally, dynamic reinsertion simulations are executed over industry-inspired instances to assess the quality of the solutions. The results show that integrating the reinsertion process in addition to considering vehicle failures can keep reducing the work overload by around 20% while significantly decreasing the waiting time of the failed vehicles.

Dedication

To my life companion, Nayda Ates, and my beloved family,

Your unwavering love, support, and encouragement have been the bedrock of my journey. Through the highs and lows, you have been my guiding light and source of strength. This achievement is a testament to our shared dreams and the endless belief you have in me. With heartfelt gratitude, I dedicate this dissertation to you, my greatest sources of inspiration and love.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Mary Elizabeth Kurz, for their unwavering support, guidance, and invaluable expertise throughout the entire process of this dissertation. Their dedication to my academic growth and willingness to share their knowledge and insights have been instrumental in shaping this research. I am truly grateful for their constant support, encouragement, and exceptional mentorship.

I am also indebted to my dissertation committee co-chair, Dr. Hamed Rahimian, and committee members, Dr. Scott J. Mason, and Dr. Kevin Taaffe, for their intellectual rigor, critical feedback, and constructive suggestions that have played a pivotal role in shaping the direction and depth of my research. I sincerely appreciate their expertise and commitment to academic excellence. A special mention is owed to my committee co-chair Dr. Hamed Rahimian, who provided invaluable contributions to the third chapter of this dissertation. His insightful perspectives, meticulous attention to detail, and willingness to engage in thoughtful discussions have greatly enriched the quality and depth of my research findings. I would like to extend my sincere appreciation to my undergraduate advisor, Dr. Emine Yaylali, for her unwavering support and guidance throughout my early academic journey. Her invaluable mentorship not only allowed me to explore my passion field but also provided me with invaluable perspectives that have greatly enriched my academic growth.

I am also grateful to my significant other, my family, and my friends for their unwavering support and understanding throughout this challenging journey. Their love, encouragement, and belief in my abilities have been a constant source of motivation.

Finally, I would like to extend my heartfelt appreciation to all those who have supported me on this intellectual expedition. Your guidance, encouragement, and contributions have made an indelible impact on my academic and personal growth. Thank you for being an integral part of this transformative journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Background and Motivation

1.1 Assembly Lines

Assembly lines are flow-oriented production systems used to produce standardized products in high volume. The mass production process starts with a workpiece that is carried by a conveyor. Conveyors generally move at a constant speed or at a constant pace through the assembly line so that part installation on the workpiece is completed when the workpiece reaches the end of the line. There is a precedence relationship among tasks: one task cannot be started before another activity is finished because of the physical structure of the product or technological constraints, e.g., the seats should be installed before the doors in a vehicle assembly process. Also, there are some assembly line specific constraints like tooling, zoning, worker skill, resources, and equipment [6, 82, 116]. Assembly line balancing (ALB) is a long-term production efficiency improvement strategy that assigns predetermined tasks to workstations while respecting the precedence relationship and problem-specific constraints.

The history of the assembly lines begins when Henry Ford introduced the famous Model-T. The first assembly lines are designed for mass production of a single standardized product in order to increase the cost/time efficiency. Global competitive markets forced manufacturers to offer customized products, e.g., electric, hybrid, and non-electric vehicles, to the customers. Customized feature configuration of a product is called a *model*. Each model of a product differs from others by shape, color, size, and feature which could result in a very high number of selections, e.g., the number of variations of a German car model exceeds 10^{24} [84]. Such industries with highly diversified

product portfolios include but are not limited to automobile, furniture, clothing, home appliances, and other consumer electronics.

Building separate assembly lines to produce each model is not cost-efficient, thus manufacturers have searched for ways to incorporate customization into an efficient production system. The development in manufacturing technology enabled utilizing *mixed-model assembly lines* (MMAL) in which multiple models are produced without reducing the efficiency of the flow-production system. Optimization of MMALs includes two main decision problems. The first problem is the ALB which is a long-term, strategic, decision since it requires movement of stations and training of workers. Balancing a line results in assigning tasks to different stations which means that executing tasks in a station requires a different skill set. While frequent line balancing could increase efficiency, the process is costly. Not only the line must be stopped at least for hours/days to move the stations but also the cost of station rearrangement is too much to repeat every week or month.

The second decision problem is the *mixed-model sequencing* (MMS), which is a short-term, operational, decision. MMS decides the order of a given set of vehicles for a planning horizon. This dissertation studies the MMS focusing on the automobile industry. Detailed background information on the MMS and car sequencing (CS) which is a more practical approach to MMS is given in the remainder of this chapter.

1.2 Sequencing Problem

The global competitive environment leads companies to consider how to produce high-quality products at lower costs. Sequencing the order of models efficiently inside a planning horizon is one of the short-term actions to reduce production costs. The two underlying objectives of the sequencing problem are to prevent line stoppage which is a result of work overload at a station and to balance part usage. Consequently, the two main objectives of the sequencing problem are minimizing work overload and leveling part usage [9].

MMALs are often designed such that average station work satisfies the time allocated to each station, cycle time, but some models with work-intensive options require more than the allocated time. Work overload occurs when the required jobs cannot be completed within station borders. A sequential order of two or more models that require a high amount of work at a station induces an inevitable work overload. Accordingly, a good allocation of models is necessary to avoid work

overload. Thus, the models that require a high amount of work are needed to be distributed along the sequence to avoid work overload. For example, electric vehicles must be distributed very carefully since battery installation requires some time that is drastically higher than the cycle time. If the electric vehicles are ordered sequentially, the work overload occurs at the battery installment station. Some of these consecutive ordered electric vehicles should be alternated with non-electric models in order to prevent work overload.

The origin of the second objective, leveling part usage, is the Toyota production system, namely the 'just in time' (JIT) production system [75]. This objective focuses on line inventory management. Different models have a variety of features, accordingly, the required parts for each model are different. Thus, sequencing models correspond to arranging the part usage. Consequently, the main idea of the second type of objective is to distribute the usage of each part evenly over the planning horizon.

This dissertation focuses on the assembly sequencing problem with the work overload objective. There are two mixed-model sequencing approaches that have the primary goal of minimizing work overload, namely MMS and CS. While chapter 2 focuses on a CS, chapters 3 and 4 focus on MMS.

1.2.1 Mixed-model Sequencing Problem

The cycle time of an MMAL is set at a minimum to be the average processing time across models at each station. The launch interval of vehicles on the conveyor is set equal to the cycle time, e.g., 10 seconds of launch interval means that a new car is launched on the assembly line every 10 seconds. The processing time of some models exceeds the cycle time while others have processing times below the cycle time. Back-to-back models with high processing time yield work overload. Such a case may induce one of the following scenarios [95]: line stoppage, employing a utility worker, or offline execution of the remaining task.

The most popular method to handle work overloads while the conveyor keeps moving is using *utility workers*, also called joker workers. Utility workers are skilled workers that can execute any task on the whole or a part of the assembly line. One can think of utility workers as team leaders, they are not operators responsible for specific tasks in a station but help operators when the work cannot be completed within the station limits. There are two popular utility worker management policies in the literature; *side-by-side* and *skip policies*. The side-by-side policy assumes that a

utility worker operates alongside the station worker to complete the tasks within the station border by shortening the total time of the tasks. Skip policy assumes that the utility worker takes over all the tasks on the vehicle which causes work overload, i.e., the utility worker takes over the job of the vehicle causes work overload while the station worker waits for the next vehicle. MMS with the side-by-side policy minimizes the total work overload duration which corresponds to the total duration of the excess job, wile MMS with the skip policy minimizes the number of work overload situations. In this dissertation, the side-to-side policy is adopted as a work overload handling procedure due to the results provided by Boysen *et al.* [19] which shows that side-by-side policy is superior when the number of work overload situations is low and average work overload duration is high. Our preliminary experiments show that our case study results fit this property.

We illustrate an MMAL with the side-by-side policy in Figure 1.1 which represents a station that processes five vehicles. The left and right vertical bold lines represent the left and right borders of the station. Assume that the cycle time c is 7 and the station length l_k is 10 TU, i.e., it takes 10 TU for the conveyor to flow through the station. This specific station processes two different configurations of vehicles: configurations A and B. While configuration A requires option 1, configuration B does not, so the processing times of configurations A and B are 9 and 5, respectively. Figure 1.1 illustrates the first five vehicles in the sequence which is [A, B, B, A, A]. The diagonal straight lines represent the position of the vehicle in the station. The worker always starts working on the first vehicle at position zero, left border of the station. The second vehicle is already at position $2 = 9 - c$ when the worker completes working on the first vehicle. Note that the next vehicle enters the station borders a cycle time after the current vehicle enters the station borders. The tasks of the second vehicle are completed when the third vehicle has just entered the station. The worker has 2 TU of idle time when the tasks of the third vehicle are completed. The worker starts working on the fifth vehicle on position 2 and the processing time of the fifth vehicle is 9 TU which causes a work overload of 1 TU, $2 + 9 - l_k = 1$. The job of processing these five vehicles could not be completed within the station borders but with the help of a utility worker, we assume that the job is completed at the station border at the cost of 1 TU work overload. The worker will continue working on the sixth vehicle at position $3 = l_k - c$, and this process continues.

The length of the workstations, processing times, worker movement speed, and conveyor speed are the parameters for the MMS which means that the MMS takes the details of the production system into consideration. We need to rest on some assumptions to be able to model the problem

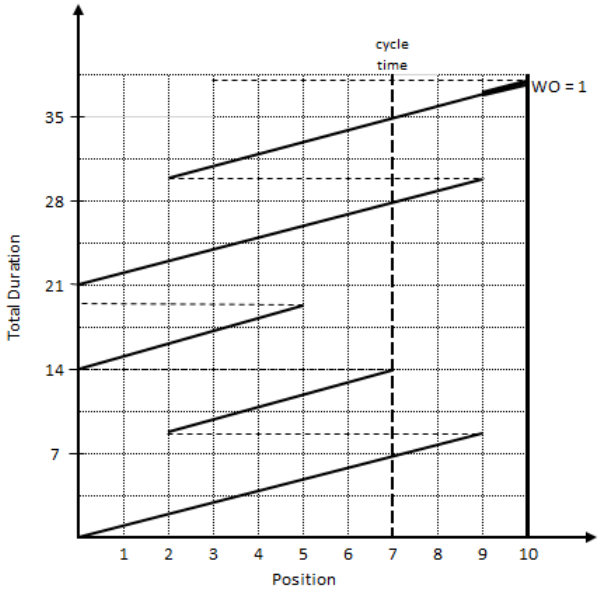


Figure 1.1: Illustration of mixed-model assembly line with five vehicles, cycle time $c = 7$, and station length $l_k = 10$. From bottom to top, the diagonal lines correspond to vehicle configurations A, B, B, A, A

mathematically. The following basic assumptions of the MMS problem as given by Bolat *et al.* [16] are also accepted in this dissertation.

- The conveyor moves from left to right at a constant speed which is one TU.
- The operators move with the conveyor while executing the specified tasks within the station's limits.
- The launching rate of products (cycle time) is a constant time (fixed rate launching).
- The workstations have closed borders. Operators cannot continue working on the workpiece outside of the station limits.
- The length of workstations is denoted by the total TU that the conveyor flows through the station from the left to right border.
- The operator movement duration is neglected since the movement of the operator is so much faster than the conveyor.
- Buffers between the stations are not allowed.
- Demand of each vehicle/model for the planning horizon (shift or day) is known before the horizon begins.

- The processing times are deterministic and vary based on the vehicle/model.
- Work overload is handled with the help of a utility worker, which means that the work overload has no effect on the next station and the operator starts working on the next vehicle at position station length minus the cycle time.
- The cycle of the planning horizons is respected. At each station, the job of the last vehicle of the current horizon should be completed before the first vehicle of the next horizon enters the station border, otherwise, work overload occurs. For example, assume that the fifth vehicle in Figure 1.1 is the last vehicle on the horizon, then four TU of work overloads would occur instead of one.

1.2.2 Car Sequencing Problem

CS is a specific version of MMS that focuses on the automobile industry. CS minimizes the work overload by controlling the sequence of the options (e.g., navigation, sunroof, electric battery). Each station is responsible for the installation of a different option. The vehicle models with an option require a high workload at the corresponding station, thus, the same models must be spaced in order not to exceed the capacity of the stations.

CS defines the *capacity rules* for each option in order to restrict the number of work-intensive options in a sub-sequence. The capacity rules control the sequence as follows: at most p_o of q_o models may contain the option o, typically written $p_o : q_o$. For example, assume that we have the $1 : 2$ capacity rule electric vehicle battery. Out of 2 consecutive models, at most 1 of them can be an electric vehicle. In Figure 1.2, there are two sequences with three vehicles; while the sequence in Figure 1.2a violates the electric vehicle capacity rule, the sequence in Figure 1.2b does not violate the rule.

(a) sequence with rule violation **(b)** sequence without rule violation

Figure 1.2: Illustration of CS capacity rules over 1:2 electric vehicle line capacity

The capacity rules could be defined by the decision makers by observing the assembly line, however, three capacity rule generation approaches are provided in the literature to detect and prevent the work overload situations [16, 45, 65]. The calculation of capacity rules is based on the operational characteristics of the assembly line.

In the first decades, CS is mostly formulated as a constraint satisfaction problem [81, 80, 58, 14] which has the assumption that there is at least one sequence that does not violate any capacity rules. Since real-world problems do not have such a solution and combinatorial optimization methods have become popular among researchers, most of the recent papers formulated CS as an optimization problem.

1.2.3 Mixed-model Sequencing vs. Car Sequencing Problems

MMS and CS have the same underlying goal of reducing line stoppages. Even though the goals of these two problems for the same system are not indistinguishable, their interpretations of the system are distinct. MMS directly minimizes work overload by focusing on tasks that require vast data like processing times, station length, conveyor speed, etc. On the other hand, CS indirectly minimizes the work overload by minimizing the number of capacity rule violations, so the only data requirement of CS is the capacity rules for each option. Accordingly, we can say that CS is a more practical version of the MMS since it requires less information collection about the assembly line.

CS is being used by car manufacturers because of its simple model representation which provides advantages in the application and maintenance [102], however, CS does not perform well on the work overload minimization, as expected. Golle *et al.* show that CS results in at least 15% more work overload compared to MMS [46]. Additionally, some car manufacturers are willing to include end-to-end supply chain optimization in the sequencing problem. In this perspective, the work of Louis *et al.* shows that the MMS model is practically more convenient for adapting supply chain optimization decisions [70].

To summarize, CS has the advantage of fewer data requirements but the vast data requirement of MMS provides a better assembly line representation which results in a more efficient and flexible model that could be adapted into real-world industrial environments such as stochastic processing times.

1.3 Gaps, Research Questions, and Contributions

In this section, we first discuss the gaps in CS and MMS literature. Then, we share the research questions on which this research is constructed. Next, we discuss the significant contributions to the related literature.

1.3.1 Gaps

The research on CS produced effective and efficient solution methodologies. State-of-art heuristics and exact solution methods have been provided during the last decades. The exact solution methods are still not capable of solving industry-sized problems. On the other hand, local search heuristics proved their ability to solve CS problems efficiently, especially with the ROADEF Challenge 2005 of which most of the finalists adopted local search heuristics. Besides, there is no work that presents a fair comparison of successful heuristics for CS. CSPLib and ROADEF Challenge 2005 datasets serve as benchmark instances and researchers present the computational results of their algorithms over these instances, however, the implementation of the algorithms has a drastic impact on the success of the results. Thus, there is a need for a fair comparison of algorithms in order to compromise on the benefits of the algorithms. This allows decision makers in the industry to select the right heuristic for their configuration. Additionally, local search heuristics in the CS literature requires fine-tuning of the transformation operators over the instances. This could be done with a huge amount of computational experiments since each industry instance has different characteristics. Executing a vast amount of experiments for the fine-tuning of the transformation operators is not feasible for manufacturers, thus there is a need to avoid this burden.

The research area of MMS focuses on solving more realistic problems in complex production environments, recently. These problems are being obtained by relaxing the assumptions given in Section 1.2.1 such as; swimming workers are allowed by relaxing closed station assumption [7], stochastic processing times are used [77, 21], workers with different skill sets are adapted [8], customer prioritization is included in the objective [88], parallel workers are considered [26], etc. A broad review of these works is given in section 3.2. Complex assembly line systems are researched but there are few existing works considering stochastic MMS models which all have uncertainty with the processing times. There is a gap in the robust MMS models considering car failures that occur frequently during the production process. The failed vehicles are pulled out of the sequence due to a

reason such as material shortage, paint, or body quality issue. Pulling out a failed vehicle may cause unexpected work overload if a robust schedule is not generated. Moreover, a failed vehicle must be reinserted back into the sequence when its issue is resolved. Currently, this process is executed via a dynamic tracking system that decides to reinsert a failed car when a suitable position is found. However, in case such positions do not occur enough, the vehicles that are waiting to be reinserted pile up, or the tracking system is forced to select positions that cause work overload. An MMS problem with the integrated reinsertion process of the failed vehicles back into the sequence is not studied in MMS literature.

1.3.2 Research Questions

The fundamentals of this research are constructed on the following research questions that are based on the previous work and the research gaps mentioned above;

1. What is the most efficient local search heuristic to solve CS? Can we avoid executing a vast amount of computational experiments in order to tune local search transformation operator weights? There is a significant gap between the lower and upper bounds of most of the CSPLib instances. Can we reduce this gap by finding better lower bounds by solving relaxed problems and better upper bounds by finding better solutions?

2. What is the impact of the failed vehicles over the assembly line? Can a robust schedule reduce possible work overloads due to failed vehicles and generate high-quality solutions? If yes, can we model MMS with stochastic failures? Is there an efficient approach to solving stochastic MMS?

3. Can we model the MMS with an integrated reinsertion process of the vehicles that fail under uncertainty? Is there an efficient approach to solving reinsertion integrated MMS with stochastic failures?

1.3.3 Contributions

Searching for the answer to our research questions leads us to develop this study which has some significant contributions to the literature.

In Chapter 2, we provide a comparison of efficient local search heuristics and the advantages and disadvantages of the heuristics are discussed. We propose an adaptive local search algorithm

that provides a significant advantage over other heuristics by not requiring tuning. The weight transformation operators are dynamically adapted at each iteration based on the success of the operators. Moreover, we present improved lower bounds for five out of 23 CSPLib instances by solving the relaxed problems.

In Chapter 3, we propose a two-stage stochastic program for the mixed-model sequencing (MMS) problem with stochastic product failures, and provide improvements to the second-stage problem. To tackle the exponential number of scenarios, we employ the sample average approximation approach and two solution methodologies. On one hand, we develop an L-shaped decomposition-based algorithm, where the computational experiments show its superiority over solving the deterministic equivalent formulation with an off-the-shelf solver. Moreover, we provide a tabu search algorithm in addition to a greedy heuristic to tackle case study instances inspired by our car manufacturer partner. Numerical experiments show that the proposed solution methodologies generate high-quality solutions by utilizing a sample of scenarios. Particularly, a robust sequence that is generated by considering car failures can decrease the expected work overload by more than 20% for both small- and large-sized instances. To the best of our knowledge, this is the first study that considers stochastic failures of products in MMS.

In Chapter 4, we propose a two-stage stochastic program and formulation improvements for a mixed-model sequencing problem with stochastic product failures and an integrated reinsertion process. Moreover, a bi-objective evolutionary optimization algorithm, a two-stage bi-objective local search algorithm, and a hybrid local search integrated evolutionary optimization algorithm are developed to tackle the proposed problem. Numerical experiments over a case study show that the hybrid algorithm is superior to others in terms of Pareto front exploration, while the local search algorithm provides more reliable results in terms of work overload objective. Finally, dynamic reinsertion simulations are executed over a case study and the results show that we can generate robust schedules that reduce the work overload by around 20% while decreasing the waiting time of the failed vehicles significantly.

1.4 Dissertation Organization

This dissertation is organized as follows. Chapter 1 introduces the research area by defining the assembly line and mixed-model assembly line. Next, mixed-model assembly line sequencing

and car sequencing problems are introduced. The system assumptions that have been made in this research are provided. The comparison of two sequencing models in terms of the work overload minimization success, applicability, and maintenance is provided. Chapter 1 is concluded with research gaps, questions, and contributions.

Chapter 2 presents a comprehensive study of some popular local search heuristics and their application on the car sequencing problem. First, the problem is introduced and the related work is covered. The performance of heuristics is discussed after corresponding heuristics and transformation operators are explained. Additionally, improved lower bounds to some of the CSPLib instances are provided.

Chapter 3 provides a mathematical formulation, and exact and heuristic solutions to MMS with stochastic failures. The motivation for the problem is presented. It is followed by related work and a two-stage stochastic mathematical model. Next, an L-shaped approach and a tabu search algorithm are presented to solve small and industry-sized problems, respectively. The chapter is concluded with computational experiments and a discussion about the efficiency of the solution approaches.

In Chapter 4, we introduce a novel variation of MMS that integrates the reinsertion of failed vehicles into the daily sequence generation process. First, the problem is formulated as a mixed-integer quadratically constrained program (MIQCP). Next, three heuristic approaches are proposed to tackle the proposed problem. The chapter is concluded with computational experiments and a discussion about the comparison of the solution approaches.

This dissertation concludes with highlights and future work suggestions.

Chapter 2

Adaptive Local Search Algorithm for Solving Car Sequencing Problem

The work presented in this chapter is documented in [116].

2.1 Introduction

The car sequencing problem (CSP) was defined and formulated by Parrello *et al.* as a version of the job-shop scheduling problem [81], specifically applied to automotive assembly lines. The CSP is a more practical approach to mixed-model sequencing problems (MMS) since it requires less information about the production system. Both CSP and MMS sequence/schedule vehicles in such a manner that "work overload" is minimized. Work overload occurs when a vehicle's tasks in a specific assembly line station cannot be completed before the vehicle leaves the station. MMS minimizes work overload by focusing on detailed scheduling of the tasks, which requires vast data like processing times, station length, conveyor speed, etc., whereas CSP minimizes work overload by influencing the sequence of vehicles with options that require a high workload. Capacity rules are defined to influence the sequence: at most p_o of q_o models may contain the high workload option o, typically represented as $p_o : q_o$. For example, a $2 : 5$ rule for electric models means that any

subsequence of length five with more than two electric cars induces a capacity rule violation. The conceptual objective of all CSP models is to minimize the total number of capacity rule violations. We explain four commonly used objective function computations for the sake of clarity as discussed in [35]; (1) The number of violated windows without side windows SW, (2) the number of violated windows with side windows SW^*, (3) the number of violations without side windows V, (4) the number of violations with side windows V^*.

The objectives are illustrated in table 2.1 which includes an example sequence with 12 vehicles subject to a single option with $2:5$ rule. The first row shows the position numbers and the second row shows the option configuration at each position, i.e., the vehicle assigned to position 1 has the option and the vehicle at position 4 does not have the option. Row 3 indicates the number of violations for each sliding window (hence the term "SW"), without considering *side windows* (the vehicles before and after the vehicles we are actively sequencing) attributed to the starting position of that window. That is, the contribution to the SW objective of the subsequence starting at position 1 and ending at 5 is 1: there is at least one violation of the capacity rule. In contrast, row 5 illustrates the V objective, which records how many violations are in the subsequence of length 5 starting at position 1; in this case, there are 2 violations. Because our sequence only has 12 cars, we cannot attribute any rule violations to subsequences starting at positions 9 and later in the SW and V objective computations. Rows 4 and 6 include *side windows*. Positions -2 to 0 represent cars that are already sequenced and cannot be changed; we need three previous positions and the first two that we sequence to compute the objective contribution for position -2. Without knowing what cars are in positions 13 and 14, we assume that they do not have the option. In all cases, we compute SW^* and V^* as we computed SW and V, we just include the impact of the side windows. The relationship among the objective values for any instance is: $SW \leq SW^* \leq V^*$ and $SW \leq V \leq V^*$. The reader may refer to Golle *et al.* [46] for an analysis of different CSP objectives and the relation between the underlying objective of CSP. This chapter adopts the SW objective due to its popularity.

This chapter is organized as follows. The related literature is presented in the next section. Then, the overview of the approach we are taking is given in Section 2.3. Local search metaheuristics and transformation operators are presented in Section 2.4. The setup of the computational experiments and the results are shared in section 2.5. Finally, the chapter is concluded with a conclusion and managerial insights in Section 2.6.

Table 2.1: Example sequence subject to 2:5 capacity rule and objective calculations

t	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Obj
2 : 5	0	0	0	1	1	1	0	1	0	0	0	1	0	0	1	0	0	
SW	-	-	-	1	1	0	0	0	0	0	0	-	-	-	-	-	-	2
SW^*	0	1	1	1	1	0	0	0	0	0	0	0	0	-	-	-	-	4
V	-	-	-	2	1	0	0	0	0	0	0	-	-	-	-	-	-	3
V^*	0	1	1	2	1	0	0	0	0	0	0	0	0	-	-	-	-	5

2.2 Related Work

Parrello *et al.* introduced the CSP as a version of the job-shop scheduling problem in 1986 [81]. In the first decades of literature, the CSP is formulated as a constraint satisfaction problem [81, 80, 58, 14]. However, recent studies formulate the CSP as an optimization problem. Therefore, the review of the CSP focuses on the optimization version of the problem.

2.2.1 Exact Solution Approaches

Most of the exact solution methods reported in the literature utilize branch and bound (B&B) algorithms. Drexl *et al.* propose a B&B algorithm to solve the constraint programming version of the CSP [34]. The branching mechanism depends on defining different states of the sub-problems. The definition of a state provides information about the options, models, and demands so that the definitions can be used to prune the node. For each level of the tree, they mainly keep track of possible options and models for each position so that the size of the tree is decreased. The main elements to trace the possible options for each position are the earliest due date and the latest due date of each occurrence of options that are calculated by using the capacity constraints. The algorithm has been tested using the non-trivial instances that are provided in [33]. Computational results show that the optimal solution is found for most of the instances and a feasible solution is found in a very short time even for larger instances.

Fliedner and Boysen present the first branch and bound algorithm that uses a similar branching method given in [34] to solve the optimization version of the CSP [38]. They define several dominance rules that thoroughly exploit the problem structure in order to fathom the nodes as early as possible. They also use a depth-first search since it is critical to search promising nodes first. Thus, the utilization rate, defined in [101], is integrated into the node selection process by selecting the node with the lowest total utilization rate. They show that the proposed algorithm outperforms

an integer program coded in CPLEX in terms of duration and upper bound for the instances that are solved optimally and terminated with a time limit, respectively.

Hybrid heuristics combined with an exact solution method have been used to solve the CSP. Zinflou *et al.* propose a hybrid genetic algorithm [124]. They integrate integer linear programming (ILP) to solve a subsequence during the crossover process. Thiruvady *et al.* use a Lagrangian relaxation (LR) of the CSP by relaxing the set of constraints that ensure each position is assigned by only one model [107]. Thiruvady *et al.* employ a large neighborhood search algorithm [108], which consists of a MIP and a Lagrangian ant colony system (LRACO) which is obtained by relaxing a set of constraints in the original problem [107]. Comparing the quality of solutions and run times with the Lagrangian-ACO algorithm given in [107] shows that the Lagrangian-ACO requires much more run time, yet the proposed method's solution quality is comparable with the LRACO.

2.2.2 Heuristic and Metaheuristic Approaches

The CSP is an NP-hard problem [43]. Although some exact solution methods are presented, the majority of the solution procedures for the CSP are based on heuristics and meta-heuristics. The quality of the initial solution has a significant impact on the efficiency of heuristic algorithms. Gottlieb *et al.* present several greedy heuristic methods to create an initial solution and integrate these methods into a local search algorithm and an ant colony optimization [48]. The proposed greedy heuristics require one car placement at each iteration, where the car selection is mainly based on the idea of utilization rate as defined in [101]. The results show the dynamic utilization rate calculated versions of the greedy heuristics perform better compared to their static variants. Furthermore, the best results are obtained from the ant colony approach.

Warick and Tsang first applied a genetic algorithm to solve the CSP and they described a lower bound formula to measure the quality of the sequences created by the genetic algorithm [112]. Zinflou *et al.* propose three new crossover operators specifically for the CSP [123]. These operators improve the efficiency of the algorithms by generating better solutions and increasing the number of feasible solutions. Sun and Fan propose a multi-objective ant colony optimization method to solve the CSP with changeover complexity, which represents the assembly complexity of options with multiple features, e.g., the occurrence of a manual sunroof or automatic sunroof in addition to the absence of a sunroof [105]. Zhang *et al.* formulate the CSP with an objective function that reflects the economic meaning of the objective, minimizing the number of utility workers required [120]. They

propose a parallel construction heuristic based on constraint propagation to solve the problem. They define three filtering rules by exploiting the problem structure, so the propagation is done after every car assignment based on the filtering rules. Finally, they adopt a parallel construction heuristic to search the solution space by allowing each model in the first position of the sequence. They run experiments on the CSPLib instances [44] by comparing the proposed method with other methods that yield good-quality solutions. They find the best solutions for 85 instances out of 109 in a very short time compared to other algorithms. Siala *et al.* form a heuristic classification structure that describes how heuristics can be designed using different strategies [97] and show the impact of each classification criterion such as branching.

Golle *et al.* present an iterative beam search (IBS) algorithm to solve the optimization version of the CSP [47]. Their main contribution is new lower bounds for the CSP so that they can prune the graph effectively. They compare the results of the proposed method with the best-known exact solution algorithm given in [38] by running both algorithms on the same computer for the same duration. IBS outperforms [38] in terms of the number of violations, the number of searched nodes, and the running time.

The French Operational Research Society (ROADEF) organizes the ROADEF challenge bi-yearly since 1999; the 2005 problem was a CSP, proposed by car manufacturer Renault. In addition to the capacity constraints of the standard CSP, paint batching constraints are introduced to minimize the solvent consumption for the painting process. State-of-the-art methods proposed to solve the challenge problem are reviewed by Solnon *et al.* [102]. Solution approaches include exact solution methods (constraint programming, integer programming), and heuristic approaches (greedy heuristics, local search heuristics, and metaheuristics). The following papers propose successful algorithms to the challenge:

- Estellon *et al.* propose a local search heuristic, which is the winning algorithm of the challenge [36]. Once an initial sequence is created by a greedy heuristic, one of five transformation methods is applied at each iteration: swap, forward insertion, backward insertion, reflection, and random shuffle. The success of this algorithm lies in choosing the proper position of transformations and fast evaluation of the new sequences (or subsequences).

- The second-ranked algorithm of the challenge is presented by Riberio *et al.* which is a hybrid heuristic that is based on the framework of iterated local search (ILS) and variable neighbor-

hood search (VNS) [92]. The intensification and diversification phases are integrated into the metaheuristics to find local optima and jump to different solution regions. They define six neighborhoods to improve the heuristics. Since the number of evaluations is much more than the solution updates, they store the current sequence in three different matrices to reduce the evaluation time. Another critical design feature of their algorithm is *ad hoc* data structures that facilitated the evaluation of each exchange in linear time.

- Prandtstetter and Raidl propose a new ILP formulation that assigns individual components to the positions, not the models [86]. They also present a VNS-based heuristic to solve larger instances. A variable neighborhood descent (VND) is employed to find the local optimum of a small neighborhood, and VND is embedded into VNS to find the local optimum of a larger neighborhood. Their main contribution is finding tight bounds and proving optimality for some instances.

Even after the challenge winner was declared, researchers have continued to improve the lower bounds and the best-found solutions for the instances. Gagne and Zinflou applied a hybrid multi-objective evolutionary algorithm to the challenge problem [42] combining a genetic algorithm and the artificial immune system (GISMOO) given in [125]. The results show that GISMOO outperforms the winner algorithm of the challenge [36] on most of the instances provided by the challenge. Jahren and Acha provide considerably better lower bounds by improving the MILP formulation and solving with a column generation approach [57].

Noting that local search is adopted by most of the heuristics/metaheuristics approaches, the following papers, which inspired us to prepare this chapter, propose the most successful local search heuristics for CSP. Puchta and Gottlieb propose three versions of a local search algorithm that applies one of six different move operators with an equal chance at each iteration [87], The first two algorithms are threshold-accepting (TA) with the first method having a geometric cooling rate and the second method having a bouncing cooling strategy. The third method is a greedy local search which accepts all solutions with an improved or unchanged objective. They compare the three approaches using six realistic benchmarks from the daily production of a car factory. The results show that the TA methods provide better solutions than the greedy local search. Estellon *et al.* propose a very fast local search (VFLS) algorithm to solve the CSP [35]. The VFLS is a greedy local search algorithm that applies a transformation operator (swap, forward insertion, backward insertion, and

inversion) at each iteration and accepts any non-deteriorated solution. They find better solutions for 3 out of 30 newer CSPLib instances by using VFLS. Hottenrott *et al.* propose an adaptive large neighborhood search (ALNS) to solve CSP with the consideration of instability of sequences in case of a failure, quality problem, or supply shortage [55]. When a failure occurs, the corresponding car is taken out of the sequence and the assembly continues with the next car. Therefore, they generate a robust model that incorporates the failure chance of each model. Accordingly, they minimize the sum of expected violations by considering a sample from all scenarios.

2.3 Experimental Approach

State-of-art IP solvers now incorporate several techniques such as B&B, presolve reduction, cutting planes, and heuristics to increase efficiency. However, exact solution methods are not sufficient for relatively large-sized sequencing problems, and most of the real-world sequencing problems are too large for this technique. Even though it provides comparable results for small and medium-sized CSPLib instances and some of the challenge problems, it cannot find good enough solutions for all instances. Consequently, there is a need for a heuristic algorithm to get satisfactory results in a reasonable amount of time. In the next section, we provide four local search approaches as a reliable method. Based on our review of the literature, we frame this chapter to explore how updated heuristics and metaheuristics perform on large CSP instances from the literature. We present these algorithms in section 2.4. The data instances are described in section 2.5

In order to evaluate the performance of the proposed algorithms, we must have either optimal solutions to the instances or strong lower bounds. We utilize the IP first proposed by Gottlieb *et al.* [48] and adopted by many researchers [38, 47] without adaptation in an attempt to find optimal solutions to the instances; please refer to [48] for details. As described in section 2.5, we used Gurobi solver to solve CSPLib and ROADEF challenge instances, but some instances were not solved to optimality. Therefore, we supplement the analysis by finding lower bounds.

A preprocessing method to find lower bounds for CSP is presented by Mayer and Walsh [72]. They provide lower bounds for 17 out of 30 CSPlib instances; for 11 of these instances, these lower bounds are used to prove that the best-known solutions are optimal. We improve this technique for finding lower bounds for some of the CSPLib instances by relaxing the problem. We relax the problem by including only a subset of options in the model. All the corresponding CSPLib instances

have five options, so the subset of options in sizes of two, three, and four are considered separately as relaxed problems. Accordingly, we end up with $\binom{5}{2} + \binom{5}{3} + \binom{5}{4} = 25$ sub-instances for each instance. Then, each relaxed problem's IP is solved using Gurobi. Note that any lower bound determined by Gurobi for the relaxed problem is also a valid lower bound for the original problem.

2.4 Local Search Algorithms

The strength of local search algorithms depends on several features: the ability to escape from local optima, fast evaluation, and exploration ability. As mentioned in section 2.2, the most successful methods to tackle CSP are pure local search algorithms or hybrid algorithms based on local search. Accordingly, we compare several local search algorithms over CSP benchmark instances.

A design consideration for local search algorithms is avoiding premature convergence by escaping locally optimal solutions. The neighborhood structure has a crucial role in escaping local optima. The proposed local search algorithm adopts nine transformation operators with varied neighborhoods so that the combination of these operators results in a more robust neighborhood structure. Additionally, local search algorithms are generally designed so that only improved solutions after a move are accepted, or non-improved solutions are accepted according to some conditions. However, accepting new solutions with the same objective value unconditionally for the CSP is crucial in order to escape local optima [87, 48, 35, 36]. This small trick lets us explore a broader neighborhood and avoid local optima since there are several solutions that have the same objective value within different neighborhoods.

The calculation of the objective value each time an operator is applied is the most time-consuming step of the algorithm. The complexity of objective calculation from scratch is related to the number of vehicles to be produced, the number of options, and the q ratio that the options have. Computing the objective has a time complexity of $O((T - Q) * O)$ where Q refers to the average of q ratios. However, the *locality* of the operators for the neighborhood of permutation-based representations is strong, i.e., small changes in the solution result in small changes in the objective. Thus, it is unnecessary to calculate the objective value from scratch every time an operator is applied. To emphasize the improvement of this fast evaluation method, consider an instance with 200 cars, five options, and an average q ratio value of 3. To calculate the number of violations using the sliding window objective from scratch, we need to check $(200 - 2) * 3 * 5 = 2970$ windows. On

		1	2	3	4	5	6	7	8	9	10	11	12	13		1	2	3	4	5	6	7	8	9	10	11	12	13
	Ratio	A	A	B	C	D	B	D	A	F	E	F	G	F	⟹	A	A	B	E	D	B	D	A	F	C	F	G	F
Option 1	1:2	1	1	0	1	1	0	1	1	0	0	0	1	0		1	1	0	0	1	0	1	1	0	1	0	1	0
Option 2	2:3	0	0	1	0	1	1	1	0	1	0	1	1	1		0	0	1	0	1	1	1	0	1	0	1	1	1
Option 3	1:3	0	0	0	1	0	0	0	0	1	1	1	1	1		0	0	0	1	0	0	0	0	1	1	1	1	1

Figure 2.1: Illustration of swap move and reevaluation

the other hand, we need to check only $2 * 2 * 3 * 5 = 60$ perturbed windows to find the new objective value when an operator is applied; this reduced computation has a time complexity of $O(Q * O)$. Note that the objective value of the initial solution is calculated from scratch and the reevaluation cost of operators K swap, K insertion, and random shuffle is higher than swap, insertion, 3opt, and inversion operators.

A swap move and the reevaluation process is illustrated in figure 2.1. First, two models are selected, model C at position 4 and model E at position 10, and swapped. Then, the windows that are impacted by the swap move are reevaluated for each option. Note that there is no need to evaluate changes related to options 2 and 3 since the selected models vary only in option 1. So, we only evaluate windows [3,4], [4,5], [9,10], and [10,11] for the new sequence since the violated windows for the incumbent solution are known. It can be seen that the number of violations for option 1 for these windows decreases from 1 to 0. Consequently, the number of violations for the new sequence decreases from 9 to 8.

Building the initial solution with a greedy heuristic instead of randomly provides a great advantage. Gottlieb *et al.* proposed five greedy heuristics from which the dynamic sum of utilization rates (DSU) heuristic produces the best solutions, especially for instances with higher utilization rates. Accordingly, we employed DSU greedy heuristic to generate an initial solution for all local search algorithms presented in the next section.

The remainder of this section first explains the compared local search algorithms and then the transformation operators used to define the neighborhoods for those local search algorithms are presented.

2.4.1 Adaptive Local Search Algorithm

We propose an adaptive local search (ALS) method that allows the weight of the operators to vary during the execution of the algorithm, depending on the operators' success. First, an initial solution is generated. Next, a transformation operator defined in section 2.4.5 is applied at

each iteration for τ seconds and the new solution s' is accepted when the objective value is not deteriorated ($v' \leq v$). A selected operator is not applied just once at each iteration since the CPU time of operators is not the same, e.g., the swap operator is faster than other operators. Thus, we allocate the same amount of time for the operators to create a more fair environment so that the adaptive selection probabilities reflect the success of operators more accurately.

Additionally, we run the iterations in batches with the size of B, i.e., a batch consists of a pre-defined number of iterations. The adaptive operator selection probabilities are updated at the end of each batch. The reason behind not updating after each iteration is that updating the adaptive parameters process is costly, so the time consumption between updating and finding new solutions is needed to be balanced. See algorithm 1.

Algorithm 1 Adaptive Local Search

Step 1) Initialization
  Set initial operator weights
  Generate an initial sequence s
  Calculate objective function value v of s
Step 2) Update
  while Stopping criteria is not met **do**
    for iteration in B **do**
      Select a transformation operator l randomly according to the adaptive probabilities
      while duration $< \tau$ **do**
        Apply the operator l to find a neighbor s'
        Calculate objective function value v' of s'
        if $v' \leq v$ **then**
          Update current sequence and violation: $s \leftarrow s'$, $v \leftarrow v'$
        end if
      end while
      Update success rate π_l of operator l
    end for
    Update weights ω_l, $l \in L$
    Update adaptive selection probabilities Θ_l, $l \in L$
  end while
Output: s, v

The advantage of the adaptive parameter control mechanism lies in controlling the operator's weights for each specific instance and each step of the algorithm. Test instances have different characteristics, and our experiments show that the operators have different success rates regarding the instances. We may have information about operators' success rates by vast experimentation and determine the parameters depending on the results. However, there is no guarantee that we will get the same results from another set of instances. Also, it is a burden to run experiments and decide

on the probabilities every time we face a new type of instance.

2.4.1.1 Adaptive Control Mechanism

While deciding which operator to apply to the incumbent solution, we adopt a roulette wheel selection mechanism in which each operator is assigned a weight. The weights of the operators are the same at the beginning of the algorithm $\omega_l = 1, \forall l \in L$. All weights are updated at the end of each batch regarding the success rate π_l of the operators during the previous iterations. The adaptive selection probability Θ_l of operator j is calculated (see equation 2.1) according to the current weight ω_l of the operator. The updates of selection probabilities and weights are executed similar to the adaptive large neighborhood search (ALNS) given in [55]. The minimum selection probability $\Theta^{min} = 2\%$ prevents operators from being eliminated during the execution of the algorithm; each operator has at least Θ^{min} chance to be selected at each iteration. The operator random shuffle always has Θ^{min} of probability because of its contribution to diversification and relatively low computation cost.

$$\Theta_l = \Theta^{min} + \frac{\omega_l}{\sum_{l \in L} \omega_l} + (1 - |J|\Theta^{min}) \tag{2.1}$$

The weights are updated as in equation 2.2. The *reaction factor* $r = 0.1$ is employed to control the reaction of the weight adjustments to changes in the success rate of the operators during the last batch of iterations [93]. The higher (lower) the reaction factor, the higher (lower) the impact of the last batch over the weight adjustment. In order to favor the more frequently called operator, we divide the success rate of operator l from the last batch by η where η is the number of times the operator l is applied.

$$\omega_l = (1 - r)\omega_l + r\frac{\pi_l}{\eta} \tag{2.2}$$

Success rates are calculated as in equation 2.3. First, the success rates are set to zero at the beginning of each batch. Next, the success rate of operator l is updated every time operator l is called depending on whether the incumbent solution is improved or a different solution with the same objective value is found or otherwise. It is always easier to improve the solution at the beginning of the algorithm since there are more violations to fix. Thus, we add the improvement rate of the objective function value to the success rate when a better solution is found. Similarly,

we reward finding a different solution with the same objective value proportional to the objective value.

$$\pi_l = \pi_l + \begin{cases} \frac{v-v'}{v}, & \text{if } v' < v; \\ \frac{1}{8v}, & \text{if } v' = v; \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

2.4.2 Simulated Annealing

Simulated annealing (SA) is one of the most used single solution-based metaheuristics. SA is inspired by the annealing process of metals. SA has been employed to solve combinatorial optimization problems due to its ability to escape local optima by accepting non-improving solutions. The proposed SA in this section, inspired by the threshold accepting algorithm in [87], is a SA with a fixed threshold of 1 since we always accept such a solution with the same objective value. The search starts with an initial solution. Next, an operator is selected depending on the weights which are extracted from the average adaptive probabilities of ALS, given in table 2.4. The selected operator is applied to generate a random neighbor at each iteration. Any neighbor with a non-deteriorated objective value is accepted. Otherwise, the acceptance of the neighbor depends on the amount of deterioration Δf of the objective value and the current temperature T. The acceptance probability is calculated using the Boltzmann distribution:

$$P(\Delta f, T) = e^{-\frac{f(s')-f(s)}{T}} \tag{2.4}$$

We utilize geometric cooling with bouncing strategy. Cooling occurs until T_{min} is reached. The temperature is set to a jump temperature T_{jump} if the current solution is not improved for a certain amount of time t_{jump}. The number of jumps is limited to the parameter n_{jump}. See algorithm 2.

2.4.3 Variable Neighborhood Search

Variable neighborhood search (VNS) systematically searches over a set of predetermined neighborhoods. The VNS consists of shaking, improvement, and neighborhood change steps.

1. **Improvement procedure:** We adopted an improvement procedure as a local search similar to the Variable neighborhood descent (VND) search which is based on the fact that a local

Algorithm 2 Simulated Annealing with Bouncing Strategy

Step 1) Initialization
Set operator selection probabilities
Set parameters T, T_{min}, α, T_{jump}, t_{jump}, n_{jump}
Generate an initial sequence s
Calculate objective function value v of s
Step 2) Update
while Stopping criteria is not met **do**
Apply a transformation operator $l \in L$ to generate a new sequence s'
Calculate objective function value v' of s'
if $v' \leq v$ **then**
Update current sequence and objective value: $s \leftarrow s'$, $v \leftarrow v'$
else
Accept s' with a probability $e^{-\frac{v-v'}{T}}$
end if
if The current solution is not improved for t_{jump} seconds **then**
$T \leftarrow T_{jump}$
else
$T \leftarrow \max(\alpha T, T_{min})$
end if
end while
Output: s, v

optimum point for several neighborhoods is more likely to be a global optimum point than a local optimum point for a single neighborhood [52]. However, instead of searching for all solutions of each neighborhood, the adopted procedure navigates among plateaus by accepting a non-deteriorated solution immediately.

2. **Neighborhood change procedure:** The order of the neighborhoods to be explored is crucial, so the search process needs to be guided. In this chapter, we employed a sequential search with the first improvement strategy. More precisely, the search starts at the first neighborhood based on the defined order and continues with the next neighborhood at the end of the allotted time. If the incumbent solution is improved, the search continues with the first neighborhood, otherwise, the search continues with the next until all neighborhoods are searched. The pseudo-code of this procedure is given in algorithm 3.

3. **Shaking procedure:** The shaking procedure is used to escape from local optimum points. We select a random neighbor from a randomly selected neighborhood regardless of the objective value of the neighbor. Shaking is applied each time a local optimum solution is reached within the improvement procedure.

Algorithm 3 Improvement Procedure of VNS

Step 1) Initialization
Input: A solution s and the objective function value v
Step 2) Update
Set $l \leftarrow 1$
while $l \leq l_{max}$ **do**
$v* \leftarrow v$
while duration $< \tau$ **do**
Apply the operator l to find a neighbor s'
Calculate objective function value v' of s'
if $v' < v$ **then**
$s \leftarrow s'$, $v \leftarrow v'$
end if
end while
if $v < v^*$ **then**
$l \leftarrow 1$
else
$l \leftarrow l + 1$
end if
end while
Output: s, v

The most common improvement procedure for VNS is variable neighborhood search (VND). A significant strategy of successful local search algorithms for CSP is the acceptance of solutions that have the same or improved objective values, which helps explore different fitness landscapes by using plateaus. Therefore, we accept solutions with the same or improved objective values to escape local optima. VNS repetitively calls improvement and shaking procedures until the stopping criteria is met; see algorithm 4. In the end, the best-found solution during the search is reported.

Algorithm 4 Variable Neighborhood Search

Step 1) Initialization
Set order of neighborhoods N_o where $o = 1, .., |N|$
Generate an initial sequence s
Calculate objective function value v of the initial sequence s
Step 2) Improvement
Set $s_{best} \leftarrow s$, $v_{best} \leftarrow v$
while Stopping criteria is not met **do**
$s, v \leftarrow improvement\_procedure(s, v)$
if $v < v_{best}$ **then**
$v_{best} \leftarrow v$, $s_{best} \leftarrow s$
end if
$s, v \leftarrow shaking(s, v, random(l))$
end while
Output: s_{best}, v_{best}

Table 2.2: The average number of attempts of operators per second

Swap	Insertion	Inversion	3opt	K Insertion	K swap	Random
4725	1856	1405	1631	1090	890	1032

2.4.4 Very Fast Local Search

Very fast local search (VFLS) algorithm is proposed by Estellon *et al.* [35]. The VFLS is a hill-climbing algorithm that applies a transformation operator $l \in L'$ at each iteration and accepts the new solution as long as the objective function value is not worse (see Algorithm 5). The algorithm's success depends on the state-of-art application of operators, which generates millions of moves in a minute, which induces the algorithm's convergence to be very fast. Accordingly, they provided 3 out of 30 best-known solutions of latter CSPLib instances.

Algorithm 5 Very Fast Local Search

Step 1) Initialization
  Set operator selection probabilities
  Generate an initial sequence s
  Calculate objective function value v of s
Step 2) Update
  while Stopping criteria is not met **do**
    Apply a transformation operator l where $l \in L'$ to generate a new sequence s'
    Calculate objective function value v' of s'
    if $v' \leq v$ **then**
      Update current sequence and objective value: $s \leftarrow s'$, $v \leftarrow v'$
    end if
  end while
Output: s, v

We include VFLS in our comparisons since it has provided the best-known solutions in the shortest amount of time for the CSPLib and ROADEF instances. However, the algorithm's speed depends on the programming skills and programming language in which the algorithm is coded. The average number of attempts of each operator for CSPLib instances is given in table 2.2 so that the reader can make the comparison. In VFLS, we only implement a subset of operators L' (swap, backward insertion, forward insertion, and inversion) in order to implement VFLS similar to the original algorithm as given in [35].

2.4.5 Transformation Operators

Several operators in the literature explore a neighborhood of a given solution. We employ the following operators: swap, backward and forward insertions, inversion (2-opt), 3-opt, backward and forward K insertions, K swap, and random shuffle. The transformations are illustrated in figure 2.2. The swap operator interchanges the position of randomly selected two cars. Insertion removes a car from position i and inserts it at position j. Insertion is applied in two different directions; backward and forward. In case of $i > j$, the insertion is called a backward insertion and all the vehicles between the positions j and i move one right (later). As the opposite, forward insertion occurs when $i < j$ and all the vehicles between the positions i and j move one left (earlier). In CSP literature, researchers adopt the linear version of the 2-opt, also called lin2opt or inversion, in which 2 points are randomly selected in the sequence and the subsequence between the selected points is reversed. We adopt the linear version of 3-opt in which two points, i and j, are randomly selected where $i > 2$ and $j < T-1$. Instead of inverting the subsequence between the two selected points, the subsequences $[1, i]$ and $[j, T]$, are inverted. K insertion is similar to insertion but a subsequence of K cars is inserted. Similarly, the K swap interchanges the positions of two subsequences of the same length. Finally, random shuffle is used to decrease the chance of being stuck at local optima. Two random points are selected similar to the 2-opt, yet the subsequence between the selected points is shuffled randomly.

2.5 Computational Experiments

2.5.1 Experimental Setup

We use one data set from CSPLib and two data sets from ROADEF challenge 2005 to run our experiments [100, 102]. We employ only the latest data set from CSPLib since the prior instances are relatively smaller and the optimality of all best-known solutions is proved. As given in table 2.3, the first data set which is provided by Gravel *et al.* [49] has 30 relatively larger instances with 200 to 400 vehicles, 5 options, and from 19 to 26 models. Only seven out of 30 instances are solved to zero violations, and the best-known solutions are not proven to be optimal as of our knowledge.

The second and third data sets are from the ROADEF challenge. We simplified challenge instances in three ways to extract simpler CSP instances: (1) by eliminating paint shop constraints,

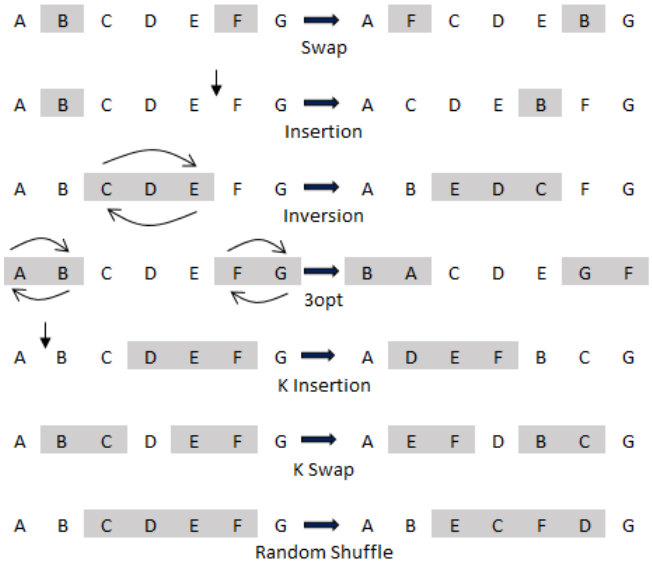


Figure 2.2: Illustration of local search operators

i.e., only the assembly line capacity restrictions are considered, (2) by removing option priorities, and (3) by removing the previous day's vehicles since the objective in this work does not consider side windows. The simplified ROADEF instances, which are similar to the CSPLib instances, let us explore different aspects of the algorithms with various production cycles, options, and models. The given numbers in the table 2.3 are from the simplified instances. 50 trials are done for each instance and each algorithm using the computing nodes of the Clemson University supercomputer with 8 cores and 30 GB of memory.

Table 2.3: Data sets

	Set	Production Cycles	Options	Models	# of Problems in data set
CSPLib	*Gravel*	200-400	5	19-26	30
ROADEF	A	335-1315	5-22	12-190	8
Challenge	X	65-1319	1-26	2-194	19

The LS algorithms are implemented in Python 3 and their performance is compared both among themselves and with the best-known solutions of the benchmark instances. In literature, the stopping criteria is commonly set to a time limit of 600 seconds since the CSP is a short-term decision problem and needs to be solved for each planning horizon. However, we are able to attempt a few hundred thousand transformations while Estellon *et al.* show that they can attempt several

millions of transformations per minute [35]. Accordingly, the time limit is set to 3600 seconds for our experiments to investigate the compared algorithms' strengths. Additionally, the convergence of the algorithms within 7200 seconds of time limit is illustrated in Appendix A in order to justify our 3600 seconds time limit.

The weight of each operator is the same at the beginning of the ALS algorithm. The weights for the operators in the other algorithms need to be set according to the success of the operators. We use the average of resulting adaptive operator selection probabilities from ALS as input to the other algorithms. The average resulting adaptive probabilities are calculated for CSPLib and ROADEF challenge instances separately (see table 2.4). This provides an advantage to the algorithms (except ALS) by providing tuning on the weights of operators. Note that the sum of the average of probabilities is 98% since the probability of random shuffle is fixed to 2% for all algorithms (except VFLS). Other parameters for each algorithm are as follows:

Table 2.4: Operator weights provided by ALS

	CSPLib Instances			ROADEF Challenge Instances		
	Min	Average	Max	Min	Average	Max
Swap	0.12	0.26	0.38	0.12	0.48	0.84
Insertion Forward	0.02	0.06	0.14	0.02	0.06	0.13
Insertion Backward	0.02	0.06	0.14	0.02	0.06	0.22
Inversion	0.19	0.33	0.54	0.02	0.14	0.28
3opt	0.02	0.02	0.05	0.02	0.05	0.12
K Insertion Forward	0.02	0.06	0.11	0.02	0.05	0.13
K Insertion Backward	0.02	0.06	0.11	0.02	0.05	0.12
K swap	0.07	0.14	0.24	0.02	0.07	0.12

- **ALS**: The parameters batch size B and the duration that a operator run each time called τ are set to $B = 100$ and $\tau = 0.2$. So, the adaptive probabilities are updated every 20 seconds. These two parameters are set according to our experiments but the reader should consider the stopping criteria and the CPU time balance between transformations and the adaptive probability updates.

- **SA with bouncing strategy**: The jump temperature is $T_{jump} = 1$. The jump occurs when the solution is not improved for $t_{min} = 60$ seconds, and the number of jumps is limited to $n_{jump} = 5$.

- **VNS**: The order of the neighborhoods is determined according to the operator selection prob-

abilities provided by ALS as in table 2.4. For example, the order of operators for the CSPLib instances is: inversion, swap, K insertion, K swap, insertion, 3opt, and random shuffle.

- **VFLS**: The operator selection probabilities are normalized for the VFLS since only a subset of operators are employed $\Theta_l = \frac{\Theta_l}{\sum \Theta_l}$ where $l \in L'$.

2.5.2 Results

We present the experimental results in this section. Tables 2.5 and 2.6 report the results for the CSPLib instances and ROADEF challenge instances, respectively. Both tables have the same setup, of which the columns 'T', 'O', and 'M' stand for the number of cars, number of options, and number of models, respectively. The column 'Obj*' denotes the best found or known solution; while the table 2.5 shows the best-known solutions from the literature, table 2.6 shows the best-found solutions by our experiments. Additionally, the best objective values proved to be optimal by IP or in literature are stated using the asterisk (*). The minimum, mean, and maximum objective values found by the local search algorithms for 50 trials are included. Also, the number of best-found solutions of minimum and maximum objective values by each algorithm is given. The DSU greedy heuristic is adapted by all algorithms so that the DSU results demonstrate the summary of 250 trials. The best-known solutions found by the algorithms are bold-faced.

We use Gurobi commercial solver to solve the IP model of each instance with a time limit of 3600 seconds. The IP solution provides comparable solutions for most of the CSPLib instances; see table 2.5. The best-known solutions are found for 24 out of 30 CSPLib instances. A solution with a small gap to the best-known solution is not found only for instances pb_200_03, pb_300_05, pb_400_02. However, this method fails to find good solutions as the size of the problem increases; the IP cannot find a satisfactory solution for six out of 27 instances in table 2.6. This illustrates that the IP model is not reliable for real-life industry-size problems, even if the problems are simplified to the standard CSP.

On the other hand, we obtain improved lower bounds by solving CSPLib instances with the subset of options as described in section 2.3. The best-known lower bounds in literature [72] and best-found lower bounds by us are given under the columns LB* and LB', respectively in table 2.5. We have found better bounds than the best-known lower bounds for five out of 23 instances.

In table 2.5, the results of ALS and VFLS are comparable and they outperform the other

Table 2.5: Results for CSPLib Instances

							ALS			VNS			VFLS			SA			DSU			Lower Bounds		
	Instance Name	T	O	M	Obj*	IP	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	LB*	LB'	Options
1	pb_200_01	200	5	25	0	4	**0**	0.10	2	**0**	1.42	4	**0**	0.06	1	**0**	0.2	1	9	15.68	23	-	-	-
2	pb_200_02	200	5	25	2	**2**	**2**	2.00	**2**	**2**	3.16	5	**2**	2	**2**	**2**	2	**2**	10	13.66	18	2	-	-
3	pb_200_03	200	5	25	3	6	4	5.17	7	5	7.86	13	**3**	4.7	6	4	5.78	8	20	24.87	32	-	**1**	2, 3
4	pb_200_04	200	5	24	7	**7**	**7**	7.00	**7**	**7**	8.76	12	**7**	7	**7**	**7**	7.04	8	14	17.39	22	7	5	0, 1
5	pb_200_05	200	5	23	6	**6**	**6**	6.00	**6**	**6**	6.62	8	**6**	6	**6**	**6**	6	**6**	10	13.32	17	-	**5**	0, 1
6	pb_200_06	200	5	23	6	**6**	**6**	6.00	**6**	**6**	6.08	7	**6**	6	**6**	**6**	6	**6**	14	17.34	22	6	-	-
7	pb_200_07	200	5	23	0	**0**	**0**	0.00	**0**	**0**	0	**0**	**0**	0	**0**	**0**	0	**0**	10	13.08	17	-	-	-
8	pb_200_08	200	5	20	8	**8**	**8**	8.00	**8**	**8**	8.24	11	**8**	8	**8**	**8**	8	**8**	19	22.38	28	8	-	-
9	pb_200_09	200	5	24	10	**10**	**10**	10.00	**10**	**10**	10.1	11	**10**	10	**10**	**10**	10	**10**	16	20.55	23	10	4	0, 1, 2
10	pb_200_10	200	5	19	19	**19**	**19**	19.13	21	**19**	20.02	22	**19**	19.06	21	**19**	19.16	21	35	42.72	52	17	6	0, 2
11	pb_300_01	300	5	25	0	**0**	**0**	0.90	2	2	3.96	9	**0**	0.82	2	**0**	1.34	3	8	14.41	24	-	-	-
12	pb_300_02	300	5	25	12	**12**	**12**	12.11	13	**12**	13.24	18	**12**	12	**12**	**12**	12.2	13	17	25.46	29	-	**9**	0, 1, 2
13	pb_300_03	300	5	25	13	**13**	**13**	13.00	**13**	**13**	13.56	17	**13**	13	**13**	**13**	13	**13**	26	28.43	34	13	-	-
14	pb_300_04	300	5	24	7	**7**	**7**	8.44	10	8	9.62	13	**7**	8.04	9	8	8.96	12	23	31.90	41	7	-	-
15	pb_300_05	300	5	20	27	52	29	30.47	33	30	33.98	39	28	29.76	33	29	31.8	35	51	59.32	66	2	**12**	0, 3, 4
16	pb_300_06	300	5	25	2	**2**	**2**	3.48	5	4	6.76	11	**2**	2.78	4	3	4.3	7	13	20.04	31	2	-	-
17	pb_300_07	300	5	24	0	**0**	**0**	0.00	**0**	**0**	0.88	5	**0**	0	**0**	**0**	0.22	2	7	11.24	19	-	-	-
18	pb_300_08	300	5	23	8	**8**	**8**	8.00	**8**	**8**	8.2	10	**8**	8	**8**	**8**	8	**8**	15	20.22	25	8	-	-
19	pb_300_09	300	5	21	7	**7**	**7**	7.60	9	**7**	9.48	13	**7**	7.6	9	**7**	8.64	10	15	20.46	24	7	-	-
20	pb_300_10	400	5	19	21	25	**21**	21.35	22	**21**	22.74	26	**21**	21.02	22	**21**	21.32	23	44	49.02	57	3	**6**	0, 3
21	pb_400_01	400	5	25	1	**1**	**1**	2.23	3	2	5.32	10	**1**	1.84	3	**1**	2.8	4	10	15.50	19	-	-	-
22	pb_400_02	400	5	22	15	28	**15**	16.59	18	16	20.24	28	**15**	16.08	18	**15**	17.06	20	27	34.32	44	15	1	
23	pb_400_03	400	5	23	9	12	**9**	9.78	11	**9**	10.52	13	**9**	9.68	11	**9**	9.5	10	18	22.67	37	-	1	0, 2, 3, 4
24	pb_400_04	400	5	26	19	**19**	**19**	19.01	20	**19**	20.08	23	**19**	19.02	20	**19**	19.22	21	45	48.96	54	19	-	-
25	pb_400_05	400	5	20	0	**0**	**0**	0.00	**0**	**0**	0.06	2	**0**	0	**0**	**0**	0	**0**	11	14.81	20	-	-	-
26	pb_400_06	400	5	23	0	**0**	**0**	0.01	1	**0**	0.52	3	**0**	0	**0**	**0**	0.12	3	11	14.06	20	-	-	-
27	pb_400_07	400	5	23	4	**4**	**4**	4.76	7	5	8.74	16	**4**	4.42	6	**4**	5.46	7	12	18.80	28	-	-	-
28	pb_400_08	400	5	21	4	**4**	**4**	4.29	5	**4**	6.6	11	**4**	4.02	5	**4**	4.6	6	12	21.32	46	4	-	-
29	pb_400_09	400	5	24	5	**5**	**5**	7.31	9	6	9.36	15	**5**	7.08	9	7	9.42	12	31	36.44	47	-	-	-
30	pb_400_10	400	5	25	0	**0**	**0**	0.24	2	**0**	2.88	6	**0**	0.02	1	**0**	0.72	2	13	17.98	23	-	-	-
	Average:				**7.17**	8.90	7.27	7.77	8.67	7.63	9.30	12.70	**7.20**	**7.60**	**8.40**	7.40	8.10	9.37	18.87	24.21	31.40			
	# of best found:					24	28		11	21		1	29		13	25		9	0		0			

Table 2.6: Results for ROADEF Challenge Instances

								ALS			VNS			VFLS			SA			DSU		
		Instance Name	T	O	M	Obj*	IP	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
	1	022_3_4	485	9	17	0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	2	024_38_3	1260	13	49	27	289	35	51.0	63	45	56.0	70	**27**	39.1	55	40	55.4	69	137	157.0	189
	3	024_38_5	1315	13	43	33	123	43	51.0	64	47	59.8	75	**33**	39.2	46	40	52.2	65	173	197.9	238
	4	025_38_1	1004	22	190	114	2335	119	150.8	185	**114**	145.8	195	176	199.3	235	211	239.2	283	317	413.9	549
SET A	5	039_38_4	954	5	20	0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	6	048_39_1	600	17	155	58	556	61	65.4	71	65	72.2	80	**58**	64.8	71	63	73.2	85	104	128.3	150
	7	064_38_2_ch1	875	9	25	363	364	**363**	363.0	364	363	363.0	364	**363**	363.2	364	**363**	363.1	364	**363**	363.2	364
	8	064_38_2_ch2	335	6	12	0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	9	022	704	12	20	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	10	023	1260	12	78	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	11	024	1319	18	102	0	0	3	3.7	5	3	3.9	6	3	3.6	4	3	3.7	6	3	4.0	9
	12	025	996	20	194	11	766	**11**	24.5	43	13	31.9	49	21	32.7	51	34	51.8	73	47	99.2	148
	13	028_CH1	325	26	116	91	799	**91**	104.9	120	95	113.1	133	99	112.8	133	115	129.9	151	139	173.7	217
	14	028_CH2	65	6	7	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	15	029	780	7	13	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	2	2.0	2
	16	034_VP	921	8	12	11	11	20	23.7	28	22	28.2	34	19	23.0	27	23	28.8	37	39	40.2	43
	17	034_VU	231	8	7	10	13	**10**	12.2	14	11	12.8	15	11	11.9	13	12	12.9	14	19	24.1	27
SET X	18	035_CH1	90	1	2	9*	9	**9**	9.0	9	**9**	9.0	**9**	**9**	9.0	**9**	**9**	9.0	**9**	**9**	9.5	10
	19	035_CH2	376	2	3	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	20	039_CH1	1247	12	85	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	21	039_CH3	1037	12	45	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	22	048_CH1	519	22	125	0	0	**0**	0.2	2	**0**	1.5	5	**0**	1.5	5	1	5.9	15	48	52.1	57
	23	048_CH2	459	20	118	16*	16	**16**	17.5	19	**16**	17.4	19	**16**	17.5	19	**16**	17.8	20	**16**	17.7	20
	24	064_CH1	875	11	37	60*	60	**60**	60.0	61	**60**	61.1	64	**60**	62.3	66	65	73.4	83	77	82.0	93
	25	064_CH2	273	6	10	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	26	655_CH1	264	5	3	0	0	**0**	0.0	0	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**	**0**	0.0	**0**
	27	655_CH2	219	4	5	46	46	**46**	46.0	46	**46**	46.0	**46**	**46**	46.0	**46**	**46**	46.0	**46**	56	56.0	56
		Average:				**31.8**	199.5	**32.9**	**36.4**	**40.5**	33.7	37.8	43.1	34.9	38.0	42.4	38.6	43.0	48.9	57.4	67.4	80.4
		# of best found:					24	21		14	19		14	21		14	16		14	14		11

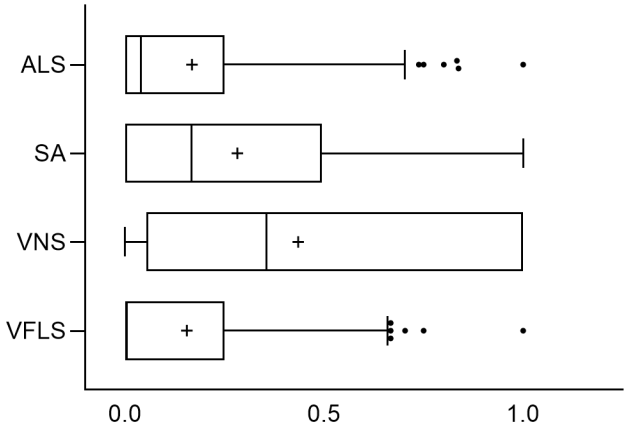


Figure 2.3: Box plots based on the normalized objective values of each instance

algorithms; we can see that the VNS is inferior among all. VFLS performs slightly better than ALS by finding 29 out of 30 best-known solutions at least once, providing 13 best-known solutions for all 50 runs. There is not much difference among the minimum objective values found by each algorithm. However, VNS and SA find worse solutions in some of the trials. On the other hand, the average minimum values of algorithms are very close. VFLS provides better solutions by 2% (3%), 6% (10%), and 18% (34%) than ALS, SA, and VNS on average of means (average maximum values).

In table 2.6, similar to the CSPLib results, the ALS and VFLS are superior to the other algorithms. The characteristics of the ROADEF challenge instances vary compared to the CSPLib instances which induce a wider range of the operator success rates, see table 2.4. Accordingly, ALS performs slightly better than VFLS and VNS due to its adaptive mechanism. On the average of means, ALS provides better solutions by %2, %6, %8 than VNS, VFLS, and SA, respectively.

We use the Mann-Whitney U test to investigate whether the distribution of the results provided by the algorithms is significantly different or not. A pairwise comparison of the algorithms shows that the ALS and VFLS outperform VNS with a 95% confidence level in terms of maximum values and there is not enough evidence to show that different results are obtained when considering the minimum and mean values. When all values found by the algorithms are normalized between [0,1] using the best known or found and worst found solutions for each instance, including CSPLib and ROADEF, as shown in figure 2.3, we see that ALS and VFLS obtain better results than the other algorithms.

2.6 Conclusion

This chapter presented a comparison of local search metaheuristics, namely adaptive local search (ALS), very fast local search (VFLS), variable neighborhood search (VNS), and simulated annealing with geometric cooling with bouncing strategy (SA). We used the sliding-window (SW) technique as the objective function and experimental tests were executed on two sets, including the latest CSPLib and simplified ROADEF challenge 2005 instances. The commercial solvers can provide good enough solutions for CSPLib instances with up to 400 cars and improved lower bounds are provided by solving the relaxed problem using Gurobi. However, the results of larger instances induced a need for a more reliable method.

We suspect that the reason behind the success of ALS and VFLS is that accepting worse solutions causes SA to spend more time in the search process away from the optimal solution which is not necessary for CSP since accepting the solutions with the same objective value creates the advantage of escaping local optima. The bouncing strategy within SA slightly improves the algorithm which proves that spending more time to escape local optima during the search process close to the optimal solution is more efficient. On the other hand, the poor performance of VNS shows that the systematic search of a very large neighborhood is more unfavorable than the random search for this problem. We saw that the VNS is worse on the CSPLib but comparable on ROADEF instances which shows that the time spent on a systematic search of neighborhoods may become more efficient for larger instances and for more complex versions of the CSP.

2.6.1 Managerial Insights

The instances in CSPLIB are very similar since they are artificially generated: all have five options, the capacity rules are the same, and utilization rates are very close. Similar performance across the operators and instances can be expected. However, in the car manufacturing facilities, the vehicle mix of each planning horizon is unique which induces unique characteristics. As demonstrated in this chapter through ROADEF instances that are obtained from industry, different characteristics of product mix and capacity rules cause transformation operators to act varied during the improvement process, see Table 2.4. Hence, for each planning horizon, vast experiments are required for the parameter tuning process in order to keep the solution quality (production efficiency) high.

In this chapter, we show that management can avoid these vast experiments, without sac-

rificing production efficiency, by utilizing adaptive parameters instead of using static parameters. Although, our study focus on the local search, it is worth noting that using adaptive parameters is equally applicable to any parameter-dependent optimization algorithm.

Chapter 3

Mixed-model Sequencing with Stochastic Failures: A Case Study for Automobile Industry

The work presented in this chapter is documented in [118].

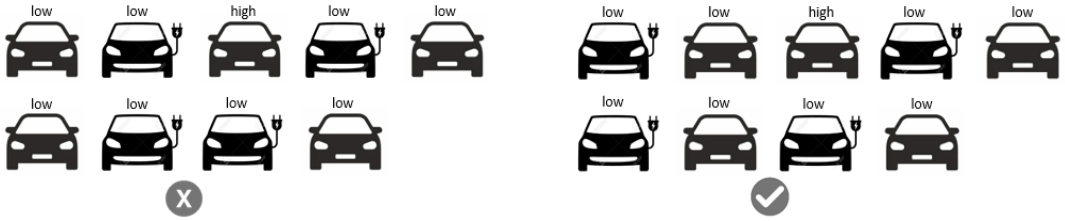
3.1 Introduction

Mixed-model assembly lines (MMAL) are capable of producing several configurations, models, of a product. The number of models increases drastically as the complexity and customizability of the product expand. The number of theoretical configurations of vehicles from a German car manufacturer is up to 10^{24} [84]. Different configurations require distinct tasks at each station which induces high variation in the processing times, though each station has a fixed maximum time available. In fact, station workload is distributed through line balancing such that each station's *average* workload conforms to this maximum time. When a station has more work allocated to it for a particular model (*work overload*), interventions are needed to maintain the flow of products in the assembly line, thereby avoiding line stoppages. Interventions can be considered in advance, through sequencing decisions, or at the time of disruption, through utility workers. When these interventions fail, the line may stop production until the situation is resolved. Thus, it is essential to distribute

the high work-load models along the planning horizon to avoid line stoppages.

The mixed-model sequencing (MMS) problem sequences products in an MMAL to minimize work overload at the stations. Data from our car manufacturing partner shows that the variation in processing times is high when customization appears on a main part, e.g., engine type: electric, diesel, gasoline, or hybrid. Car manufacturers have adapted their assembly lines for the mixed-model production of vehicles with diesel and gasoline engines. However, the assembly of electric vehicles (EV) in the same line has brought new challenges, while not eliminating the production of vehicles with diesel or gasoline engines. Unlike other vehicles, electric and hybrid vehicles have large batteries which causes a huge difference in tasks, e.g., at the station where the battery is loaded. As the proportion of electric and hybrid vehicles grows in a manufacturer's mix, the impact of supply problems increases. Sometimes, a part is delayed from a supplier, so a designed sequence of vehicles will have a missing vehicle. Even if this vehicle has a gasoline or diesel engine, its absence may impact the battery-intensive stations. As a manufacturer's mix of vehicles grows more specialized with more time-consuming content for a large subset, without alternative tasks for the vehicles without the specialized content, the impact of missing vehicles on a carefully designed sequence grows.

Some vehicles in a production sequence may not be ready for assembly on the production day for various reasons, such as the body not being ready, paint quality issues, or material shortage. Such vehicles, referred to as *failed vehicles*, need to be pulled out of the sequence. The resulting gap is closed by moving the succeeding vehicles forward. This process and the resulting additional work overload occurrence is illustrated in Figure 3.1 for a battery loading station. The processing time at this station is longer than the cycle time for EVs and shorter than the cycle time for non-EVs, and assume that back-to-back EVs cause work overload. We schedule five vehicles, two electric and three non-electric. One of the non-EVs (third in both scheduled sequences) has a high failure probability. The initial sequences with no failures, while different, both will lead to no work overload. Assuming the third vehicle fails, we have different consequences for the resultant sequence of vehicles. In the non-robust sequence, removing the failed non-EV results in two EVs in a row, which will cause a work overload. However, the robust sequence, which is composed of the same vehicles in a different order, can withstand the failure of the third vehicle without causing a work overload. We refer to this sequence as the "robust" sequence because no work overload occurs when the vehicle with high failure probability is pulled out of the sequence.



(a) non-robust sequence **(b)** robust sequence

Figure 3.1: Illustration of a non-robust and robust sequence to stochastic failures

In this chapter, we generate robust sequences that consider the vehicles' potential failures to reduce additional work overloads. We focus on the final assembly line, assuming that vehicles follow the same sequence as they arrive from the paint shop and resequencing is not an option; when a vehicle is removed from the sequence, the following vehicles close the gap. The contributions of this study are as follows:

- We provide a two-stage stochastic program for a MMS problem with stochastic product failures, and we provide improvements to the second-stage problem. To the best of our knowledge, this is the first study that considers stochastic failures of products in MMS.
- We adopt the sample average approximation (SAA) approach to tackle the exponential number of scenarios. The numerical experiments show that we can generate robust solutions with an optimality gap of less than 1% and 5% by utilizing a sample of scenarios, for the small-sized and industry-sized instances, respectively.
- We develop an L-shaped decomposition-based algorithm to solve small-sized instances. The numerical results show that the L-shaped algorithm outperforms an off-the-shelf solver, solving the deterministic equivalent formulation (DEF), in terms of both quality and computational time.
- To solve industry-sized instances, we propose a greedy heuristic and a tabu search (TS) algorithm that is accelerated in convergence with problem-specific tabu rules, and in objective reevaluation each time a new solution is visited.
- We conduct a case study with the data inspired by our car manufacturer industrial partner. The numerical experiments show that we can reduce the work overload by more than 20% by considering stochastic car failures and solving the corresponding problem with the proposed

solution methodologies.

The remainder of this chapter is structured as follows. MMS related literature is reviewed in Section 3.2. The tackled problem is defined, and the mathematical formulation of the proposed problem is presented in Section 3.3. Exact and heuristic solution approaches in addition to the SAA approach are presented in Section 3.4. In Section 3.5, we execute numerical experiments to analyze the performance of proposed solution methodologies and present the results. Finally, the chapter is concluded in Section 3.6.

3.2 Related Work

Manufacturers use various design configurations of MMALs to maximize their revenue. The optimization process of the assembly line sequencing takes these design configurations into account. The first paper that articulates the MMS was presented by Kilbridge and Wester [59]. The researchers tackle the MMS with varied characteristics which required a systematic categorization of the components and the operating system of MMS problems. Dar-EL categorizes the MMAL into four categories based on their main characteristics of assembly lines: product transfer system, product mobility on the conveyor, accessibility among adjacent stations, and the attribute of the launching period [29]. An analytic framework for the categorization of Dar-El is given by Bard *et al.* [9]. Later, a survey is presented by Boysen *et al.*, where they define tuple notation for the sequencing problems based on more detailed characteristics of assembly lines, including work overload management, processing time, concurrent work, line layout, and objective in addition to the main characteristics [18].

Several objectives are employed to evaluate the performance of the assembly line sequence. The most common objective in the literature, also adopted in this chapter, is minimizing the total work overload duration, proposed by Yano *et al.* [115]. Tsai describes hiring utility workers to execute tasks so that production delays are avoided, which leads to the objective of minimizing the total utility work duration [110]. Fattahi and Salehi minimize the total idle time in addition to utility work [37]. Boysen *et al.* propose minimizing the number of utility workers instead of the total utility work duration in order to improve utility worker management [19].

A few exact solution methods are proposed in the literature to solve the deterministic MMS problem. Scholl *et al.* proposes a decomposition approach that uses patterns of different sequences,

called pattern-based vocabulary building [96]. They use a column generation method to solve the linear relaxation of the formulation and an informed tabu search is adapted to determine the pattern sequence. Bolat proposes a job selection problem that is solved prior to the sequencing problem [15]. They employ a due date-oriented cost function as an objective, and the work overload is restricted as a hard constraint. They develop a branch-and-bound (B&B) algorithm that is improved with some dominance criteria, a procedure to compare sequences based on the quality, which can select 50 jobs out of 100 in seconds. Kim and Jeong present a B&B algorithm to solve the MMS problem with sequence-dependent setup times [60]. They calculate a lower bound on the work overload of the current sequence and the minimum possible work overload of the unconsidered configurations. The model can solve instances with up to five stations and 30 configurations. Boysen *et al.* integrate a skip policy for utility work into the MMS and formulate the new problem as a mixed-integer linear program (MILP) [19]. They propose a B&B algorithm with improved lower bound calculations and dominance rules.

There are several heuristic and meta-heuristic approaches related to the MMS. The most popular algorithm is the genetic algorithm (GA) which is adapted in several ways to solve MMS problems [56, 61, 85, 66, 25, 2, 119]. Akgunduz and Tunali review GA-based MMS solution approaches [3]. Other popular evolutionary algorithms that are used to solve MMS include ant colony optimization [4, 122, 63], particle swarm [90, 74, 113], and scatter search algorithm [89, 24, 68].

The standard MMS assumptions given in Chapter 1 create a gap between the industrial applications and the research aspects of the problem. In the last decades, researchers have narrowed this gap by publishing case study papers that relax some assumptions. In the remainder of this section, we review MMS-related case studies that make a significant contribution in this regard.

Bautista and Cano, partnering with car manufacturer Nissan, consider interruption of operations due to control management process [12]. They introduce an interruption rule and integrate this into the problem formulation so that the interruption of production does not result in unexpected delays or incomplete jobs. They solve the problem with a commercial solver and also propose a bounded DP. In addition to incorporating line interruption, the authors formulate an extended version of the MMS which allows operators to work in the same workstation arranged in parallel [23]. They also introduce workload regularity constraints to balance the work among the operators. Moreover, Bautista and Pozo [11] include mixed product restrictions, which limits the number of products in a sub-sequence, in the problem given in [23].

The normal working pace of the workers is the pace at which a worker can repeat the same tasks during the whole shift. In literature, it is assumed that the workers always work at the same pace, however, it is usual to expect workers to work at a faster pace in case of work overload. Bautista *et al.* consider the human element of the assembly line to minimize the work overload by increasing the pace of the workers when it is necessary [10]. To incorporate the work pace factor into the MMS they use the *Methods and Time Measurement (MTM)* system to determine standard processing times of tasks. Additionally, they introduce a pace function that adapts the Yerkes-Dudson law (inverted U shape) which assumes that the performance of workers gradually increases until fatigue starts. Finally, they formulate the MMS with the work pace factor as an MILP and solve it using a commercial solver. The results show that a significant cost reduction can be provided by increasing the work pace of the workers inside the limits of the collective agreement.

When the whole production time of a single product (lead time) is large, some models need to be fixed to a certain position or a pre-specified range of positions because of a due date requirement. Buergin *et al.* present a case study of sequencing Airbus A320 models with a weighted objective of order-related costs, work overload costs, and part usage variation costs [22]. They also consider varied penalty costs due to open border stations, e.g., the penalty of a work overload that can be compensated by an idle worker from the neighbor station is lower than the penalty of a work overload that requires overtime work. They formulate the problem as an MIP and solve it by using CPLEX. The overall costs are decreased by 97%.

Aroui *et al.* conduct a case study from the truck industry which considers three types of operators that handle different tasks [8]. Type 1 operators perform tasks on each model. Type 2 operators perform tasks for only pre-defined models and the duration of their tasks may be multiple cycle times, e.g., two type 2 operators are assigned to stations 1 and 2 together but operator 1 works on models 1,2 and operator 2 works on models 3,4. Lastly, type 3 operators are assigned to indivisible tasks (a single task occurs for each model and lasts multiple cycle times). They also consider that multiple operators can work at the same workstation. They propose an MILP formulation, and two metaheuristic approaches (GA and SA). They solve a case study of Volvo's truck production plant. The computational results show that they improve work overload by an average of 36.5%.

Tanhaie *et al.* propose a modified multi-objective particle swarm optimization (MOPSO) approach to solve a multi-objective MMS integrated into a make-to-order (MTO) environment [106]. Before sequencing the orders they first determine the prioritization of the orders by considering

criteria like customer reputation, customer loyalty, critical ratio, and customer-firm communication. Three minimization objectives are considered, namely the total set-up time, the number of work overload, and the total earliness and tardiness. They compare the MOPSO with four other well-known evolutionary algorithms and the results show that the MOPSO is the most suitable algorithm for the MMS with demand management.

While the majority of the MMS literature focuses on models with deterministic parameters, there are a few studies that consider stochastic parameters on either processing times or demand. The seminal study with stochastic processing times is proposed by Zhao *et al.* [121]. They provide a Markov chain based approach that minimizes the expected work overload duration. This approximation is done by generating sub-intervals of possible positions of workers within the stations. The expected work overload is calculated based on the lower and upper bounds of the intervals. Mosadegh *et al.* propose a heuristic approach, inspired by Dijkstra's algorithm, to tackle a single-station MMS with stochastic processing times [76]. They formulate the problem as a shortest path problem. Mosadegh *et al.* formulate a multiple-station MMS with stochastic processing times as an MILP [77]. They provide a Q-learning-based simulated annealing (SA) heuristic to solve industry-sized problems and show that the expected work overload is decreased compared to the deterministic problem. Bramer *et al.* propose a reinforcement learning approach to solve MMS by negatively rewarding work overload occurrences [21]. They show that the proposed approach provides at least 7% better solutions than SA and GA. Moreover, stochastic parameters are considered in integrated mixed-model balancing and sequencing problems as well: in processing times [1, 79, 32] and in demand [98]. Sikora *et al.* propose a Bender's decomposition algorithm to solve an integrated assembly line balancing and sequencing as a two-stage problem with stochastic demand [98]. The balancing decisions and the sequencing decisions are made in the first and second stages, respectively. They reduce the balancing problem domain by finding the earliest and latest possible stations for each task using the tasks' average duration. Then, valid inequalities, which are proposed to strengthen the master problem, set a lower bound on the work overload for each scenario using the unavoidable work overloads and idle times. Finally, they present cuts based on the relaxation by solving the sub-problem for only a subset of stations which provides a lower bound on the work overload.

Although numerous studies have been conducted on the sequencing problems, only Hottenrott *et al.* consider the product failures in sequence planning, yet in the car sequencing structure [55]. To the best of our knowledge, there is no research available that establishes robust sequences

in the MMS structure that can withstand work overloads caused by product failures.

3.3 Problem Statement and Mathematical Formulation

In Section 3.3.1, we define the MMS with stochastic failures and illustrate the problem with an example. Then, in Section 3.3.2, we provide a two-stage stochastic program for our problem.

3.3.1 Problem Statement

In an MMAL, a set of workstations are connected by a conveyor belt. Products, launched with a fixed rate, move along the belt at a constant speed of one time unit (TU). The duration between two consecutive launches is called the cycle time c, and we define the station length $l_k \geq c$ in TU as the total TU that the workpiece requires to cross the station $k \in K$. Operators work on the assigned tasks and must finish their job within the station length, otherwise, the line is stopped or a so-called utility worker takes over the remaining job. The excess work is called *work overload*. The sequence of products therefore has a great impact on the efficiency of the assembly line. MMS determines the sequence of a given set of products V by assigning each product $v \in V$ to one of the positions $t \in T$.

Formulating the MMS problem based on the vehicle configurations instead of vehicles is usual [16, 9, 96], however, automobile manufacturers offer billions of combinations of options. When this high level of customization is combined with a short lead time promised to the customers, each vehicle produced in a planning horizon becomes unique. In this chapter, the vehicles are sequenced instead of configurations since the case study focuses on an automobile manufacturing facility with a high level of customization. In order to do so, we define a binary decision variable x_{vt}, which takes value of 1 if vehicle $v \in V$ is assigned to position $t \in T$. The processing time of vehicle $v \in V$ at station $k \in K$ is notated by p_{kv}. The starting position and work overload of the vehicle at position $t \in T$ for station $k \in K$ are represented by z_{kt} and w_{kt}, respectively. Table 3.1 lists all the parameters and decision variables used in the proposed model. While second-stage decision variables are scenario-dependent, we drop such dependency for notation simplicity throughout the chapter unless it is needed explicitly for clarity.

In this chapter, we adopt the side-by-side policy as a work overload handling procedure. A utility worker is assumed to work with the regular worker side-by-side, enabling work to be

Table 3.1: List of parameters and decision variables used in the model

Sets and Index	
V, v	Vehicles
K, k	Stations
T, t	Positions
Ω, ω	Scenarios
Parameters	
p_{kv}	The processing time of vehicle $v \in V$ at station $k \in K$
l_k	The length of station $k \in K$
c	The cycle time
f_v	The failure probability of vehicle $v \in V$
$e_{v\omega}$	1 if vehicle $v \in V$ exists at scenario $\omega \in \Omega$, 0 otherwise
First-Stage Decision variables	
x_{vt}	1 if vehicle $v \in V$ is assigned to position $t \in T$, 0 otherwise
Second-Stage Decision variables	
w_{kt}	The work overload at station $k \in K$ at position $t \in T$
z_{kt}	Starting position of operator at station $k \in K$ at the beginning of position $t \in T$
b_{kt}	The processing time at station $k \in K$ at position $t \in T$

completed within the station borders. The objective of MMS with the side-by-side policy is to minimize the total duration of work overloads, i.e., the total duration of the remaining tasks that cannot be completed within the station borders. The regular operator stops working on the piece at the station border so they can start working on the next workpiece at position $l_k - c$ in the same station.

We note that the vehicles usually go through the body shop and paint shop in the scheduled sequence before the assembly process. Hence, the failed vehicles must be pulled out of the sequence, and its position cannot be compensated, i.e., resequencing is not an option. It is assumed that each vehicle $v \in V$ (with its unique mix of configurations) has a failure probability f_v, and failures are independent of each other. The failures are related to the specifications of the vehicle, e.g., the increased production rate of EVs may induce higher failure rates, or painting a vehicle to a specific color may be more problematic. In our numerical experiments in Section 3.5, we estimate the failure probabilities from the historical data by doing feature analysis and using logistic regression.

3.3.2 Mathematical Model Under Uncertainty

In this section, first, we provide a two-stage stochastic program for our problem. Next, we discuss improvements of the proposed formulation.

Motivated by the dynamics of an MMAL, we formulate our problem as a two-stage stochastic program. The sequence of vehicles is decided in the first stage (here-and-now), before the car failures are realized. Once the car failures are realized, the work overload is minimized by determining the second-stage decisions (wait-and-see) given the sequence. First-stage decisions are determined by

assigning each vehicle to a position such that the expected work overload in the second stage is minimized.

To formulate the problem, suppose that various realizations of the car failures are represented by a collection of finite scenarios Ω. As each vehicle either exists or fails at a scenario, we have a total of $2^{|V|}$ scenarios. We let $\Omega = \{\omega_1, \ldots, \omega_{2^{|V|}}\}$, with ω indicating a generic scenario. To denote a scenario ω, let $e_{v\omega} = 1$ if vehicle v exists and $e_{v\omega} = 0$ if vehicle v fails at scenario $\omega \in \Omega$. We can then calculate the probability of scenario ω as $\rho_\omega = \prod_{v=1}^{|V|} f_v^{1-e_{v\omega}}(1-f_v)^{e_{v\omega}}$ such that $\sum_{\omega\in\Omega} \rho_\omega = 1$, where f_v denotes the failure probability of vehicle $v \in V$.

A two-stage stochastic program for the *full-information problem*, where all possible realizations are considered, is as follows:

$$\min_{x} \quad \sum_{\omega\in\Omega} \rho_\omega Q(x,\omega) \tag{3.1a}$$

$$\text{s.t.} \quad \sum_{v\in V} x_{vt} = 1, \qquad t \in T \tag{3.1b}$$

$$\sum_{t\in T} x_{vt} = 1, \qquad v \in V \tag{3.1c}$$

$$x_{vt} \in \{0,1\}, \qquad t \in T, \;\; v \in V \tag{3.1d}$$

where

$$Q(x,\omega) = \min_{z,w,b} \quad \sum_{k\in K}\sum_{t\in T_\omega} w_{kt} \tag{3.2a}$$

$$\text{s.t.} \quad b_{kt} = \sum_{v\in V} p_{kv} x_{vt}, \qquad k \in K, \;\; t \in T_\omega \tag{3.2b}$$

$$z_{kt} - z_{k(t+1)} - w_{kt} \le c - b_{kt}, \qquad k \in K, \;\; t \in T_\omega, \tag{3.2c}$$

$$z_{kt} - w_{kt} \le l_k - b_{kt}, \qquad k \in K, \;\; t \in T_\omega, \tag{3.2d}$$

$$z_{k0} = 0, \qquad k \in K, \tag{3.2e}$$

$$z_{k(|T_\omega|+1)} = 0, \qquad k \in K, \tag{3.2f}$$

$$z_{kt}, w_{kt} \ge 0, \qquad k \in K, \;\; t \in T_\omega, \tag{3.2g}$$

In the first-stage problem (3.1), the objective function represents the expected work overload, i.e., the cost associated with the second-stage problem. Constraint sets (3.1b) and (3.1c) ensure that ex-

actly one vehicle is assigned to each position and each position has exactly one vehicle. respectively. Constraint set (3.1d) presents the domain of the binary first-stage variable. The second-stage problem (3.2) minimizes the total work overload throughout the planning horizon, given the sequence and scenario $\omega \in \Omega$. Note that T_ω denotes the set of positions of non-failed vehicles at scenario $\omega \in \Omega$, which is obtained by removing failed vehicles. Constraint set (3.2b) determines the processing time b_{kt} at station k at position t. The starting position and workload of the vehicles at each station are determined by constraint sets (3.2c) and (3.2d), respectively. The constraint set (3.2e) ensures that the first position starts at the left border of the station. Constraint set (3.2f) builds regenerative production planning, in other words, the first position of the next planning horizon can start at the left border of the station. The constraint set (3.2g) defines the second-stage variables as continuous and nonnegative.

Several remarks are in order regarding the proposed two-stage stochastic program (3.1) and (3.2). First, the number of decision variables and the set of constraints in (3.2) are scenario-dependent, as the valid positions T_ω are obtained based on the failure scenario $\omega \in \Omega$. Second, the proposed two-stage stochastic program (3.1) and (3.2) has a simple recourse. That is, once the sequence is determined and the failures are realized, the work overloads are calculated from the sequence of the existing vehicles, without resequencing.

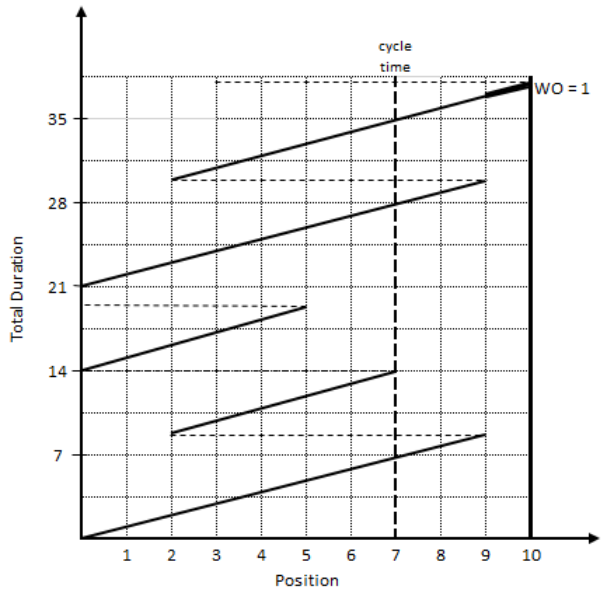
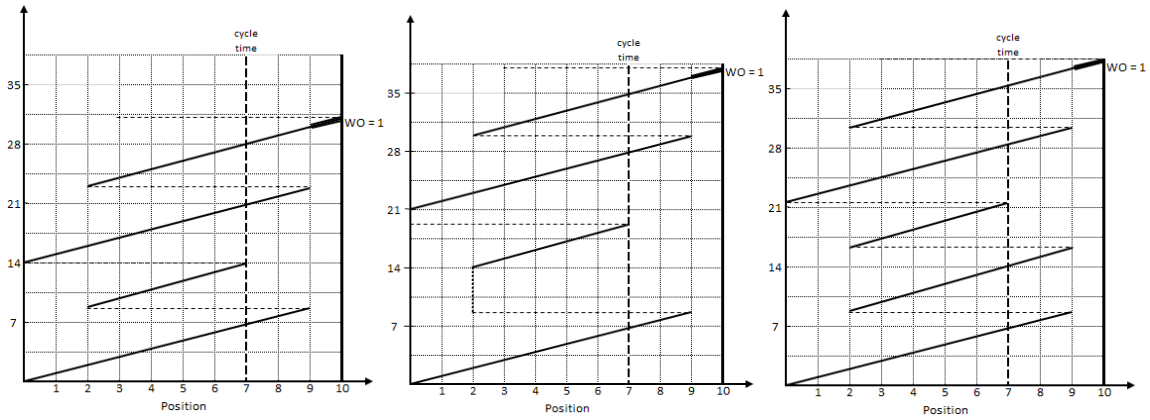


Figure 3.2: Illustration of mixed-model assembly line with five vehicles

In the remainder of this section, we first provide two modified models for the second-stage problem so that the number of decision variables and the set of constraints are no longer scenario-dependent. Then, we provide two monolithic MILP formulations for the deterministic equivalent formulation (DEF) of the two-stage stochastic program of MMS with stochastic failures.

For each scenario, we modify the second-stage problem by updating the processing times of failed vehicles instead of removing the failed vehicles. In Figure 3.3, we demonstrate how the original model (3.2) and modified models represent car failures. To this end, we consider the example given in Figure 3.2 (refer to Figure 1.1 for the explanation) and assume that the vehicle at the second position fails. In the original model, the failed vehicles are removed from the sequence and the succeeding vehicles moved forward (Figure 3.3a). The proposed modified models, referred to as *standard model* and *improved model*, are explained below. In Section 3.5.2, we discuss the impact of these modified models on the computational time and solution quality.



(a) Original model **(b)** (Modified) standard model **(c)** (Modified) improved model

Figure 3.3: Assembly line illustration of proposed models

Standard Model: In a preprocessing step, the processing time of vehicle v is set to zero for all stations if the vehicle fails in scenario ω. Accordingly, since this modification is conducted by adding uncertainty to the processing times, the scenario index ω is added to the processing time. That is, $e_{v\omega} = 0 \Rightarrow p_{kv\omega} = 0 \;\; k \in K$. Based on this modification, the second-stage problem for a

given scenario ω can be presented as

$$Q(x,\omega) = \min_{w,z,b} \quad \sum_{k\in K}\sum_{t\in T} w_{kt} \tag{3.3a}$$

$$\text{s.t.} \quad b_{kt} = \sum_{v\in V} p_{kv}x_{vt}, \qquad k \in K,\ t \in T, \tag{3.3b}$$

$$z_{kt} + b_{kt} - w_{kt} - z_{k(t+1)} \leq c, \qquad k \in K,\ t \in T, \tag{3.3c}$$

$$z_{kt} + b_{kt} - w_{kt} \leq l_k, \qquad k \in K,\ t \in T, \tag{3.3d}$$

$$z_{k0} = 0, \qquad k \in K, \tag{3.3e}$$

$$z_{k(T+1)} = 0, \qquad k \in K, \tag{3.3f}$$

$$z_{kt} - \beta_k b_{kt} - z_{k(t+1)} \leq 0, \qquad k \in K,\ t = \{1,\dots,T-1\} \tag{3.3g}$$

$$z_{kT} - w_{kT} - \beta_k b_{kT} \leq 0, \qquad k \in K, \tag{3.3h}$$

$$z_{kt}, w_{kt}, b_{kt} \geq 0, \qquad k \in K,\ t \in T, \tag{3.3i}$$

The objective function (3.3a) and the constraints (3.3b)–(3.3f) are the same as in formulation (3.2), except the set of positions and the length of the sequence are not scenario-dependent anymore. Constraint set (3.3g) guarantees that the starting position at station k at position $t+1$ equals the starting position of position t when the vehicle assigned to position t fails. Constraint set (3.3h) assures that the regenerative production planning is kept in case the vehicle at the end of the sequence fails. The parameter β is calculated in a way that $\beta_k b_{ktn} > z_{ktn}$ which makes constraint sets (3.3g) and (3.3h) non-effective for the positions that have existing vehicles. Hence, β_k equals the maximum possible starting position divided by the minimum processing time for station k, $\beta_k = \frac{l_k - c}{\min_{v\in V}\{p_{kv}\}} > 0$. Note that the processing times in this calculation are the actual processing times before the preprocessing step. Also, β_k is well-defined as the minimum processing time is strictly greater than zero. Figure 3.3b demonstrates that in the standard model, the processing time of the second vehicle is set to zero, so the operator starts working on the third vehicle at position two where the operator was going to start working on the second vehicle if it had not failed.

Improved Model: In order to reduce the size of the standard model, we modify this model as follows. During the preprocessing step, the processing time of vehicle v is set to the cycle time for all stations if a vehicle fails at scenario ω. Let us refer to the vehicles with processing time equal to cycle time for all stations as "neutral" because these vehicles do not have any impact on the schedule

in terms of work overload (see Proposition 3.1 and its proof). In other words, we transform failed vehicles into *neutral* vehicles, i.e., $e_{v\omega} = 0 \Rightarrow p_{kv\omega} = c \;\; k \in K$.

Proposition 3.1. *A neutral vehicle has the same starting position as its succeeding vehicle at all stations. That is,* $b_{kt} = c \;\; \Rightarrow \;\; z_{k(t+1)} = z_{kt}$.

Proof. The operator's starting position of the vehicle at $t+1$ is $z_{k(t+1)} = z_{kt} + b_{kt} - c - w_{kt}$. Assume that the vehicle at position t is a neutral vehicle. We have $z_{k(t+1)} = z_{kt} - w_{kt}$. Hence, showing that the neutral vehicles never cause a work overload, $w_{kt} = 0$, completes the proof. We know that the maximum starting position at a station is $\max_{t \in T}\{z_{kt}\} = l_k - c$, which is a result of two extreme cases: an operator finishes working on a workpiece at the right border of a station or the operator cannot finish the work so we have a work overload. The starting position is less than $l_k - c$ for other cases. Therefore, a vehicle with a processing time less than or equal to c at a station cannot cause any work overload. This completes the proof. □

As a result of Proposition 3.1, constraints (3.3g) and (3.3h) can be removed from the standard model. Hence, the problem size is reduced. Figure 3.3c contains an illustration for Proposition 3.1. The second vehicle becomes neutral when its processing time is set to cycle time so that the third vehicle starts at the same position as the second vehicle.

Using the standard or improved model, the DEF for MMS with stochastic failures can be obtained by adding the first-stage constraints (3.1b)–(3.1d) to the corresponding second-stage formulation, and by adding copies of all second-stage variables and constraints. We skip the details for brevity.

3.4 Solution Approaches

In Sections 3.4.1 and 3.4.2, we propose an L-shaped decomposition-based algorithm and a tabu search algorithm in addition to a greedy heuristic, respectively, to solve the models presented in Section 3.3.2. Then, in Section 3.4.3, the SAA approach is motivated and a solution quality assessment scheme is presented.

3.4.1 Exact Solution Approach

For the ease of exposition, we consider an abstract formulation of the two-stage stochastic program presented in Section 3.3.2 as follows:

$$z^* = \min_{x \in X} \mathbb{E}[Q(x, \xi_\omega)], \tag{3.4}$$

where x denotes the first-stage decision variables and $X := \{x \in \{0,1\}^{|V|\times|T|} \colon Ax = b\}$ is the feasible region of decision variables x, i.e., the set of points satisfying constraints (3.1b) - (3.1d). Moreover, we represent the second-stage problem for the standard or improved model, presented in Section 3.3.2, as

$$Q(x, \xi_\omega) = \min_y \{q^\top y | Dy \geq h_\omega - T_\omega x,\ y \geq 0\}, \tag{3.5}$$

where y represents the second-stage decision variables and $\xi_\omega = (h_\omega, T_\omega)$. The expectation of the recourse problem becomes $\mathbb{E}[Q(x, \xi_\omega)] = \sum_{\omega\in\Omega} \rho_\omega Q(x, \xi_\omega)$.

The L-shaped method is a procedure that has been successfully used to solve large-scale two-stage stochastic programs. Note that for any $\omega \in \Omega$, function $Q(x, \xi_\omega)$, defined in (3.5), is convex in x because x appears on the right-hand side of constraints. Hence, we propose to iteratively construct its underestimator. To this end, for each $\omega \in \Omega$ and a given first-stage decision $x \in X$, we consider a *subproblem* that takes the form of (3.5). Moreover, we create a *relaxed master problem*, which contains a partial, but increasingly improving, representation of $Q(x, \xi_\omega)$, for each $\omega \in \Omega$, through the so-called *Benders' cuts*. Recall that our proposed two-stage stochastic programs have a relative complete recourse, that is, for any first-stage decision x, there is a feasible second-stage variable y. Thus, an underestimator of $Q(x, \xi_\omega)$ can be constructed by only the so-called *Benders' optimality cuts*.

We now describe more details on our proposed L-shaped algorithm. We form the relaxed master problem for formulation (3.4) and (3.5) as follows:

$$\min_{x,\theta} \quad \sum_{\omega\in\Omega} \rho_\omega \theta_\omega \tag{3.6a}$$

$$\text{s.t.} \quad x \in X \tag{3.6b}$$

$$\theta_\omega \geq G_\omega^\iota x + g_\omega^\iota, \quad \iota \in \{1, \ldots, l\},\ \omega \in \Omega, \tag{3.6c}$$

where the auxiliary variable θ_ω approximates the optimal value of the second-stage problem under scenario $\omega \in \Omega$, i.e., $Q(x, \xi_\omega)$, through cuts $\theta_\omega \geq G_\omega^\iota x + g_\omega^\iota$ formed up to iteration l.

Let $(\hat{x}^\iota, \hat{\theta}^\iota)$ be an optimal solution to the relaxed master problem (3.6). For each scenario $\omega \in \Omega$, we form a subproblem (3.5) at $\hat{x}^\iota$. Suppose that given $\hat{x}^\iota$, $\hat{\pi}_\omega^\iota$ denotes an optimal dual vector associated with the constraints in (3.5). That is, $\hat{\pi}_\omega^\iota$ is an optimal extreme point of the dual subproblem (DSP)

$$\max_{\pi} \{\pi_\omega^\top (h_\omega - T_\omega \hat{x}^\iota) | \pi_\omega^\top D \leq q^\top,\ \pi_\omega \geq 0\}, \tag{3.7}$$

where π_ω is the associated dual vector. Then, using linear programming duality, we generate an optimality cut as

$$\theta_\omega \geq G_\omega^\iota x + g_\omega^\iota, \tag{3.8}$$

where $G_\omega^\iota = -(\hat{\pi}_\omega^\iota)^\top T_\omega$ and $g_\omega^\iota = (\hat{\pi}_\omega^\iota)^\top h_\omega$.

Our proposed L-shaped algorithm iterates between solving the relaxed master problem (3.6) and subproblems (3.5) (one for each $\omega \in \Omega$) until a convergence criterion on the upper and lower bounds is satisfied. This algorithm results in an L-shaped method with *multiple* cuts.

In order to exploit the specific structure of the MMS problem and to provide improvements on the dual problem, let us define variables π^{sp}, π^{wo}, π^{fs}, π^{ch}, π^{sf}, and π^{cf} corresponding to starting position constraints (3.3c), work overload constraints (3.3d), first station starting position constraints (3.3e), regenerative production planning constraints (3.3f), starting position of the vehicles following a failed vehicle (3.3f), and regenerative production planning with failed vehicles (3.3g), respectively. The DSP for scenario $\omega \in \Omega$ at a candidate solution $\hat{x}^\iota$, obtained by solving a relaxed master

problem, can be formulated as follows:

$$\max_{\pi} \quad \sum_{k\in K}\sum_{t\in T}\pi^{sp}_{kt}(\sum_{v\in V} p_{kv}\hat{x}_{vt} - c) + \pi^{wo}_{kt}(\sum_{v\in V} p_{kv}\hat{x}_{vt} - l_k) \tag{3.9a}$$

$$\text{s.t.} \quad \pi^{sp}_{k0} + \pi^{wo}_{k0} + \pi^{fs}_{k} + \pi^{sf}_{k0} \leq 0, \qquad k \in K \tag{3.9b}$$

$$\pi^{sp}_{kt} - \pi^{sp}_{k(t+1)} - \pi^{wo}_{k(t+1)} + \pi^{sf}_{kt} - \pi^{sf}_{k(t+1)} \leq 0, \qquad k \in K,\ \ t \in \{1,..,T-1\} \tag{3.9c}$$

$$\pi^{sp}_{kT} - \pi^{ch}_{k} \leq 0, \qquad k \in K \tag{3.9d}$$

$$\pi^{sp}_{kt} + \pi^{wo}_{kt} \leq 1, \qquad k \in K,\ \ t \in \{1,..,T-1\} \tag{3.9e}$$

$$\pi^{sp}_{kT} + \pi^{wo}_{kT} + \pi^{cf}_{k} \leq 1, \qquad k \in K \tag{3.9f}$$

$$\pi^{sp}_{kt}, \pi^{wo}_{kt}, \pi^{sf}_{kt}, \pi^{cf}_{k} \geq 0, \qquad k \in K,\ \ t \in T \tag{3.9g}$$

$$\pi^{fs}_{k}, \pi^{ch}_{k} \text{unrestricted}, \qquad k \in K \tag{3.9h}$$

We provide improvements to the dual problem in several ways. The dual variables π_{fs} and π_{cf} are removed since the corresponding subproblem constraints (3.3f) and (3.3g) are eliminated in the improved model. The dual variables π^{fs} and π^{ch} are not in the objective function and are unrestricted, which means that we can remove these variables and the constraints with those variables from the formulation without altering the optimal value of the problem. In our preliminary computational studies, we improved the dual subproblem by removing these variables. However, we observed that most of the DSPs have multiple optimal solutions, and as the number of vehicles and stations increase, it is more likely to have multiple optimal solutions. This naturally raises the question of what optimal dual vector provides the strongest cut, if we add only one cut per iteration per scenario. One can potentially add the cuts corresponding to all optimal dual extreme points, however, this results in an explosion in the size of the relaxed master problem after just a couple of iterations. While there is no reliable way to identify the weak cuts [91], we executed experiments in order to find a pattern for strong cuts. Our findings showed that adding the cut corresponding to the optimal dual extreme point with the most non-zero variables results in the fastest convergence. Thus, we added an ℓ_1 regularization term to the objective function of the DSP, hence, the new objective is encouraged to choose an optimal solution with the most non-zero variables. Accordingly, we propose

an improved DSP formulation as follows:

$$\max_{\pi} \quad \sum_{k\in K}\sum_{t\in T} \pi^{sp}_{kt}(\sum_{v\in V} p_{kv}\hat{x}_{vt} - c + \epsilon) + \pi^{wo}_{kt}(\sum_{v\in V} p_{kv}\hat{x}_{vt} - l_k + \epsilon) \tag{3.10a}$$

$$\text{s.t.} \quad \pi^{sp}_{kt} - \pi^{sp}_{k(t+1)} - \pi^{wo}_{k(t+1)} \leq 0, \qquad k \in K,\ t \in \{1,..,T-1\} \tag{3.10b}$$

$$\pi^{sp}_{kt} + \pi^{wo}_{kt} \leq 1, \qquad k \in K,\ t \in T \tag{3.10c}$$

$$\pi^{sp}_{kt}, \pi^{wo}_{kt} \geq 0, \qquad k \in K,\ t \in T \tag{3.10d}$$

Determining the dual variables is not trivial. Therefore, we propose Algorithm 10 in the Appendix B to find the optimal dual variables. This heuristic approach is not employed in this study since our preliminary experiments show that using Gurobi to solve each SP as an LP is faster than our implementation on Python, yet one can be interested in using the heuristic approach.

3.4.2 Heuristic Solution Approach

MMS is an NP-hard problem, and stochastic failures of products (cars) increases the computational burden of solving the problem drastically. Hence, it is essential to create efficient heuristic procedures in order to solve industry-sized problems. In this section, we provide a fast and easy-to-implement greedy heuristic to find a good initial feasible first-stage decision (i.e., a sequence of vehicles) and an efficient tabu search (TS) algorithm to improve the solution quality. Although all $|V|!$ vehicle permutations are feasible, the proposed greedy heuristic aims to find a good initial feasible solution. To achieve this, a solution is generated for the deterministic counterpart of the proposed MMS problem, which excludes vehicle failures. We refer to this problem as the *one-scenario problem*, since the corresponding problem has a single scenario with no failed vehicles. Assuming that the failure probability of each vehicle is less than or equal to 0.5, the scenario with no failed vehicles has the highest probability. Once such a feasible sequence of vehicles is generated, the TS algorithm improves this solution in two parts: first, over the one-scenario problem, and then, over the full-information problem.

3.4.2.1 Greedy Heuristic

It is important for a local search heuristic algorithm to start with a good quality solution. A naive approach to generate an initial solution (sequence) is to always select the vehicle that causes

the minimum new work overload for the next position. However, this approach is myopic since it only considers the current position. We remediate this issue by decreasing future work overloads which includes considering idle times and dynamic utilization rates. Accordingly, in order to generate a good initial solution, we propose utilizing an iterative greedy heuristic that follows a priority rule based on the work overload, idle time, and weighted sum of processing time, to be defined shortly.

Before explaining our proposed greedy heuristic, let us define some technical terms. The *idle time* refers to the duration that an operator waits for the next vehicle to enter the station borders. The weight of processing times is determined by using station *utilization rates*, which is inspired by car sequencing problem utilization rates [101, 48]. We describe the utilization rate of a station as the ratio between the average processing time on a station and the cycle time, so the utilization rate of station k is $\sum_{v \in V} p_{kv}/(|V| * c)$. At each iteration, after a new assignment of a vehicle, the dynamic utilization rates are calculated by considering only the unassigned vehicles. Accordingly, the weighted sum of the processing time of a vehicle v is calculated using (3.11):

$$\frac{\sum_{k \in K} p_{kv} \sum_{i \in \hat{V}} p_{ki}}{|K| * |\hat{V}| * c}, \tag{3.11}$$

where $\hat{V}$ denotes the set of unassigned vehicles. If the utilization rate of a station is greater than 1, then the average processing time is more than the cycle time, which induces an unavoidable work overload. On the other hand, a utilization rate close to 0 indicates that the average processing time is minimal compared to the station's allocated time.

Our proposed greedy heuristic builds a sequence iteratively, one position at a time, starting from the first position and iterating over positions. We use t to denote an iteration. At each iteration, $t = 1, \ldots, T$, the unassigned vehicles that cause the minimum new work overload are determined, denoted by $V_{t,wo}$. Ties are broken by selecting the vehicles from the set $V_{t,wo}$, which causes the minimum new idle time, the new set of vehicles is denoted by $V_{t,idle}$. In the case of ties, the vehicle with the highest weighted sum of processing time from the set $V_{t,idle}$ is assigned to the position t of the sequence. Note that the first vehicle of the sequence is the vehicle with the highest weighted sum of processing time among the set $V_{0,idle}$ since there is no work overload initially.

Finally, we enhance the proposed greedy heuristic by considering the category of the vehicles. Motivated by our case study, we categorize the vehicles based on the engine type, electric or non-electric, because the engine type is the most restrictive feature due to the high EV ratio (number of

EVs divided by the number of all vehicles). Moreover, the engine type leads to different processing times on a specific station. Hence, we modify our greedy heuristic to first decide whether an EV or a non-EV should be assigned to the next position at each iteration. Accordingly, first, the EV ratio is calculated, and an EV is assigned to the first position. The procedure always follows the EV ratio. For example, if the EV ratio is 1/3, an EV will be assigned to the positions $1 + 3t$ where $t = \{1, \ldots, \frac{|T|}{3} - 1\}$. In case of a non-integer EV ratio, the position difference between any two consecutive EVs is the integer part of the ratio plus zero or one; randomly decided based on the decimal part of the ratio. Once the vehicle category is decided throughout the entire sequence, the specific vehicle to be assigned is selected based on the above-described procedure. We note that this enhancement in the greedy heuristic may be applied for any restrictive feature that causes large variations in processing times.

Table 3.2: Illustration of greedy heuristic

Vehicle	Engine	p_1	p_2
A	Electric	15	4
B	Electric	16	3
C	Gasoline	2	10
D	Gasoline	3	8
E	Gasoline	2	9
F	Gasoline	4	7

To describe the greedy heuristic, consider an example with six vehicles and two stations. The processing times and engine types of vehicles are given in Table 3.2. The cycle time is 7 TU, and the length of the stations are 20 TU and 10 TU, respectively. The EV ratio is 1/3. We consider only EVs for the first position, vehicle A is designated to the first position since it causes less idle time than vehicle B. Next, none of the non-EVs causes work overload or idle time, so we assign the vehicle with the highest weighted sum of processing times to the second position, vehicle C. The procedure continues with another non-EV, and vehicle F is assigned to the third position because it is the only vehicle that does not cause any work overload. Consistent with the 1/3 EV ratio, an EV must be assigned to the fourth position, and vehicle B is assigned to this position as it is the only EV left. Vehicle E is assigned to the fifth position due to its higher weighted sum of processing times. Finally, vehicle D is assigned to the last position. The resulting sequence is *A-C-F-B-E-D* with a work overload of 3 TU, only at position 6 at station 2.

3.4.2.2 Tabu Search Algorithm

This section proposes a simulation-based local search algorithm on a very large neighborhood with tabu rules. The TS algorithm starts at the initial feasible solution (sequence) generated by the iterative greedy heuristic, and improves the initial solution via iterative improvements within the designed neighborhood. At each iteration of the TS, a transformation operator is randomly selected based on operator weights and applied to the incumbent solution to visit a random neighbor, respecting the tabu rules. The candidate solution is accepted if the objective function value is non-deteriorated, i.e., the candidate solution is rejected only if it has more total work overload. Then, another random operator is applied to the incumbent solution. This process repeats until the stopping criterion is met.

As aforementioned, the TS has two parts. The first part acts as the second step of the initial solution generation procedure since it improves the solution provided by the greedy heuristic for the one-scenario problem. In our preliminary numerical experiments we observed that this step can drastically improve the initial solution quality. Hence, we conduct this step for a duration τ_{one}. Next, the algorithm makes a transition to the full-information problem and reevaluates the objective function value of the incumbent solution—the sequence generated by the first part of TS. In the second part of the TS algorithm, the objective function value corresponding to the sequence is evaluated for the full-information problem. To do this, we calculate the total work overload for all realizations $\omega \in \Omega$, given the first-stage decision (sequence). That is, we calculate the objective function of (3.3) for each realization $\omega \in \Omega$ and take the weighted sum, each multiplied by the probability of the scenario. Observe from (3.3) that once the first-stage decision is fixed, the problem decomposes in scenarios and stations. Accordingly, the solution evaluation process is parallelized over scenarios and stations.

The TS algorithm continues evaluating the solution for the full-information problem for a duration τ_{full}. The allocated time for the second part, τ_{full}, is much larger than that of the first part, τ_{one}, since iterating over one-scenario problem is much faster than that over a set of realizations. In the reminder of this section, we explain various components of the TS algorithm.

Objective Evaluation The objective function of the problem for a given scenario is the same as the objective given in (3.3a), total work overload over all stations and positions. Evaluation of the objective, after each movement, is the bottleneck of our algorithm since the new total work

overload needs to be determined. Note that the objective evaluation starts at the first position and is executed iteratively since there is a sequence dependency. Accordingly, we propose to reduce the computational burden in two ways.

First, reevaluating the whole sequence is unnecessary since transformation operators make local changes in the sequence, i.e., some parts of the sequence remain unaltered and do not require reevaluation. Hence, we apply partial reevaluation after each movement. To explain partial reevaluation, assume that the vehicles at positions t_1 and t_2 are swapped. We certainly know that the subsequence corresponding to positions $[1, t_1 - 1]$ is not impacted by the swap operation; hence, we do not reevaluate these positions. Additionally, we may not have to reevaluate all the positions in $[t_1, t_2 - 1]$ and the positions in $[t_2, |T|]$. In each of these subsequences, there could be a *reset position* which ensures that there is no change in the objective from that position until the end of the subsequence. Since the rest of the subsequence after the reset position is not changed, we can jump to the end of the subsequence. To highlight how a partial reevaluation may improve the objective reevaluation process, suppose that the vehicles at positions 350 and 380 are swapped. We certainly know the subsequence corresponding to positions [1, 349] is not impacted by the swap. Additionally, in the case that there is a reset point before position 380 (and $|T|$), we do not have to reevaluate all the positions between 350 and 380, and the positions between 380 and $|T|$.

Second, we calculate the objective function in an accelerated way. Traditionally work overload and starting position for position t at station k, respectively w_{kt} and $z_{k(t+1)}$, are calculated as: $w_{kt} = z_{kt} + b_{kt} - l_k$ and $z_{k(t+1)} = z_{kt} + b_{kt} - w_{kt} - c$, where $w_{kt}, z_{kt} \geq 0$. Instead of calculating work overload and starting position vectors separately, we propose using a single vector to extract these information, which in fact is a different representation of the starting position vector z. If there is a work overload at position t, then $z_{k(t+1)} = l_k - c$. Otherwise, if there is not any work overload at position t, then $z_{k(t+1)} = z_{kt} + b_{kt} - c$, or we can equivalently write $z_{k(t+1)} = z_{k(t-1)} + b_{k(t-1)} - c + b_{kt} - c - w_{k(t-1)}$. Again, if there is a work overload at position $t-1$, then $z_{k(t+1)} = (l_k - c) + b_{kt} - c$, otherwise, if there is not any work overload at $t-1$, then $z_{k(t+1)} = z_{k(t-1)} + (b_{k(t-1)} - c) + (b_{kt} - c)$. Since we know that $z_{k0} = 0$, we can generalize it as $z_{k(t+1)} = \sum_{h=1}^{t}(b_{kh} - c)$, which is the cumulative sum of vector $\eta_k = b_k - c$ up to and including position t. However, this generalization has the assumption that there is not any work overload or idle time up to position t. We note that there is an idle time at position $t+1$ when $z_{kt} + b_{kt} - c < 0$. Accordingly, we can write a general formula as $z_{k(t+1)} = \max(0, \min(l_k - c, z_{kt} + \eta_{k(t+1)}))$, which

is referred to as the conditional cumulative sum of η_k up to position t. Intuitively, the conditional cumulative sum is defined as follows: starting from position 0, the cumulative sum is calculated iteratively within the closed range $[0, l_k - c]$. Whenever the cumulative sum exceeds the lower bound zero or the upper bound $l_k - c$, we set the cumulative sum to the corresponding bound's value. If the cumulative sum is below the lower bound, the excess value is equal to the idle time. Otherwise, if the cumulative sum is above the upper bound, the excess value is equal to the work overload. For example, if the cumulative sum is -2 at a position, the cumulative sum is set to zero and there is a 2 TU of idle time at that position.

In light of the proposed improvements, the partial reevaluation process is executed in two subsequences, $[t_1, t_2)$ and $[t_2, |T|]$, assuming that t_1 and t_2 are the two selected positions by any transformation operator and $t_1 < t_2$. The process starts at the first position, called position one, of the corresponding subsequence. We set $z_{k0} = \eta_{k1}$ and we calculate the starting position, work overload, and idle time for the positions in the subsequence iteratively as mentioned above. The reevaluation of the subsequence is completed when either a reset position is found or the whole subsequence is iterated. A reset position occurs at position t differently in two cases as follows: 1) if $z_{k,t+1} = 0$ when the processing time on the starting position t_1 (or t_2) is decreased, 2) if the sum of idle time and work overload up to position t in the current subsequence exceeds the total increased processing time at the corresponding starting position t_1 (or t_2) when the processing time on the starting position t_1 (or t_2) is increased.

Transformation Operators In this section, we explain the details of the transformation operators. We employ swap, forward and backward insertions, and inversion operators. The swap operator interchanges the positions of two randomly selected cars. Insertion removes a car from position i and inserts it at position j. Insertion is applied in two different directions, backward and forward. When $i > j$, the insertion is called a backward insertion, and all the vehicles between the positions j and i move one position to the right , i.e., scheduled later. On the contrary, forward insertion occurs when $i < j$ and all the vehicles between the positions i and j move one position to the left, i.e., scheduled earlier. Inversion takes two randomly selected positions in the sequence, and the subsequence between the selected positions is reversed. A repetitive application of these operators creates a very large neighborhood which helps the improvement procedure to escape local optima, especially when it is combined with a non-deteriorated solution acceptance procedure. The

latter enables the algorithm to move on the plateaus that consist of the solutions with the same objective function value (see Section 3.5.3.2 for numerical experiments).

Tabu List We design the tabu list in a non-traditional manner. This list includes the movements that induce undesired subsequences. Based on our observations, we define an undesired subsequence as back-to-back EVs because consecutive EVs cause a tremendous amount of work overload at the battery loading station. Accordingly, any movement that results in back-to-back EVs is a tabu. For the sake of clarity, we describe tabu movements in detail for each operator separately in C.

3.4.3 SAA Approach and Solution Quality Assessment

In (3.4), it is assumed that the probability of each scenario is known a priori, which may not hold in practice. In addition, the exponential growth of the number of scenarios causes an explosion in the size of the stochastic program. Hence, we utilize the SAA approach to tackle these issues. Consider the abstract formulation (3.4). The SAA method approximates the expected value function with an identically and independently distributed (i.i.d) random sample of N realizations of the random vector $\Omega_N := \{\omega_1, \ldots, \omega_N\} \subset \Omega$ as follows:

$$z_N = \min_{x \in X} \frac{1}{N} \sum_{\omega \in \Omega_N} Q(x, \xi_\omega). \tag{3.12}$$

The optimal value of (3.12), z_N, provides an estimate of the true optimal value [62]. Let $\hat{x}_N$ and x^* denote an optimal solution to the SAA problem (3.12) and the true stochastic program (3.4), respectively. Note that $\mathbb{E}[Q(\hat{x}_N, \xi_\omega)] - \mathbb{E}[Q(x^*, \xi_\omega)]$ is the optimality gap of solution $\hat{x}_N$, where $\mathbb{E}[Q(\hat{x}_N, \xi_\omega)]$ is the (true) expected cost of solution $\hat{x}_N$ and $\mathbb{E}[Q(x^*, \xi_\omega)]$ is the optimal value of the true problem (3.4). A high quality solution is implied by a small optimality gap. However, as x^* (and hence, the optimal value of the true problem) may not be known, one may obtain a statistical estimate of the optimality gap to assess the quality of the candidate solution $\hat{x}_N$ [53]. That is, given that $\mathbb{E}[z_N] \leq \mathbb{E}[Q(x^*, \xi_\omega)]$, we can obtain an upper bound on the optimality gap as $\mathbb{E}[Q(\hat{x}_N, \xi_\omega)] - \mathbb{E}[z_N]$. We employ the multiple replication procedure of Mak and Egger [71] in order to assess the quality of a candidate solution by estimating an upper bound on its optimality gap. A pseudo-code for this procedure is given in Algorithm 6. We utilize MRP, in Section 3.5, to assess the quality of solutions generated by different approaches.

Algorithm 6 Multiple Replication Procedure $\text{MRP}_\alpha(\hat{x})$

Input: Candidate solution $\hat{x}$, replication size M, and $\alpha \in (0,1)$.
Output: A normalized $\%100(1-\alpha)$ upper bound on the optimality gap of $\hat{x}$.
for $m = 1, 2, \ldots, M$. **do**
  Draw i.i.d. sample Ω_N^m of realizations ξ_ω^m, $\omega \in \Omega_N^m$.
  Obtain $z_N^m := \min_{x \in X} \frac{1}{N} \sum_{\omega \in \Omega_N^m} Q(x, \xi_\omega^m)$.
  Estimate the out-of-sample cost of $\hat{x}$ as $\hat{z}_N^m := \frac{1}{N} \sum_{\omega \in \Omega_N^m} Q(\hat{x}, \xi_\omega^m)$.
  Estimate the optimality gap of $\hat{x}$ as $G_N^m := \hat{z}_N^m - z_N^m$.
end for
Calculate the sample mean and sample variance of the gap as
  $\bar{G}_N = \frac{1}{M} \sum_{m=1}^M G_N^m$ and $s_G^2 = \frac{1}{M-1} \sum_{m=1}^M (G_N^m - \bar{G}_N)^2$.
Calculate a normalized $\%100(1-\alpha)$ upper bound on the optimality gap as
$\frac{1}{\bar{z}_N} \left(\bar{G}_N + t_{\alpha;M-1} \frac{s_G}{\sqrt{M}} \right)$, where $\bar{z}_N = \frac{1}{M} \sum_{m=1}^M z_N^m$.

In addition, we propose an MRP integrated SAA approach for candidate solution generation and quality assessment, given in Algorithm 7. On the one hand, a candidate solution is generated by solving an SAA problem with a sample of N realizations. Then, we use the MRP to estimate an upper bound on the optimality gap of the candidate solution. If the solution is ϵ-optimal, i.e., estimated upper bound on its optimality gap is less than or equal to a ϵ threshold, the algorithm stops. Otherwise, the sample size increases until a good quality solution is found. The algorithm returns a candidate solution and its optimality gap.

Algorithm 7 MRP integrated SAA

Input: List of sample sizes N_{list} and $\epsilon, \alpha \in (0,1)$.
Output: Solution $\hat{x}$ and OptGap.
for N in N_{list} **do**
  Obtain a candidate solution $\hat{x}_N$ by solving the SAA problem (3.12).
  Calculate a normalized $\%100(1-\alpha)$ upper bound on the optimality gap as $\text{MRP}_\alpha(\hat{x}_N)$.
  if $\text{MRP}_\alpha(\hat{x}_N) \leq \epsilon$ **then**
    $\hat{x} \leftarrow \hat{x}_N$ and OptGap $\leftarrow \text{MRP}_\alpha(\hat{x}_N)$.
    exit for loop
  end if
end for

We end this section by noting that each of the DEF, presented in Section 3.3.2, the L-shaped algorithm, presented in Section 3.4.1, and the heuristic algorithm, presented in Section 3.4.2, can be used to solve the SAA problem and obtain a candidate solution. However, the probability of scenarios $\omega \in \Omega$, ρ_ω, must change in the formulations so that it reflect the scenarios in a sample Ω_N. Let $\hat{N}$ and n_ω represent the set of unique scenarios in Ω_N and the number of their occurrences. Thus, in the described DEF, L-shaped algorithm, and TS algorithm, $\sum_{\omega \in \Omega} \rho_\omega(\cdot)$ changes to $\frac{1}{N} \sum_{\omega \in \Omega_N}(\cdot)$ or equivalently, $\frac{1}{N} \sum_{\omega \in \hat{N}} n_\omega(\cdot)$. Accordingly, in the L-shaped method, we generate one optimality cut for each unique scenario $\omega \in \hat{N}$ by solving $|\hat{N}|$ number of subproblems at each iteration.

3.5 Numerical Experiments

In Section 3.5.1, we describe the experimental setup. Then, in Section 3.5.2 and 3.5.3, we assess solution quality and computational performance of the proposed L-shaped and heuristic algorithms applied to a SAA problem, respectively.

3.5.1 Experimental Setup

We generated real-world inspired instances from our automobile manufacturer partner's assembly line and planning information. As given in Table 3.3, we generated three types of instances: (1) small-sized instances with 7-10 vehicles to assess the performance of L-shaped algorithm, (2) medium-sized instances with 40 vehicles to assess the performance of the TS algorithm for the one-scenario problem, (3) large-sized instances with 200, 300, and 400 vehicles to evaluate the performance of the TS algorithm. All instances have five stations, of which the first one is selected as the most restrictive station for EVs, the battery loading station. The rest are selected among other critical stations that conflict with the battery loading station.

Table 3.3: Data sets

Instance Type	$\|V\|$	$\|K\|$	Number of Instances
Small	7, 8, 9, 10	5	30×4
Medium	40	5	30×1
Large	200, 300, 400	5	30×3

The cycle time c is 97 TU, and the station length l is 120 TU for all but the battery loading station, which is two station lengths, 240 TU. The information about the distribution of the processing times is given in Table 3.4. It can be observed that the average and maximum processing times for each station are lower than the cycle time and the station length, respectively. Moreover, the ratio of the EVs is in the range of [0.25, 0.33] across all instances.

Table 3.4: Processing times distribution

	Time (s)		
Station ID	Min	Mean	Max
1	42.6	94.1	117.2
2	7.9	84.3	197.9
3	57.8	96.2	113.3
4	26.9	96.9	109.7
5	57.8	96.2	114.3

We derived the failure rates from six months of historical data by performing predictive feature analysis on vehicles. Based on the analysis, two groups of vehicles are formed according

to their failure probabilities, low-risk and high-risk vehicles, whose failure probabilities are in the range of [0.0, 0.01] and [0.2, 0.35], respectively. The failure probability is mostly higher for recently introduced features, e.g., the average failure probability of EVs is 50% higher than that of other vehicles. High-risk vehicles constitute [0.03, 0.05] of all vehicles. However, this percentage increases to [0.15, 0.25] for the small-sized instances in order to have a higher rate of failed vehicles. We note that the failures are not considered for the medium-sized instances since these instances are used for only the one-scenario problem, which does not involve failures by definition.

The number of failure scenarios, $2^{|V|}$, increases exponentially in the number of vehicles. Thus, we generated an i.i.d random sample of N realizations of the failure scenarios; hence, formed a SAA problem. For each failure scenario and vehicle, we first chose whether the vehicle was high risk or low risk (based on their prevalence). Then, depending on being a high-risk or low-risk vehicle, a failure probability was randomly selected from the respective range. Finally, it was determined whether the vehicle failed or not. In order to have a more representative sample of scenarios for large-sized instances, no low-risk vehicle was allowed to fail at any scenario.

For each parameter configuration, we generated 30 instances. The vehicles of each instance were randomly selected from a production day, respecting the ratios mentioned above. The algorithms were implemented in Python 3. For solving optimization problems we used Gurobi 9.0. The time limit is 600 seconds for all experiments unless otherwise stated. We run our experiments on computing nodes of the Clemson University supercomputer. The experiments with the exact solution approach were run on nodes with a single core and 15 GB of memory, and the experiments with the heuristic solution approach were on on nodes with 16 cores and 125 GB of memory.

3.5.2 Exact Solution Approach

In this section, we present results for the solution quality and computational performance of the L-shaped algorithm. We used MRP scheme, explained in Section 3.4.3, to assess the solution quality. We also compared the computational performance of the L-shaped algorithm with that of solving the DEF. We present the results for 120 small-sized instances consisting of 7 to 10 vehicles. We do not present the results for large-sized instances as our preliminary experiments showed that the number of instances that could be solved to optimality decreases drastically.

We also point that instead of solving a relaxed master problem to optimality at each iteration of the L-shaped algorithm, one can aim for just obtaining a feasible solution $\hat{x} \in X$. This may result

in saving a significant amount of computational time that would be otherwise spent on exploring solutions that are already eliminated in previous iterations. This kind of implementation, referred to as *branch-and-Benders-cut* (B&BC), is studied in the literature, see, e.g., [54, 109, 27]. In our implementation of our proposed L-shaped algorithm, we used Gurobi's Lazy constraint callback to generate cuts at a feasible integer solution found in the course of the branch-and-bound algorithm.

3.5.2.1 Solution Quality

Figure 3.4 shows the impact of sample size on the solution quality of the SAA problem. Observe that the improvement in the upper bound of the optimality gap, the MRP output, as the sample size increases from 100 to 1000 progressively. We set the number of replications M to 30 and $\alpha = 0.05$ (95% confidence interval). While the mean of the optimality gap decreases gradually from 0.76% to 0.12%, a drastic enhancement is observed with the variance. We have 36 out of 120 solutions with an optimality gap of larger than 1% when the sample size is 100. However, all of the obtained solutions have less than a 1% optimality gap when the sample size is 1000. It can be seen in the figure that good solutions can be obtained with a sample size of 100, yet it is not assured due to the high variance of the approximation. Consequently, the results suggest that the sample size should be increased until the variance of objective estimation is small enough.

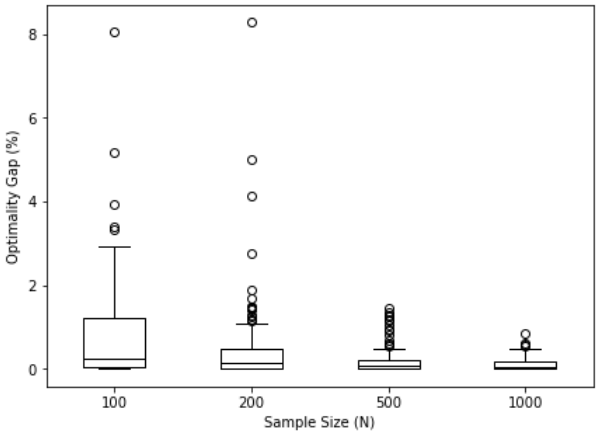


Figure 3.4: Solution quality of the SAA problem based on sample sizes

Based on the results in Figure 3.4, we implemented the MRP integrated SAA scheme, presented in Section 3.4.3 and Algorithm 7, to balance the required computational effort and solution quality. We set $\alpha = 0.05$ (95% confidence interval) and $\epsilon = 0.01$ in MRP. While it is ensured that

we obtain solutions within a 1% optimality gap, most of the solutions are found with the least computational effort, e.g., at the first iteration with a sample size of 100. In Table 3.5, we provide key performance results to show the performance of the MRP integrated SAA scheme, where the number of replications M is 30 and the MRP sample size N is 5000. The average value for the optimal value, accepted candidate solution's expected objective value, and optimality gap are presented in Table 3.5. The sample size of the accepted candidate solutions is 84, 20, 11, and 5 for the sample sizes $N_{list} = \{100, 200, 500, 1000\}$, respectively. The average optimality gap is 0.2% which shows that SAA can produce high-quality solutions.

Table 3.5: Solution quality of the MRP integrated SAA

Statistical Lower Bound $\mathbb{E}[z_N]$	Estimated Objective Value $\mathbb{E}[Q(\hat{x}_N, \xi_\omega)]$	Optimality Gap $\bar{G}_N$
55.80	55.91	0.11 (0.2%)

Additionally, we assess the solution quality of the one-scenario problem (i.e., the deterministic MMS problem without any car failure). We first optimally solved the one-scenario problem for each instance by using Gurobi. Then, we assessed the quality of the obtained solutions by utilizing MRP. Observe from Figure 3.5 that the average optimality gap is 23%, the maximum optimality gap is 274%, and the standard deviation is 39%. Comparing the performance of the SAA and the one-scenario problems shows that we can generate robust solutions by considering vehicle failures, which helps reduce work overloads by more than 20%.

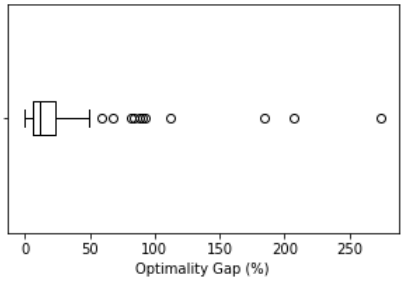


Figure 3.5: Solution quality of the one-scenario problem

3.5.2.2 Computational Performance

In this section, we conduct different experiments to compare the DEF and L-shaped algorithm. On the one hand, we assess the impact of using the *improved* model, described in Section 3.3.2, obtained by setting the processing time of failed vehicles to the cycle time, and compare the

results with those obtained using the standard model. The DEF corresponding to the standard and improved models are denoted as D_{std} and D_{imp}, respectively. Similarly, the L-shaped algorithm corresponding to the standard and improved are denoted as L_{std} and L_{imp}. On the other hand, we assess the impact of our proposed cut selection strategy, described in Section 3.4.1, obtained using ℓ_1 norm regularization to find a cut with the least number of zero coefficients. We used the cut selection strategy for the improved model, and denote the corresponding L-shaped algorithm as L_{imp-cs}.

In Table 3.6, we present the results on the impact of the improved model and cut selection strategy to compare the DEF and L-shaped algorithm for solving the SAA problem. We report the average and standard deviation of the computational time (in seconds), labelled as μ_t and σ_t, respectively, and the optimality gap, labelled as *Gap* (in percentage). Additionally, the number of instances that could not be solved optimally within the time limit is given in parenthesis under the *Gap* columns. All time results are the average of instances (out of 30 instances) that could be solved optimally within the time limit, while the *Gap* results are the average of the instances that could not be solved optimally. Based on the results in Section 3.5.2.1, we conducted the computational experiments on the SAA problem with sample sizes 100, 200, 500, and 1000.

Table 3.6: Computational performance of the DEF and L-shaped algorithms for the SAA problem of small-sized instances

		D_{std}			D_{imp}			L_{std}			L_{imp}			L_{imp-cs}		
$\|V\|$	N	μ_t (s)	σ_t (s)	Gap (%)	μ_t (s)	σ_t (s)	Gap (%)	μ_t (s)	σ_t (s)	Gap (%)	μ_t (s)	σ_t (s)	Gap (%)	μ_t (s)	σ_t (s)	Gap (%)
7	100	6.3	3.8	-	1.4	1.1	-	4.9	3.2	-	1.2	0.5	-	1.1	0.5	-
	200	10.2	6.4	-	2.0	1.1	-	7.9	5.7	-	1.9	0.9	-	1.6	0.6	-
	500	15.0	8.8	-	3.2	1.8	-	10.2	6.5	-	2.4	1.0	-	2.1	0.8	-
	1000	19.5	11.2	-	4.4	2.5	-	11.9	8.0	-	2.8	1.1	-	2.4	0.9	-
8	100	29.3	19.7	-	11.4	10.5	-	51.7	59.4	-	9.4	6.6	-	5.2	4.4	-
	200	50.5	31.5	-	19.1	16.5	-	83.2	83.4	-	16.7	22.9	-	8.1	6.9	-
	500	88.9	53.3	-	32.3	24.9	-	145.0	146.2	-	24.9	26.4	-	13.4	10.9	-
	1000	127.9	81.9	-	44.3	35.9	-	159.1	150.8	(1) 0.31	30.7	26.5	-	15.6	14.2	-
9	100	170.3	157.6	(2) 0.33	35.3	27.3	-	187.5	130.3	(9) 0.54	43.1	39.0	-	25.0	19.1	-
	200	238.9	165.1	(7) 0.34	68.4	53.1	-	315.6	170.2	(14) 0.54	80.0	96.4	-	41.1	49.4	-
	500	263.6	140.3	(12) 0.38	122.6	120.3	(1) 0.20	357.0	170.0	(17) 0.55	105.0	79.9	(1) 0.16	58.6	49.1	-
	1000	317.7	140.4	(16) 0.44	204.2	153.5	(1) 0.13	366.3	198.4	(22) 0.55	155.6	100.9	(1) 0.07	87.1	66.5	-
10	100	279.1	159.7	(12) 0.28	91.8	61.7	-	486.5	253.8	(25) 0.61	169.3	137.1	(3) 0.16	133.7	152.2	-
	200	258.5	161.2	(23) 0.34	160.9	117.9	-	344.2	361.7	(28) 0.61	238.5	179.6	(4) 0.18	158.8	138.3	(2) 0.16
	500	565.2	58.4	(26) 0.48	245.3	151.4	(6) 0.13	283.9	0.0	(29) 0.69	264.4	193.9	(12) 0.18	223.5	170.9	(7) 0.16
	1000	479.0	89.4	(28) 0.53	294.1	149.2	(13) 0.18	191.0	0.0	(29) 0.72	266.0	170.8	(18) 0.26	273.6	189.6	(8) 0.19

Observe from Table 3.6 that using the improved model instead of the standard model decreased the computational time and optimality gap of both the DEF and L-shaped algorithm drastically. In particular, the solution time decreased for all instances. On average (over different $|V|$ and N), we observe a 67% and 70% decrease for the DEF and L-shaped algorithm, respectively. Additionally, the decrease in the standard deviation is around 64% and 74% for the DEF and L-shaped algorithm, respectively, when non-optimal solutions are left out. Moreover, the number of instances

that could not be solved optimally is reduced by using the improved model: on average, 83% and 78% of those instances are solved optimally with D_{imp} and L_{imp}, respectively. Additionally, the remaining non-optimal solutions are enhanced as a reduction in optimality gaps is achieved.

Another drastic improvement is provided by our cut selection strategy. Comparing L_{imp} and L_{imp-cs} in Table 3.6 shows that the mean and standard deviation of the computational time, and optimality gap are reduced by 35%, 33%, and 18%, on average. Furthermore, an optimal solution is found by L_{imp-cs} for 56% of the instances that could not be solved optimally within the time limit by L_{imp}.

Finally, we compare D_{imp} and L_{imp-cs}. Observe from Table 3.6 that L_{imp-cs} resulted in lower mean computational time by 31%, 59%, 45%, and 1% for instances with 7, 8, 9, and 10 vehicles, respectively, and by 15%, 28%, 38%, and 45% for instances with 100, 200, 500, 1000 scenarios, respectively. In the same order, the variance decreased by 56%, 58%, 38%, and -52% for instances with 7,8,9, and 10 vehicles, respectively, and by -1%, 18%, 41%, and 42% for instances with 100, 200, 500, 1000 scenarios, respectively. We conclude that our L-shaped algorithm outperforms the DEF for instances with up to 9 vehicles, and they provide comparable results for instances with 10 vehicles. Additionally, the superiority of the L-shaped algorithm over the DEF escalates as the number of scenarios increases.

3.5.3 Heuristic Solution Approach

In this section, we present results for the solution quality and computational performance of the TS algorithm. The solution quality is evaluated by employing the MRP scheme, explained in Section 3.4.3. We also assess the computational performance of the TS algorithm in various aspects.

We set the operator selection probabilities (weights) based on our preliminary experiments. The weights of swap, forward insertion, backward insertion, and inversion are 0.45, 0.10, 0.15, 0.30, respectively. We set the time limit for the one-scenario problem to 10 seconds, i.e., $\tau_{one} = 10$ seconds, which leaves $\tau_{full} = 590$ seconds. Finally, based on the results of the quality assessment, we set the sample size N to 1000 for the computational performance assessments.

3.5.3.1 Solution Quality

We solved the SAA problem of large-sized instances using the TS algorithm. To assess the solution quality, we then used the proposed MRP integrated SAA scheme, given in Algorithm 7, with

the replication $M = 30$, MRP sample size $N = 20000$, $\alpha = 0.05$ (95% confidence interval), $\epsilon = 0.05$, and the list of sample sizes $N_{list} = \{1000, 2500, 5000\}$. Table 3.7 reports the key performance result to show the performance of the MRP integrated SAA scheme. While the maximum optimality gap is 3.7%, the average optimality gap is 0.28% which indicates that solving the SAA problem with the proposed TS algorithm can produce high-quality solutions. Figure 3.6 further shows that the optimality gap for most of the solutions is less than 1% with only five outliers out of 90 instances.

Table 3.7: Solution quality of the MRP integrated SAA

Statistical Lower Bound $\mathbb{E}[z_N]$	Estimated Objective Value $\mathbb{E}[Q(\hat{x}_N, \xi_\omega)]$	Optimality Gap $\bar{G}_N$
239.59	240.26	0.67 (0.28%)

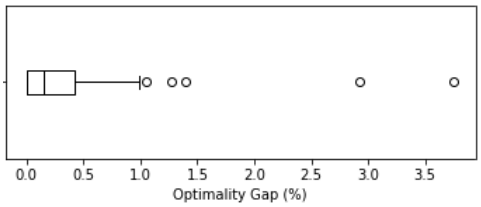


Figure 3.6: Solution quality of the SAA problem

Moreover, we assess the solution quality of the one-scenario problem over large-sized instances by using the same procedure in order to show its robustness. Observe from Figure 3.7 that the average optimality gap is 24.8%, the maximum optimality gap is 76.5%, and the standard deviation is 10.8%. Comparing the performance of the SAA and one-scenario problems demonstrates that we can generate robust solutions by considering vehicle failures. Accordingly, we reassure with the industry-sized instances that we can reduce the work overloads by more than 20% by considering stochastic car failures and solving the corresponding problem efficiently.

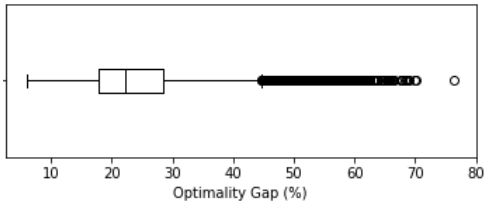


Figure 3.7: Solution quality of one-scenario problem

3.5.3.2 Computational Performance

To assess the computational performance of the TS algorithm, we conducted different tests: 1) compared the solution found with the TS algorithm with the solution found by an off-the-shelf solver for the one-scenario problem of medium-sized instances, 2) compared the solution found with the TS algorithm with the optimal solution found by L_{imp-cs} approach for the SAA problem of small-sized instances, 3) compared the solution found with the TS algorithm with that of a simulated annealing (SA) algorithm for the SAA problem of large-sized instances, 4) and analyzed the convergence of TS algorithm for the one-scenario and SAA problems of large-sized instance. We executed 30 runs for each of the instances and tests.

Table 3.8 reports the results for the first set of computational experiments for the one-scenario problem. We note that we generated 30 instances that could be solved within three hours time limit with Gurobi (solving the DEF). The minimum, average, maximum, and standard deviation of the computational time (in seconds) are shown for the TS algorithm and Gurobi. Additionally, the average number of movements for the TS algorithm is reported in order to provide some insight about the efficiency of the implementation. The optimal solutions, for all 30 instances and for all 30 runs, are found in under 10 seconds by the TS algorithm. The average computational times are 1140 and 0.33 seconds for the Gurobi solver and TS algorithm, respectively. These results show that the proposed TS algorithm can consistently provide optimal solutions to the one-scenario problem in a very short amount of time by avoiding any local optima.

Table 3.8: Computational performance of Gurobi and TS for the one-scenario problem of medium-sized instances

	Time (s)				Move (#)
Method	Min	Mean	Max	Std. Dev.	Mean
Gurobi	2.37	1140.91	9373.08	2613.95	-
TS	6e-4	0.33	9.44	0.81	16940

Table 3.9: Computational performance of Gurobi and TS for the SAA problem of small-sized instances

	Time (s)				Optimality Gap (%)			Optimally Solved (%)	Move (#)
Method	Min	Mean	Max	Std. Dev.	Mean	Max	Std. Dev.		Mean
L_{imp-cs}	5.02	68.91	210.06	50.99	0	0	0	100	-
TS	1e-4	0.08	0.28	0.08	0.14	2.79	0.41	83	628

Recall that as demonstrated in Section 3.5.2.2, L_{imp-cs} outperformed Gurobi in solving the SAA problem for small-sized instances. In the second set of experiments, we compared the computational effectiveness of solving the SAA problem of small-sized instances with the TS algorithm and

L_{imp-cs}. To this end, we chose a sample size of 1000 and solved 30 small-sized instances optimally by using L_{imp-cs}. We then solved the same set of instances using the TS algorithm until either an optimal solution was obtained or the time limit was reached. Table 3.9 shows that TS algorithm was able to find optimal solutions in 83% of the experiments, 746 out of 900. However, the average optimality gap for these non-optimal solutions is 0.14%, indicating that the TS algorithm is reliable in terms of optimality. For the TS algorithm, we recorded the computational time as the time until the last improvement is done, so we can see that the TS algorithm is very efficient, with an average runtime of 0.08 seconds.

We observed that the TS algorithm terminated at a local optima often when the number of vehicles is very small. Our hypothesis is that there are plateaus with the same objective value when the number of vehicles is large. However, this is not the case when there are only 10 vehicles as there are a limited number of sequences and each sequence has a different objective function value. Hence, in the third set of experiments, we compared the TS algorithm and a SA algorithm for large-sized instances to analyze the impact of the tabu list and accepting worse solutions on escaping a local optima. We utilized the same local search algorithm for the SA with two differences: 1) disabled the tabu rules and 2) enabled accepting worse solutions based on the acceptance criterion. For the SA algorithm, we set the starting temperature $T_{init} = 10$. We also adopted a geometric cooling with the cooling constant $\alpha = 0.999$. The acceptance ratio is calculated using the Boltzmann distribution $P(\Delta f, T) = e^{-\frac{f(x')-f(x)}{T}}$, where x' is a new solution, x is the incumbent solution, and T is the current temperature.

We note that the Kruskal-Wallis test shows no significant difference between the computational performance of the TS and SA algorithms at a 95% confidence level. However, as illustrated in Figure 3.8, the TS algorithm produces better results and converges faster than the SA algorithm (averaged over of 900 runs). This result shows that the proposed TS algorithm is capable of exploiting the search space while generally avoiding premature convergence to local optima. Accordingly, we conclude that there is no need to accept worse solutions in the local search.

Finally, in the last set of experiments, we conducted an analysis in order to provide insight into the reliability of the proposed TS algorithm's convergence. In particular, in Figure 3.9, we presented the box plots of the standard deviation of the objective values for the one-scenario and SAA problems of large-sized instances. Each data point represents the standard deviation of 30 runs (for each of the 90 large-sized instances). The average standard deviations for the one-scenario and

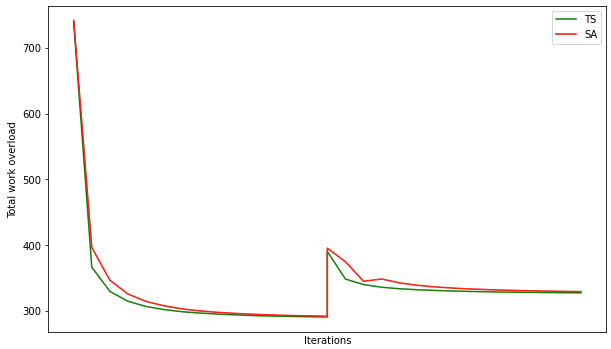


Figure 3.8: Convergence comparison of the TS and SA algorithms

SAA problems are 0.19 and 0.93, while the means of the objective values are 212.77 and 239.69, respectively. Accordingly, the average coefficient of variations, the ratio between the average standard deviation and mean, are 9e-4 and 4e-3, which indicates that the proposed TS algorithm provides highly reliable results for both of the problems.

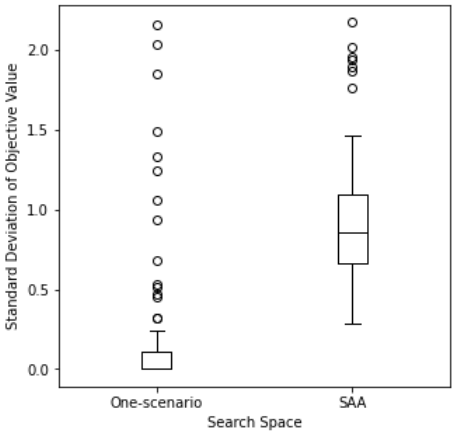


Figure 3.9: Convergence of the objective value with TS algorithm for the one-scenario and SAA problems

3.6 Conclusion

This chapter studied mixed-model sequencing (MMS) problem with stochastic failures. To the best of our knowledge, this is the first study that considers stochastic failures of products in MMS. The products (vehicles) fail according to various characteristics and are then removed from the sequence, moving succeeding vehicles forward to close the gap. Vehicle failure may cause extra work overloads that could be prevented by generating a robust sequence at the beginning. Accordingly,

we formulated the problem as a two-stage stochastic program, and improvements were presented for the second-stage problem. We employed the sample average approximation approach to tackle the exponential number of scenarios. We developed L-shaped decomposition-based algorithms to solve small-sized instances. The numerical experiments showed that the L-shaped algorithm outperforms the deterministic equivalent formulation, solved with an off-the-shelf solver, in terms of both solution quality and computational time. To solve industry-sized instances efficiently, we developed a greedy heuristic and a tabu search algorithm that is accelerated with problem-specific tabu rules. Numerical results showed that we can provide good quality solutions, with less than a 5% statistical optimality gap, to industry-sized instances in under ten minutes. The numerical experiments also indicated that we can generate good quality robust solutions by utilizing a sample of scenarios. In particular, we can reduce the work overload by more than 20%, for both small- and large-sized instances, by considering possible car failures.

3.6.1 Managerial Insights

Car manufacturers are facing several challenges due to the increasing ratio of EVs in production. EVs have significant differences compared to non-EVs which require specific treatment when creating the sequence for the mixed-model assembly line. One of the main challenges is the battery installation process. Unlike traditional vehicles, EVs have large and heavy batteries that need to be installed in a specific order to avoid damaging the vehicle or the battery itself. Accordingly, a huge processing time variation at the battery loading station arises from this difference, which requires special treatment to ensure that the assembly line can continue to produce high-quality vehicles efficiently.

We have observed that consecutive EVs induce a significant amount of work overload, which generally requires line stoppage even with the help of utility workers. Planning the sequence by ruling out back-to-back EVs does not guarantee that there will not be any occurrence of consecutive EVs. The failure of vehicles disrupts the planned sequence, and the necessity of considering car failures during the planning process increases as the difference between product types expands. Accordingly, in this chapter, we focused on generating robust schedules that take into account the possible deleterious effects resulting from the divergence between electric and non-electric vehicles. However, it is worth noting that our proposed solution methodologies are equally applicable to any feature that causes similar variations at specific work stations.

As aforementioned, work overload is handled by either stopping the line or employing utility workers. Thus, reducing work overload by more than 20% results in drastic production efficiency and cost-saving improvements in the assembly line since there will be fewer line stoppages and a reduction in the number of required utility workers.

Chapter 4

Mixed-model Sequencing with Reinsertion of Failed Vehicles: A Case Study for Automobile Industry

4.1 Introduction

An assembly line is a manufacturing system in which a product is progressively built by moving it through a sequence of workstations, each specializing in a specific task. Mixed-model assembly lines (MMAL) are developed to produce multiple product variations or models on the same assembly line, allowing for flexibility and customization while maximizing efficiency and productivity. The variety of models produced significantly increases as the product becomes more complex and customizable. An automobile is a useful example of such a product, e.g., a German car manufacturer has up to 10^{24} potential configurations for their vehicles [84]. Some products may require additional steps, specialized equipment, or longer assembly processes due to their unique attributes, resulting in substantial variations in processing times. The tasks are distributed through the assembly line balancing, ensuring that the average workload of each station aligns with the established *cycle time*,

the fixed duration between two consecutive product launches. However, a poor sequence of products may result in uneven workloads across workstations, excessive work allocation (*work overload*), or idle time, due to the varying processing times of products. When such a work overload situation occurs, interventions are needed such as stopping the line or utilizing more workers. In order to avoid or minimize such interventions, the assembly process is balanced by sequencing the products. The corresponding sequencing problem is called as a mixed-model sequencing (MMS) problem which minimizes the work overload duration across all workstations.

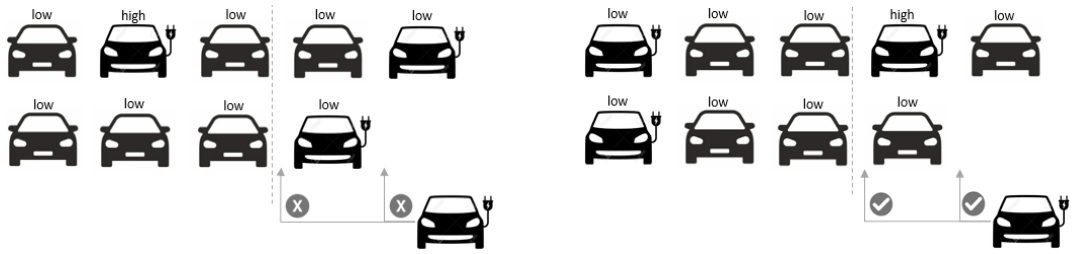
In the automobile industry, car manufacturers have adjusted their assembly lines to accommodate the mixed-model production of vehicles with diesel and gasoline engines, however, the starting to produce electric vehicles (EV) (or hybrid vehicles) on the same line has posed new challenges. In contrast to other vehicles, EVs have large batteries, resulting in substantial differences in tasks, particularly at the station where the battery is loaded. Accordingly, the position of EVs in a sequence needs to be carefully determined since too many EVs in a subsequence may result in a significant work overload at the corresponding stations. To illustrate further, consider an MMAL where a station is responsible for an electric battery system. The processing time for installing a battery system on an EV might take, on average, three minutes, while tasks on a non-EV might take around 30 seconds. Without proper sequencing, the battery loading station could experience work overloads and idle times due to the drastic processing time difference. By solving the MMS problem for each planning horizon, the workload can be distributed in a way that accommodates the varying processing times, ensuring efficient utilization of resources and maximizing production output.

As car manufacturers increase the proportion of the EVs in the product mix in response to market demand, the issues that cause a vehicle not to be produced in the planned sequence become more significant. For instance, if a part is delayed from a supplier, or if there is a paint issue with a vehicle, a planned sequence of vehicles may have a missing vehicle, which is referred to as *failed vehicles*. Even if this missing vehicle has a gasoline or diesel engine, its absence can still affect the stations that require intense battery-related tasks. The failed vehicles are pulled out of the sequence and the resulting gap is closed either by moving the succeeding vehicles forward or by filling with another vehicle that failed previously and waiting to be reinserted, we call these vehicles, failed but ready for reinsertion, as *reinstating vehicles* throughout this chapter. In a traditional MMS, the failure of vehicles is not considered which means that the failure of vehicles may result in a

significant additional work overload.

Moreover, in automobile manufacturing facilities, the reinsertion process is executed dynamically. Ideally, a failed vehicle that is ready for reinsertion is reinserted into the sequence only if an appropriate position is found, i.e., reinsertion is executed only if it does not cause an extra work overload. The vehicles to be reinserted can accumulate if appropriate positions do not occur enough. Once the number of reinstating vehicles exceeds a threshold, the line is stopped and all the reinstating vehicles are produced back to back which generally impacts production efficiency. As a result, we propose integrating the possibility of reinsertion into the sequencing model in order to minimize the possible work overload due to vehicle failures and reinsertions and to avoid line stoppages due to the excessive number of reinstating vehicles.

In Figure 4.1, we motivate a robust sequence that considers vehicle failures and the reinsertion process of failed vehicles for a battery installation station where back-to-back electric vehicles cause a large amount of work overload. In the example, there are five vehicles to be sequenced, three non-EVs and two EVs, and one of the EVs has a high failure probability. There is one failed vehicle that will be ready for reinsertion starting from position 3. Assuming the high-risk vehicle fails, different consequences occur based on the initial sequence. In the non-robust sequence, the reinsertion of the reinstating vehicle results in work overload for either of the possible positions. On the other hand, in the robust sequence, the reinsertion to any possible position does not cause any work overload.



(a) non-robust sequence **(b)** robust sequence

Figure 4.1: Illustration of a non-robust and robust sequence to stochastic failures and reinsertion process

This chapter aims to generate robust sequences that take into account the potential vehicle failures and their reinsertion back into the sequence, in order to avoid additional workloads. We focus on the final assembly line, where we assume that vehicles must follow the original sequence they arrive in from the paint shop, without the option of resequencing at the point where the vehicle

failures are realized. Accordingly, when a vehicle is taken out of the sequence, the subsequent vehicles fill the resulting gap. The key contributions of this chapter are as follows:

- A bi-objective two-stage stochastic program is presented for an MMS problem with stochastic product failures and reinsertion process, and formulation improvements are provided. This is the first study that considers the product reinsertion process in MMS, to the best of our knowledge.
- We develop a bi-objective evolutionary algorithm, a two-stage bi-objective local search algorithm, and a local search integrated bi-objective evolutionary algorithm to tackle the proposed problem. The numerical results show that the local search integrated evolutionary algorithm outperforms other algorithms by providing work reliable solutions in terms of work overload objective. On the other hand, the hybrid local algorithm provides a better exploration of solution space
- A case study with the data inspired by our car manufacturer industrial partner is conducted via dynamic reinsertion simulations. The numerical experiments show that the work overload can be reduced by around 20% by considering stochastic car failures and the reinsertion process, while significantly decreasing the waiting time of failed vehicles until their reinsertion.

The remainder of this chapter is structured as follows. The related literature is reviewed in Section 4.2. The proposed problem is defined, and mathematically formulated in Section 4.3. Efficient solution approaches to tackle the problem are presented in Section 4.4. The numerical experiments that analyze the performance of presented solution approaches are executed, and the results are shared in Section 4.5. Finally, the chapter is concluded with the highlights and a discussion over a managerial point of view in Section 4.6.

4.2 Related Literature

The efficiency of an MMAL primarily relies on the management of two key decision problems [9]. As a long-term decision problem, line balancing has a strategic role, as it involves allocating the workload across the workstations in the assembly line [13, 17, 99, 82, 116]. As a short-term decision problem, the sequencing problem has an operational role, as it addresses daily production planning. Combining mixed-model assembly line balancing and sequencing problems has garnered

significant interest among researchers since considering strategic and operational decisions together provides benefits to the line efficiency [78, 69, 83]. For the sequencing problem, researchers have developed two main categories of objective functions: those related to work overload and those focused on just-in-time objectives. Additionally, the literature presents three solution approaches: level-scheduling [104], car-sequencing [31, 102, 117], and MMS. A comprehensive survey, including classification schemes and detailed explanation, is given by Boysen *et al.* [18].

The problem proposed in this chapter can be considered a combined mixed-model sequencing and resequencing problem. The resequencing problem minimizes the work overload while reinserting a set of failed vehicles into a given sequence [20]. There are several studies in the literature that tackle the resequencing problem in the automobile industry. Franz *et al.* consider a resequencing setting in the automotive industry [40]. They compare the performance of three local search heuristic features approaches while minimizing the total work overload occurrences. A variable neighborhood search combined with tabu search outperforms other feature combinations. Franz *et al.* consider the resequencing problem in a dynamic setting, dynamically supplied failed vehicles, and compares two reinsertion strategies integrated into local search heuristics; reinsert the vehicle as soon as it is ready and wait for a good reinsertion position [41]. They show that both strategies perform better, compared to real-world application that does not utilize such strategy. Gujjula and Gunther consider the resequencing problem in the level-scheduling setting [50]. Gunay and Kula propose a two-stage stochastic model for a resequencing problem that considers the post-paint restoring buffer, which is used to restore the initial sequence before the paint quality issues [51]. Leng *et al.* propose a multi-objective reinforcement learning approach to solve combined resequencing and color-batching problems [64].

While most of the existing literature on MMS primarily focuses on models that have deterministic parameters, there are a limited number of studies that examine stochastic parameters, only related to processing times or demand. Zhao *et al.* introduces a Markov chain-based approach that minimizes the expected total work overload duration, by generating sub-intervals that represent possible positions of workers within the stations based on the stochastic processing times [121]. Mosadegh *et al.* proposes a heuristic approach inspired by Dijkstra's algorithm to address a single-station MMS with stochastic processing times, formulated as a shortest path problem [76]. Mosadegh *et al.* formulates a multiple-station MMS with stochastic processing times as a mixed-integer linear programming (MILP) problem [77]. They introduce a Q-learning-based simulated

annealing heuristic to solve real-world-sized problems and demonstrate a decrease in expected work overload compared to the deterministic problem. Brammer *et al.* suggests a reinforcement learning approach for solving MMS with stochastic processing times [21]. Their results indicate that the proposed method outperforms SA and GA by at least 7% in terms of solution quality. Furthermore, stochastic parameters are also considered in integrated mixed-model balancing and sequencing problems. Specifically, stochastic processing times are studied in [1, 79, 32] and stochastic demand is studied in [98].

To the best of our knowledge, there is currently no existing research that focuses on establishing robust sequences that handle work overloads resulting from product failures by considering the reinsertion process within any sequencing structure. The only studies that consider the stochastic product failure, but not the reinsertion of failed products, are given by Hottenrott *et al.* within car sequencing structure [55], and Yilmazlar *et al.* within MMS structure [118].

4.3 Problem Statement and Mathematical Formulation

In Section 4.3.1, we define the MMS with reinsertion and illustrate the problem with an example. Then, in Section 4.3.2, we provide a mixed-integer quadratic program for the proposed problem.

4.3.1 Problem Statement

In an MMAL (Mixed-Model Assembly Line), a conveyor belt is used to interconnect a group of workstations. Products are launched at a fixed rate, and the duration between two consecutive products is referred to as the cycle time c. The products move along the belt with a constant speed of one time unit (TU). We assume that the length of station $k \in K$ is $l_k \geq c$ in TU, which represents the total time required for a workpiece to traverse the station. Operators are responsible for carrying out assigned tasks within the station length, i.e., closed-border stations. If they fail to complete their tasks within the allocated time, a utility worker intervenes to complete the remaining work. The additional workload that remains unfinished is known as *work overload*. The efficiency of the assembly line is greatly impacted by the sequence of products. Therefore, MMS determines the sequence of a given set of products V by assigning each product $v \in V$ to a specific position $t \in T$.

In this chapter, we consider that each vehicle is unique since our automobile manufacturer

Table 4.1: List of parameters and decision variables used in the model

Sets and Index	
V, v	Vehicles that are initially planned to be produced in the current horizon
F_{new}, i	Failed vehicles that were planned to be produced in the current horizon
F_{old}, i	Failed vehicles that were planned to be produced in a previous horizon
K, k	Stations
T, t	Positions
Ω, ω	Scenarios
Parameters	
p_{kv}	The processing time of vehicle v at station k
l_k	The length of station k
c	The cycle time
f_v	The failure probability of vehicle v
e_{vn}	1 if vehicle v exists at scenario n, 0 otherwise
f_{max}	The maximum number of failed cars allowed at the end of the planning horizon
g_i	The number of days between the planned and current period for the failed car $i \in F_{old}$
d_i	The number of extra days until the delivery date that the failed car $i \in F_{old}$ had when failed
λ	The minimum number of positions between each reinsertion
r_i	The number of positions required for failed vehicle $i \in F$ to be ready for reinsertion
First-Stage Decision Variables	
x_{vt}	1 if vehicle $v \in V$ is assigned to position $t \in T$, 0 otherwise
δ_{vt}	1 if vehicle $v \in V$ is positioned after position t before reinsertions, 0 otherwise
δ'_{vt}	1 if vehicle $v \in V$ is positioned at or after position t before reinsertions, 0 otherwise
Second-Stage Decision Variables	
w_{kt}	The work overload at station $k \in K$ at cycle $t \in \bar{T}$
z_{kt}	Starting position of operator at station $k \in K$ at the beginning of cycle $t \in \bar{T}$
b_{kt}	The processing time at station $k \in K$ at cycle $t \in \bar{T}$
y_{it}	1 if failed vehicle $i \in F$ is inserted to position $t = \{1, \ldots, \lvert T\rvert + 1\}$, 0 otherwise
s_{vt}	1 if vehicle $i \in \bar{V}$ is sequenced at position $t \in \bar{T}$ after reinsertions, 0 otherwise
γ_{it}	1 if failed vehicle $i \in F$ is inserted after position $t \in T$, 0 otherwise

partner produces vehicles based on custom orders and customization offers billions of different options. Hence, we sequence vehicles instead of configurations. Accordingly, we define the first-stage binary decision variable x_{vt}, which takes the value of 1 if vehicle $v \in V$ is assigned to position $t \in T$. Additionally, the second-stage decision variables are defined to determine the reinsertion of failed vehicles and the final position of all vehicles after the fails are realized and the reinsertions are decided. The binary decision variable y_{it} is 1 if failed vehicle $i \in F$ reinserted at position t, and binary decision variable s_{vt} is 1 if the final position of vehicle $v \in V \cup F$ is t. We denote the processing time of vehicle $v \in V$ at station $k \in K$ by p_{kv}. The starting position and work overload of the vehicle at position $t = 1, \ldots, |T| + |F_{old}|$ for station $k \in K$ are denoted by z_{kt} and w_{kt}, respectively. In Table 4.1, we present all the sets, parameters, and decision variables used in the proposed mathematical formulation. The second-stage decision variables are scenario-dependent, however, such dependency is dropped for the sake of notation simplicity.

There are two types of failed vehicles that require special consideration for the reinsertion process. The first type is the vehicles that had failed in a previous production horizon F_{old}, have not been reinserted yet, and become ready for reinsertion at the beginning or during the current horizon. The second type is the vehicles that fail in the current production horizon F_{new}. In addition

to the sets defined in Table 4.1, we define some other useful sets, based on the relation of the already defined sets. The set $F = F_{new} \cup F_{old}$ presents the set of all failed vehicles. The set $V^{'} = V \setminus F_{new}$ denotes the set of vehicles that were planned for the current production horizon and do not fail. The sets $\bar{V} = V \cup F_{old}$ and $\bar{T}$ denote the set of all vehicles and positions after the reinsertions, respectively.

In this chapter, while we adopt the basic assumptions of the MMS problem as given by Bolat *et al.* [16], additional rules/assumptions for the vehicle failures and the reinsertion process are defined as follows:

- It is assumed that each vehicle $v \in V$ has a failure probability f_v, and failures are independent of each other. In our numerical experiments in Section 4.5, the failure probabilities are estimated from the historical data by doing feature analysis and using logistic regression.

- The vehicles go through the body shop and paint shop in the scheduled sequence before the assembly process. Hence, a failed vehicle must be pulled out of the sequence, however, its position can be compensated by a reinstating vehicle.

- Once the failure of a vehicle occurs due to any reason, there is an uncertain amount of time (measured in the number of positions) required for the failure reason to be fixed, i.e., the number of positions required for a failed vehicle to become a reinstating vehicle. For example, repainting of the vehicle if the issue was paint quality, or arrival of material if it was a supply chain issue. This time, referred to as *being ready time*, is modeled as a discrete uniform distribution in a closed range.

- There is an upper limit on the number of reinstating vehicles at the end of each production horizon, f_{max}. In the car manufacturing facilities, if f_{max} is to be exceeded, the whole line stops and only produces the reinstating vehicles. The motivation behind this application is to avoid the late delivery of failed vehicles.

- The number of failed vehicles at the beginning of a production horizon is predicted randomly using a discrete uniform distribution between 0 and f_{max}.

- It is assumed that only one reinsertion can be done into each subsequence of length λ.

- A failed vehicle from a previous planning horizon must be reinserted if the current horizon is

the last day for it to be started, $g_i = d_i$, so that the vehicle can be delivered to the customer on time.

The integration of the reinsertion process into the sequencing problem introduces conflicting objectives. The number of vehicles reinserted during a planning horizon conflicts with the total work overload since the reinsertions are made into an already optimized sequence, minimized work overload. Throughout the chapter, these objectives are referred to as *work overload objective* and *reinsertion objective*, respectively. Accordingly, we employ two objectives for our problem: minimizing total work overload duration and minimizing the sum of squared waiting days of each not reinserted failed vehicle. For the work overload handling procedure, we adopt the side-by-side policy which assumes that the regular operator stops working on the workpiece once the workpiece reaches the station border. The remaining job of the workpiece is completed by the so-called utility worker, and the regular operator starts working on the next workpiece at position $l_k - c$ in the same station.

4.3.2 Mathematical Model Under Uncertainty

The operational dynamics of the reinsertion process, which is executed after the realization of vehicle failures, motivated us to formulate our problem as a two-stage stochastic program. In the first stage (here-and-now), the vehicle sequence is decided before any car failures are realized. Subsequently, when the car failures become known, the work overload is minimized through the second-stage decisions (wait-and-see) based on the initial sequence. While the determination of first-stage decisions involves assigning each vehicle to a specific position with the objective of minimizing the anticipated objective function value in the second stage, the second-stage decisions involve the reinsertions with the work overload and reinsertion objectives. For the sake of clarity, throughout the chapter, we call the decisions on either a failed vehicle reinserted or not as *binary reinsertion decisions*, on the other hand, the decision on the reinsertion positions as *reinsertion position decisions*.

In the proposed problem, the uncertainty lies in several factors: 1. vehicle failures in the current horizon, each vehicle either exists or fails, 2. the number of vehicles in F_{old}, 3. the number of days between the planned and current period g_i for each failed car $i \in F_{old}$, 4. The number of positions required for each failed vehicle $i \in F$ to be ready for reinsertion r_i. There are a total of $2^{|V|}$, $(f_{max} + 1)$, $G^{|F_{old}|}$ (assume that G is the lead time of the facility), and $|V| * |F|$ scenarios for

the factors 1, 2, 3, and 4, respectively. In order to decrease the degree of scenario space, we predict factors 3 and 4 and fix them as deterministic parameters. That is, g_i and r_i are known beforehand and used in case of vehicle i failure. Accordingly, we have a total of $2^{|V|} * (f_{max} + 1)$ scenarios.

To formulate the problem, we represent varying realizations of vehicle failures and failed vehicles by a set of finite scenarios Ω. Each scenario $\omega \in \Omega$ is denoted by the vehicle failure, let $e_{v\omega} = 0$ if vehicle v fails and $e_{v\omega} = 1$ if vehicle $v \in V$ exists at scenario $\omega \in \Omega$, and the number of failed vehicles from previous horizons $|F_{old}|$ is uniformly distributed in $[0, f_{max}]$. The probability of scenario ω can be calculated as $p_w = \frac{1}{f_{max}+1} \prod_{v=1}^{|V|} f_v^{1-e_{v\omega}} (1 - f_v)^{e_{v\omega}}$.

We note that the sequence length depends on the vehicle failures and the second-stage decisions (reinsertions), and such a problem cannot be solved using state-of-art commercial solvers. Hence, we develop some methods to improve the formulation so that we can formulate the second-stage as a mixed-integer quadratically constrained program (MIQCP). First, we create a *dummy position* at $|T| + 1$ to where we make dummy reinsertions, in other words, we assume that all failed vehicles $i \in F$ that are actually not reinserted are assigned to position $|T| + 1$. This trick helps each scenario have a fixed sequence length that is no longer decision dependent. The sequence length of scenario ω is $|T| + |F^{\omega}_{old}|$. Next, dummy reinsertions, that are made at the end of the sequence, should be exempt from the work overload calculation since they actually do not exist in the final sequence (not reinserted), and they will be produced in a future horizon. In order to tackle this issue, we turn each vehicle that is reinserted at the dummy position into a *neutral vehicle* by setting the processing time equal to the cycle for each station. We refer to these vehicles as neutral vehicles because they do not have any impact on the schedule in terms of work overload. Refer to Section 3.3.2 for an extensive explanation of neutral vehicles.

Finally, we present a two-stage stochastic program for the *full-information problem* where all possible realizations are considered.

$$\min_{x,\delta,\delta^{'}} \quad \sum_{\omega\in\Omega} \rho_{\omega} Q((x,\delta,\delta^{'}),\omega) \tag{4.1a}$$

$$\text{s.t.} \quad \sum_{v\in V} x_{vt} = 1, \qquad t \in T \tag{4.1b}$$

$$\sum_{t\in T} x_{vt} = 1, \qquad v \in V \tag{4.1c}$$

$$\sum_{t\in T} \delta_{vt} - x_{vt}t = 0, \qquad v \in V \tag{4.1d}$$

$$\delta_{vt} - \delta_{v(t-1)} \leq 0, \qquad v \in V, \quad t = 2,\ldots,|T| \tag{4.1e}$$

$$\delta^{'}_{v0} = 0, \qquad v \in V \tag{4.1f}$$

$$\delta^{'}_{vt} - \delta_{v(t-1)} = 0, \qquad v \in V, \;\; t = 2,\ldots,|T| \tag{4.1g}$$

$$x \in \{0,1\}, \tag{4.1h}$$

$$\delta,\delta^{'} \geq 0 \tag{4.1i}$$

where $Q(x,\omega) =$

$$\min_{y,s,\gamma,z,w,b,\beta} \quad \sum_{t\in\bar{T}}\sum_{k\in K} w_{kt} + \sum_{i\in F}(1-\sum_{t\in T} y_{it})(g_i+1)^2 \tag{4.2a}$$

$$\text{s.t.} \quad x_{it} - \sum_{h=\min(|T|,t+r_i)}^{|T|+1} y_{ih} \le 0 \qquad i \in F_{new},\ \ t \in T \tag{4.2b}$$

$$\sum_{t=1}^{|T|+1} y_{it}t \ge r_i, \qquad i \in F_{old} \tag{4.2c}$$

$$\sum_{t\in T} y_{it} = 1, \qquad i \in \{F_{old} | g_i = d_i\} \tag{4.2d}$$

$$\sum_{h=t}^{t+\lambda-1}\sum_{i\in F} y_{ih} \le 1, \qquad t = 1,\ldots,|T|-\lambda+1 \tag{4.2e}$$

$$\sum_{t\in T}(1-y_{it}) \le f_{max}, \qquad i \in F \tag{4.2f}$$

$$\sum_{t=1}^{|T|+1} y_{it} = 1, \qquad i \in F \tag{4.2g}$$

$$\sum_{t\in T} \gamma_{it} - y_{it}t = 0, \qquad i \in F \tag{4.2h}$$

$$\gamma_{it} - \gamma_{i(t-1)} \le 0, \qquad i \in F,\quad t = 2,\ldots,|T| \tag{4.2i}$$

$$\sum_{v\in\bar{V}} s_{vt} = 1, \qquad t \in \bar{T} \tag{4.2j}$$

$$\sum_{t\in\bar{T}} s_{vt} = 1, \qquad v \in \bar{V} \tag{4.2k}$$

$$\sum_{t\in\bar{T}} s_{vt}t - \sum_{t\in T} x_{vt}t - \sum_{t\in T}\sum_{i\in F} y_{it}\delta'_{vt} + \sum_{i\in F_{new}}\sum_{t\in T} x_{it}\delta_{vt}(1-e_i) = 0, \qquad v \in V' \tag{4.2l}$$

$$\sum_{t\in\bar{T}} s_{it}t - \sum_{t\in T} y_{it}t - \sum_{t\in T}\sum_{j\in F} y_{jt}\gamma_{it} + \sum_{j\in F_{new}}\sum_{t\in T} x_{jt}\gamma_{it}(1-e_j) - y_{vT}\beta_i = 0, \qquad i \in F \tag{4.2m}$$

$$b_{kt} - \sum_{v\in V'} p_{kv}s_{vt} - \sum_{i\in F} s_{it}(p_{ki}(1-y_{i(|T|+1)}) + cy_{i(|T|+1)}) = 0, \qquad k \in K,\quad t \in \bar{T} \tag{4.2n}$$

$$z_{kt} + b_{kt} - z_{k(t+1)} - w_{kt} \leq c, \qquad k \in K, \quad t \in \bar{T} \tag{4.2o}$$

$$z_{kt} + b_{kt} - w_{kt} \leq l_k, \qquad k \in K, \quad t \in \bar{T} \tag{4.2p}$$

$$z_{k0} = 0, \qquad k \in K \tag{4.2q}$$

$$z_{k(|\bar{T}|+1)} = 0, \qquad k \in K \tag{4.2r}$$

$$y, s, \gamma \in \{0, 1\}, \tag{4.2s}$$

$$z, w, b, \beta \geq 0 \tag{4.2t}$$

The first-stage problem (4.1) minimizes the expected cost associated with the second-stage problem. Constraint sets (4.1b) and (4.1c) ensure that each position has one vehicle assigned and each vehicle is assigned to one position, respectively. Constraint sets (4.1d) and (4.1e) builds the relationship between x and δ variables. The δ variables help to determine the number of failed vehicles up to each position in constraint set (4.2l). Constraint sets (4.1f) and (4.1g) build the relationship between $\delta^{'}$ and δ variables. The $\delta^{'}$ variables help to determine the number of reinserted vehicles up to each position in constraint set (4.2l). Constraint sets (4.1h) and (4.1i) represent the domain of the first-stage variables.

In the second-stage problem (4.2), the objective function minimizes the sum of the total work overload duration and the sum of the square of the number of waiting days of the failed vehicles that arenot to be reinserted, given the first-stage decision (sequence) and scenario $\omega \in \Omega$. Constraint sets (4.2b) - (4.2f) force the restrictions about the reinsertion. The constraint sets (4.2b) and (4.2c) ensure that new and previously failed vehicles are not reinserted before they are ready, respectively. Constraint set (4.2d) guarantees that a failed vehicle at its due must be reinserted. The constraint set (4.2e) assures that each subsequence in length λ has at most one reinsertion. Constraint set (4.2f) assures that the limit on the maximum number of not reinserted vehicles is not exceeded. Constraint set (4.2g) ensures that all failed vehicles are reinserted; the reinsertions at position $|T| + 1$ are the dummy reinsertions. Constraint sets (4.2h) and (4.2i) build the relationship between x and γ variables. The γ variables help to determine the number of failed and reinserted vehicles up to each position in constraint set (4.2m). Constraint sets (4.2j) and (4.2k) ensure that, after the reinsertions, each position has one vehicle assigned and each vehicle is assigned to one position, respectively. Note that $\bar{V}$ and $\bar{T}$ include the previously failed vehicles and additional positions for them, respectively. Constraint sets (4.2l) and (4.2m) determine the final position of

each non-failed and failed vehicle, respectively. Note that $V^{'}$ denotes the set of vehicles that were planned for the current production horizon and do not fail. In constraint set (4.2l), the final position of a non-failed vehicle is set to the original position plus the number of reinsertions up to its original position minus the number of vehicle failures up to its original position. In constraint set (4.2m), the final position of a failed vehicle is set to its reinsertion position plus the number of reinsertions up to its reinsertion position minus the number of fails up to its reinsertion position. The last term $y_{vT}\beta_i$ is added because the order of the vehicles reinserted to the dummy position $|T|+1$ is irrelevant. Constraint set (4.2n) determines the processing times of vehicles based on the final sequence. We note that the processing time of the dummy reinserted vehicles is set to cycle time for each station (as neutral vehicles). Constraint sets (4.2o) - (4.2p) determine the starting position and work overload at each position based on the final sequence, respectively. Constraint sets (4.2q) and (4.2r) guarantee a regenerative production plan. Constraint sets (4.2s) and (4.2t) represent the domain of the second-stage variables.

One can obtain the deterministic equivalent formulation (DEF) for MMS with stochastic failures and reinsertion by adding the first-stage constraints (4.1b)-(4.1i) to the second-stage formulation, and by adding copies of all second-stage variables and constraints.

For our problem, integrating the reinsertion process into MMS presents a bi-objective environment since inserting additional vehicles into an already optimized sequence conflicts with the work overload minimization. The two objectives are given as a single objective in the summation form in (4.2) for the sake of clarity. However, we tackle these two conflicting objectives via a bi-objective approach which is adapted into the proposed solution approaches in Section 4.4. In a multi-objective environment, the objectives conflict with each other, making it impossible to find a solution that optimizes all objectives simultaneously. In order to balance the tradeoffs among the objectives, the concept of *Pareto optimality* is defined. The set of Pareto optimal solutions is called *Pareto front*. A solution is Pareto optimal if the solution is not dominated by any other solutions. The domination criteria is defined as follows. Let $F(x) = f_1(x), .., f_m(x)$ be the set of objectives, where m is the number of objectives. Solution x dominates solution $x^{'}$ if $x_i \leq x_i^{'}$ for all $i \in 1, \ldots, m$ and $x_j \leq x_j^{'}$ for at least one of the objectives. The goal of multi-objective algorithms is to find the best representation of the true Pareto front.

Let us define the objectives separately before proposing our solution approaches. The *work overload* objective (4.3) minimizes total work overload duration for the final sequence (after vehicle

failure realization and reinsertions) across all scenarios, stations, and positions.

$$\min \quad \sum_{\omega \in \Omega} \rho_\omega \sum_{t \in \bar{T}} \sum_{k \in K} w_{kt\omega} \tag{4.3}$$

The *reinsertion objective* (4.4) minimizes the sum of the squared number of waiting days of the failed vehicles that are not reinserted at the end of the current horizon, across all scenarios. The waiting time is calculated as the number of days between the current production horizon and the horizon that the vehicle was originally planned to be produced. The motivation behind this objective is meeting the delivery deadlines and ensuring customer satisfaction since each car is produced based on a specific order.

$$\min \quad \sum_{\omega \in \Omega} \rho_\omega \sum_{i \in F} (1 - \sum_{t \in T} y_{it\omega})(g_i + 1)^2 \tag{4.4}$$

4.4 Solution Approaches

MMS, a problem known for its NP-hard nature [110], becomes significantly more challenging to solve when faced with stochastic product (car) failures and integrated reinsertion process in a multi-objective setting. Therefore, it is essential to develop effective heuristic methods to tackle large-scale problems. Accordingly, in this section, we first explain a feasibility enhancement procedure that is utilized in the evolutionary algorithm, and some efficiency improvements that are utilized for all three proposed solution approaches, then we explain our sampling approach that tackles the exponentially increasing number of scenarios. Next, we propose three efficient metaheuristic approaches to solve our problem. First, a population-based evolutionary optimization algorithm utilizing the NSGA-II structure is proposed in Section 4.4.1. Then, a two-stage multi-objective local search is presented in Section 4.4.2. Finally, a hybrid approach, a local search integrated evolutionary algorithm utilizing NSGA-II structure, is proposed in Section 4.4.3. Evolutionary and local search algorithms are proposed since they have proved their success on multi-objective problems and MMS problems, respectively.

Feasibility Enhancement Each solution (sequence) is feasible in a standard MMS problem. However, in our problem, with the addition of vehicle failures and the reinsertion process, it is not the case. Although each first-stage decision (sequence before failures and reinsertion) is feasible, the

second-stage decisions (final sequence after reinsertions for each scenario) can be infeasible due to one of the following five restrictions regarding the constraints (4.2b) - (4.2f). The first three restrictions (4.2b) - (4.2d) are tackled while making the decision of the reinsertion position to ensure that a failed vehicle is not reinserted before it is ready, referred to as *being ready feasibility*. On the other hand, the latter two restrictions (4.2e) and (4.2f) are tackled once the final sequence is determined. In case of a restriction (4.2e) violation, which means the number of not reinserted failed vehicles exceeds the limit, we check all positions until we find appropriate positions for a number of vehicles equal to the excess amount. If we cannot accomplish enough reinsertions, then we do the reinsertions with the sacrifice of violating the restriction (4.2f). Hence, the only restriction that we allow to be violated is (4.2f), referred to as *lambda feasibility*, however, in case of a violation of this restriction, we try to eliminate (or minimize) it by exploring possible movements (one vehicle movement at a time) without violating other restrictions. In the end, we keep the number of violated subsequences (the subsequences length of λ with more than one reinsertion) in order to use the constrained-domination process within the evolutionary algorithms.

Efficiency Improvements The work overload and the reinsertion objective function values are calculated as given in (4.3) and (4.4), respectively. The computational cost associated with evaluating the objective function and checking and maintaining the feasibility is high. These procedures consume a significant portion of the algorithm's execution time, potentially limiting their overall efficiency and speed in finding a good representation of the Pareto front. We utilize several improvement techniques to improve the computational efficiency of the proposed algorithms. First, we employ an accelerated objective evaluation technique to speed up the objective evaluation of each new solution. Next, the tabu rules given in Appendix C are employed to increase the chance of finding a good reinsertion position. That is, every time we generate a random reinsertion position or every time we apply a transformation operator to the second-stage solution during the local search, we apply the tabu rules. Finally, in order to decrease the computational cost of maintaining feasibility, we keep track of the first-stage and reinsertion positions of failed vehicles, and also a reinsertion map of the sequence for each scenario. The reinsertion map (rm) is a positional binary mapping of the sequence, $rm_t = 1$ if there is a reinserted vehicle at position t, 0 otherwise. Accordingly, we obtain three improvements as follows: (1) having reinsertion positions of failed vehicles decreases the single iteration cost of the second-stage improvement from $O(N)$ to $O(1)$, (2) having positions of failed

vehicles at the first-stage solution decreases the maintaining being ready feasibility cost from $O(N)$ to $O(1)$, (3) having reinsertion position of failed vehicles decreases the maintaining lambda feasibility cost from $O(|F|\lambda)$ to $O(\lambda)$. We note that while improvements (2 and 3) are for all three approaches, improvement (1) is for the local search and local search integrated evolutionary approaches.

Sampling Approach Due to the exponential increase in the number of scenarios, the stochastic program becomes overwhelmingly large. Consequently, we utilize a sampling approach to address this challenge. We notice that the majority of failure scenarios have marginal probabilities. Rather than explicitly exploring all potential failure scenarios $\omega \in \Omega$, we approximate the expected value function of the two-stage stochastic program $z^* = \min_{x \in X} \mathbb{E}[Q(x, \xi_\omega)]$ with an identically and independently distributed (i.i.d) random sample of N realizations of the random vector $\Omega_N := \{\omega_1, \ldots, \omega_N\} \subset \Omega$. We no longer take into account the probabilities associated with each scenario. However, it is possible for more probable scenarios to occur multiple times within our sample Ω_N. We assess the average work overloads and wait times in the concluding sequences of all failure scenarios included in the sample. Let $\hat{N}$ and n_ω represent the set of unique scenarios in Ω_N and the number of their occurrences. Hence, the objective function (4.1a) $\sum_{\omega \in \Omega} \rho_\omega(\cdot)$ changes to $\frac{1}{N} \sum_{\omega \in \Omega_N}(\cdot)$ or equivalently, $\frac{1}{N} \sum_{\omega \in \hat{N}} n_\omega(\cdot)$.

4.4.1 Evolutionary Optimization Algorithm

We employ the NSGA-II framework which is developed by [30]. NSGA-II has demonstrated its efficiency in solving multi-objective problems in various fields [111, 67, 114].

In order to tackle the second-stage feasibility issue, we employ two procedures. First, we apply *feasibility enhancement procedure* that transforms (if possible) an infeasible sequence into a feasible sequence as explained earlier in this section. Second, we employ the constraint-domination principle defined in [30], which considers the feasibility of each sequence while deciding domination between two solutions. Utilizing the constrained-domination principle leads to the outcome where feasible solutions surpass infeasible solutions in terms of nondomination rank. Feasible solutions are evaluated and ranked based on their nondomination level using objective function values. Nevertheless, when comparing two infeasible solutions, the solution with a lower degree of constraint violation achieves a higher rank. In case two infeasible solutions have the same degree of constraint violation, then they are compared based on their objective function values.

In addition to the improvement procedures that we employed for the NSGA-II framework as mentioned above, the novelty of the proposed approach lies in the problem-specific chromosome design and next-generation creation methods: crossover and mutation.

Initial Population, First-Stage We generate the first-stage initial population by employing iterative greedy heuristics. One solution is generated using the greedy heuristic given in Section 3.4.2.1, let us refer to it as the utilization rate greedy heuristic throughout this chapter. The rest is generated by using a naive greedy heuristic which makes one assignment at a time, starting from the first position. The vehicle for the first position is selected randomly, then, for the next position, the vehicles that cause the minimum work overload are determined. Out of this group, the vehicles that cause the minimum idle time are determined. A random vehicle is selected from the final group and assigned to the next position. This procedure is repeated until the sequence is completed.

Initial Population, Second-Stage The second stage solutions are generated in two steps. We first determine the binary reinsertion decisions of each failed vehicle for each scenario $\omega \in \Omega_N$. Then, the reinsertion positions of the failed vehicles that are decided to be reinserted are determined. Note that if a failed vehicle is decided to be reinserted but it does not become ready before the end of the horizon, then the binary reinsertion decision is changed. The feasibility enhancement procedure is applied to each scenario of each solution, once the second-stage decisions are determined. All second-stage decisions are randomly determined (both binary reinsertion and reinsertion position decisions), except for two solutions: 1. none of the failed vehicles are reinserted (except the ones that have to be reinserted) for the solution that is generated using the utilization rate greedy heuristic, 2. all of the failed vehicles are reinserted (except the ones that cannot be reinserted due to infeasibility) for one of the other solutions.

Chromosome Design The chromosome representation of each solution is designed in two parts which represent the first and second-stage decisions, respectively. As illustrated in Figure 4.2, while the first part consists of a single sequence, the second part consists of $|\Omega_N|$ sequences, each representing a final sequence for scenario $\omega \in \Omega_N$, note that we illustrate only three scenarios in the figure. The first part of the chromosome encodes the corresponding first-stage sequence. Then, vehicle failures are realized which is summarized under the scenario information column. Next, second-stage decisions (reinsertions) are encoded in the second part of the chromosome. From the

scenario information, we know that vehicles $\{3, 6\}$, $\{3\}$, and $\{6, 7\}$ are in F for the scenarios 1,2, and 3, respectively. The first and second numbers of each gene (tuple) correspond to the id of a failed vehicle and the reinsertion decision for that vehicle, respectively. If a failed vehicle is not reinserted, then the second number equals zero, otherwise equals the reinsertion position. For example, in scenario 2, the failed vehicle 3 is reinserted in position 3.

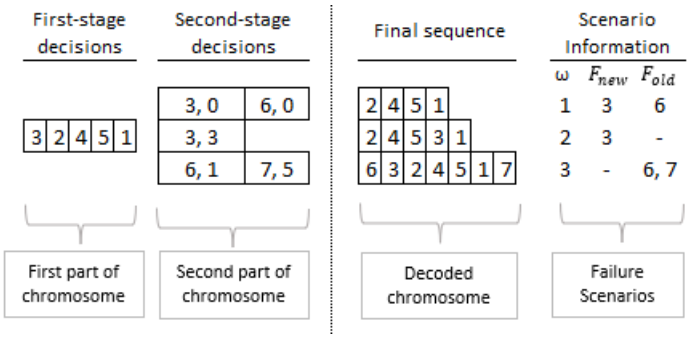
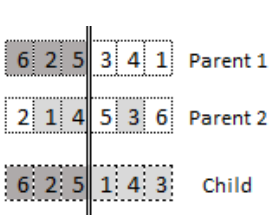
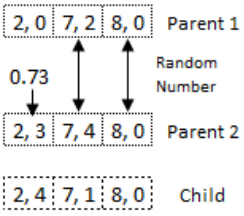


Figure 4.2: Chromosome illustration over an example with three scenarios

Crossover The crossover is executed for the first and second parts of the chromosomes separately. First, two solutions from the parent population are randomly selected. Then, in order to generate a single child by using the two selected parents, a modified partially mapped crossover (PMX) is applied to the first part of the chromosome as illustrated in Figure 4.3a. In order to generate a single child chromosome, a random single point is selected. While the genes up to that point are selected from the first parent, the rest of the genes are selected from the second parent, based on the order that occurred on the second parent, and the genes were not selected from the first parent.



(a) PMX for the first part



(b) Uniform crossover for the second part of a given scenario

Figure 4.3: Illustration of proposed crossover methods

The second part of a child chromosome is generated in two stages. First, a uniform crossover is employed to decide the decimal numbers in the second part of the child's chromosome, however, as a binary selection on binary reinsertion decision. Thus, for each gene in each scenario, if the

binary reinsertion decisions are the same in two-parent genes, the child gets the same decision. Otherwise, we generate a random number to decide on the dominant parent for that specific gene. Next, the reinsertion position is determined independently from parents since otherwise, most of the children chromosomes become infeasible. The second part crossover procedure is illustrated in Figure 4.3b. The parents have the same binary decision for the second and third genes, thus the child gets the same binary reinsertion decision, vehicle 7 is reinserted and vehicle 8 is not reinserted. The parents have different decisions for the first gene, thus a random number is generated to decide which decision the child gets. Since the random number is greater than 0.5, the child gets the decision from the second parent which is reinserting vehicle 2. The reinsertion positions of vehicles 2 and 7 are randomly decided independent of corresponding parent genes, but obeying being ready feasibility restrictions.

Mutation We apply mutations to the first and second-stage solutions separately. For the first-stage solutions, an inversion operator is employed as a mutation in order to avoid local optima and to increase the diversity [2]. The inversion operator is applied to a randomly selected subsequence (the subsequence between randomly selected two points). For the second-stage solutions. a single gene mutation is applied to a gene that carries the binary reinsertion decision. Accordingly, we generate a random number for each child chromosome, if the random number is smaller than the mutation threshold, then we apply mutation to a randomly selected gene on a randomly selected scenario. On one hand, if the selected gene is a negative binary reinsertion decision (not reinserted), then it changes to a positive decision (reinsert), so that the corresponding failed vehicle is reinserted to a random position. On the other hand, if the selected gene is a positive binary reinsertion decision, then it changes to a negative decision, so that the corresponding vehicle is removed from the final sequence and the succeeding vehicles are moved forward to close the gap.

4.4.2 Two-stage Bi-objective Local Search Algorithm

This section proposes a simulation-based two-stage bi-objective local search (STMLS) algorithm which is inspired by the TS algorithm presented in Section 3.4.2.2. A pseudo-code for STMLS is given in Algorithm 8. To summarize the algorithm, STMLS gets an initial solution that is generated for the deterministic counterpart of the proposed MMS problem, which excludes vehicle failures and reinsertions, this problem is referred to as *one-scenario problem*. Then, the initial solution is

improved over the one-scenario problem for just a couple of seconds. Next, the improvement loop for the full-information problem starts once the second-stage solutions are generated. During this loop until meeting the termination criterion, STMLS alternates between improving the first-stage and second-stage solutions on work overload objective and introduces random changes to the binary reinsertion decisions to explore the reinsertion objective space. The algorithm aims to find a good representation of the Pareto front by solving the proposed problem in a bi-objective environment.

Algorithm 8 Pseudo-code of STMLS

Input: The initial first-stage solution generated by utilization rate greedy heuristic, parameters: operator weights, θ, τ_f, and τ_s.
Output: External population: a set of non-dominated solutions that represent the Pareto front
Improve the initial first-stage solution over one-scenario problem
External population $\leftarrow \emptyset$
Generate random second-stage solutions for each $\omega \in \Omega_N$ (all possible binary reinsertion decisions are 1), and apply feasibility enhancement
Calculate the objective function value of each solution in the initial population
Improve second-stage solutions on work overload objective by applying second-stage solution improvement procedure
iteration $\leftarrow 1$
while termination criterion is not reached **do**
  if iteration % $\theta = 0$ **then**
    Update external population
    for each $\omega \in \Omega$ **do**
      Select a failed vehicle randomly and change the binary reinsertion decision
      If the change is from 0 zero to 1, then find a random feasible position for the reinsertion (if possible)
      Calculate the new objective function values and accept the new solution as the incumbent solution, regardless of objective values
      Apply second-stage improvement procedure for τ_s duration
    end for
  else
    Apply first-stage improvement procedure for τ_f duration
  end if
  iteration $\leftarrow iteration + 1$
end while

Local Search Specific Efficiency Improvements Since the movements made via transformation operators make local changes to the sequence, there is no need to reevaluate the whole sequence. Hence, we utilized partial objective reevaluation during a local search (both first-stage and second-stage), in addition to the accelerated reevaluation, as given in Section 3.4.2.2.

Transformation Operators We apply one randomly selected operator at each iteration of the improvement procedures explained below. The random selection of operators is based on the operator weights which are determined based on our preliminary experiments. The swap operator is used to interchange the positions of two randomly chosen cars in the sequence. The insertion operator removes a car from its current position i and inserts the car to a position j. There are two types of insertions: backward insertion and forward insertion. Backward insertion occurs when $i > j$, where the car is inserted at position j and all vehicles between positions j and i are shifted one position to the right (scheduled later). On the other hand, forward insertion happens when $i < j$, where the car is inserted at position j and all vehicles between positions i and j are shifted one position to the left (scheduled earlier). Inversion reverses a subsequence between two randomly selected positions in the whole sequence. Even though there are more operators proposed in the literature that could be used to transform a permutation-based representation, a local search across the neighborhood defined over these four transformation operators is shown in Chapter 2 to be very efficient to solve sequencing problems.

First-stage Improvement Procedure Given a full solution (first and second-stage solutions), this procedure executes a local search within the neighborhood defined above, by applying one operator at a time on the first-stage solution. Each time a neighbor first-stage solution is visited, the feasibility of each second-stage solution is checked and a new reinsertion position is randomly generated (if possible) for the ones that cannot be reinserted to the previously determined position anymore due to the change on the first-stage solution. We note that tabu rules are utilized while selecting the reinsertion positions during the random second-stage solution generation process, additionally, the binary reinsertion decisions are not changed during this procedure unless due to a feasibility issue. If the new solution is not deteriorated based on the work overload objective, then the new solution is accepted as the incumbent solution, rejected otherwise. To summarize, this procedure improves the work overload objective while keeping the reinsertion objective unchanged (when possible).

Second-stage Improvement Procedure Given a second-stage solution for a scenario $\omega \in \Omega_N$, this procedure executes a local search on the second-stage reinsertion position decisions by applying swap and insertion operators on the reinserted vehicles only. That is because, a movement of vehicle

$v \in V$ means changing the first-stage solution, and there is no valid inversion movement when $\lambda > 1$. A new solution is accepted if it is not deteriorated in terms of work overload objective.

4.4.3 Local Search Integrated Evolutionary Algorithm

In this section, we propose a hybrid approach which is a local search integrated evolutionary algorithm, *LS-NSGA-II*. We integrate local search improvement procedures to the NSGA-II structure proposed in Section 4.4.1 in two places. First, each first-stage solution in the initial population is improved over the one-scenario problem. Second, the second-stage improvement procedure is applied to each child chromosome when it is generated. Executing a local search on each child chromosome increases the computational complexity of the algorithm, hence, it is necessary to balance the time spend on local search and evolution. In order to accomplish this, we decrease the population size P and set a shorter time limit τ_s for the second-stage improvement on each chromosome. A pseudo-code for the proposed approach is given in Algorithm 9.

Algorithm 9 Pseudo-code of Local Search Integrated NSGA-II

Input: The first stage initial population generated by greedy heuristic, parameters: population size (P), mutation probability, and τ_s

Output: A set of non-dominated solutions that represent the Pareto front

*Improve each first-stage solution over one-scenario problem

*EP $\leftarrow \emptyset$

For each first-stage solution, randomly generate second-stage solutions for each $\omega \in \Omega$, and apply feasibility enhancement

Calculate the objective function value of each solution in the initial population

while Termination criterion is not reached **do**

  Create a child population of size P using the parent population via applying crossover, mutation, and feasibility enhancement

  *Apply second stage improvement procedure, given in Section 4.4.2, to each child chromosome for τ_s duration

  Calculate the objective function value of each child solution

  Combine parent and child populations which results in a set of solutions in size $2P$

  Determine nondomination rankings using the constrained domination principle

  Calculate crowding-distance of each solution

  Select the best P solutions (next generation's parent population) based on the nondomination ranking and crowding distance in the given order

  *Update EP: Add all non-dominated solutions to the EP and remove all dominated solutions from the EP

end while

**Only with the LS integrated NSGA-II structure*

One of the drawbacks of NSGA-II is that the number of non-dominated solutions (solutions in the first Pareto front) is limited with the population size P. In order to tackle this drawback we

collect all non-dominated solutions that are explored along the search in an external set (external population, EP), which is originally proposed by [73]. We note that, we utilize this improvement only with the LS-NSGA-II structure since our preliminary experiments show that the number of non-dominated solutions does not exceed the population size with the standard NSGA-II structure.

4.5 Numerical Experiments

In this section, we first describe the experimental setup in Section 4.5.1. Then, the computational performance of the proposed solution approaches is assessed in Section 4.5.2. Next, the solution quality of the solutions obtained by solving the proposed problem is assessed in Section 4.5.3, over dynamic reinsertion simulations, by comparing them with the solutions obtained by solving the one-scenario problem and the problem given in Chapter 3.

We note that the traditional MMS problem is NP-hard [110], hence, the problem proposed in this Chapter is NP-hard since it is a more complicated version of the traditional MMS. Considering the stochastic failure of vehicles poses a significant complexity increase in the model, as shown in Chapter 3. Additionally, integrating vehicle reinsertions to MMS adds significant complexity. Accordingly, we did not propose any exact solution approaches since they are inadequate to solve even small instances. e.g., Gurobi can optimally solve instances with up to 5 vehicles, 5 stations, and 100 scenarios even with a single objective, in a one-hour time limit.

4.5.1 Experimental Setup

In this section, the large-sized real-world inspired instances generated in Chapter 3 are utilized for the numerical experiments, refer to Section 3.5.1 for the details of the instances.

There are additional data derived since we consider the reinsertion process in this chapter. The maximum number of failed cars allowed at the end of the planning horizon is set to $f_{max} = |V| * 0.05$. The reinsertion window length $\lambda = 10$. The order lead time is 9. The number of days between the planned and current period for the failed car $i \in F_{old}$, g_i, is discrete uniformly distributed in [1, 9]. The number of extra days until the delivery date that the failed car $i \in F_{old}$, d_i, is discrete uniformly distributed in $[g_i, 9]$. The number of positions required for failed vehicle i to be ready for reinsertion, r_i, is discrete uniformly distributed in [10, $|V|$-10] for $i \in F_{new}$ and in [0, $|V|$-10] for $i \in F_{old}$. As aforementioned, these parameters are generated for each instance as a

deterministic parameter, in other words, they are the same for each scenario $\omega \in \Omega$.

We utilized a sampling approach by generating an i.i.d random sample of N realizations of failure scenarios. For each failure scenario $\omega \in \Omega_N$ and vehicle $v \in V$, we first determine whether a vehicle failed or not as explained in Section 3.5.1. Then, the number of vehicles in F_{old} is determined for each scenario from a discrete uniform distribution in $[0,\ f_{max}]$. Once $|F_{old}|$ is determined for each scenario, the vehicles in F_{old} are randomly selected from a predetermined set of vehicles in size f_{max}. We set the sample size $N = 100$, unless otherwise stated, considering the problem complexity and the results given in Section 3.5.2.

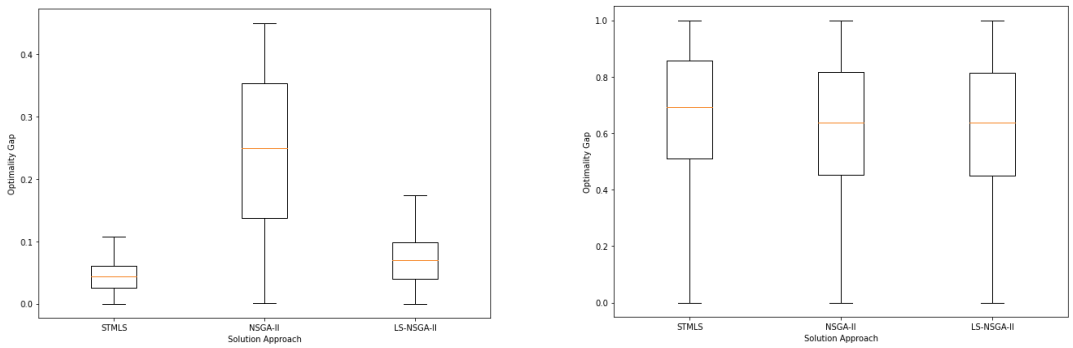
The proposed algorithms are implemented in Python 3. In order to meet the industry requirements that expect to obtain a good quality solution in a limited time, the termination criterion is set to 600 seconds for all of the algorithms. The number of replications (runs) is set to 30, i.e., each instance is solved by each solution approach for 30 times with a different seed. The computing nodes, with 16 cores and 125 GB of memory, of the Clemson University supercomputer are utilized to execute the numerical experiments.

4.5.2 Computational Performance

In this section, we assessed the performance of the proposed algorithms: STMLS, NSGA-II, and LS-NSGA-II. The performance assessment is executed in two parts. We first evaluate the solution approach on each objective, separately. Then, a multi-objective comparison over the non-dominated sets provided by each algorithm is executed.

Single-objective Analysis After completing each run of every solution method, we obtained a set of non-dominated solutions: the first Pareto front of the last generation for the NSGA-II and the external populations for the STMLS and the LS-NSGA-II. In order to evaluate the reliability of the solution methods for each objective, we then compared each non-dominated set with the best-found value of each objective across all runs. Let us call the best found values across all runs as *heuristic ideal points*, because finding ideal points (the optimal values of each objective when the problem solved explicitly for each objective) is not possible. This comparison is executed as follows: for each instance, we first determine the heuristic ideal points. Then, we determine the best-found objective values of each non-dominated set, one value per objective from each run. Next, the gap between the heuristic ideal points and the best-found in a single run is calculated for each objective. The box

plots that show the distribution of the gaps are given in Figure 4.4.



(a) work overload objective **(b)** reinsertion objective

Figure 4.4: Convergence comparison of the algorithms in terms of each objective separately

For the work overload objective, STMLS and LS-NSGA-II provide more reliable solutions, each within an 11% and 18% gap, respectively. While STMLS provides slightly better solutions compared to LS-NSGA-II, there is no significant difference based on the Kruskal-Wallis test with 95% confidence interval. On the other hand, they both outperform the NSGA-II which provides solutions in a wider objective value range, with up to 44% gap. Moreover, in terms of reinsertion objective, all three solution approaches provide similar solution quality based on this analysis. Since, for most of the instances, the best-found reinsertion objective is either zero or very close to zero, even a small fraction of the increase in the reinsertion objective value induces a large gap, as in the figure, each approach have on average more than 60% gap. However, the average ideal point of reinsertion objective is 0.56 and the average best-found of each non-dominated set is 1.55, 1.34, and 1.34, for the STMLS, NSGA-II, and LS-NSGA-II, respectively. We can conclude the single-objective analysis by stating that STMLS and LS-NSGA-II can provide reliable solutions in terms of both objectives and NSGA-II can provide reliable solutions only for the reinsertion objective.

Multi-objective Analysis The performance of the proposed solution approaches was compared in a multi-objective optimization environment. That is, a comparison of non-dominated sets of solutions provided by each approach (Pareto front). The quality of the Pareto front generated by an algorithm is assessed based on criteria such as the number of non-dominated solutions explored, the distribution of the solutions in the non-dominated set, and the distance among the non-dominated solutions and Pareto-optimal solutions or ideal points [126, 5, 94, 56, 28, 103]. In this section, we first present empirical attainment surface graphs in order to have an insight into the Pareto exploration and

quality comparison of the algorithms. The attainment function provides a description of a random non-dominated point set's location distribution. The attainment function can be estimated by empirical attainment functions (EAF) [39], which is utilized to interpret the statistical performance of stochastic multi-objective optimizers. Next, we employed the following performance metrics to assess the multi-objective performance comparison of the solution approaches. We note that the objective function values are normalized for each instance separately and the normalized values are used throughout this section in order to tackle different scales associated with each objective.

- *The number of non-dominated solutions (NNS)* evaluates the number of solutions within the Pareto front that is generated by an algorithm, i.e., the number of non-dominated solutions explored by an algorithm.

- *Covered size of space (CSS)* compares two sets of non-dominated solutions provided by two algorithms. The CSS metric, also known as *C metric*, is introduced by Zitzler *et al.* [126]. Given two sets of non-dominated solutions, X and Y, $C(X,Y)$ calculates the ratio between the number of solutions in set Y that are dominated by any solution from set X and the total number of solutions in set Y, given in (4.5). That is, $C(X,Y) = 1$ means that all the solutions in set Y are dominated by at least one of the solutions from set X.

$$C(X,Y) = \frac{\mid a_Y \in Y \,;\, \exists a_X \in X,\, a_X \leq a_Y \mid}{Y} \tag{4.5}$$

- *Mean ideal distance (MID)* measures the distance between the ideal point, denoted as z^*, and a given set of non-dominated solutions X, as given in (4.6), assume that the m is the number of objectives and n is the number of solutions in set X. Remember that we employed the best-found objective function values across all runs as heuristic ideal points.

$$MID = \frac{\sum_{i=1}^{n} \sqrt{\sum_{j \in m} (X_{ij} - z_m^*)^2}}{n} \tag{4.6}$$

- *Spread of non-dominance solutions (SNS)* evaluates the distribution of non-dominated solutions based on the MID value. The SNS is used to assess the variability of the distance between each solution and the ideal point. Note that a higher SNS value is desirable as it indicates a

more uniform distribution of solutions across the Pareto front.

$$SNS = \sqrt{\frac{\sum_{i=1}^{n}\left(MID - \sqrt{\sum_{j\in m}(X_{ij} - z_m^*)^2}\right)^2}{(n-1)}} \tag{4.7}$$

In Figure 4.5, we compare the statistical performance of the algorithms over nine instances by utilizing the attainment surfaces. We randomly selected three production days (three instances for each day, each with a different number of vehicles) for this analysis since it is not convenient to share plots for each of the 90 instances.

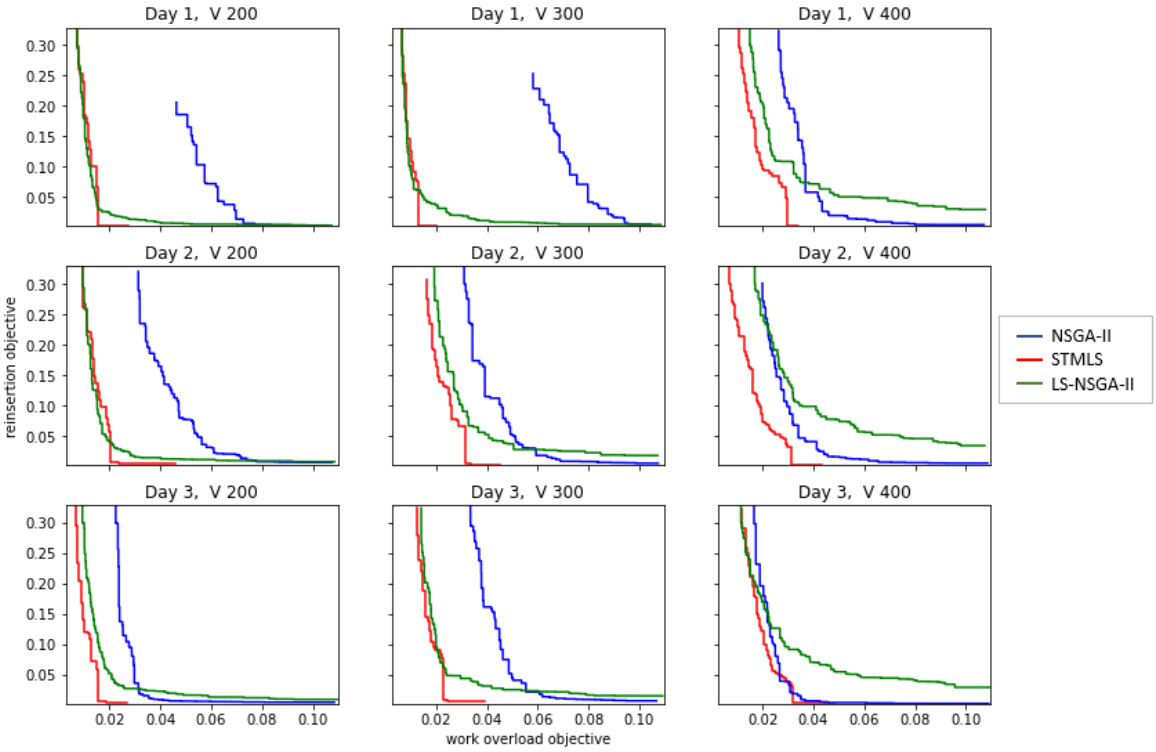


Figure 4.5: Comparison of the 50%-attainment surfaces of the proposed algorithms, with a 95% confidence level. The illustrated region is limited with up to 0.1 and 0.3 for the work overload and reinsertion objectives, respectively

The STMLS and LS-NSGA-II algorithms provide better-quality Pareto fronts compared to NSGA-II. It is interesting to note that integrating local search procedures within the NSGA-II structure have improved the NSGA-II algorithm, this was at the expense of degradation in the right-below area, where the work overload objective is high and the reinsertion objective is low, especially as the number of vehicles increases. This is interpreted as the increasing algorithm complexity may

hurt the exploration in some areas as the size of the problems increase. It is also important to point out that LS-NSGA-II explores a greater region in its Pareto front compared to STMLS and NSGA-II, i.e., better exploration of the edge cases where one of the objectives is very low while the other objective is very high. It is difficult to demonstrate the statistical comparison of the algorithms for whole space in a single graph, hence, Figure 4.5 is limited to focus on the left-below region where more balanced solutions, in terms of objective function values, are provided. Refer to Appendix E for the global counterpart of Figure 4.5.

Table 4.2: NNS, MID, and SNS performance metrics comparison of the NSGA-II, STMLS, and LS-NSGA-II, averaged across all instances and all runs

Solution Method	NNS	MID	SNS
NSGA-II	46.18	0.23	0.10
STMLS	12.99	0.26	0.08
LS-NSGA-II	40.76	0.24	0.12

In Tables 4.2 and 4.3, we compare the algorithms in terms of the performance metrics that are explained earlier in this section. The presented values correspond to the average values across all runs and instances, yet for the CSS metrics, the instances are aggregated based on instance size since the performance of the algorithms, for the CSS metric, is impacted by the size of the instances. Table 4.2 shows that the STMLS performs worse in terms of all three metrics, NNS, MID, and SNS, compared to the evolutionary algorithms. STMLS generates less number of non-dominated solutions. While the generated solutions have a greater distance to the heuristic ideal points, the smaller SNS value shows that the spread of the solutions is low. This is because the non-dominated solutions provided by STMLS have, on average, high reinsertion objective value since STMLS spends most of the computational effort on the minimization of the work overload objective. On the other hand, LS-NSGA-II and NSGA-II perform similarly in terms of NNS, MID, and SNS metrics, yet higher SNS and lower MID values of LS-NSGA-II support that it explores more solutions in the edge regions.

CSS metric provides a reliable pair-wise comparison among sets of non-dominated solutions. Table 4.3 shows that STMLS consistently achieves high CSS values, which means that STMLS provides better quality Pareto fronts. Although STMLS provides a significantly less number of non-dominated solutions compared to evolutionary algorithms, the high CSS values show that STMLS-provided solutions dominate, on average, 85% of NSGA-II-provided solutions and 68% of LS-NSGA-II-provided solutions. LS-NSGA-II performs relatively well compared to NSGA-II. LS-NSGA-II-

Table 4.3: CSS performance metric comparison of the NSGA-II, STMLS, and LS-NSGA-II, averaged across all instances and all runs

Solution Method	V	NSGA-II	STMLS	LS-NSGA-II
NSGA-II	200	-	0.07	0.22
	300	-	0.06	0.23
	400	-	0.05	0.27
STMLS	200	0.84	-	0.61
	300	0.86	-	0.65
	400	0.88	-	0.80
LS-NSGA-II	200	0.72	0.31	-
	300	0.71	0.26	-
	400	0.66	0.12	-

provided solutions dominate, on average, 70% of NSGA-II-provided solutions, so we can say that integration of local search improvement procedure into the NSGA-II structure drastically enhanced the algorithm performance, yet with some decrease with the increasing instance size.

4.5.3 Solution Quality

In this section, we evaluated the solution quality of the solutions obtained by solving the problem proposed in this Chapter, *full-information problem with failures and reinsertion* (FFR), over dynamic reinsertion simulations by comparing them with the solutions obtained by solving the one-scenario problem and the problem given in Chapter 3, *full-information problem with failures* (FF). We utilized the first part of the TS algorithm given in Section 3.4.2 to solve the one-scenario problem. The complete version of the TS algorithm is employed to solve the FF problem. Finally, the STMLS algorithm is used to solve the FFR problem. We preferred STMLS over LS-NSGA-II due to the reliable performance of STMLS in terms of work overload objective.

The dynamic reinsertion simulations are executed based on the industry application which is to reinsert a reinstating vehicle as soon as a convenient position is found within the sequence. The simulation procedure is explained as follows: we first solved each problem with the corresponding solution method for all 90 instances and for 30 runs, within a time limit of 600 seconds, and with a sample size $N = 100$, note that the sample size is valid only for the full-information problems. Then, another set of 1000 samples is generated as test simulations, and the solutions obtained from each method are simulated over these samples by executing reinsertions dynamically. The reinsertions are made in different work overload thresholds, while also obeying feasibility rules defined in this Chapter. For each solution, each position is checked for each reinstating vehicle, starting from the

first position. Once a convenient position for a reinstating vehicle is found that causes less than or equal to the work overload threshold, reinsertion is made. The simulation for a solution is completed when all the positions are checked and the possible reinsertions are made.

Table 4.4: Comparison of variants of MMS problem over dynamic reinsertion simulations

	One-scenario		Full-information with failures		Full-information with failures and reinsertion	
WO Threshold	Obj WO	Obj RE	Obj WO	Obj RE	Obj WO	Obj RE
0	214.96	113.96	**178.94**	133.15	187.18	**107.15**
3	225.38	90.32	**186.28**	106.39	191.23	**82.33**
5	232.55	81.91	**191.75**	97.16	195.65	**73.51**
10	255.43	50.51	212.27	60.87	**200.48**	**41.36**
15	282.07	19.29	239.47	23.80	**212.45**	**16.32**
30	317.15	1.84	276.34	1.82	**238.15**	**1.52**

Table 4.4 presents the numerical results of the solution quality simulations. The objective function values were averaged across all instances and all runs and for the FFR problem across all solutions in the corresponding non-dominated set. The *Obj WO* and *Obj RE* columns correspond to the work overload and reinsertion objectives, respectively. The results show that the one-scenario problem solutions are outperformed, in terms of work overload objective, by the FF problem by around 17% over the simulations with the work overload threshold of 0, 3, and 5, and by the FFR problem by 22%, 22%, and 25% when the work overload threshold is 10, 15, 30, respectively. It is important to point out that the FF problem solutions outperform the FFR problem solutions, in terms of work overload objective, by around 4% when the work overload threshold is less than 10. It aligns with the problem characteristics. FF problem solutions provide better work overload results when the work overload threshold is low, the reinsertion is not made at the cost of a large work overload sacrifice. It is because the FF problem minimizes work overload over scenarios that consider the failures but not the reinsertions. On the other hand, the FFR problem solutions outperform the FF problem solutions, in terms of work overload, by 6%, 11%, and 14% when the work overload threshold is 10, 15, and 30, respectively. Additionally, FFR problem solutions provide the best reinsertion objective results across all work overload thresholds. Accordingly, we can conclude that the FFR problem can generate good-quality solutions in terms of both objectives, since the motivation of the FFR problem is to generate robust solutions that decrease possible work overloads while performing a high rate of reinsertions.

4.6 Conclusion

This chapter studied MMS problem with stochastic failures and integrated reinsertion process. This is the first study that integrates the reinsertion process into a sequencing problem, to the best of our knowledge. We formulated the proposed problem as a two-stage stochastic program and proposed formulation improvements. Three bi-objective optimization algorithms were presented to tackle the problem. The numerical experiments showed that while the two-stage bi-objective local search algorithm provides reliable solutions in terms of work overload objective, the hybrid local search integrated evolutionary optimization algorithm provides a better exploration of solution space. To assess the quality of the solutions, dynamic reinsertion simulations are executed over industry-inspired instances. The results show that we can reduce the work overload by around 20% while decreasing the waiting time of the failed vehicles drastically.

4.6.1 Managerial Insights

The challenges that car manufacturers face due to the increasing ratio of EVs, which are the main motivation of Chapters 3 and 4, are discussed in Section 3.6.1. A planned sequence is disrupted first by the failure of vehicles, then by the reinsertion of the failed vehicles. In this chapter, we focused on generating robust schedules considering the vehicle failures and the reinsertion process which have an escalating impact on the production efficiency of the assembly lines as the difference between product types expands, currently due to electric vehicles, yet could be any other new development that induces similar results.

The proposed problem offers the potential to generate high-quality solutions that generate a balance between minimizing work overload and waiting time of failed vehicles. This makes the problem a promising approach for real-world operational scenarios where reducing work overload and ensuring efficient reinsertions are critical considerations. By considering both failures and reinsertions, the problem addresses the inherent complexities of operations management, leading to improved performance and enhanced robustness in practical settings.

As aforementioned, the work overload reduction of around 20% results in significant production efficiency and cost-saving enhancements in the assembly line. This is because there will be fewer line stoppages and a decrease in the number of utility workers needed. Additionally, reducing the waiting time of failed vehicles means an increase in the reinsertion ratio, which benefits decision-

makers in two important ways. First, shorter waiting times lead to higher customer satisfaction as the late delivery of failed vehicles decreases. Second, car manufacturers impose a limit on the number of reinstating vehicles, as having a large number of piled-up vehicles disrupts the production line, such as inventory management or buffer issues. Thus, when the number of reinstating vehicles reaches the limit, the standard production stops, and only the reinstating vehicles are produced, resulting in a significant decline in production efficiency. Therefore, an increased reinsertion ratio improves production efficiency by decreasing the likelihood of such a scenario occurring.

Chapter 5

Conclusions and Future Research

In this chapter, we present the highlights and potential future research of each chapter explicitly.

Chapter 2 presented a comparison of local search metaheuristics, namely adaptive local search (ALS), very fast local search (VFLS), variable neighborhood search (VNS), and simulated annealing with geometric cooling with bouncing strategy (SA). We used the sliding-window (SW) technique as the objective function and experimental tests were executed on two sets, including the latest CSPLib and simplified ROADEF challenge 2005 instances. The commercial solvers can provide good enough solutions for CSPLib instances with up to 400 cars and improved lower bounds are provided by solving the relaxed problem using Gurobi. However, the results of larger instances induced a need for a more reliable method.

We suspect that the reason behind the success of ALS and VFLS is that accepting worse solutions causes SA to spend more time in the search process away from the optimal solution which is not necessary for CSP since accepting the solutions with the same objective value creates the advantage of escaping local optima. The bouncing strategy within SA slightly improves the algorithm which proves that spending more time to escape local optima during the search process close to the optimal solution is more efficient. On the other hand, the poor performance of VNS shows that the systematic search of a very large neighborhood is more unfavorable than the random search for this problem. We saw that the VNS is worse on the CSPLib but comparable on ROADEF instances which shows that the time spent on a systematic search of neighborhoods may become

more efficient for larger instances and for more complex versions of the CSP.

Future research may provide a similar comparison of local search metaheuristics with a more complex CSP. Additionally, in CSP literature, population-based metaheuristics are also proven to be efficient, thus a comparison over population-based metaheuristics including genetic algorithm and ant colony optimization should be addressed.

Chapter 3 studied mixed-model sequencing (MMS) problem with stochastic failures. To the best of our knowledge, this is the first study that considers stochastic failures of products in MMS. The products (vehicles) fail according to various characteristics and are then removed from the sequence, moving succeeding vehicles forward to close the gap. Vehicle failure may cause extra work overloads that could be prevented by generating a robust sequence at the beginning. Accordingly, we formulated the problem as a two-stage stochastic program, and improvements were presented for the second-stage problem. We employed the sample average approximation approach to tackle the exponential number of scenarios. We developed L-shaped decomposition-based algorithms to solve small-sized instances. The numerical experiments showed that the L-shaped algorithm outperforms the deterministic equivalent formulation, solved with an off-the-shelf solver, in terms of both solution quality and computational time. To solve industry-sized instances efficiently, we developed a greedy heuristic and a tabu search algorithm that is accelerated with problem-specific tabu rules. Numerical results showed that we can provide good quality solutions, with less than a 5% statistical optimality gap, to industry-sized instances in under ten minutes. The numerical experiments also indicated that we can generate good quality robust solutions by utilizing a sample of scenarios. In particular, we can reduce the work overload by more than 20%, for both small- and large-sized instances, by considering possible car failures.

One direction for future research is to include stochastic processing times in addition to stochastic product failures. This may potentially generate more robust schedules, particularly in case a connection between failures and processing times is observed.

Chapter 4 studied MMS problem with stochastic failures and integrated reinsertion process. This is the first study that integrates the reinsertion process into a sequencing problem, to the best of our knowledge. We formulated the proposed problem as a two-stage stochastic program and proposed formulation improvements. Three bi-objective optimization algorithms were presented to tackle

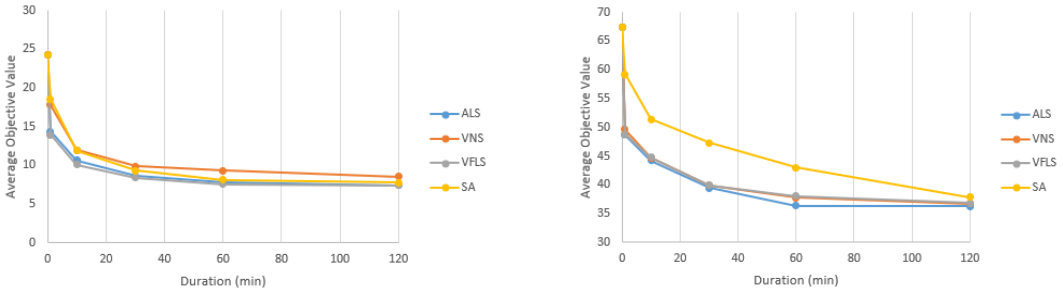
the problem. The numerical experiments showed that while the two-stage bi-objective local search algorithm provides reliable solutions in terms of work overload objective, the hybrid local search integrated evolutionary optimization algorithm provides a better exploration of solution space. To assess the quality of the solutions, dynamic reinsertion simulations are executed over industry-inspired instances. The results show that we can reduce the work overload by around 20% while decreasing the waiting time of the failed vehicles drastically.

There are similarities between MMS and some variants of the traveling salesman problem (TSP). Since the TSP is one of the most studied combinatorial optimization problems, the state-of-art solution methodologies presented for TSP may be adapted to MMS. Refer to our pilot study, given in Appendix D, that provides a mathematical formulation of MMS interpretation as a one-commodity pickup and delivery traveling salesman problem (1-PDTSP).

Appendices

Appendix A Convergence of Local Search Algorithms

In this section, we share convergence plots of the local search algorithms, given in Chapter 2, for two hours. While the y-axis is the runtime, the x-axis is the average objective function value across all 50 trials. Notice that we can justify our one-hour time limit since the most efficient algorithm for both of the datasets, VFLS for CSPLib and ALS for ROADEF, makes a significant improvement during 1 hour. In the next one hour, we do not observe significant improvements, VFLS makes 2% and ALS makes 1% improvement within the last one-hour of experiments for the CSPLib and ROADEF instances, respectively.



(a) CSPLib **(b)** ROADEF

Figure 1: Convergence of Local Search Algorithms

Appendix B Heuristic Approach to Find Optimal DSP Solutions

In this section, we provide a heuristic approach that provides optimal dual variable values for the dual problem formulated in (3.10). First, we calculate the starting positions (z) and work overloads (w) for the given first-stage decisions x for a scenario n. Next, we use algorithm 10 to find an extreme point of DSP. The DSP variables are non-negative and it is a maximization problem. Thus, the variables get a positive value only when the corresponding primal constraint is binding. For each station and for each cycle time, starting from the last cycle to the first one, the algorithm assigns 1 to the π^{sp} and π^{wo} variables to find an extreme point. In the objective, $\pi^{wo}_{kt} - l_k$ is always non-positive since we assume that any processing time is less than or equal to the station length, which means that $\pi^{wo}_{kt}(p^n_{kv}\bar{x}_{vt} - l_k)$ is always non-positive. Accordingly, we maximize our dual problem by minimizing the π^{sp} values when the corresponding constraint in the primal problem is binding. Also, the solution has to be feasible. The algorithm prioritizes the π^{sp}_{kt} variable to assign 1 since the variables π^{sp}_{kt} and π^{wo}_{kt} can not equal 1 at the same time 3.9b. Hence, the algorithm assign $\pi^{wo}_{kt} = 1$ every time constraint 3.2b is not binding and 3.2c is binding because of constraint 3.10b which does not let $\pi^{sp}_{kt} = 1$ if both $\pi^{sp}_{k(t+1)}$ and $\pi^{wo}_{k(t+1)}$ are zero.

Algorithm 10 Find Extreme Dual Values

Input SP solution for scenario n
  for each station k **do**
    if Constraint 3.2b is binding **then**
      $\pi^{sp}_{kT} \leftarrow 1$
    else if Constraint 3.2c is binding **then**
      $\pi^{wo}_{kT} \leftarrow 1$
    end if
    for each cycle from T-1 to 1 **do**
      if ($\pi^{sp}_{k(t+1)} = 1$ **or** $\pi^{wo}_{k(t+1)} = 1$) **and** constraint 3.2b is binding **then**
        $\pi^{sp}_{kt} \leftarrow 1$
      else if Constraint 3.2c is binding **then**
        $\pi^{wo}_{kt} \leftarrow 1$
      end if
    end for
  end for
Output π^{sp}_n, π^{wo}_n

Appendix C Tabu List for Local Search Algorithm

In this section, we explain the tabu rules mentioned in Section 3.4.2.2. Assume that we have two positions is selected for any operator to be applied, $t_1, t_2 | t_1 < t_2$. The tabu movements for each operator is described below under two circumstances; the vehicle at the position t_1 is 1) an electric vehicle 2) is not an electric vehicle.

Swap

1) The position t_2 cannot be a neighbor of an electric vehicle, e.g., the vehicles at the positions $t_2 - 1$ and $t_2 + 1$ cannot be an electric vehicle.

2) The vehicle at the position t_2 cannot be an electric vehicle if the position t_1 is a neighbor of an electric vehicle.

Forward Insertion

1) The vehicles at the positions t_2 or $t_2 - 1$ cannot be an electric vehicle.

2) The both of the vehicles at the positions $t_1 - 1$ and $t_1 + 1$ cannot be electric vehicle.

Backward Insertion

1) The vehicles at the positions t_1 or $t_1 - 1$ cannot be an electric vehicle.

2) The both of the vehicles at the positions $t_2 - 1$ and $t_2 + 1$ cannot be electric vehicle.

Inversion

1) The position t_2 cannot be a left neighbor of an electric vehicle, e.g., the vehicle at the position $t_2 + 1$ cannot be an electric vehicle.

2) If the vehicle at the position t_1 is a right neighbor of an electric vehicle, then the second position cannot be an electric vehicle.

Appendix D MMS Interpretation as TSP

In this section, we interpret MMS as a one-commodity pickup and delivery traveling salesman problem (1-PDTSP) at which each customer provides or requires a given non-zero amount of product, and the vehicle in a depot has a given capacity. Each customer and the depot must be visited exactly once by the vehicle supplying the demand while minimizing the total travel distance. We interpret the work pieces to be produced in MMS as customers to be visited (1-PDTSP), station length as vehicle capacity, starting position of vehicles as load level of vehicles after visiting each customer, and processing time of vehicles minus the cycle time as the demand of the customers. The main difference is that the 1-PDTSP has the objective of minimizing the cost of the tour while MMS minimizes the work overload which would be minimizing vehicle overload for 1-PDTSP. Hence, we alter the vehicle overloading from hard constraint to soft constraint and have it in the objective function,

$$\max \quad \sum_{i\in V}\sum_{j\in V}\sum_{k\in K} w_{ijk} \tag{1}$$

$$\sum_{j\in V} x_{ij} = 1 \qquad \forall i \in V \tag{2}$$

$$\sum_{i\in V} x_{ij} = 1 \qquad \forall j \in V \tag{3}$$

$$(SubtourEliminationConstraints) \tag{4}$$

$$z_{ijk} - w_{ijk} - \sum_{r\in V, r\neq i,j} z_{jrk} - x_{ij} * (c - p_{jk}) \leq 0 \qquad \forall i \in V, \qquad \forall j \in V, \quad \forall k \in K \tag{5}$$

$$z_{ijk} - w_{ijk} - x_{ij} * (L - p_{jk}) \leq 0 \qquad \forall i \in V, \qquad \forall j \in V, \quad \forall k \in K \tag{6}$$

$$z_{0jk} = 0, \qquad \forall k \in K, \quad \forall j \in V \tag{7}$$

$$w_{0jk} = 0, \qquad \forall k \in K, \quad \forall j \in V \tag{8}$$

$$z_{j0k} = 0, \qquad \forall k \in K, \quad \forall j \in V \tag{9}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in V, \qquad \forall j \in V \tag{10}$$

$$z_{ijk}, w_{ijk} \geq 0 \qquad \forall i \in V, \qquad \forall j \in V, \quad \forall k \in K \tag{11}$$

Appendix E Attainment Surface Comparison of NSGA-II, STMLS, and LS-NSGA-II

In this section, we compare the attainment surface of the algorithms proposed in Chapter 4. Figure 2, which is the global counterpart of Figure 4.5, demonstrates that the LS-NSGA-II is superior to the other algorithms in terms of exploration ability.

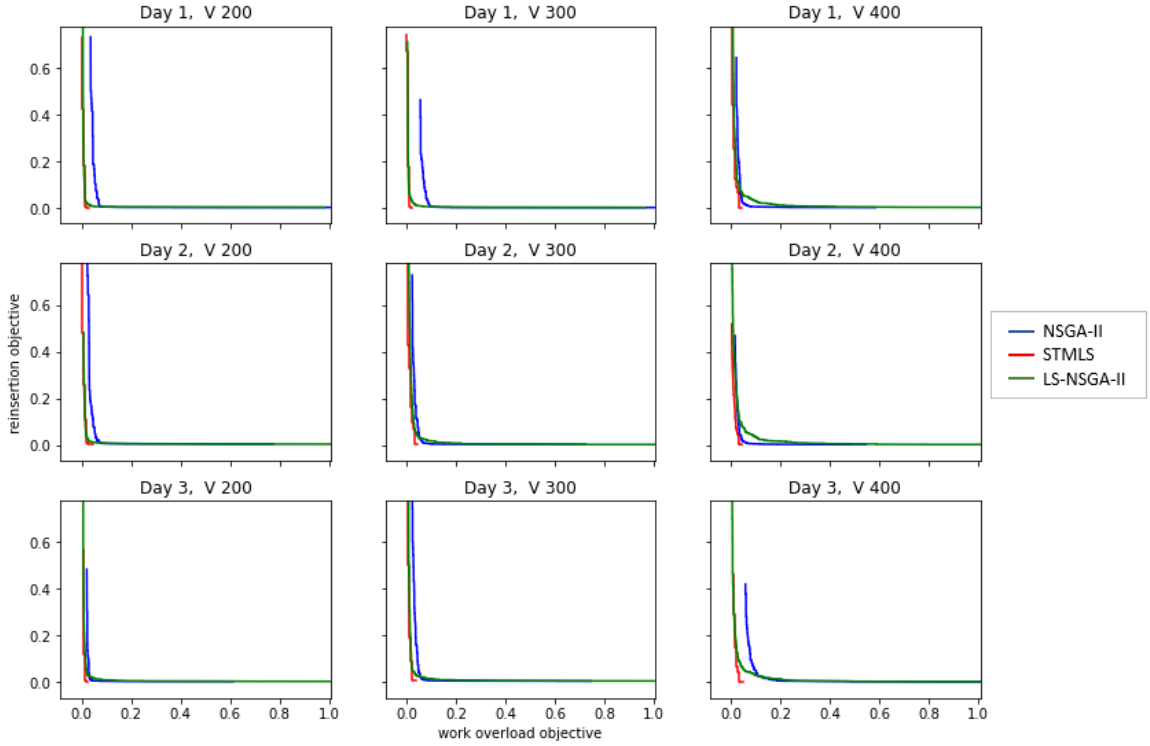


Figure 2: Comparison of the 50%-attainment surfaces of the proposed algorithms, with a 95% confidence level

Bibliography

[1] S Agrawal and MK Tiwari. A collaborative ant colony algorithm to stochastic mixed-model u-shaped disassembly line balancing and sequencing problem. *International journal of production research*, 46(6):1405–1429, 2008.

[2] Onur Serkan Akgündüz and Semra Tunalı. An adaptive genetic algorithm approach for the mixed-model assembly line sequencing problem. *International Journal of Production Research*, 48(17):5157–5179, 2010.

[3] Onur Serkan Akgündüz and Semra Tunalı. A review of the current applications of genetic algorithms in mixed-model assembly line sequencing. *International Journal of Production Research*, 49(15):4483–4503, 2011.

[4] Sener Akpınar, G Mirac Bayhan, and Adil Baykasoglu. Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks. *Applied Soft Computing*, 13(1):574–589, 2013.

[5] Mohammad Alaghebandha, Vahid Hajipour, and Mojtaba Hemmati. Optimizing multi-objective sequencing problem in mixed-model assembly line on just-in-time: particle swarm optimization algorithm. *International Journal of Management Science and Engineering Management*, 12(4):288–298, 2017.

[6] Anas Alghazi and Mary E Kurz. Mixed model line balancing with parallel stations, zoning constraints, and ergonomics. *Constraints*, 23(1):123–153, 2018.

[7] Anas Alsayed Alghazi. *Balancing and sequencing of mixed model assembly lines*. PhD thesis, Clemson University, 2017.

[8] Karim Aroui, Gülgün Alpan, and Yannick Frein. Minimizing work overload in mixed model assembly lines: A case study from truck industry. In *5th International Conference on Information Systems, Logistics and Supply Chain Connecting worlds ILS 2014*, 2014.

[9] Jonathan F Bard, EZEY Dar-Elj, and Avraham Shtub. An analytic framework for sequencing mixed model assembly lines. *The International Journal of Production Research*, 30(1):35–48, 1992.

[10] Joaquín Bautista, Rocío Alfaro, and Cristina Batalla. Modeling and solving the mixed-model sequencing problem to improve productivity. *International Journal of Production Economics*, 161:83–95, 2015.

[11] Joaquín Bautista and Rocío Alfaro-Pozo. A grasp algorithm for quota sequences with minimum work overload and forced interruption of operations in a mixed-product assembly line. *Progress in Artificial Intelligence*, 7(3):197–211, 2018.

[12] Joaquín Bautista and Alberto Cano. Solving mixed model sequencing problem in assembly lines with serial workstations with work overload minimisation and interruption rules. *European Journal of Operational Research*, 210(3):495–513, 2011.

[13] Christian Becker and Armin Scholl. A survey on problems and methods in generalized assembly line balancing. *European journal of operational research*, 168(3):694–715, 2006.

[14] Michael E Bergen, Peter Van Beek, and Tom Carchrae. Constraint-based vehicle assembly line sequencing. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 88–99. Springer, 2001.

[15] Ahmet Bolat. A mathematical model for selecting mixed models with due dates. *International Journal of Production Research*, 41(5):897–918, 2003.

[16] Ahmet Bolat and Candace A Yano. Scheduling algorithms to minimize utility work at a single station on a paced assembly line. *Production Planning & Control*, 3(4):393–405, 1992.

[17] Nils Boysen, Malte Fliedner, and Armin Scholl. A classification of assembly line balancing problems. *European journal of operational research*, 183(2):674–693, 2007.

[18] Nils Boysen, Malte Fliedner, and Armin Scholl. Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373, 2009.

[19] Nils Boysen, Mirko Kiel, and Armin Scholl. Sequencing mixed-model assembly lines to minimise the number of work overload situations. *International Journal of Production Research*, 49(16):4735–4760, 2011.

[20] Nils Boysen, Armin Scholl, and Nico Wopperer. Resequencing of mixed-model assembly lines: Survey and research agenda. *European Journal of Operational Research*, 216(3):594–604, 2012.

[21] Janis Brammer, Bernhard Lutz, and Dirk Neumann. Stochastic mixed model sequencing with multiple stations using reinforcement learning and probability quantiles. *OR Spectrum*, 44(1):29–56, 2022.

[22] Jens Buergin, Sina Helming, Jan Andreas, Philippe Blaettchen, Yannick Schweizer, Frank Bitte, Benjamin Haefner, and Gisela Lanza. Local order scheduling for mixed-model assembly lines in the aircraft manufacturing industry. *Production Engineering*, 12(6):759–767, 2018.

[23] Alberto Cano et al. Modeling and solving a variant of the mixed-model sequencing problem with work overload minimisation and regularity constraints. an application in nissan's barcelona plant. *Expert systems with applications*, 39(12):11001–11010, 2012.

[24] Jaime Cano-Belmán, Roger Z Ríos-Mercado, and Joaquín Bautista. A scatter search based hyper-heuristic for sequencing a mixed-model assembly line. *Journal of Heuristics*, 16:749–770, 2010.

[25] G Celano, S Fichera, V Grasso, U La Commare, and G Perrone. An evolutionary approach to multi-objective scheduling of mixed model assembly lines. *Computers & Industrial Engineering*, 37(1-2):69–73, 1999.

[26] Parames Chutima and Sathaporn Olarnviwatchai. A multi-objective car sequencing problem on two-sided assembly lines. *Journal of Intelligent Manufacturing*, 29(7):1617–1636, 2018.

[27] Gianni Codato and Matteo Fischetti. Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[28] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007.

[29] Ezey M Dar-El. Mixed-model assembly line sequencing problems. *Omega*, 6(4):313–323, 1978.

[30] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[31] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *ECAI*, volume 88, pages 290–295, 1988.

[32] Jietao Dong, Linxuan Zhang, Tianyuan Xiao, and Huachao Mao. Balancing and sequencing of stochastic mixed-model assembly u-lines to minimise the expectation of work overload time. *International Journal of Production Research*, 52(24):7529–7548, 2014.

[33] Andreas Drexl and Alf Kimms. Sequencing jit mixed-model assembly lines under station-load and part-usage constraints. *Management Science*, 47(3):480–491, 2001.

[34] Andreas Drexl, Alf Kimms, and Lars Matthießen. Algorithms for the car sequencing and the level scheduling problem. *Journal of Scheduling*, 9(2):153–176, 2006.

[35] Bertrand Estellon, Frédéric Gardi, and Karim Nouioua. Large neighborhood improvements for solving car sequencing problems. *RAIRO-Operations Research*, 40(4):355–379, 2006.

[36] Bertrand Estellon, Frédéric Gardi, and Karim Nouioua. Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research*, 191(3):928–944, 2008.

[37] Parviz Fattahi and Mohsen Salehi. Sequencing the mixed-model assembly line to minimize the total utility and idle costs with variable launching interval. *The International Journal of Advanced Manufacturing Technology*, 45:987–998, 2009.

[38] Malte Fliedner and Nils Boysen. Solving the car sequencing problem via branch & bound. *European Journal of Operational Research*, 191(3):1023–1042, 2008.

[39] Carlos M Fonseca and Peter J Fleming. On the performance assessment and comparison of stochastic multiobjective optimizers. In *International conference on parallel problem solving from nature*, pages 584–593. Springer, 1996.

[40] Christian Franz, Eric Caap Hällgren, and Achim Koberstein. Resequencing orders on mixed-model assembly lines: Heuristic approaches to minimise the number of overload situations. *International Journal of Production Research*, 52(19):5823–5840, 2014.

[41] Christian Franz, Achim Koberstein, and Leena Suhl. Dynamic resequencing at mixed-model assembly lines. *International Journal of Production Research*, 53(11):3433–3447, 2015.

[42] Caroline Gagné and Arnaud Zinflou. An hybrid algorithm for the industrial car sequecing problem. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[43] Ian P Gent. Two results on car-sequencing problems. *Report University of Strathclyde, APES-02-98*, 7, 1998.

[44] Ian P Gent and Toby Walsh. Csplib: a benchmark library for constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 480–481. Springer, 1999.

[45] Uli Golle, Nils Boysen, and Franz Rothlauf. Analysis and design of sequencing rules for car sequencing. *European Journal of Operational Research*, 206(3):579–585, 2010.

[46] Uli Golle, Franz Rothlauf, and Nils Boysen. Car sequencing versus mixed-model sequencing: A computational study. *European Journal of Operational Research*, 237(1):50–61, 2014.

[47] Uli Golle, Franz Rothlauf, and Nils Boysen. Iterative beam search for car sequencing. *Annals of Operations Research*, 226(1):239–254, 2015.

[48] Jens Gottlieb, Markus Puchta, and Christine Solnon. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In *Workshops on Applications of Evolutionary Computation*, pages 246–257. Springer, 2003.

[49] Marc Gravel, Caroline Gagne, and Wilson L Price. Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56(11):1287–1295, 2005.

[50] Rico Gujjula and Hans-Otto Gunther. Resequencing mixed-model assembly lines under just-in-sequence constraints. In *2009 International Conference on Computers & Industrial Engineering*, pages 668–673. IEEE, 2009.

[51] Elif Elcin Gunay and Ufuk Kula. A stochastic programming model for resequencing buffer content optimisation in mixed-model assembly lines. *International Journal of Production Research*, 55(10):2897–2912, 2017.

[52] Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[53] Tito Homem-de Mello and Güzin Bayraksan. Monte carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science*, 19(1):56–85, 2014.

[54] John Hooker. *Logic-based methods for optimization: combining optimization and constraint satisfaction*. John Wiley & Sons, 2011.

[55] Andreas Hottenrott, Leon Waidner, and Martin Grunow. Robust car sequencing for automotive assembly. *European Journal of Operational Research*, 291(3):983–994, 2021.

[56] Chul Ju Hyun, Yeongho Kim, and Yeo Keun Kim. A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. *Computers & Operations Research*, 25(7-8):675–690, 1998.

[57] Eivind Jahren and Roberto Asín Achá. A column generation approach and new bounds for the car sequencing problem. *Annals of Operations Research*, 264(1-2):193–211, 2018.

[58] Carsten Jordan and Andreas Drexl. A comparison of constraint and mixed-integer programming solvers for batch sequencing with sequence-dependent setups. *ORSA Journal on Computing*, 7(2):160–165, 1995.

[59] M Kilbridge. The assembly line model-mix sequencing problem. In *Proceedings of the 3rd Int. Conf. on Operations Research, 1963*, 1963.

[60] Siwon Kim and Bongju Jeong. Product sequencing problem in mixed-model assembly line to minimize unfinished works. *Computers & Industrial Engineering*, 53(2):206–214, 2007.

[61] Yeo Keun Kim, Chul Ju Hyun, and Yeongho Kim. Sequencing in mixed model assembly lines: a genetic algorithm approach. *Computers & Operations Research*, 23(12):1131–1145, 1996.

[62] Anton J Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

[63] Ibrahim Kucukkoc and David Z Zhang. Simultaneous balancing and sequencing of mixed-model parallel two-sided assembly lines. *International Journal of Production Research*, 52(12):3665–3687, 2014.

[64] Jinling Leng, Xingyuan Wang, Shiping Wu, Chun Jin, Meng Tang, Rui Liu, Alexander Vogl, and Huiyu Liu. A multi-objective reinforcement learning approach for resequencing scheduling problems in automotive manufacturing systems. *International Journal of Production Research*, pages 1–20, 2022.

[65] Aymeric Lesert, Gülgün Alpan, Yannick Frein, and Stephane Noire. Definition of spacing constraints for the car sequencing problem. *International Journal of Production Research*, 49(4):963–994, 2011.

[66] Yow-Yuh Leu, Lance A Matheson, and Loren Paul Rees. Sequencing mixed-model assembly lines with genetic algorithms. *Computers & Industrial Engineering*, 30(4):1027–1036, 1996.

[67] Dong Liu, Qiang Huang, Yuanyuan Yang, Dengfeng Liu, and Xiaoting Wei. Bi-objective algorithm based on nsga-ii framework to optimize reservoirs operation. *Journal of Hydrology*, 585:124830, 2020.

[68] Qiong Liu, Wen-xi Wang, Ke-ren Zhu, Chao-yong Zhang, and Yun-qing Rao. Advanced scatter search approach and its application in a sequencing problem of mixed-model assembly lines in a case company. *Engineering Optimization*, 46(11):1485–1500, 2014.

[69] Thiago Cantos Lopes, Celso Gustavo Stall Sikora, Adalberto Sato Michels, and Leandro Magatão. An iterative decomposition for asynchronous mixed-model assembly lines: combining balancing, sequencing, and buffer allocation. *International Journal of Production Research*, 58(2):615–630, 2020.

[70] Adèle Louis, Gülgün Alpan, Bernard Penz, and Alain Benichou. Mixed-model sequencing versus car sequencing: comparison of feasible solution spaces. *International Journal of Production Research*, pages 1–20, 2022.

[71] Wai-Kei Mak, David P Morton, and R Kevin Wood. Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24(1-2):47–56, 1999.

[72] Valentin Mayer-Eichberger and Toby Walsh. Sat encodings for the car sequencing problem. In *POS@ SAT*, pages 15–27, 2013.

[73] Krzysztof Michalak. Improving the nsga-ii performance with an external population. In *Intelligent Data Engineering and Automated Learning–IDEAL 2015: 16th International Conference, Wroclaw, Poland, October 14-16, 2015, Proceedings 16*, pages 273–280. Springer, 2015.

[74] Seyed Mohammed Mirghorbani, Masoud Rabbani, Reza Tavakkoli-Moghaddam, and Alireza R Rahimi-Vahed. A multi-objective particle swarm for a mixed-model assembly line sequencing. In *Operations Research Proceedings 2006: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Jointly Organized with the Austrian Society of Operations Research (ÖGOR) and the Swiss Society of Operations Research (SVOR) Karlsruhe, September 6–8, 2006*, pages 181–186. Springer, 2007.

[75] Yasuhiro Monden. *Toyota production system: an integrated approach to just-in-time*. CRc Press, 2011.

[76] Hadi Mosadegh, SMT Fatemi Ghomi, and Gürsel A Süer. Heuristic approaches for mixed-model sequencing problem with stochastic processing times. *International Journal of Production Research*, 55(10):2857–2880, 2017.

[77] Hadi Mosadegh, SMT Fatemi Ghomi, and Gürsel A Süer. Stochastic mixed-model assembly line sequencing problem: Mathematical modeling and q-learning based simulated annealing hyper-heuristics. *European Journal of Operational Research*, 282(2):530–544, 2020.

[78] Uğur Özcan, Hakan Çerçioğlu, Hadi Gökçen, and Bilal Toklu. Balancing and sequencing of parallel mixed-model assembly lines. *International Journal of Production Research*, 48(17):5089–5113, 2010.

[79] Uğur Özcan, Talip Kellegöz, and Bilal Toklu. A genetic algorithm for the stochastic mixed-model u-line balancing and sequencing problem. *International Journal of Production Research*, 49(6):1605–1626, 2011.

[80] Bruce Parrello. Car wars:(almost) birth of an expert system. *AI expert*, 3(1):60–64, 1988.

[81] Bruce D Parrello, Waldo C Kabat, and Larry Wos. Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated reasoning*, 2(1):1–42, 1986.

[82] Bryan W Pearce, Kavit Antani, Laine Mears, Kilian Funk, Maria E Mayorga, and Mary E Kurz. An effective integer program for a general assembly line balancing problem with parallel workers and additional assignment restrictions. *Journal of Manufacturing Systems*, 50:180–192, 2019.

[83] Yunfang Peng, Lijun Zhang, Beixin Xia, and Yajuan Han. Research on balancing and sequencing problems of flexible mixed model assembly lines with alternative precedence relations. *International Journal of Production Research*, pages 1–17, 2022.

[84] Frits K Pil and Matthias Holweg. Linking product variety to order-fulfillment strategies. *Interfaces*, 34(5):394–403, 2004.

[85] SG Ponnambalam, P Aravindan, and M Subba Rao. Genetic algorithms for sequencing problems in mixed model assembly lines. *Computers & industrial engineering*, 45(4):669–690, 2003.

[86] Matthias Prandtstetter and Günther R Raidl. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *European Journal of Operational Research*, 191(3):1004–1022, 2008.

[87] Markus Puchta and Jens Gottlieb. Solving car sequencing problems by local optimization. In *Workshops on Applications of Evolutionary Computation*, pages 132–142. Springer, 2002.

[88] Masoud Rabbani, Leyla Aliabadi, and Hamed Farrokhi-Asl. A multi-objective mixed model two-sided assembly line sequencing problem in a make–to-order environment with customer order prioritization. *Journal of Optimization in Industrial Engineering*, 12(2):1–20, 2019.

[89] A Rabbani Rahimi-Vahed, Masoud Rabbani, Reza Tavakkoli-Moghaddam, S Ali Torabi, and Fariborz Jolai. A multi-objective scatter search for a mixed-model assembly line sequencing problem. *Advanced Engineering Informatics*, 21(1):85–99, 2007.

[90] AR Rahimi-Vahed, SM Mirghorbani, and Masoud Rabbani. A new particle swarm algorithm for a multi-objective mixed-model assembly line sequencing problem. *Soft computing*, 11:997–1012, 2007.

[91] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017.

[92] Celso C Ribeiro, Daniel Aloise, Thiago F Noronha, Caroline Rocha, and Sebastián Urrutia. A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints. *European Journal of Operational Research*, 191(3):981–992, 2008.

[93] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[94] J David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the first international conference on genetic algorithms and their applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers, 1985.

[95] Armin Scholl et al. Balancing and sequencing of assembly lines. Technical report, Darmstadt Technical University, Department of Business Administration ..., 1999.

[96] Armin Scholl, Robert Klein, and Wolfgang Domschke. Pattern based vocabulary building for effectively sequencing mixed-model assembly lines. *Journal of Heuristics*, 4(4):359–381, 1998.

[97] Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. A study of constraint programming heuristics for the car-sequencing problem. *Engineering Applications of Artificial Intelligence*, 38:34–44, 2015.

[98] Celso Gustavo Stall Sikora. Benders' decomposition for the balancing of assembly lines with stochastic demand. *European Journal of Operational Research*, 292(1):108–124, 2021.

[99] Panneerselvam Sivasankaran and P Shahabudeen. Literature review of assembly line balancing problems. *The International Journal of Advanced Manufacturing Technology*, 73:1665–1694, 2014.

[100] Barbara Smith. CSPLib problem 001: Car sequencing. `http://www.csplib.org/Problems/prob001`.

[101] Christine Solnon. Solving permutation constraint satisfaction problems with artificial ants. In *ECAI*, pages 118–122. Citeseer, 2000.

[102] Christine Solnon, Alain Nguyen, Christian Artigues, et al. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927, 2008.

[103] Nitin Srinath. A study of scheduling problems with sequence dependent restrictions and preferences. 2022.

[104] George Steiner and Scott Yeomans. Level schedules for mixed-model, just-in-time processes. *Management science*, 39(6):728–735, 1993.

[105] Hui Sun and Shujin Fan. Car sequencing for mixed-model assembly lines with consideration of changeover complexity. *Journal of manufacturing systems*, 46:93–102, 2018.

[106] F Tanhaie, M Rabbani, and N Manavizadeh. Sequencing mixed-model assembly lines with demand management: problem development and efficient multi-objective algorithms. *Engineering Optimization*, pages 1–18, 2020.

[107] Dhananjay Thiruvady, Andreas Ernst, and Mark Wallace. A lagrangian-aco matheuristic for car sequencing. *EURO journal on computational optimization*, 2(4):279–296, 2014.

[108] Dhananjay Thiruvady, Kerri Morgan, Amiza Amir, and Andreas T Ernst. Large neighbourhood search based on mixed integer programming and ant colony optimisation for car sequencing. *International Journal of Production Research*, 58(9):2696–2711, 2020.

[109] Erlendur S Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *International conference on principles and practice of constraint programming*, pages 16–30. Springer, 2001.

[110] Li-Hui Tsai. Mixed-model sequencing to minimize utility work and the risk of conveyor stoppage. *Management science*, 41(3):485–495, 1995.

[111] Penghong Wang, Jianrou Huang, Zhihua Cui, Liping Xie, and Jinjun Chen. A gaussian error correction multi-objective positioning model with nsga-ii. *Concurrency and Computation: Practice and Experience*, 32(5):e5464, 2020.

[112] Terry Warwick and Edward PK Tsang. Tackling car sequencing problems using a generic genetic algorithm. *Evolutionary Computation*, 3(3):267–298, 1995.

[113] LIU Wei-qi, LIU Qiong+, ZHANG Chao-yong, and SHAO Xin-yu. Hybrid particle swarm optimization for multi-objective sequencing problem in mixed model assembly lines. *Computer Integrated Manufacturing System*, 17(12):0, 2011.

[114] Chuanbo Xu, Yiming Ke, Yanbin Li, Han Chu, and Yunna Wu. Data-driven configuration optimization of an off-grid wind/pv/hydrogen system based on modified nsga-ii and critic-topsis. *Energy Conversion and Management*, 215:112892, 2020.

[115] Candace Arai Yano and Ram Rachamadugu. Sequencing to minimize work overload in assembly lines with product options. *Management science*, 37(5):572–586, 1991.

[116] I Ozan Yilmazlar, Adarsh Jeyes, Alexis Fiore, Apurva Patel, Chelsea Spence, Chase Wentzky, Nicole Zero, Mary E Kurz, Joshua D Summers, and Kevin M Taaffe. A case study in line balancing and simulation. *Procedia Manufacturing*, 48:71–81, 2020.

[117] I Ozan Yilmazlar and Mary E Kurz. Adaptive local search algorithm for solving car sequencing problem. *Journal of Manufacturing Systems*, 2023.

[118] I Ozan Yilmazlar, Mary E Kurz, and Hamed Rahimian. Mixed-model sequencing with stochastic failures: A case study for automobile industry. *arXiv preprint arXiv:2306.12618*, 2023.

[119] Beikun Zhang, Liyun Xu, and Jian Zhang. A multi-objective cellular genetic algorithm for energy-oriented balancing and sequencing problem of mixed-model assembly line. *Journal of Cleaner Production*, 244:118845, 2020.

[120] Xiangyang Zhang, GAO Liang, WEN Long, and Zhaodong Huang. Parallel construction heuristic combined with constraint propagation for the car sequencing problem. *Chinese Journal of Mechanical Engineering*, 30(2):373–384, 2017.

[121] Xiaobo Zhao, Jianyong Liu, Katsuhisa Ohno, and Shigenori Kotani. Modeling and analysis of a mixed-model assembly line with stochastic operation times. *Naval Research Logistics (NRL)*, 54(6):681–691, 2007.

[122] Qiong Zhu and Jie Zhang. Ant colony optimisation with elitist ant for sequencing problem in a mixed model assembly line. *International Journal of Production Research*, 49(15):4605–4626, 2011.

[123] Arnaud Zinflou, Caroline Gagné, and Marc Gravel. Crossover operators for the car sequencing problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 229–239. Springer, 2007.

[124] Arnaud Zinflou, Caroline Gagné, and Marc Gravel. Genetic algorithm with hybrid integer linear programming crossover operators for the car-sequencing problem. *INFOR: Information Systems and Operational Research*, 48(1):23–37, 2010.

[125] Arnaud Zinflou, Caroline Gagné, and Marc Gravel. Gismoo: A new hybrid genetic/immune strategy for multiple-objective optimization. *Computers & Operations Research*, 39(9):1951–1968, 2012.

[126] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.