On the Car Sequencing Problem: Analysis and Solution Methods

Dissertation
zur Erlangung des Grades eines Doktors der
wirtschaftlichen Staatswissenschaften
(Dr. rer. pol.)
des Fachbereichs Rechts- und Wirtschaftswissenschaften
der Johannes Gutenberg-Universität Mainz

vorgelegt von
Dipl.-Wirt.-Inf. Uli Golle
in Mainz

im Jahre 2011

Tag der mündlichen Prüfung: 19.10.2011

Für Tamara

Contents

List of Papers

- Uli Golle[1], Prof. Dr. Nils Boysen[2], Prof. Dr. Franz Rothlauf[1] (2010). Analysis and design of sequencing rules for car sequencing. *European Journal of Operational Research*, 206 (3), pp. 579-585
- Uli Golle[1], Prof. Dr. Franz Rothlauf[1], Prof. Dr. Nils Boysen[2] (2011). Car sequencing versus mixed-model sequencing: A computational study.
- Prof. Dr. Nils Boysen[2], Uli Golle[1], Prof. Dr. Franz Rothlauf[1] (2011). The car resequencing problem with pull-off tables. *BuR - Business Research*, 4 (2), pp. 1-17
- Uli Golle[1], Prof. Dr. Franz Rothlauf[1], Prof. Dr. Nils Boysen[2] (2011). Iterative beam search for car sequencing. *submitted to Annals of Operations Research*
- Uli Golle[1] (2011). Fitness landscape analysis and design of metaheuristics for car sequencing. *submitted to IEEE Transactions on Evolutionary Computation*

[1]Johannes Gutenberg-Universität Mainz, Lehrstuhl für Wirtschaftsinformatik und BWL, Jakob-Welder-Weg 9, D-55128 Mainz, Germany
[2]Friedrich-Schiller-Universität Jena, Lehrstuhl für Operations Management, Carl-Zeiß-Straße 3, D-07743 Jena, Germany

List of Figures

List of Tables

Chapter 1

Introduction

"Any customer can have a car painted any color that he wants so long as it is black"

Henry Ford

At the time of the famous remark by Henry Ford about his model T, the emergence of assembly lines marked the beginning of mass production. Due to standardized commodities, specialized machinery and the division of labour on the line, economies of scale could be realized, leading to falling production costs and, therefore, low product selling prices. However, the widely transition from sellers' to buyers' markets in various economic branches (e.g., the automitive industry) throughout the second half of the 20th century lead to an increasing demand for products customized to individual needs at still low prices. The concept of mass customization emerged (Pine, 1993). Mass customization implies that manufacturers offer an increased variety of their products, but still employ the efficiency of mass production, as all variants of a common base product are jointly manufactured on the same assembly line, also referred to as mixed-model assembly line[1]. Common examples of manufacturers using mass customization can be found in the built-to-order electronic as well as automotive industry. Especially in the latter, the product variety is large and ever increasing. For instance, already in the 1980's approximately 2.5 mio unique configurations of the Ford Escort (Weiner, 1985) and 20 mio variants of the Volkswagen Golf (Dichtl et al., 1983) could be ordered by customers. In 2004, BMW theoretically provided up to 10^{32} different variants (Meyr, 2004). The high variety implies a heterogeneous model mix in production, for example, out of 1.1 mio Mercedes A-Class built between the years 2003 and 2005 in the plant in Raststatt/Germany only two cars were identical (Schlott, 2005).

As the variants are differentiated by a number of selected options, they require different assembling tasks at the stations of the line. To deal with the variety in production, the mixed-model assembly production involves various

[1] Assembly lines where variants of more than one base product are jointly produced are called multi-model assembly lines

planning problems. The long-term decision of assigning assembly tasks to stations of the line is called the assembly line balancing problem (Baybars, 1986; Becker and Scholl, 2006). Based on forecasts for the demand of each variant, the objective is to balance the work content among all stations while maintaining precedence relations between tasks. When feasible line balances are known, the medium-term problem of assigning incoming orders to lines and production periods (usually days or shifts) is referred to as order selection or master scheduling problem (Hindi and Ploszajski, 1994; Bolat, 2003). It primarily aims at minimizing the deviation between the production date and the due date of each order to avoid inventory costs or penalties due to late delivery. The order selection problem results in production plans for each line and planning period and is generally repeated in rolling horizons. After production plans are defined, the short-term sequencing problem determines an optimal production sequence of the orders assigned to the same planning period and line (Boysen et al., 2009). Since the assembly line is balanced to an average variant, the objective is to find a sequence of orders with no peaks in the work load and material demand throughout the production period. Unforeseen events, like rush orders or machine breakdowns, can stir the initially planned sequence and necessitate a resequencing. The short-term resequencing problem either follows the same objectives as the sequencing problem or tries to restore the original planned sequence while considering additional constraints such as buffer capacities (Bolat, 2003).

The literature on the sequencing problem of mixed-model assembly lines deals with various approaches, which differ in their pursued objective as well as their level of detail (Boysen et al., 2009). Two general objectives for sequencing are distinguished, material supply-oriented and workload-oriented objectives. The first one is related to the just-in-time principle. Since an assembly line is in general coupled with preceding levels of the supply chain, which deliver required materials for production just-in-time or just-in-sequence to the line (Meyr, 2004), the model sequence directly influences the material supply. In order to avoid peaks in the material demand and, therefore, large safety stocks of parts and supplies, the material supply-oriented objective focuses on finding a sequence with an even material demand over the whole production planning period. This results in the level scheduling approach, which is mainly followed by Asian car manufacturers (Monden, 1998).

The second objective considered for sequencing is the minimization of work overload and is primarily employed by European car manufacturers. It deals with the different work load induced by the variants. Since the available assembly time in a station is based on an average variant, a sequence of consecutive work-intensive variants can lead to work overload of the respective line operators. Thus, an operator can not finish his/her assigned tasks within the limits of a station. Work overload has to be compensated by other strategies such

as employing additional utility workers, stopping the line in order to allow the operator to finish the tasks or remove the model from the sequence, finish the tasks offline, and resequence it in a later position. Two approaches for the minimization of work overload have been proposed in the literature and can be distinguished by their level of detail. The mixed-model sequencing (MMS) problem (Wester and Kilbridge, 1964) explicitly addresses the minimization of work overload in its objective function. Therefore, it considers different operational characteristics of the line, such as station lengths, operator movements, processing times etc., and results in a detailed time schedule for production.

The scope of this thesis is on the second approach, the car sequencing (CS) problem (Parrello et al., 1986). CS is originally derived from practical applications in the automotive industry and is still employed by automobile original equipment manufacturers such as Renault (Solnon et al., 2008). Compared to MMS, it is a more aggregated approach for sequencing models on a mixed-model assembly line, as it only implicitly considers the minimization of work overload. Instead, CS controls the occurrence of selected options (e.g., airconditioning or sunroof in case of an automobile) accompanied with the models in the sequence by applying so-called sequencing rules. A sequencing rule for each option is defined by $H : N$ (pronounced 'H out of N') and restricts the occurrence of the respective option to at most H in any subsequence of N successive models. If more than H out of N models contain the respective option, a violation occurs. The aim is to find a sequence of models, which respects all sequencing rules - or, if not existent - which minimizes the number of sequencing rule violations.

The literature implicitly assumes the minimization of sequencing rule violations being related to the minimization of work overload. However, even though CS is widely adopted, both in research and practical applications, it has never been examined, if this assumption applies. Since CS uses a surrogate objective, instead of explicitly addressing the minimization of work overload like MMS, several simplifying modeling assumptions have to be made, when using CS. For instance, by applying a sequencing rule of type $H : N$ for each option, a station on the line is allowed to assemble only one option at once, as no interaction between options in a station can be represented. Furthermore, an option in CS is distinguished in merely two states, variants can contain the option or not. Yet, it is unknown how the simplifications made by CS affect its solution quality. Thus, on the one hand side, researchers and practitioners can not estimate the consequences of using CS, and on the other hand are faced with the question why applying CS instead of MMS at all.

1.1 Purpose of the thesis

The purpose of this thesis is to shed some light on the CS problem and its suitability for workload-oriented sequencing as well as to adjust it to the resequencing problem. Furthermore, this work aims at developing different exact and metaheuristic solution methods in order to solve the CS problem.

The first step in analyzing the suitability of CS is to quantify its solution quality in comparison to the direct workload-oriented approach of MMS. Therefore, this work considers CS as both, a constraint satisfaction problem as well as a combinatorial optimization problem, and adresses the analysis and the design of appropriate sequencing rules and objective functions with regard to CS' underlying goal of minimizing the amount of work overload. Thereby, a new approach for generating effective sequencing rules is developed. In order to increase the applicability of CS for practical problem settings, the incorporation of multistate options into the CS model and their effect on the solution quality are shown.

At present, CS is only applicable for the sequencing problem although its objective is also pursued in resequencing scenarios as well. Since the resequencing problem alters an already existing sequence during production by using special buffers, it faces additional contraints compared to sequencing. Thus, this work adjusts the CS model to a resequencing problem with so-called pull-off tables, which are direct accessible buffers that are often found in the automotive industry. The resulting car resequencing (CRS) problem is formalized and applied in a real world example.

This thesis also develops state-of-the-art exact and metaheuristic solution methods to solve both, CS and CRS, with the focus being on CS. Therefore, a new graph representation of CS is presented and existing lower bounds are improved. Both are applied by an exact iterative beam search (IBS) algorithm, which is based on a truncated breadth-first search. Furthermore, the results of a fitness landscape analysis of a set of frequently applied CS instances are used to develop two efficient metaheuristics for CS, a variable neighborhood search (VNS) as well as a memetic algorithm (MA), which is a conjunction of an evolutionary algorithm with local search.

1.2 Structure of the thesis

This thesis by publication is composed of five articles, that are partly published in or submitted to scientific journals. The structure of the thesis is as follows.

Chapter 2 studies CS as a constraint satisfaction problem and analytically as well as empirically analyzes the solution quality of CS. Thus, it examines to what extend CS feasible solutions are feasible in MMS and vice versa. A

new sequencing rule generation approach for CS is developed in Section 2.4 increasing the solution quality by introducing multiple sequencing rules per option. Furthermore, Section 2.5 shows how multistate options (options having more than two processing times at a station) can be incorporated into the CS model.

Chapter 3 examines the solution quality of CS compared to MMS when both are considered as combinatorial optimization problems. Thereby, two sequencing rule generation approaches as well as three different objective functions from the literature of CS are applied. Furthermore, a new weighting factor to distinguish between option violations in the objective function of CS is introduced and its benefit is assessed in experiments as well. The evaluation covers the linear relationship of CS and MMS objective functions and a comparison of the resulting amount of work overload induced by solutions of both approaches.

Chapter 4 outlines the adaption of CS for the resequencing problem using pull-off tables. The resulting CRS problem is formalized as binary linear program (Section 4.2) and transformed into a shortest path problem (Section 4.3), for which different exact and metaheuristic solution approaches are proposed in Section 4.4. An additional lower bound and dominance rule are introduced. The performance of the solution approaches are compared to the commercial standard solver CPlex and the effect of varying the number of pull-off-tables is assessed in experiments. Finally, CRS is applied to a real-world resequencing scenario of a major German truck manufacturer.

In Chapter 5, the graph representation of Chapter 4, is adapted to the CS problem. Existing lower bounds in the literature of CS are improved and applied. The resulting shortest path problem is solved by an efficient iterative beam search algorithm (IBS) which supplementary uses two different utilization rates in order to guide the search. Experiments are conducted on widely used CS instances from the CSPLib and the performance of IBS is compared to the currently best known exact solution approach for CS, a scattered branch-and-bound algorithm.

Chapter 6 studies the fitness landscapes of a set of CS problem instances induced by four different neighborhood operators and three distance metrics. Thereby, the autocorrelation as well as fitness-distance correlation is analyzed and the results are used to design two metaheuristics for CS, a VNS and a MA. The efficiency of the proposed algorithms is shown in experiments on instances of the CSPLib.

Finally, Chapter 7 summarizes and concludes the thesis.

Chapter 2

Analysis and design of sequencing rules for car sequencing

Uli Golle, Nils Boysen, Franz Rothlauf

Abstract

This paper presents novel approaches for generating sequencing rules for the car sequencing (CS) problem in cases of two and multiple processing times per station. The CS problem decides on the succession of different car models launched down a mixed-model assembly line. It aims to avoid work overloads at the stations of the line by applying so-called sequencing rules, which restrict the maximum occurrence of labor-intensive options in a subsequence of a certain length. Thus to successfully avoid work overloads, suitable sequencing rules are essential. The paper shows that the only existing rule generation approach leads to sequencing rules which misclassify feasible sequences. We present a novel procedure which overcomes this drawback by generating multiple sequencing rules. Then, it is shown how to apply both procedures in case of multiple processing times per station. For both cases analytical and empirical results are derived to compare classification quality.

2.1 Introduction

Mixed-model assembly lines allow car manufacturers to produce a large variety of different models of a common base product on a single production line. The sequence of models is important since it affects economic parameters. A major cost driver are work overloads of assembly workers, which can occur if several labor-intensive models are scheduled consecutively on the line. Work overloads

need to be compensated by cost-intensive strategies, e.g., application of utility workers or line stoppage. Therefore, car manufacturer are interested in finding model sequences with minimum work overload. Common approaches are car sequencing (CS) and mixed-model sequencing (MMS).

CS (Parrello et al., 1986; Solnon et al., 2008) introduces sequencing rules $H_o : N_o$ for each labor-intensive option o, which restrict the occurrence of this option to at most H_o in any subsequence of N_o successive models. The goal is to find a sequence, which does not violate any of the given sequencing rules or – if such a sequence is not existent – minimizes rule violations. The alternative MMS approach (Wester and Kilbridge, 1964) evaluates a sequence by explicitly considering operation times, worker movements, station borders, and other operational characteristics of a line. This way, work overloads can exactly be quantified and, thus, be minimized. A sequence is called feasible if no work overload occurs during the execution of the sequence and unfeasible otherwise.

Both, MMS and CS, try to minimize work overload. MMS directly determines the work overload resulting into high effort for data collection, data preciseness, and computation time. In contrast, CS, which is based on a surrogate objective for work overload by applying sequencing rules, is simple to apply and data requirements are low. However, if the applied sequencing rules are not suitable, CS can be less accurate than MMS and wrongly classify sequences to be either feasible or unfeasible. Therefore, the generation of adequate rules for CS is important. The CS research mainly focuses on the development of efficient solution procedures (Solnon et al., 2008; Boysen et al., 2009) for finding optimal sequences. Only little work deals with the definition of sequencing rules. Drexl and Kimms (2001) provide a rather intuitive example:

> Assume that 60% of the cars manufactured on the line need the option "sun roof". Moreover, assume that five cars (copies) pass the station where the sun roofs are installed during the time for the installation of a single copy. Then, three operators (installation teams) are necessary for the installation of sun roofs. Hence, the capacity constraint of the final assembly line for the option "sun roof" is three out of five in a sequence, or 3:5 for short.

Bolat and Yano (1992) presented the only analytical approach on how to derive sequencing rules. It is limited to cases where two different processing times occur at each station. In this paper, we study how well sequencing rules obtained by the Bolat and Yano (BY) approach correctly classify sequences to be either feasible or unfeasible. Since our analysis shows that a large percentage of feasible sequences are classified as unfeasible, we develop a multiple sequencing rule approach (MSR) that generates CS rules that correctly classify sequences as feasible and unfeasible, respectively. Finally, we consider multiple

processing times per station, discuss how the BY and MSR approach can be used for such scenarios, and study how well sequences are correctly classified.

Section 2.2 introduces fundamental assumptions and defines measures for the classification quality of sequencing rules. Then, we study the classification quality of the BY approach (Sect. 2.3) as well as the novel MSR approach (Sect. 2.4). Section 2.5 presents different possibilities of how to develop sequencing rules for stations with multiple processing times and studies how well the BY and MSR approach correctly classifies sequences to be either feasible or unfeasible. The paper ends with concluding remarks.

2.2 Assumptions and classification quality

We introduce fundamental assumptions of the mixed-model assembly line and define measures on the classification quality of sequencing rules.

- The model-mix, i.e., the demand for models throughout the planning horizon, is known with certainty. Thus, rush orders or breakdowns do not occur, so that only static sequencing problems are considered.
- Workpieces are moved with constant velocity through the station which are successively arranged along the line. W.l.o.g. lines flow from left to right. We assume no buffers between stations.
- Fixed rate launching is applied, so that consecutive units are placed on the line at the same intervals equal to cycle time c.
- We assume closed stations, so working across the stations' boundaries is not possible.
- Assembly workers return with infinite velocity to the next workpiece. This is an adequate simplification whenever the conveyor speed is much slower than the walking speed of workers. Otherwise, cycle time can ex ante be reduced by a constant return time. Furthermore, processing starts instantaneously once worker and workpiece meet inside a station.
- We assume a deterministic problem since processing times p_{mk} per model $m \in M$ and station $k \in K$ are known with certainty. Moreover, all possible models can be processed inside a station when starting work at the left-hand border: $p_{mk} \leq l_k \; \forall \; m \in M; \; k \in K$, with l_k being the length of station k. Otherwise, sequence-independent work overload exists and a station inevitably is overloaded with any occurrence of the respective model. Cycle times range in between the minimum and maximum processing times at each station: $\min_{m \in M}\{p_{mk}\} \leq c \leq \max_{m \in M}\{p_{mk}\} \forall \, k \in K$.

- Work overload occurs whenever an assembly operator is not able to finish his/her present workpiece (with normal processing velocity) before reaching the right-hand station border. Then, in the real-world some kind of compensation, e.g., line stoppage or applying cross-trained utility workers, is required.

These assumptions define the status of an assembly system for a given model sequence over the complete planning horizon. Therefore, for any given sequence, MMS can accurately quantify the resulting work overload and correctly classify a sequence to be either feasible or unfeasible.

Throughout the paper, we restrict ourselves to feasibility problems and label feasible sequences as *MMS-feasible*. If a sequence does not violate any sequencing rule of a CS approach, it is denoted as *CS-feasible*. Both approaches, CS and MMS, are equivalent if CS-feasibility induces MMS-feasibility (CS-feasible$\rightarrow$MMS-feasible) and vice versa (MMS-feasible$\rightarrow$CS-feasible). For CS, such a one-to-one mapping is desirable since it correctly classifies all sequences to be either feasible or unfeasible. However, there are two possible types of misclassifications:

- CS-feasible$\nrightarrow$MMS-feasible. The sequencing rules of CS classify sequences to be feasible although they are unfeasible. Sequencing rules are not strict enough and do not identify all sequences that cause work overload. This case causes major problems in the real-world since additional costs for dealing with unforeseen work overloads occur.

- CS-feasible$\nleftarrow$MMS-feasible. CS classifies sequences to be unfeasible although they are feasible. There are sequences that cause no work overload but violate at least one sequencing rule. In this case, no unforeseen work overloads occur but CS excludes feasible sequences from consideration. Obviously, this impedes the search for any solution procedure and potentially excludes optimal solutions if the feasibility version of CS is coupled with an additional objective function (Drexl and Kimms, 2001). As worst case scenario, CS wrongly classifies all feasible solutions, so no feasible solution can be found.

The lower the percentage of misclassifications the better the classification quality of the applied sequencing rules and, thus, the better the rule generation approach that lead to this rules.

2.3 Classification quality of the Bolat and Yano approach

To our best knowledge, the only analytical approach on how to derive sequencing rules was proposed by Bolat and Yano (1992). They assumed one

option per station with two processing times. All models containing the option (denoted as *option models*) require processing time p^+ and all basic models without the option require processing time p^-, with $p^- < c < p^+ \leq l$, where c is the cycle time and l the station length. Bolat and Yano proposed to generate sequencing rules $H : N$ that restrict the number of option models to at most H within a subsequence of N consecutive models, where

$$H = \left\lfloor \frac{l-c}{p^+ - c} \right\rfloor \text{ and} \tag{2.1}$$

$$N = H + \left\lceil \frac{H \cdot (p^+ - c)}{c - p^-} \right\rceil . \tag{2.2}$$

H are the maximum possible number of consecutive option models without outreaching the right-hand station border. N adds to H the number of basic models required to reset the workers starting point to the left-hand station border after H successive option models.

For an example with $l = 15$, $c = 10$, $p^+ = 12$ and $p^- = 7$, the BY approach returns the sequencing rule $2 : 4$. Therefore, at most $H = 2$ option models can be produced consecutively, before $N - H = 2$ basic models are required to reset the subsequence to the left border again. The following properties of the BY approach hold:

Proposition 2.1. *For the BY approach, CS-feasible→MMS-feasible.*

Proof: A sequence is CS-feasible if the $H : N$-rule is not violated (i) inside any subsequence of N cars and (ii) when concatenating subsequences to form a longer sequence. In the worst case, all H option models succeed in a row. In a sequence of N cars, the worker starts processing a subsequence of H consecutive option models at the left-hand border and ends at $H(p^+ - c)$. According to (2.1), this end lies before station's length l. (2.2) calculates the sequence length N such that a sequence of $N - H$ basic models resets position from the right most point $H(p^+ - c)$ back to the left-hand border. Therefore, (i) holds. If only feasible subsequences are concatenated, there is no interplay between the subsequences and (ii) holds. □

Proposition 2.2. *For the BY approach, CS-feasible↚MMS-feasible.*

Proof: We prove by giving a counterexample to the contradiction of our proposition: CS-feasible←MMS-feasible. For the above example with $l = 15$, $c = 10$, $p^+ = 12$ and $p^- = 7$, the sequence $\pi =< +, +, -, +]$ is MMS-feasible (compare movement diagram of Figure 2.1). "+" and "-" represents an option model and basic model, respectively. In the movement diagram, workers accompanying

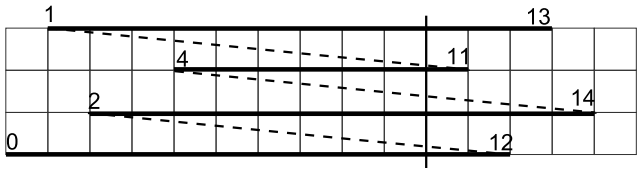


Figure 2.1: Counterexample for CS-feasible ← MMS-feasible

cycle time c	$\in [10, 20]$
station length l	$\in (c, 40]$
processing time of option models p^+	$\in (c, l]$
processing time of basic models p^-	$\in [1, c)$

Table 2.1: Parameter settings

their workpiece are solid horizontal lines; return movements are dashed diagonal lines. The BY approach yields a $2:4$-rule, which is violated by sequence π. Thus, π is MMS-feasible but not CS-feasible and the proposition holds. □

A MMS-feasible sequence is only considered by the BY approach as CS-feasible, if it allows the worker to reset to the left-hand border again after processing at most N consecutive models. Feasible sequences where this is not the case (see example in Figure 2.1) violate at least one sequencing rule and won't be found by CS applying the BY rules. Although CS-feasible↚MMS-feasible holds in general, there are two special cases for which CS-feasible←MMS-feasible holds:

1. $H = 1$ and $l - p^+ < r$, with

$$r = \begin{cases} (p^+ - c) \mod (c - p^-) & \text{if } (p^+ - c) \mod (c - p^-) > 0, \\ c - p^- & \text{if } (p^+ - c) \mod (c - p^-) = 0. \end{cases}$$

   After each option model, at least $N - H$ basic models must be processed before the next option model can follow. Therefore, every feasible sequence allows the worker to reset after at most N successive models.

2. $N - H = 1$.
   The worker resets immediately to the left-hand border by processing one basic model. Since only H option models can be processed consecutively without inducing work overload, every feasible sequence allows the worker to reset after at most $H + 1 = N$ successive models.

For the one and two station case, we study the number of sequences in percent for which CS-feasible↚MMS-feasible holds. For each sequence length

T, we create $1,000$ random problem instances. The parameters settings c, l, p^+, and p^- of each instance are chosen randomly according to Table 2.1. For one station (two stations) and $T \leq 22$ (≤ 12), we consider all possible sequences by enumeration; for larger T, at least $1,000,000$ random sequences are generated for each instance. If necessary, the sampling size is increased until it contains at least 10 MMS-feasible sequences.

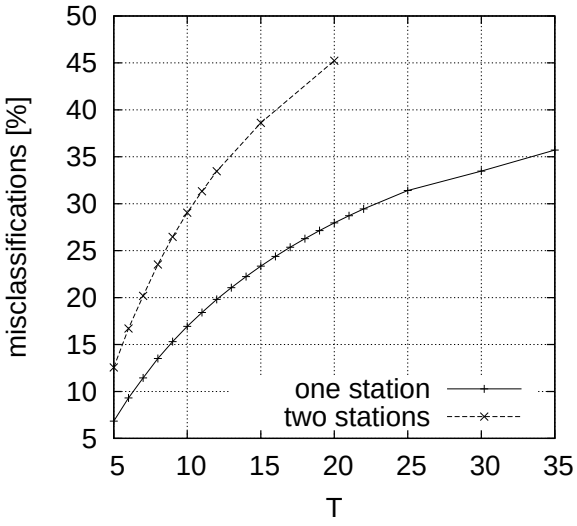


Figure 2.2: Number of misclassified sequences where CS-feasible↚MMS-feasible (in %)

For each sequence, we determine the CS-feasibility using the sequencing rules obtained by the BY approach as well as the MMS-feasibility. Figure 2.2 shows the number of sequences (in percent) that are MMS-feasible but CS-unfeasible. For example, when considering only one station and a sequence length of $T = 30$, then about 33% of MMS-feasible solutions are classified by the BY approach as unfeasible. The percent of sequences where CS-feasible↚MMS-feasible increases with T and with the number of stations.

In summary, all sequences that are CS-feasible are also MMS-feasible. However, a large portion of MMS-feasible solutions are incorrectly classified by the sequencing rules obtained from the BY approach to be CS-unfeasible. Therefore, using this sequencing rules in optimization approaches that search for feasible solutions can be problematic since many feasible solutions are excluded from the search space.

2.4 MSR - A novel rule generation approach

We propose a new rule generation approach with multiple sequencing rules (MSR) for stations with one option and two processing times. The new approach generates a number of sequencing rules which correctly classifies sequences to be either feasible or unfeasible.

The MSR approach calculates k_{min}, which is the maximum number of successive option models, as

$$k_{min} = \left\lfloor \frac{l-c}{p^+ - c} \right\rfloor . \tag{2.3}$$

k_{min} equals H from the BY approach (2.1). Consider a sequence of length T with k option models and $T-k$ basic models. After processing the whole sequence a worker can be at most $l-c$ time units away from the left-hand station border without inducing work overload. Thus, $l-c \geq k(p^+-c)-(T-k)(c-p^-)$ holds. Rewriting this inequation leads to the maximum number of option models k_{max} that may occur in a sequence of length T:

$$k_{max} = \left\lfloor \frac{T(c-p^-)+(l-c)}{p^+ - p^-} \right\rfloor . \tag{2.4}$$

MSR generates $k_{max}-k_{min}+1$ different sequencing rules $H^{k-k_{min}+1} : N^{k-k_{min}+1}$ with

$$H^{k-k_{min}+1} = k \text{ and } \; N^{k-k_{min}+1} = k+m \tag{2.5}$$

$\forall k \in [k_{min}, k_{max}]$ and

$$m = \left\lceil \frac{k(p^+-c)-(l-p^+)}{c-p^-} \right\rceil . \tag{2.6}$$

Eq. (2.6) is a modification of (2.2) of the BY approach and calculates the minimum number of basic models m that are required after k option models before another option model can be processed. For an example with $l=17$, $c=10$, $p^+=13$, and $p^-=5$, the BY approach returns a $2:4$ rule. For a sequence of $T=4$, MSR generates two rules $H^1 : N^1 = 2:3$ and $H^2 : N^2 = 3:4$. A sequence that satisfies all rules generated by MSR is CS-feasible. For the situation that three option models (+) and one basic model (-) need to be assembled, Figure 2.3 shows the four different sequences $(\pi_1, \ldots, \pi_4)$ which are correctly classified by the MSR approach to be either feasible or unfeasible. The BY approach would wrongly classify all four sequences as CS-unfeasible.

Proposition 2.3. *For the MSR approach, CS-feasible → MMS-feasible.*

Proof: We prove by contraposition and show CS-unfeasible←MMS-unfeasible. A MMS-unfeasible sequence contains at least one option model at position j

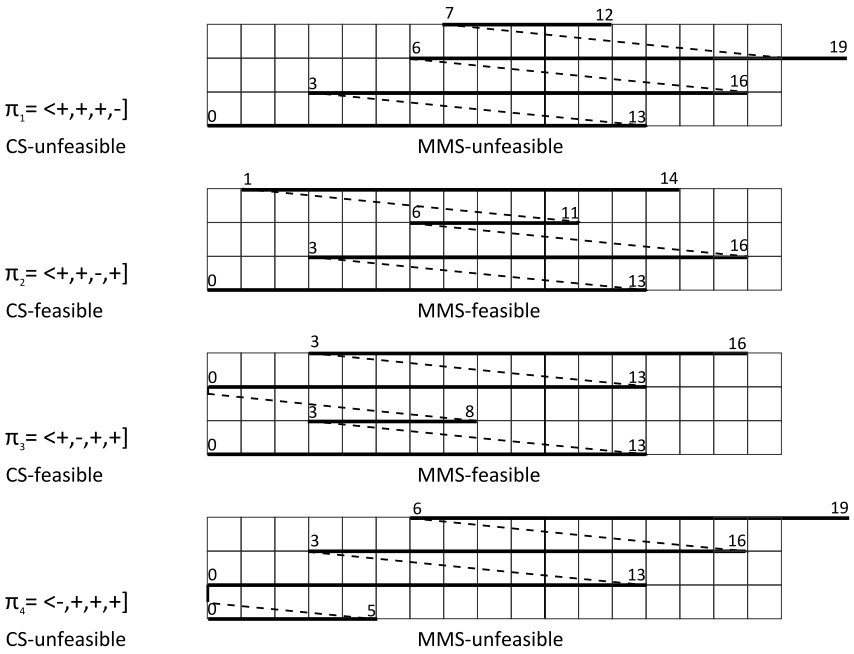


Figure 2.3: Example for the MSR approach with sequences that assemble three option models (+) and one basic model (-)

with starting time $s_j > l - p^+$. Processing the option model at position j leads to work overload. W.l.o.g. j is the first position in the sequence, where work overload occurs. Position $i = max(x|x < j \wedge s_x = 0)$ is the last time where the worker resets before j. i always exists, since the sequence starts with $s_1 = 0$. We neglect all models at positions less than i since they have no influence on s_j. Furthermore, we only consider the subsequence $\pi_{[i,j)}$ excluding the option model at position j. Since no reset or work overload occurs in $\pi_{[i,j)}$, the starting time s_j is the sum of the displacements of all models in $\pi_{[i,j)}$. With $\overline{k}$ option models and $\overline{m}$ basic models, $s_j = \overline{k}(p^+ - c) + \overline{m}(p^- - c) = \overline{k}(p^+ - c) - \overline{m}(c - p^-) > l - p^+$. Since $k_{min} \leq \overline{k} \leq k_{max}$, a sequencing rule exists for $\overline{k}$ with $N^{\overline{k}-k_{min}+1} = \overline{k} + m$. According to (2.6), this sequencing rule demands $l - p^+ \geq \overline{k}(p^+ - c) - m(c - p^-)$. Therefore, $\overline{k}(p^+ - c) - \overline{m}(c - p^-) > \overline{k}(p^+ - c) - m(c - p^-)$ which leads to $m > \overline{m}$. The subsequence $\pi_{[i,j)}$ has a length lower than $N^{\overline{k}-k_{min}+1}$ but already contains $\overline{k}$ option models. Having an option model at position j does not only induce work overload but also leads to a violation of the respective sequencing rule, since the subsequence $\pi_{[i,j]} = \pi_{[i,j)} + \{j\}$ has length $\leq N^{\overline{k}-k_{min}+1}$ but contains $\overline{k} + 1$ option models. This completes the proof that CS-unfeasible←MMS-unfeasible and therefore CS-feasible→MMS-feasible. □

Proposition 2.4. *For the MSR approach, CS-feasible ← MMS-feasible.*

Proof: We use a contraposition proof and show CS-unfeasible → MMS-unfeasible. A CS-unfeasible sequence violates at least one sequencing rule. Thus, a subsequence π of length $t = k + m$ with $k \in [k_{min}, k_{max}]$ contains at least $k+1$ option models and $m-1$ basic models. W.l.o.g. the violation occurs at the last position in π. Therefore, in the first $t-1$ positions of π, there are k option models and $m-1$ basic models. The first position i in π has starting time s_i. Assuming that no work overload occurs in the first $t-1$ positions, the starting time s_j of the last position j in π must be $\geq s_i+k(p^+-c)-(m-1)(c-p^-)$. For $s_i \geq 0$ and $k(p^+ - c) - (m-1)(c-p^-) > l - p^+$ (compare (2.6)), we get $s_i + k(p^+ - c) - (m-1)(c-p^-) > l - p^+$ and hence $s_j > l - p^+$. Therefore, the option model at position j induces work overload. This completes the proof that CS-unfeasible→MMS-unfeasible and therefore CS-feasible←MMS-feasible. □

MSR has the disadvantage that the number $k_{max} - k_{min} + 1$ of sequencing rules is relatively large and some of the rules are redundant in the sense that they are covered by other rules. We introduce the concept of strictness.

Definition 2.1. A sequencing rule $H : N$ is *stricter* than another sequencing rule $P : Q$, if all possible permutations of option and basic models for a given model mix which are feasible under $H : N$ are also feasible under $P : Q$ but at least one permutation exists which is unfeasible under $H : N$, but feasible under $P : Q$.

A test for strictness compares the maximum number of allowed option models within a subsequence of length N and Q, respectively. In a sequence of length T, the maximal number of option models allowed by sequencing rule $H : N$ is $H \left\lfloor \frac{T}{N} \right\rfloor + \min(T \mod N; H)$ (Fliedner and Boysen, 2008). Thus, $H : N$ is stricter than $P : Q$, if

$$H \left\lfloor \frac{Q}{N} \right\rfloor + \min(Q \mod N; H) \leq P, \text{ and} \tag{2.7}$$

$$H \neq P \vee N \neq Q \tag{2.8}$$

(2.7) ensures that the number of option occurrences under rule $H : N$ never exceeds P for any feasible subsequence of length Q. Thus, any permutation which does not violate the $H : N$ rule is also feasible for $P : Q$. Inequalities (2.8) ensure that both rules are not identical. For an example with $l = 20$, $c = 10$, $p^+ = 20$, $p^- = 0$, and $T = 10$, MSR generates five rules $1 : 2$, $2 : 4$, $3 : 6$, $4 : 8$, and $5 : 10$. According to strictness, the latter four rules are redundant and the rule set can be reduced to a single $1 : 2$ rule.

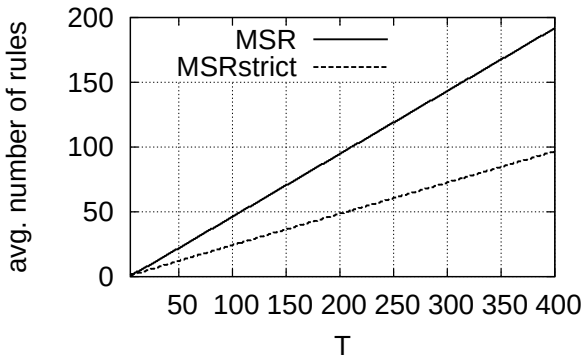


Figure 2.4: Average number of sequencing rules against sequence length T for MSR and MSRstrict, which removes redundant rules.

Note that redundant rules can not be identified by interpreting rules as fractions and arguing that rule $H : N$ is stricter than rule $P : Q$ if $H/N < P/Q$. Such an argumentation may lead to wrong results. For example, in a feasible sequence of length $T = 4$, we assume two option models and two basic models leading to six possible permutations. A $1 : 2$ classifies three of them as feasible. The conjecture that a $2 : 5$ rule is more strict ($2/5 < 1/2$) is wrong since $2 : 5$ classifies all six permutations as feasible.

For different T, we study the number of redundant sequencing rules for problems with one station. We create 1,000 random problem instances with parameters from Table 2.1. Figure 2.4 shows the average number of sequencing rules generated by MSR against the sequence length T with and without eliminating redundant sequencing rules. The MSR approach using strictness for the elimination of redundant sequencing rules is denoted as MSRstrict. When eliminating redundant rules, the number of sequencing rules can approximately be cut in half. For $T = 50$, MSR generates on average 22.5 sequencing rules per station; MSRstrict, which eliminates redundant rules, only produces on average 12.6 rules.

In summary, the new MSR approach for one option per station with two processing times generates multiple sequencing rules. It correctly classifies all sequences either as feasible or unfeasible. The definition of strictness is able to halve the number of necessary rules for MSR by eliminating redundant rules.

2.5 Multiple processing times

Although the literature focused on rule generation approaches for stations with only two possible options (Bolat and Yano, 1992), in real life, options with

multiple processing times per station are common. For example, in automobile assembly, sunroofs can come in three options with no sunroof, manual, or electric; transmission may be manual or automatic in 4 to 6 speeds; onboard electronic devices, such as stereo systems can come in various configurations. Furthermore, multiple options (parts or modules) might need to be installed per station.

We assume more than two models $m \in M$ with diverging processing times p_m. M is split into two subsets $M^+ = \{m \in M | p_m > c\}$ and $M^- = \{m \in M | p_m < c\}$ containing all models with processing time larger and smaller than cycle time c, respectively. All models with $p_m = c$ are excluded from consideration, as their production does not modify the ending position within the station. Thus, they can never produce additional work overload and can be scheduled at facultative sequence positions.

We reduce the multiple options case to the two option case by introducing one *virtual option* per station. All models $m \in M^+$ require the virtual option with processing time p^{v+}; all $m \in M^-$ with processing time p^{v-} do not require the virtual option. For the virtual processing times p^{v+} and p^{v-}, there are different possibilities: using the maximal (MAX), average (AVG), or minimal (MIN) processing times of each set:

$$\text{MAX:} \quad p^{v+} = \max_{m \in M^+} \{p_m\}, \qquad p^{v-} = \max_{m \in M^-} \{p_m\} \tag{2.9}$$

$$\text{AVG:} \quad p^{v+} = \frac{\sum_{m \in M^+} \{p_m\}}{|M^+|}, \qquad p^{v-} = \frac{\sum_{m \in M^-} \{p_m\}}{|M^-|} \tag{2.10}$$

$$\text{MIN:} \quad p^{v+} = \min_{m \in M^+} \{p_m\}, \qquad p^{v-} = \min_{m \in M^-} \{p_m\} \tag{2.11}$$

Having two options (either the virtual option or not) and corresponding processing times, sequencing rules can be derived using either the BY or MSR approach. We give an example with four models to be processed at a station with length $l = 10$ and cycle time $c = 5$. $M^+ = \{1, 2\}$ contains two models with processing times $p_1 = 8$ and $p_2 = 6$. Models 3 and 4 with processing times of $p_3 = 4$ and $p_4 = 2$ belong to set M^-. Models 1 and 2 require the virtual option; models 3 and 4 are declared as basic models. Using the maximal processing times (MAX), we get $p^{v+} = 8$ and $p^{v-} = 4$. For a sequence length $T = 4$, the BY approach results into a $1 : 4$ rule and the MSR approach to rules $1 : 2$ and $2 : 6$.

Proposition 2.5. *For the virtual option approach with MAX aggregation, CS-feasible→MMS-feasible.*

Proof: CS-feasible→MMS-feasible holds for both the BY and MSR approach with one option and processing times $p^+ = p^{v+} = \max_{m \in M^+}\{p_m\}$ and $p^- =$

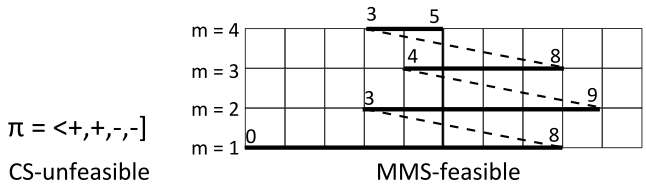


Figure 2.5: Counterexample for CS-feasible↞MMS-feasible

$p^{v-} = \max_{m \in M^-}\{p_m\}$. Having additional options $m \in M^+$ with processing $p_m < p^{v+}$ and $m \in M^-$ with processing $p_m < p^{v-}$ can only move ending times to the left and never lead to additional work overload. □

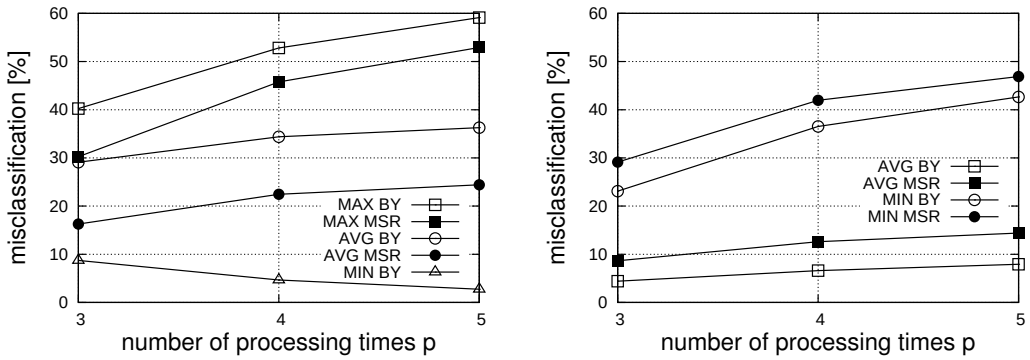
In contrast, for the AVG and MIN aggregation, CS-feasible↠MMS-feasible does not hold, which could simply be proven by a counterexample.

Proposition 2.6. *For the virtual option approach with MAX aggregation, CS-feasible↞ MMS-feasible holds.*

Proof: We prove by a counterexample to the contradiction of our proposition: CS-feasible←MMS-feasible. Consider the example with $l = 10$, $c = 5$, $p_1 = 8$, $p_2 = 6$, $p_3 = 4$, $p_4 = 2$, $M^+ = \{1, 2\}$, and $M^- = \{3, 4\}$. Any possible sequence of four different models is MMS-feasible. The BY approach returns a $1 : 4$ rule. Since model 1 and 2 require the virtual option, no CS-feasible sequence exists. The MSR approach leads to rules $1 : 2$ and $2 : 6$. Therefore, all sequences where model 1 directly follows model 2 and vice versa, are CS-unfeasible. Figure 2.5 shows a sequence that is MMS-feasible but CS-unfeasible for both, the BY and MSR approach. □

For the BY approach, both AVG and MIN aggregation lead to CS-feasible↞MMS-feasible. For MSR and AVG aggregation, CS-feasible↞MMS-feasible. Combining MSR with MIN aggregation results into CS-feasible←MMS-feasible. Since CS-feasible←MMS-feasible holds for the case with only two processing times, it also holds for the virtual option case since all options $m \in M^+$ and $m \in M^-$ have processing times larger than p^{v+} and p^{v-}, respectively.

For the three aggregation possibilities, we study the classification quality as the number of sequences in percent for which CS-feasible↞MMS-feasible and CS-feasible↛MMS-feasible holds, respectively. Experimental design follows Section 2.3. For either three, four, or five different processing times (and thus models) and various sequence lengths $T \in \{10, 15, 20\}$, we generate $1,000$ random problem instances according to the parameters given in Table 2.1. For $T = 10$, we consider all possible sequences; for larger T, at least $1,000,000$ random sequences for each instance are sampled. Sampling size is iteratively



(a) CS-feasible↚MMS-feasible (b) CS-feasible↛MMS-feasible

Figure 2.6: Number of misclassified sequences (in percent) for $T = 10$

increased by a factor of $1,000,000$ until it contains at least 10 MMS-feasible sequences.

For each sequence, we determine MMS-feasibility. Furthermore, for the virtual option approach with MAX, AVG, and MIN aggregation, we determine CS-feasibility using either the BY or MSR approach. For $T = 10$, Figure 2.6(a) shows the number of sequences (in percent) that are MMS-feasible but not CS-feasible (CS-feasible↚MMS-feasible) in relation to the number of processing times p. Table 2.2 lists complete results for different T and p.

Using MAX aggregation, the number of feasible solutions that are wrongly classified as CS-unfeasible is highest. For MAX and AVG aggregation, the percentage of wrongly classified sequences increases the more processing times p are considered; for MIN it decreases. In comparison to BY, MSR misclassifies a lower percentage of sequences. As mentioned before, combining MSR and MIN leads to CS-feasible←MMS-feasible.

Analogously, Figure 2.6(b) presents the number of sequences (in percent) that are CS-feasible but not MMS-feasible (CS-feasible↛MMS-feasible) against p for $T = 10$. In comparison to AVG aggregation, MIN aggregation leads to a higher percentage of wrongly classified sequences. The BY approach correctly classifies a higher number of sequences than the MSR approach.

A tradeoff can be observed between CS-feasible↚MMS-feasible and CS-feasible↛MMS-feasible. Aggregations that misclassify a large number of MMS-feasible sequences as CS-unfeasible, misclassify a lower number of MMS-unfeasible sequences as CS-feasible and vice versa. This tradeoff applies to both the BY and MSR approach.

Wrongly classifying an MMS-unfeasible sequence as CS-feasible is problematic when searching for feasible sequences. Executing such an MMS-unfeasible sequence would lead to unforeseen overhead and causes large additional cost.

p	agg.	classification error	BY approach T=10	BY approach T=15	BY approach T=20	MSR approach T=10	MSR approach T=15	MSR approach T=20
3	MAX	CS-feasible $\nleftarrow$ MMS-feasible	40,23	48,57	53,98	30,24	36,86	41,43
	AVG	CS-feasible $\nleftarrow$ MMS-feasible	29,10	36,35	41,22	16,26	21,21	24,96
		CS-feasible $\nrightarrow$ MMS-feasible	4,42	6,17	7,50	8,64	12,11	14,82
	MIN	CS-feasible $\nleftarrow$ MMS-feasible	8,71	12,28	15,29	0,00	0,00	0,00
		CS-feasible $\nrightarrow$ MMS-feasible	23,11	27,91	31,13	29,14	35,29	39,30
4	MAX	CS-feasible $\nleftarrow$ MMS-feasible	52,83	62,21	67,41	45,76	54,57	59,87
	AVG	CS-feasible $\nleftarrow$ MMS-feasible	34,38	43,11	48,96	22,44	29,21	33,95
		CS-feasible $\nrightarrow$ MMS-feasible	6,61	9,59	11,09	12,60	17,73	21,73
	MIN	CS-feasible $\nleftarrow$ MMS-feasible	4,65	6,95	8,64	0,00	0,00	0,00
		CS-feasible $\nrightarrow$ MMS-feasible	36,52	44,13	49,13	41,97	50,36	55,57
5	MAX	CS-feasible $\nleftarrow$ MMS-feasible	59,12	68,70	73,99	52,93	62,48	68,04
	AVG	CS-feasible $\nleftarrow$ MMS-feasible	36,27	45,47	51,60	24,41	31,89	37,16
		CS-feasible $\nrightarrow$ MMS-feasible	7,93	11,14	13,55	14,41	20,11	24,67
	MIN	CS-feasible $\nleftarrow$ MMS-feasible	2,73	3,83	4,89	0,00	0,00	0,00
		CS-feasible $\nrightarrow$ MMS-feasible	42,67	51,61	57,47	46,89	56,24	62,04

Table 2.2: Percentage of sequences where CS-feasible$\nleftarrow$MMS-feasible and CS-feasible$\nrightarrow$MMS-feasible, respectively

Therefore, for multiple processing times, we only find MAX aggregation useful since it guaranties that every CS-feasible sequence is also MMS-feasible. Combining MAX aggregation with MSR finds a larger number of MMS-feasible solutions in comparison to the BY approach. Again, this comes for the price of an enlarged rule set.

2.6 Conclusion

This paper investigates the classification quality resulting from existing and novel procedures for generating car sequencing rules, where quality is measured by the fraction of sequences for which a generated rule set properly predicts whether or not work overload occurs. Analytical and empirical results show a superior classification quality of our novel MSR approach for both the two and multiple processing times case. However, this comes for the price of additional sequencing rules to be introduced per instance. Thus, to benefit from more accurate rules, solution procedures are required which are able to handle large rule sets. Furthermore, future research should investigate the optimization version of minimizing work overload. Here, special rule generation procedures are required which additionally derive option-specific penalty values weighting

rule violations according to the resulting amount of work overload.

Bibliography

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

Boysen, N., Fliedner, M., and Scholl, A. (2009). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.

Drexl, A. and Kimms, A. (2001). Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints. *Management Science*, 47(3):480–491.

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

Parrello, B. D., Kabat, W. C., and Wos, L. (1986). Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2(1):1–42.

Solnon, C., Cung, V., Nguyen, A., and Artigues, C. (2008). 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.

Wester, L. and Kilbridge, M. D. (1964). The assembly line model-mix sequencing problem. In *Proceedings of the Third International Conference on Operations Research*.

Chapter 3

Car sequencing versus mixed-model sequencing: A computational study

Uli Golle, Franz Rothlauf, Nils Boysen

Abstract

The paper deals with the two most important mathematical models for sequencing products on a mixed-model assembly line in order to minimize work overload, the mixed-model sequencing (MMS) model and the car sequencing (CS) model. Although both models follow the same underlying objective, only MMS directly addresses the work overload in its objective function. CS instead applies a surrogate objective using so-called sequencing rules which restrict labor-intensive options accompanied with the products in the sequence. The CS model minimizes the number of violations of the respective sequencing rules, which is widely assumed to lead to the minimization of work overload. In this paper, we present a first experimental investigation of CS compared to MMS in order to quantify the gap in the solution quality between both models. Therefore, several variants of CS are applied, combining different sequencing rule generation approaches and objective functions found in the literature as well as a newly introduced weighting factor. The linear relation between the objectives of both models is analyzed using Pearson's correlation coefficient and the performance of the different models is evaluated on several random test instances. Although both objectives are positive linearly related, our results show that a sequence found by CS contains on average at least 15% more work overload than a solution found by MMS.

3.1 Introduction

On a mixed-model assembly line, various car models are produced in facultative sequence with a lot size of one. These models vary in a number of options and, therefore, require different processing times at the stations of the line. Since a station is balanced to an average production time, a sequence with consecutive work intensive models could lead to work overload of the line operators as the assembly task can not be finished within the station limits. Work overload has to be compensated, e.g., by employing additional utility workers or stopping the line. In order to minimize the amount of work overload, two approaches are present in the literature, mixed-model sequencing (MMS) and car sequencing (CS). Although both pursue the same underlying objective, their approaches are very different.

MMS (Wester and Kilbridge, 1964) directly minimizes the amount of work overload. Therefore, it explicitly considers processing times, worker movements, stations borders, and other operational characteristics of a line. This level of detail results in a high effort for data collection and computation time. On the other hand, CS (Parrello et al., 1986) uses a more simplified surrogate objective for work overload. It applies so-called sequencing rules, which restrict the maximum occurrence of options in any subsequence of defined length. The aim is to find a production sequence with a minimum number of rule violations. Since CS is more aggregated than MMS, its data requirements are lower and the evaluation of sequences is faster. Furthermore, sequencing rules seem more intuitive to human decision makers.

In the past decade, CS experiences more attention in the literature mainly due to its practical relevance. For instance, the automobile manufacturer Renault launched a contest in 2005 resulting in 51 inscriptions, where the aim was to optimize different real-world CS instances from Renault's plants (Solnon et al., 2008). Puchta and Gottlieb (2002) apply other real-world examples from a German car manufacturer as well. However, the literature on CS almost exclusively deals with different solution approaches, leaving the question open to what extend the surrogate objective of CS actually leads to the underlying goal of minimizing the amount of work overload. The accuracy of CS in identifying sequences with minimum work overload clearly depends on the suitability of the applied sequencing rules and how rule violations are evaluated in the objective function. Two rule generation approaches, one by Bolat and Yano (1992b) and the Multiple Sequencing Rule approach (MSR) of Chapter 2, exist. Furthermore, three frequently used objective functions for CS have been proposed in the literature.

In this paper, we examine the solution quality of CS compared to MMS. CS is applied with different combinations of sequencing rules and objective functions as well as a newly introduced weighting factor. First, we perform

a correlation analysis in order to investigate the linear relationship between both objectives, the number of sequencing rule violations and the amount of work overload. Second, by analyzing optimal solutions and evaluating the performances of both models on various test instances, we determine the gap in the solution quality that comes along with CS compared to MMS. Our results reveal the following insights:

- The objectives of CS and MMS are positive linearly related using Pearson's product moment correlation coefficient.
- A sequence found by CS contains on average at least 23% more work overload than a solution found by MMS. If inadequate sequencing rules and/or objective functions for CS are used, this gap can increase up to 75%.
- The gap in the solution quality can be decreased to 15% by considering our new weighting factor.

Overall, our results show that CS with its surrogate objective aggregates too much from the underlying objective of minimizing the work overload. Therefore, the CS model is not able to produce competitive results in terms of solution quality compared to MMS. Our contribution in this regard is to provide estimates of the resulting gap of work overload when using CS and not MMS. Based on our findings, we encourage both, researchers and practitioners, to concentrate more on MMS instead of CS in future research and practical applications, respectively.

The remainder of the paper is structured as follows. Section 3.2 states the MMS approach and assumptions made for the paper on hand. The CS model is introduced in Section 3.3 together with the rule generation approaches, the objective functions as well as the weighting factor considered in this paper. The computational study along with its results is presented in Section 3.4, followed by a critical discussion about the advantages of CS in Section 3.5. Finally, Section 3.6 concludes the paper.

3.2 Mixed-model sequencing

MMS was introduced by (Wester and Kilbridge, 1964) and aims to find a sequence of different models $m \in M$ with minimum total work overload based on a detailed schedule. Work overload occurs, whenever an operator is not able to finish assembly operations on a current workpiece within the station's boundaries. MMS explicitly takes several operational characteristics of the line into account, such as processing times, station borders, operator movements, and others (for an overview see Boysen et al., 2009). The range of characteristics

and, thus, MMS models makes it impossible to examine the appropriateness of CS for any possible MMS setting. Therefore, we apply the basic MMS problem as our benchmark model, which is based upon the following assumptions (see Scholl, 1999, p. 96; Boysen et al., 2009)

T	number of production cycles (index t)
M	number of models (index m)
K	number of stations (index k)
c	cycle time
d_m	demand for model m
l_k	length of station k
p_{mk}	processing time of model m in station k
s_{kt}	starting time of the tth model in station k
w_{kt}	work overload induced by the tth model in station k
x_{mt}	binary variables: 1, if model m is produced in slot t, 0 otherwise

Table 3.1: Notation

- Without loss of generality assembly lines flow from left to right.
- All stations are closed, i.e. operators are not allowed to work beyond the station borders.
- All stations are consecutively arranged along the line and models move with constant speed through the stations.
- The processing of a model starts early, i.e. as soon as operator and model meet within the station boundaries.
- An operator has zero return times, i.e. when having finished one model moves with infinite speed to the next model. This assumption is adequate, since the speed of the conveyor is usually much slower than the operators walking speed. Otherwise the cycle time c can be reduced to include the average return time.
- Fixed rate launching is applied, i.e. consecutive models are placed on the conveyor at the same interval equal to the cycle time c.
- The demands d_m of the models are given for the whole planning period and remain unchanged, i.e. no rush orders are allowed.
- The number of stations K and their respective lengths l_k, with $k \in K$, are given, e.g., from the solution of the preceding balancing problem.

- All processing times p_{mk} of models $m \in M$ in stations $k \in K$ are static and deterministic.
- $p_{mk} \leq l_k \quad \forall m \in M; k \in K$ holds, otherwise sequence-independent work overload would occur whenever the respective model is processed.
- Work overload w_{kt} induced by the tth model of the sequence in station k is immediately compensated by additional utility workers within the station borders, such that the starting time $s_{k+1,t}$ of the respective model in the subsequent station $k+1$ is not affected.

Considering the aforementioned assumptions and the notation summarized in Table 3.1, we get the following MMS model (Scholl, 1999, p. 105; Boysen et al., 2009):

$$\text{(MMS) Minimize} \quad \text{obj}_{\text{wo}} = \sum_{k=1}^{K} \sum_{t=1}^{T} w_{kt} \tag{3.1}$$

subject to

$$\sum_{t=1}^{T} x_{mt} = d_m \quad \forall m \in M \tag{3.2}$$

$$\sum_{m \in M} x_{mt} = 1 \quad \forall t = 1, \ldots, T \tag{3.3}$$

$$s_{kt} \geq s_{k,t-1} + \sum_{m \in M} p_{mk} \cdot x_{m,t-1} - c - w_{k,t-1} \quad \forall k = 1, \ldots, K; t = 2, \ldots, T \tag{3.4}$$

$$s_{kt} + \sum_{m \in M} p_{mk} \cdot x_{mt} - w_{kt} \leq l_k \quad \forall k = 1, \ldots, K; t = 1, \ldots, T \tag{3.5}$$

$$s_{kt} \geq 0 \quad \forall k = 1, \ldots, K; t = 1, \ldots, T \tag{3.6}$$

$$w_{kt} \geq 0 \quad \forall k = 1, \ldots, K; t = 1, \ldots, T \tag{3.7}$$

$$s_{k1} = 0 \quad \forall k = 1, \ldots, K \tag{3.8}$$

$$x_{mt} \in \{0, 1\} \quad \forall m \in M; t = 1, \ldots, T \tag{3.9}$$

The total work overload over all stations $k \in K$ and production slots t is minimized (3.1). Equations (3.2) and (3.3) ensure that the demand d_m for

each model is met and each slot in the sequence contains exactly one model. Furthermore, the processing of a model at position t in station k cannot start until the respective operator has processed the preceding model at position $t-1$ (3.4). Potential work overload is immediately compensated and, thus, is not affecting subsequent starting times (3.5). (3.5) together with (3.6) also model closed stations. Work overload is required to be nonnegative (3.7). The line is assumed to be in an initial condition before processing the entire sequence (3.8), thus, an operator awaits the first model at the left-hand station border. Note, that for comparative reasons we do not require to reach this initial condition again after processing the entire sequence, since this constraint can not be modeled with CS. Finally, x_{mt} take only binary values (3.9).

Example: Consider an example with a sequence length of $T = 11$, one station $K = 1$ and two models $M = 2$. Furthermore, we have the cycle time $c = 5$, the length of the single station $l_1 = 12$ and model 0 (model 1) with a demand of $d_0 = 7$ ($d_1 = 4$) and processing time $p_{01} = 3$ ($p_{11} = 10$). Figure 3.1 shows the movement diagram for the example sequence $< 01110001000 >$. In a movement diagram, the operator movements within a station necessary to process the sequence are visualized. Processing times are represented by horizontal lines, return times by diagonal dashed lines. The operator starts processing at the left-hand station border at position 0. The example sequence contains a copy of model 0 at the first slot. After assembling this model, the operator returns back to position 0 and processes model 1 at slot 2. Since model 1 requires a processing time of $p_{11} = 10$, the operator fails to reach the left-hand station border again after finishing it. Instead he meets the subsequent model, which already crossed the left-hand station border, at position 5. Again a copy of model 1 needs to be processed which leads to a work overload of 3. It is assumed that the work overload is immediately compensated, therefore, the operator meets the 4th model at position 7 in the station and so on. Note, that the example sequence leads to a total work overload of 8, since the operator is not able to finish the respective models at slots 3 (work overload $w_{13} = 3$) and 4 ($w_{14} = 5$) in time.

3.3 Car sequencing

Compared to MMS, CS is a more aggregate approach for finding a production sequence with minimum work overload. Presupposed is a pool of various models $m \in M$ as well as their required demands d_m. The models can be distinguished by several binary options (such as a car having an air conditioning or not). CS employs $H_o : N_o$ sequencing rules in order to restrict the occurrences of each option $o \in O$ in any subsequence of N_o succeeding models to at most H_o. For example, a sequencing rule of $2 : 3$ for the option "air con-

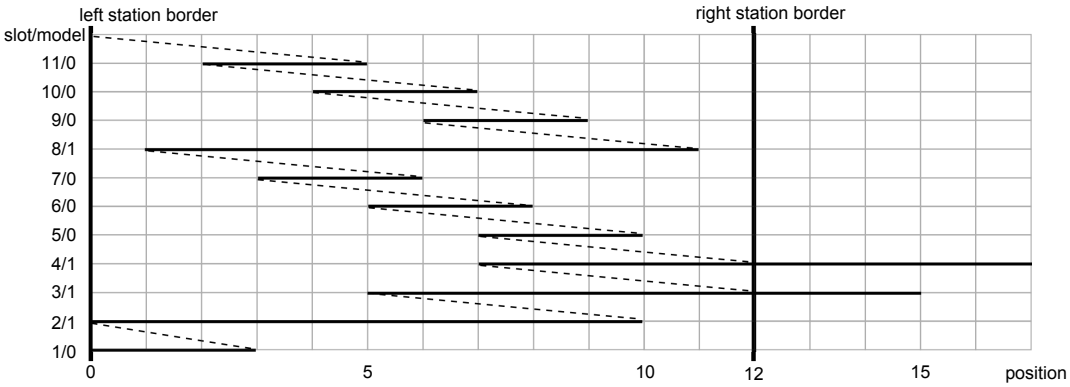


Figure 3.1: Movement diagram for example sequence

T	number of production slots (index t)
O	number of options (index o)
$H_o : N_o$	sequencing rule for option o: at most H_o out of N_o successively sequenced models require option o
p_o^+	processing time of models with option o
p_o^-	processing time of models without option o
a_{om}	binary variables: 1, if model m contains option o, 0 otherwise
x_{mt}	binary variables: 1, if model m is produced in slot t, 0 otherwise
Q_o	number of sequencing rules for option o (index q)
$H_o^q : N_o^q$	qth sequencing rule for option o

Table 3.2: Additional notation for CS

ditioning" requires that in any subsequence of 3 consecutive models at most 2 models contain this option, otherwise a violation occurs. The objective of CS is to minimize the number of sequencing rule violations over all options.

The existing CS literature implicitly assumes that the surrogate objective of minimizing the number of rule violations is associated with the minimization of work overload. However, to our best knowledge it has never been examined if this assumption applies. By all means, suitable sequencing rules and an appropriate objective function are required in order to accomplish the underlying objective of minimizing work overload. We identify two sequencing rule generation approaches and three frequently applied objective functions in the literature that are considered in this paper.

3.3.1 Sequencing rule generation approaches

Bolat and Yano sequencing rule approach (BYSR)

Bolat and Yano (1992b) derive sequencing rules for CS based on operational characteristics of the assembly line. For this purpose, they assume that exactly one option $o \in O$ is processed at each station $k \in K$, therefore we use the notations o and O synonymously for k and K, respectively. Since CS applies binary options, each option has merely two processing times. A model requiring option o has processing time p_o^+ larger than the cycle time c, while a model without this option needs processing time p_o^-, which is lower than c. Thus, we have $p_o^- < c < p_o^+ \leq l_o$. Given these assumptions, the Bolat and Yano Sequencing Rule (BYSR) approach computes a sequencing rule $H_o : N_o$ for option o by:

$$H_o = \left\lfloor \frac{l_o - c}{p_o^+ - c} \right\rfloor \text{ and} \tag{3.10}$$

$$N_o = H_o + \left\lceil \frac{H_o \cdot (p_o^+ - c)}{c - p_o^-} \right\rceil \quad \forall o \in O. \tag{3.11}$$

H_o is the maximum possible number of consecutive models having option o that can be processed without inducing work overload. N_o is computed by adding to H_o the number of succeeding models without option o that are required to shift the operator back to the left-hand station border after processing H_o models with option o.

Example: Consider the aforementioned example with $l_1 = 12, c = 5, p^- = p_{01} = 3, p^+ = p_{11} = 10$. Then, H_o amounts to 1 and N_o to 4, and therefore the BYSR approach returns a sequencing rule of $1 : 4$.

Multiple sequencing rules approach (MSR)

In Chapter 2, it was shown that the BYSR approach has some defects regarding the identification of feasible sequences, considering MMS and CS as constraint satisfaction problems. The rules generated by BYSR are too restrictive. Therefore, a certain amount of actually feasible sequences, that contain no work overload, is missclassified by CS, as they violate at least one sequencing rule. This impedes the search process of CS, since the search space of overall feasible solutions is reduced. Therefore, Section 2.4 introduced the multiple sequencing rules (MSR) approach to diminish the defects.

MSR builds up on the same assumptions as the BYSR approach. Hence, exactly one option $o \in O$ is processed at each station $k \in K$ with two pro-

cessing times p_o^+ for models with option o and p_o^- for models without o, with $p_o^- < c < p_o^+ \leq l_o$. Instead of a single rule per option $o \in O$, MSR introduces multiple sequencing rules $H_o^q : N_o^q$, with $q = 1, \ldots, Q_o$ and Q_o being the total number of sequencing rules for option o. Q_o amounts to $q_o^{max} - q_o^{min} + 1$ with:

$$q_o^{min} = \left\lfloor \frac{l_o - c}{p_o^+ - c} \right\rfloor \quad \forall o \in O \tag{3.12}$$

$$q_o^{max} = \left\lfloor \frac{T(c - p_o^-) + (l_o - c)}{p_o^+ - p_o^-} \right\rfloor \quad \forall o \in O. \tag{3.13}$$

q_o^{min} equals H_o from the BYSR approach and returns the maximum number of successive models containing option o without inducing work overload. q_o^{max} computes the maximum number of models with option o that can occur in a sequence of length T without work overload.

Then, $\forall q_o \in [q_o^{min}, q_o^{max}]$, MSR generates a sequencing rule $H_o^{q_o - q_o^{min}+1} : N_o^{q_o - q_o^{min}+1}$ with:

$$H_o^{q_o - q_o^{min}+1} = q_o \quad \forall o \in O \tag{3.14}$$

$$N_o^{q_o - q_o^{min}+1} = H_o^{q_o - q_o^{min}+1} + \left\lceil \frac{q_o(p_o^+ - c) - (l - p_o^+)}{c - p_0^-} \right\rceil \quad \forall o \in O. \tag{3.15}$$

The last term of (3.15) computes the minimum number of models without option o that are required after processing q_o models with o in order to process another model containing o without inducing work overload.

Example: Consider again our example with $l_1 = 12$, $c = 5$, $p^- = p_{01} = 3$, $p^+ = p_{11} = 10$ and assume a sequence length $T = 11$. Then, q_o^{min} is 1 and q_o^{max} amounts to 4, thus, four sequencing rules are needed overall. With (3.14) and (3.15), we get $H_o^1 : N_o^1 = 1 : 3$, $H_o^2 : N_o^2 = 2 : 6$, $H_o^3 : N_o^3 = 3 : 10$ and $H_o^4 : N_o^4 = 4 : 13$.

3.3.2 Objective functions and model formulation

Several objective functions for CS have been proposed in the literature (Boysen et al., 2009). In this paper, we consider the three most widely used, the sliding window (SW) objective function (Gottlieb et al., 2003; Gagne et al., 2006; Fliedner and Boysen, 2008), an objective function introduced by Fliedner and Boysen (2008) and one by Bolat and Yano (1992a). Furthermore, we adjust these objective functions to incorporate more than one sequencing rule per option and introduce additional weights that allow to differentiate between

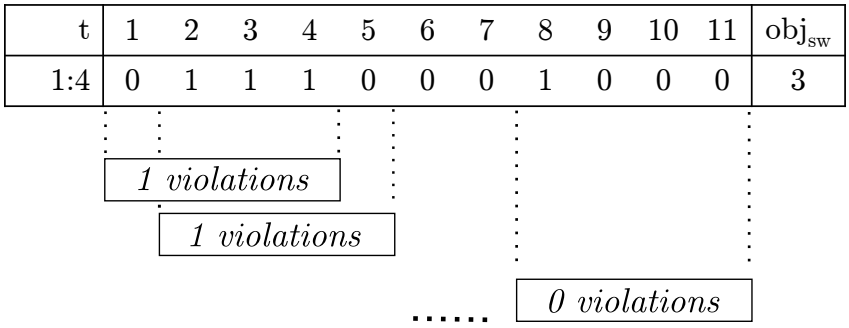


Figure 3.2: Evaluation of example sequence with SW objective function

the impact of violations of different options on the resulting work overload.

CS evaluates a sequence by sliding through the sequence and summing up the violations of each relevant subsequence, also called windows. A violation occurs, if a window for option o contains more than H_o occurrences of the respective option. The three objective functions differ in the windows considered as well as in the amount of violations a single window can provoke. The SW objective function counts all windows of size N_o that are violated. Every violated window leads to exactly one violation in the objective function, regardless how many excessive option occurrences it contains. With notations from Table 3.2, the SW objective function results in the following model (Gottlieb et al., 2003; Gagne et al., 2006; Fliedner and Boysen, 2008):

$$\text{Minimize} \quad \text{obj}_{\text{sw}} = \sum_{o \in O} v_o \tag{3.16}$$

with

$$v_o = \sum_{t=1}^{T-N_o+1} \min \left\{ 1; \max \left\{ \sum_{t'=t}^{t+N_o-1} \sum_{m \in M} a_{om} \cdot x_{mt'} - H_o; 0 \right\} \right\} \quad \forall\, o \in O \tag{3.17}$$

subject to (3.2),(3.3) and (3.9).

(3.16) minimizes the total number of violations over all options. (3.17) assesses all violated subsequences of size N_o. Figure 3.2 considers our aforementioned example sequence from Figure 3.1 together with the 1:4 sequencing rule obtained by the BYSR approach. The first window from slot 1 to 4 is violated due to three occurrences of option o. So is the second window from slot 2 to 5. Overall, the SW objective function amounts to three violations.

Fliedner and Boysen (2008) introduce a different objective function (FB),

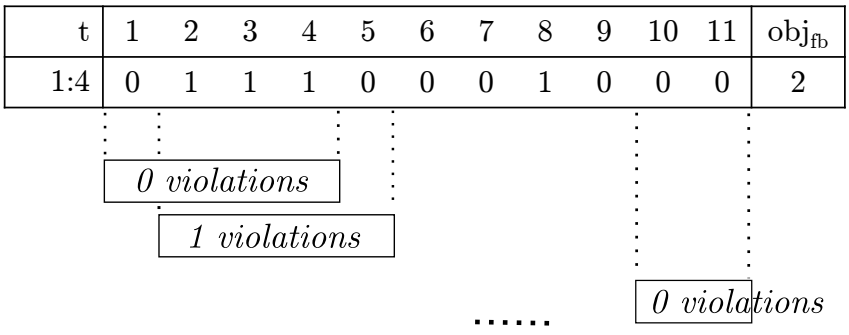


Figure 3.3: Evaluation of example sequence with FB objective function

an improvement of a quite similar function by Gagne et al. (2006). In order to eliminate some defects of the SW approach, where violations at the beginning and at the end of the sequence are favored, as fewer windows are violated, the FB objective function considers windows of size $< N_o$ at the end of the sequence. It only considers a violated window, if the first model in the window contains the respective option. Thereby, the FB objective function counts the option occurrences that actually lead to a violation of a subsequence. The associated model is as follows (Fliedner and Boysen, 2008):

$$\text{Minimize} \quad \text{obj}_{\text{fb}} = \sum_{o \in O} v_o \tag{3.18}$$

with

$$v_o = \sum_{t=1}^{T-H_o} \min \left\{ \sum_{m \in M} a_{om} \cdot x_{mt}; \max \left\{ \sum_{t'=t}^{\min\{t+N_o-1;T\}} \sum_{m \in M} a_{om} \cdot x_{mt'} - H_o; 0 \right\} \right\} \quad \forall\, o \in O \tag{3.19}$$

subject to (3.2),(3.3) and (3.9).

Again, (3.18) minimizes the total number of violations over all options. (3.19) evaluates all violated subsequences with 1, if the first model in the subsequence has option o, and 0 otherwise. Note, that we slightly modified the formula in Fliedner and Boysen (2008), since the first sum of (3.19) only needs to run until $T - H_o$, instead of T, i.e. no violation can occur in subsequences of size H_o and less. Figure 3.3 shows the example sequence with the FB objective function. Although the first window contains excessive option occurrences, it induces no violation in the objective function, as the first model in the window doesn't require the option. In contrast, the second window from

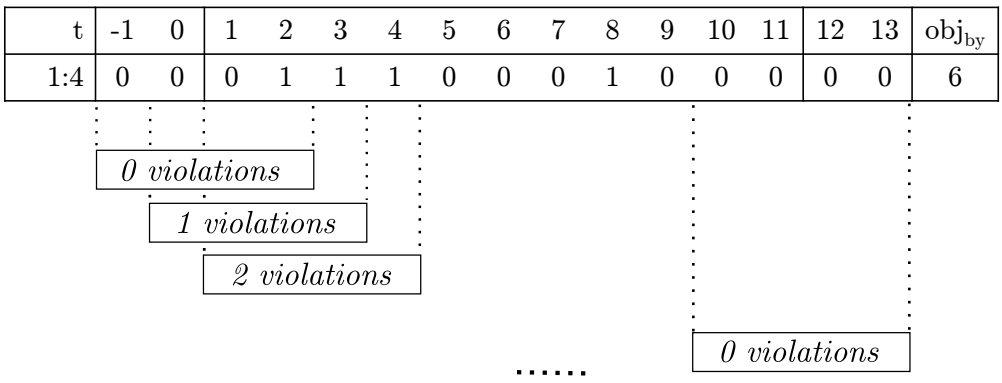


Figure 3.4: Evaluation of example sequence with BY objective function

slot 2 to 5 is counted with one violation by the FB objective function, since the first model has the respective option. Note, that the evaluation runs until slot $T - H_o = 11 - 1 = 10$, thus considers subsequences of size less than 4 at the end of the sequence, as the windows beginning at slot 9 and 10 have size 3 and 2, respectively. Overall, the FB objective function results in two violations.

In order to capture the extent to which each window is violated, Bolat and Yano (1992a) propose an objective function (BY), where each excessive option occurrence in a window leads to one additional violation. Therefore, a window can induce at most $N_o - H_o$ violations. The BY objective function results in the following model (Bolat and Yano, 1992a; Benoist, 2008) (we follow the compact notation of Benoist (2008)):

$$\text{Minimize} \quad \text{obj}_{\text{by}} = \sum_{o \in O} v_o \tag{3.20}$$

with

$$v_o = \sum_{t=H_o-N_o+2}^{T-H_o} \max \left\{ \sum_{t'=t}^{t+N_o-1} \sum_{m \in M} a_{om} \cdot x_{mt'} - H_o; 0 \right\} \quad \forall\, o \in O \tag{3.21}$$

subject to (3.2),(3.3) and (3.9).

(3.20) minimizes the total number of violations over all options. (3.21) counts the number of excessive occurrences of option o in each subsequence. Note, that all subsequences have a size equal to N_o and the evaluation can go beyond the actual sequence, as also models prior to the sequence beginning at slot $H_o - N_o + 2$ and models after the sequence up to slot $T - H_o + N_o - 1$ are

t	1	2	3	4	5	6	7	8	9	10	11	obj_{sw}	obj_{fb}	obj_{by}	obj_{wo}
Sequence 1	0	1	0	1	0	0	0	1	0	1	0	4	2	4	3
Sequence 2	0	1	1	1	0	0	0	1	0	0	0	3	2	6	8

Figure 3.5: Example sequences subject to 1:4

included. All models beyond the sequence are assumed to not having option o. Figure 3.4 shows the evaluation of the example sequence with the BY objective function. The evaluation starts at slot -1 with the first window not being violated. In contrast, the second window is violated, since it has one excessive occurrence of option o, thus, leads to one violation. The third window has even two excessive option occurrences and is evaluated with two violations. Furthermore, in our example, two models after the actual sequence at slots 12 and 13, both not requiring the respective option, are considered for the last two windows starting at slot 9 and 10, respectively. The BY objective function leads to six violations overall.

Example: We reconsider our example from Section 3.2 with the sequencing rule 1 : 4 according to the BYSR approach (see Section 3.3.1). Figure 3.5 shows two example sequences and their respective violations according to the three aforementioned objective functions together with the actual work overload. The sequences are assessed very differently by the three objective functions. While the SW objective function favors sequence 2 and the BY objective function finds sequence 1 superior to sequence 2, the FB objective function is indifferent between both sequences. In this example, the BY objective function leads to the right result as the actual work overload for sequence 1 is lower than for sequence 2.

In order to apply the MSR approach with the aforementioned objective functions, we have to adjust (3.16), (3.18) and (3.20) such that Q_o sequencing rules for each option o are incorporated:

$$\text{Minimize} \quad \text{obj}_{\text{sw}} = \text{obj}_{\text{fb}} = \text{obj}_{\text{by}} = \sum_{o \in O} \frac{1}{Q_o} \sum_{q \in Q_o} v_{oq} \tag{3.22}$$

Since the number of sequencing rules Q_o for each option can vary, (3.22) considers the average violations over all sequencing rules. Furthermore, we introduce in (3.17), (3.19) and (3.21) additional indices for H_o and N_o, respectively, leading to:

t	1	2	3	4	5	6	7	8	9	10	11	obj_{sw}	obj_{fb}	obj_{by}	obj_{wo}
Sequence 1	0	1	0	1	0	0	0	1	0	1	0	1	0.75	1	3
Sequence 2	0	1	1	1	0	0	0	1	0	0	0	2	1.25	3.25	8

Figure 3.6: Example sequences subject to 1:3, 2:6, 3:10 and 4:13

$$v_{oq} = \sum_{t=1}^{T-N_o^q+1} \min\left\{1; \max\left\{\sum_{t'=t}^{t+N_o^q-1} \sum_{m\in M} a_{om} \cdot x_{mt'} - H_o^q; 0\right\}\right\} \quad \forall\, o \in O;\ q \in Q_o \tag{3.23}$$

for the SW objective function,

$$v_{oq} = \sum_{t=1}^{T-H_o^q} \min\left\{\sum_{m\in M} a_{om} \cdot x_{mt}; \max\left\{\sum_{t'=t}^{\min\{t+N_o^q-1;T\}} \sum_{m\in M} a_{om} \cdot x_{mt'} - H_o^q; 0\right\}\right\} \quad \forall\, o \in O;\ q \in Q_o \tag{3.24}$$

for the FB objective function and

$$v_{oq} = \sum_{o\in O} \sum_{t=H_o^q-N_o^q+2}^{T-H_o^q} \max\left\{\sum_{t'=t}^{t+N_o^q-1} \sum_{m\in M} a_{om} \cdot x_{mt'} - H_o^q; 0\right\} \quad \forall\, o \in O;\ q \in Q_o \tag{3.25}$$

for the BY objective function.

Example: Again we consider our example from Section 3.2, this time with the four sequencing rules (1:3, 2:6, 3:10 and 4:13) generated by the MSR approach (see Section 3.3.1). In Figure 3.6 the same two example sequences as in Figure 3.5 and their respective violations according to three objective functions are shown. Using the MSR approach, all three objective functions assess the sequences right as they favor sequence 1.

3.3.3 Weighting of violations

On the one hand, the amount of imminent work overload caused by excessive occurrences of option o depends on the position of the excessive options in the window. Thus, succeeding option occurrences, that lead to a violation result in a higher work overload than non-succeeding options. This can be observed in our aforementioned example in Figures 3.5 and 3.6, where sequence 2 induces a higher work overload than sequence 1. On the other hand, the resulting work overload by a sequencing rule violation can considerably vary between options $o \in O$, as the stations, that assemble the options, have different operational characteristics, such as station length, etc. The forecited objective functions partly consider these factors. The SW and BY objective function favor non-succeeding violations, as less windows are violated (except for the SW function, when violations occur at the beginning or end of the sequence, as in Figure 3.5). Both also implicitly weight between options. An option occurrence, for instance, affects at most N_o windows, thus can lead to at most N_o violations. Therefore, violations of an option o with lower N_o are favored, and implicitly assumed to cause less work overload. However, there is no justification for this assumption. The FB objective function doesn't differentiate between violations at all. Each option occurrence that causes a violation is always assessed with one violation in the objective function.

We propose to use explicit weights λ_o for each option $o \in O$, in order to distinguish between option violations:

$$\text{Minimize} \quad \text{obj}^{\text{w}} = \sum_{o \in O} \lambda_o \frac{1}{Q_o} \sum_{q \in Q_o} v_{oq} \tag{3.26}$$

λ_o should be associated with the induced work overload, if a violation occurs. Therefore, we use the maximum work overload, that can be caused by one violation of option o. The maximum work overload occurs, if the operator reaches the right-hand station border while assembling a model and the next model in the line contains option o. Thus, the operator reaches this following model at time $l_k - c$ in the station and since $p_o^+ > c$ the resulting work overload amounts to:

$$\lambda_o = l_k - c + p_o^+ - l_k = p_o^+ - c \quad \forall\, o \in O. \tag{3.27}$$

Example: Consider two stations (options) with $l_1 = l_2 = 10$ and $c = 5$. On these stations, whether or not realizing option 1 (2) takes $p_1^+ = 10$ and $p_1^- = 0$ ($p_2^+ = 8$ and $p_2^- = 2$), respectively. Applying the BYSR approach, for both options a 1:2 rule is generated and weights amount to $\lambda_1 = 5$ and

	t	1	2	3	obj	obj^w	obj_{wo}
Sequence 1	Option 1	1	1	0	1	4	3
	Option 2	1	0	1	0	0	0
				total	1	4	3
Sequence 2	Option 1	1	0	1	0	0	0
	Option 2	1	1	0	1	3	1
				total	1	3	1

Figure 3.7: Example option weights λ_o

$\lambda_2 = 3$. Three different models (each with a single copy) are to be produced, where model 1 requires both options, model 2 only option 1, and model 3 only option 2. Consider the two different sequences depicted in Figure 3.7. While in the unweighted case (column obj) all three objective functions evaluate both sequences with the same value, in the weighted case (column obj^w) sequence 1 is properly identified as causing more work overload.

3.4 Experiments

In this section, we examine to what extent the different CS variants are able to achieve the underlying goal of minimizing work overload. First, we describe the problem instances applied throughout this section. We generate random MMS instances according to Scholl (1999) since the literature on MMS deals with random instances as well. The CS instances are then derived from the MMS instances using the sequencing rule generation approaches mentioned in Section 3.3.1. Second, we conduct a correlation analysis in order to examine the linear relationship between both objectives, the amount of work overload and the number of sequencing rule violations. Finally, we examine the solution quality and performance of the different CS variants in comparison to MMS on our problem instances.

3.4.1 Instance generation

In order to evaluate the different CS approaches and their performances compared to MMS, we define three sets of random MMS instances from which the respective CS instances are derived. Set 1 includes small instances, while set 2 contains larger instances and set 3 instances of real-world size. We vary the sequence length T, the number of stations K and number of models M with the parameters in Table 3.3. A full-factorial design of these parameters is performed leading to 8 combinations for set 1, and 18 combinations for set 2

		Set 1	Set 2	Set 3
Sequence length	T	10, 15	50, 100	500, 1000
Number of Stations	K	5, 10	20, 30, 40	20, 30, 40
Number of Models	M	5, 10	20, 30, 40	200, 300, 400
Cycle time	c		90	
Station length	l_k		[100, 150]	

Table 3.3: Parameter for random instances

as well as set 3. For each combination, we generate 10 random test instances leading to 440 instances overall. The test instances are created following the suggestions by Scholl (1999), who solely provide a general instance generator for MMS, which is also used in other recent publications (Cano-Belmán et al., 2010; Boysen et al., 2010, etc.). On all instances, the cycle time is set to $c = 90$ and the station lengths l_k are uniformly distributed in the interval [100;150]. The demand d_m of each model m is randomly chosen from the interval $[\frac{1}{2}T/M; \frac{3}{2}T/M]$ such that the total demand $\sum_{m \in M} d_m$ equals T and at least one copy of m is included. We restrict the number of processing times per station to two since CS is only applicable for binary options. The two processing times p_k^- and p_k^+ per station k are randomly defined from $[\frac{1}{2}c; c-1]$ and $[c+1; \min\{\frac{3}{2}c, l_k\}]$, respectively. Hence, $p_k^- < c$ and $p_k^+ > c$. We choose the processing times p_{mk} of each model m in station k randomly from p_k^- and p_k^+ such that each model is unique, its average processing time per station $\frac{1}{K}\sum_{k=1}^{K} p_{mk}$ is in the interval $[\frac{3}{4}c, c]$ and the average processing time of each station $\frac{1}{T}\sum_{m=1}^{M} d_m \cdot p_{mk}$ is beneath the cycle time c.

Since Chapter 2 already compared feasible instances of CS and MMS, we consider in these experiments only MMS instances, where the optimal solution contains work overload. Therefore, the instances in set 1 are optimally solved using CPLEX 12.2 and chosen such that they are MMS infeasible. However, the size of the instances in sets 2 and 3 impedes a computation of the optimal solution and, thus, their MMS infeasibility cannot be ensured. The corresponding CS instances of each MMS instance are derived using both rule generation approaches, the BYSR approach and MSR approach (see Section 3.3.1). Furthermore, we differentiate MSR by varying the number of sequencing rules Q_o used per option o. Thus, for MSR_1 we set $Q_o = 1$ and apply only the first generated rule, for MSR_2 we use the first two rules and for MSR_n we apply all generated rules per option. We combine each derived CS instance with the three aforementioned objective functions (see Section 3.3.2) once with and once without the weighting function (see Section 3.3.3). Hence, this leads to $4 \cdot 3 \cdot 2 = 24$ different CS variants overall.

3.4.2 Linear relationship between CS and MMS objectives

We evaluate the linear relationship between the objective values produced by MMS and each of the CS variants using Pearson's product-moment correlation coefficient r_{XY}, which is defined for samples $x_i \in X$ and $y_i \in Y$ as follows:

$$r_{XY} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 \cdot \sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{3.28}$$

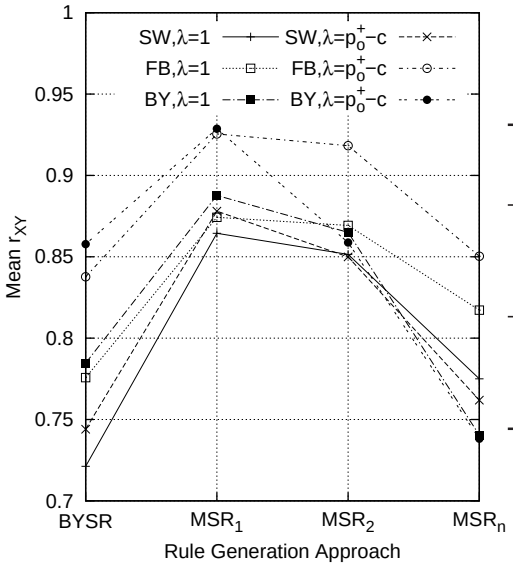
If $r_{XY} = 1$, a perfect positive linear relationship between samples X and Y exists, for $r_{XY} = 0$, X and Y are uncorrelated.

We use the first two sets of problem instances and generate $10,000$ random sequences per instance. For each sequence, we compute the amount of work overload as sample X and evaluate the number of sequencing rule violations according to CS variant v as sample $Y_v, v = 1, \ldots, 24$. Figure 3.8 shows the resulting correlation coefficient r_{XY_v} for each CS variant v.

For all CS combinations, the number of rule violations is positively correlated to the amount of work overload. However, there are differences in the degree of correlation. The BYSR approach has the lowest correlation coefficient since its generated rules are too restrictive. MSR_1 is stronger correlated than the other rule generation approaches on all objective functions. When options are restricted by more rules, violations of these options are overvalued and, thus, the correlation decreases. Therefore, MSR_2 is slightly inferior to MSR_1, while there is an even larger gap between MSR_n and MSR_1.

The BY objective function is slightly superior considering approaches that generate only one rule per option, namely BYSR and MSR_1. Depending on the size of the sequence window and to what extend the window is violated, the BY objective function is able to differentiate more between violation occurrences and, thus, implicitly also between work overload scenarios. However, the more rules are applied, the more the BY objective function amplifies the effect of overvaluation of violations. Regarding more rules per option, the FB function is actually superior to the other objective functions, as the aforementioned effect of overrating option violations applies less to it.

The weighting factor $\lambda_o = p_o^+ - c$ increases the correlation for nearly all CS variants, especially for those combined with BYSR or MSR_1 (see also Figure 3.8). Only for objective functions SW and BY each combined with MSR_2 and MSR_n, it has a minor negative effect. It seems that for these CS variants the weighting factor slightly increases the effect of overrating option violations. The resulting differences in the correlation coefficients have all been found to be statistically significant on a 0.05 level of significance.



λ_o	obj	Rule Generation Approach			
		BYSR	MSR_1	MSR_2	MSR_n
1	SW	0.7212	0.8644	0.8515	0.7750
	FB	0.7757	0.8743	0.8693	0.8172
	BY	0.7847	0.8877	0.8650	0.7404
$p_o^+ - c$	SW	0.7440	0.8780	0.8499	0.7619
	FB	0.8377	0.9255	0.9183	0.8503
	BY	0.8578	0.9287	0.8588	0.7382

Figure 3.8: Resulting Pearson's correlation coefficients averaged over set 1 and set 2

3.4.3 Solution quality of CS compared to MMS

In this section, the solution quality of the different CS variants each compared to MMS is analyzed. We first apply problem set 1, and compare optimal solutions of MMS and CS. For each instance in this set, we determine all optimal sequences of MMS and each CS variant using an iterative beam search approach, which will be described in detail in Chapter 5. For MMS, the average amount of work overload $\overline{w} = \frac{1}{|Set1|}\sum_{i\in \text{Set1}} w_i$ and the average number of optimal solutions $\overline{\eta}$ over all instances in set 1 is displayed in Table 3.4. For each CS variant v, we determine the average number of optimal solutions $\overline{\eta}_v$ as well as the deviation in the number of optimal solutions $\Delta\overline{\eta}_v = \overline{\eta}_v/\overline{\eta} \cdot 100\%$ compared to MMS. Furthermore, we compute for every optimal sequence of CS the induced amount of work overload. This leads to $\overline{w}_{iv}$, the average amount of work overload resulting from all optimal solutions for instance i and CS variant v. We average $\overline{w}_{iv}$ over all instances in set 1 by defining $\overline{w}_v = \frac{1}{|Set1|}\sum_{i\in \text{Set1}} \overline{w}_{iv}$. Then, $\Delta\overline{w}_v = \overline{w}_v/\overline{w} \cdot 100\%$ is the deviation between the average amount of work overload induced by optimal solutions of CS and MMS. For each CS variant v, Table 3.5 presents the resulting values of $\overline{w}_v, \Delta\overline{w}_v, \overline{\eta}_v$ and $\Delta\overline{\eta}_v$.

In general, the solution quality of CS is worse compared to MMS. Since CS assigns the same objective value to solutions that actually induce a different amount of work overload, it has at least twice as many optimal solutions as MMS. Among these solutions are sequences which not even come close to the optimum as can be observed in the resulting average amount of work overload

$\overline{w}$, which is higher than that of MMS. Thus, an optimal solution found by one of the CS variants (with $\lambda = 1$) results at least into 33% (MSR_2 and BY) more work overload than the actual optimal solution. There are differences regarding the CS variants. As already stated above, the sequencing rules generated by the BYSR approach are very restrictive. Therefore, it finds less solutions than the MSR approaches, but also excludes good sequences resulting in a higher $\overline{w}$. In contrast, MSR_1 assigns the same objective value to more solutions, thus, comprises more optimal solutions than BYSR, but these solutions contain less work overload leading to a better average solution quality. MSR_2 and MSR_n are able to further exclude non-optimal solutions and, thus, reduce the number of solutions as well as increase the solution quality. For the instances in set 1, MSR_n works well, since the number of generated rules depends on the sequence length T, which is relatively low. Thus, MSR_n generates not significantly more rules than MSR_2 and the aforementioned effects of overrating violations don't apply.

Again, the BY objective function performs best, as an optimal solution found by this function contains on average less work overload than solutions of other objective functions. By considering the weighting factor $\lambda_o = p_o^+ - c$, the performance of CS in general can be increased as it leads to a further exclusion of solutions with a higher amount of work overload and, therefore, reduces the overall number of CS optimal solutions as well as the average work overload induced by one of these solutions.

We further examine the performance of the different CS variants compared to MMS on problem set 2. Due to their size in terms of sequence length T, number of stations S and number of models M, the instances in set 2 cannot be solved optimally, yet. Therefore, we use a metaheuristic by Puchta and Gottlieb (2002), which was originally introduced for CS and comprises six different move operators. We adjusted this approach in order to apply it with MMS and the different CS variants as well.

For each instance in set 2, we randomly create 20 initial sequences. The metaheuristic is applied on each of these initial sequences using the MMS model and the 24 different CS variants. Since set 2 contains 180 instances, we conduct $180 \cdot 20 \cdot (1 + 24) = 90,000$ runs overall. Beginning with the initial sequence, a move is randomly chosen from one of the six operators with equal probability. The move is applied to the current sequences and the resulting sequence is maintained as new current sequence if its objective value is less or equal than the objective value of the current sequence. This process is repeated until a given maximum number of moves is reached and the current sequence is returned as best solution.

In order to decrease disruptions of solutions and, thus, intensify the search process, Puchta and Gottlieb (2002) restrict the size of the affected subsequence by one move operator to at most $T/2$. We abort every single search

$\overline{w}$	$\overline{\eta}$
14.8	361,609

Table 3.4: Performance of MMS on problem set 1

λ_o	obj	Rule Generation Approach							
		BY		MSR_1		MSR_2		MSR_n	
		$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\eta}$ $\Delta\overline{\eta}$ [%]	$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\eta}$ $\Delta\overline{\eta}$ [%]	$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\eta}$ $\Delta\overline{\eta}$ [%]	$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\eta}$ $\Delta\overline{\eta}$ [%]
1	SW	25.7 73.8	535,013 148.0	20.3 37.8	1,188,680 328.7	20.2 37.0	1,063,088 294.0	20.3 37.3	1,061,891 293.7
	FB	26.6 79.9	602,323 166.6	20.6 39.8	1,242,443 343.6	20.5 39.0	1,102,667 304.9	20.6 39.3	1,101,467 304.6
	BY	23.5 58.9	532,457 147.3	20.0 35.3	1,176,556 325.4	19.6 33.0	1,055,540 291.9	19.7 33.7	1,054,517 291.6
$p_o^+ - c$	SW	24.8 67.8	452,108 125.0	18.9 27.9	1,101,374 304.6	18.2 23.5	983,320 271.9	19.0 28.8	982,301 271.7
	FB	25.5 73.1	503,823 139.3	19.2 29.8	1,131,356 312.9	18.6 25.8	1,013,252 280.2	19.2 30.1	1,012,266 279.9
	BY	22.0 48.8	474,657 131.3	18.3 23.8	1,090,946 301.7	17.6 19.5	972,736 269.0	18.4 24.7	971,681 268.7

Table 3.5: Performance of CS variants on problem set 1

after 100,000 moves and assess the best sequence found using the MMS objective function. Thus, the resulting amount of work overload for MMS as well as the different CS variants can be compared. Again, we determine $\overline{w}$, the average amount of work overload of the best solutions found by MMS and CS and for each CS variant v the resulting average deviation of work overload $\Delta\overline{w}_v$ compared to MMS. Furthermore, we measure for each model the average time $\overline{\tau}$ required to proceed the 100,000 moves. Tables 3.6 and 3.7 show the results of MMS and CS, respectively. Furthermore, the performance of the metaheuristic was tested in preceding experiments on problem set 1, where it found the optimal solution for MMS as well as for each of the 24 CS variants in all of the $80 \cdot 20 \cdot (1 + 24) = 40,000$ runs.

The results in Tables 3.6 and 3.7 confirm our previous findings. The gap in the solution quality between CS and MMS is at least 21.5% (MSR_1 and BY and $\lambda_o = p_o^+ - c$). Again, the MSR approach shows a superior solution quality for all objective functions compared to BYSR. For the instances in set 2, MSR_1 leads to the best solutions among the CS approaches and requires the least time. MSR_2 and MSR_n also result in a better solution quality compared to BYSR. MSR_1 as well as BYSR solve the instances in little less time than

$\overline{w}$	$\overline{\tau}$ [s]
1,078.4	3.3

Table 3.6: Performance of MMS on problem set 2

λ_o	obj	Rule Generation Approach							
		BY		MSR_1		MSR_2		MSR_n	
		$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\tau}$ [s]	$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\tau}$ [s]	$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\tau}$ [s]	$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\tau}$ [s]
1	SW	1,879.6 74.3	2.2	1,445.6 34.0	2.0	1,467.0 36.0	3.7	1,609.9 49.3	55.7
	FB	1,807.0 67.6	2.0	1,444.2 33.9	1.9	1,466.1 35.9	3.2	1,626.9 50.9	52.2
	BY	1,617.9 50.0	2.5	1,375.8 27.6	2.1	1,384.3 28.4	4.2	1,507.2 39.8	134.7
$p_o^+ - c$	SW	1,812.6 68.1	2.2	1,384.0 28.3	2.0	1,407.2 30.5	3.6	1,532.8 42.1	55.3
	FB	1,704.4 58.0	2.0	1,352.8 25.4	1.9	1,375.6 27.6	3.2	1,527.3 41.6	52.0
	BY	1,506.3 39.7	2.4	1,309.8 21.5	2.1	1,323.3 22.7	4.1	1,427.1 32.3	134.5

Table 3.7: Performance of CS variants on problem set 2

MMS, which requires 3.3 seconds on average (Table 3.6). This stems from the faster evaluation of an altered sequence by CS, since only parts of the sequence need to be re-assessed. However, by applying more than one sequencing rule this advantage disappears, as can be observed for MSR_2 and MSR_n. While the solution time of MSR_2 is slightly inferior to that of MMS, MSR_n needs considerably more time and its objective values are even worse compared to MSR_1 and MSR_2. The BY objective function performs better than the other objective functions, although it requires a little more time due to the higher effort for computing its objective values. The introduced weighting factor adds to an increased solution quality for all CS variants without changing the solution time considerably. Due to the results obtained on problem set 2, we exclude MSR_n in further experiments, since its solution quality and solution time is far worse compared to the other MSR approaches.

Finally, we examine the performance of MMS and CS on problem set 3, which consists of instances of real-world size. Again, 20 initial sequences for each instance are created and optimized with the aforementioned metaheuristic, leading to $180 \cdot 20 \cdot (1 + 24) = 90,000$ runs overall. We apply the metaheuristic with the same setting as for problem set 2. Thus, moves are chosen

$\overline{w}$	$\overline{\tau}$ [s]
16,383.2	39.1

Table 3.8: Performance of MMS on problem set 3

λ_o	obj	Rule Generation Approach					
		BY		MSR_1		MSR_2	
		$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\tau}$ [s]	$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\tau}$ [s]	$\overline{w}$ $\Delta\overline{w}$ [%]	$\overline{\tau}$ [s]
1	SW	28,030.8 71.1	13.3	21,445.7 30.9	12.6	22,204.3 35.5	21.8
	FB	26,107.2 59.4	19.3	21,136.5 29.0	18.5	21,729.2 32.6	30.7
	BY	23,045.3 40.7	22.9	20,104.0 22.7	20.7	20,473.7 25.0	38.5
$p_o^+ - c$	SW	26,642.8 62.6	13.2	20,265.7 23.7	12.5	21,177.9 29.3	21.7
	FB	24,095.4 47.1	19.2	19,570.9 19.5	18.4	20,284.2 23.8	30.7
	BY	20,791.0 26.9	24.1	18,778.0 14.6	21.9	19,386.6 18.3	40.2

Table 3.9: Performance of CS variants on problem set 3

with equal probability, the size of the affected subsequence is restricted to $T/2$ and each run is aborted after 100,000 moves. Tables 3.8 and 3.9 show the respective results for MMS and CS.

The results are similar to that for problem set 2. The MSR_1 rule generation approach as well as the BY objective function show a superior solution quality among the CS approaches. The weighting factor $\lambda_o = p_o^+ - c$ further improves the solution quality. Compared to the results on problem set 2, the absolute gap in the amount of work overload between MMS and CS increases, but the deviation in percent between both values decreases as the least deviation between CS and MMS amounts to 14.6% compared to 21.5% on set 2 (MSR_1 and BY and $\lambda_o = p_o^+ - c$). Furthermore, the advantage of the CS variants regarding the solution time increases as the variants which apply only one sequencing rule per option (BYSR and MSR_1) require approximately half the time of MMS. This is founded in the increased sequence length T. After each move, the CS variants only have to reassess windows where a change occurred, which are at most N_o for each option and change. In contrast, MMS has to reassess the entire sequence beginning at the position of the first change. Thus, the longer the sequence the bigger the difference in the solution time between

both approaches.

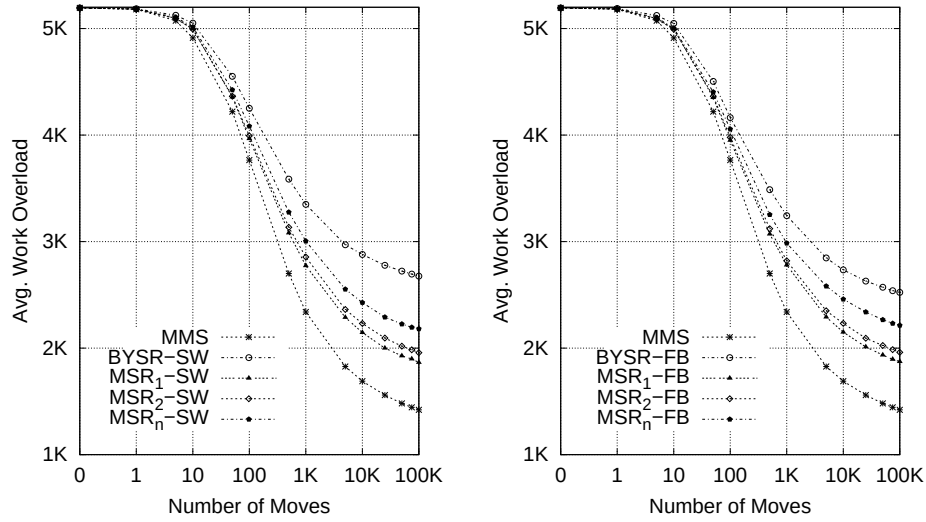
On 10 representative instances of set 2 with $T = 100, S = 30$ and $M = 30$, we show how the number of performed moves with the metaheuristic as well as the solution time influences the solution quality of MMS and CS. Again, for each instance, 20 initial solutions are created and optimized using the aforementioned metaheuristic, thus, leading to 200 runs overall for MMS and each CS variant. We examine different stopping criteria of the metaheuristic. The number of moves and the amount of time after which the solution procedure is stopped are varied between 1 and 100,000 moves and 0.1 and 60 seconds, respectively. Figures 3.9 and 3.10 show the respective results. To exemplify the results, we partitioned the CS variants into three different plots, each containing CS variants with identical objective functions. We concentrate our evaluation on CS variants with $\lambda_o = p_o^+ - c$, since the introduced weighting factor showed superior results in previous experiments.

Regarding Figures 3.9(a), 3.9(b) and 3.9(c), we can observe that the differences in the solution quality between MMS and each of the CS variants increases with the number of applied moves. The order of the considered approaches regarding the solution quality remains unchanged. Regardless at which number of moves the search process is stopped, MMS always produces sequences that contain on average less work overload than sequences found by one of the CS variants.Furthermore, the results of MSR_1 are always better than that of the other rule generation approaches regardless which objective function is applied. Figures 3.10(a), 3.10(b) and 3.10(c) show the solution quality against the elapsed time. At the beginning of the search, MSR_1 results in better solutions than MMS, since it can evaluate more sequences in the same amount of time. However, this advantage fades away at 0.05 seconds, where the MMS approach becomes superior. The solution gap between MMS and MSR_1 even increases the longer the metaheuristic runs. MSR_n shows a poor solution quality at the beginning of the search due to the large amount of sequencing rules considered. However, for the SW (Figure 3.10(a)) and FB (Figure 3.10(b)) objective function it is able to compensate this disadvantage later in the search as it produces better results than BYSR.

In summary, when applying CS, the MSR_1 rule generation approach in combination with the BY objective function and the newly introduced weighting factor $\lambda_o = p_o^+ - c$ shows the overall best performance in terms of solution quality and runtime. However, it still results in sequences that contain on average at least 15% more work overload than sequences found by MMS.

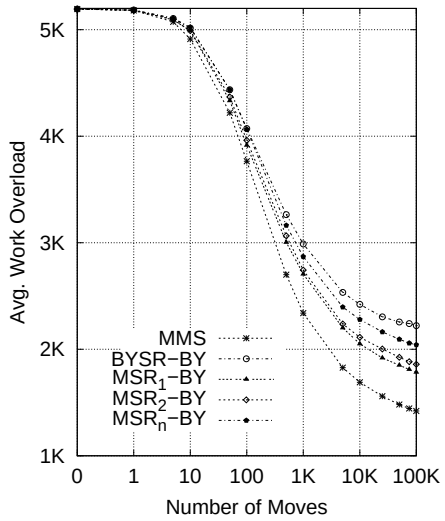
3.5 Discussion

It is not surprising that CS leads to a lower solution quality compared to MMS since it is a much more aggregated model. For the instances considered in this



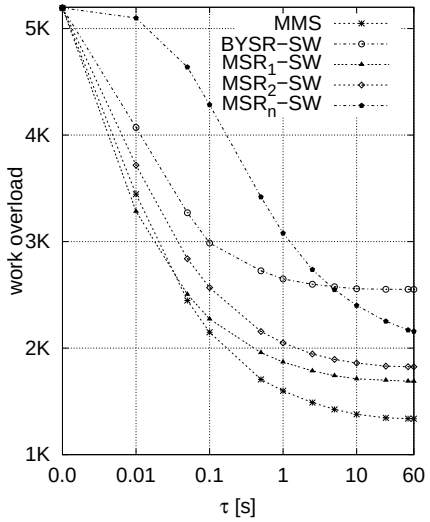
(a) CS variants with SW objective function

(b) CS variants with FB objective function

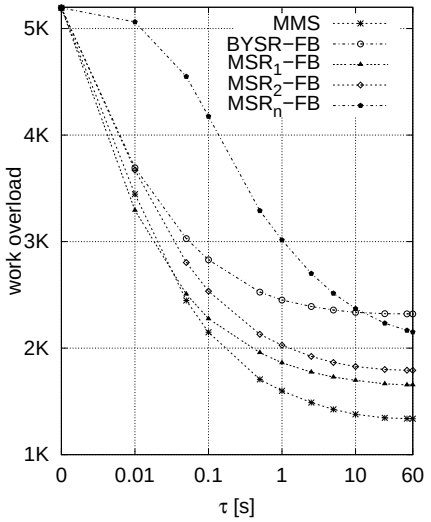


(c) CS variants with BY objective function

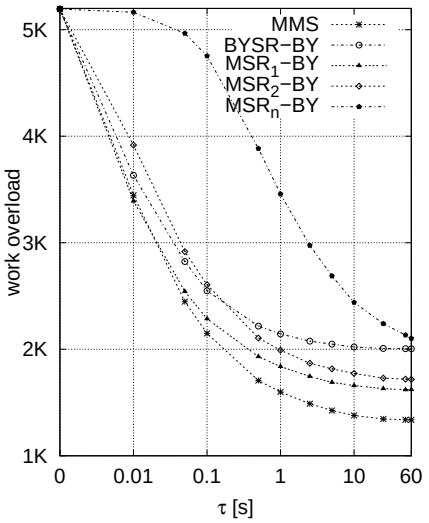
Figure 3.9: Solution quality $\overline{w}$ against avg. time $\overline{\tau}$ of MMS compared to CS variants with weighting factor $\lambda_o = p_o^+ - c$ based on instances with $T = 100, S = 30, M = 30$.



(a) CS variants with SW objective function



(b) CS variants with FB objective function



(c) CS variants with BY objective function

Figure 3.10: Solution quality $\overline{w}$ against avg. time $\overline{\tau}$ of MMS compared to CS variants with weighting factor $\lambda_o = p_o^+ - c$ based on instances with $T = 100, S = 30, M = 30$.

paper, the resulting gap in the amount of work overload between CS and MMS is at least about 15% (for CS with the MSR_1 rule generation approach and BY objective function and weighting factor $\lambda_o = p_o^+ - c$). This gap increases considerably if CS is applied with other sequencing rules and/or objective functions. For instance, using sequencing rules created by the BYSR approach in combination with the SW objective function and no weighting factor even leads to a solution gap of about 75%. The question arises which advantages of CS compared to MMS justify this gap in the solution quality?

The literature puts forward that CS is supposed to avoid the significant effort of data collection accompanied with MMS (Boysen et al., 2009). However, this is only the case when sequencing rules for CS are created by rules of thumb or intuitively as in the following example by Drexl and Kimms (2001):

> Assume that 60% of the cars manufactured on the line need the option 'sun roof'. Moreover, assume that five cars (copies) pass the station where the sun roofs are installed during the time for the installation of a single copy. Then, three operators (installation teams) are necessary for the installation of sun roofs. Hence, the capacity constraint of the final assembly line for the option 'sun roof' is three out of five in a sequence, or 3:5 for short.

The usage of such intuitive sequencing rules is very likely to put more inaccuracy into CS and, thus, increases the gap in the solution quality between CS and MMS. If sequencing rules are derived analytically and consider operational characteristics of the line, as in the BYSR and MSR rule generation approaches, the same amount of data as for MMS has to be collected in advance. As another advantage Drexl et al. (2006) mention that CS doesn't require an explicit definition of open and closed stations. In fact, the definition of $H : N$ sequencing rules implicitly assumes that all stations on the line are closed since no interaction between stations can be represented with this kind of sequencing rules. The literature also emphasizes that sequencing rules simultaneously balance the allocation of the required material as well (Drexl and Jordan, 1995; Drexl et al., 2006), which is actually the goal of the just-in-time related level scheduling model (Miltenburg, 1989). However, Boysen (2005, p. 233) experimentally revealed that the objective of level scheduling is not implicitly optimized by CS.

To our best knowledge, only two advantages of CS compared to MMS remain. First, sequencing rules are more intuitive to human decision makers. Indeed, a $H : N$ is easier to verify for humans than computing the entire schedule. However, practical instances with often more than 1000 mixed-models to be sequenced are far too complex to be solved by human decision makers. Second, CS evaluates sequences faster compared to MMS, since it considers only combinatorial aspects of the problem and not the detailed time schedule

and operator movements (Bolat and Yano, 1992a; Drexl et al., 2006). We also verified this point in our experiments, as CS variants that use one sequencing rule per option can evaluate sequences faster than MMS by a factor of about 1.5-2. However, Figure 3.10 reveals that the faster evaluation doesn't result in a superior solution quality of CS compared to MMS, if both approaches are considered to have the same amount of time (> 0.05 seconds) for solving an instance. Since the solution time in practical applications and in the literature on CS and MMS is often fixed to 600 seconds, the solution quality of MMS is supposed to be superior in these scenarios as well.

We have to recall, that CS actually benefits from the assumptions made for the test instances in this paper, namely to use only closed stations and to allow two processing times per station. In practical applications, stations are often open, hence, the operator is allowed to work beyond the station borders to a certain extend. Furthermore, most real-world options have more than two characteristics and a station often assembles more than one option at a time, which leads to more than merely two processing times per station. Both scenarios are not addressed by CS. Thus, the solution gap between CS and MMS most likely increases when both are applied to practical settings.

3.6 Conclusions

Our work is the first that compares the solution quality of CS and MMS. Therefore, we perform a comprehensive computational study using random MMS instances of various sizes. Based on these instances, we derive the related CS instances using different sequencing rule generation approaches and also apply various objective functions for CS discussed in the literature. Furthermore, we introduce an additional weighting factor for CS in order to distinguish between option violations. A full factorial design of the rule generation approaches, objective functions and weighting factor is performed leading to 24 different CS variants considered in our study. First, we show that the number of rule violations is positive linearly related to the amount of work overload using Pearson's product moment coefficient. Second, the solution quality of MMS and the different CS variants is assessed by computing the resulting amount of work overload when each model is applied on the various test instances. For the problem instances in this paper, CS results in solutions that contain at least 23% more work overload than solutions of the related MMS problem. This gap can be decreased to 15% by considering our weighting factor for CS. The difference in the solution quality between CS and MMS increases, if CS is applied with inadequate sequencing rules and/or objective functions. For instance, using sequencing rules created by the BYSR approach in combination with the sliding-window objective function even leads to a solution gap of more than 75%.

Overall, the surrogate objective followed by CS aggregates too much from the underlying goal of minimizing work overload. Therefore, the solution quality of CS is not competitive compared to MMS. Based on our experimental findings and the discussion of the advantages of CS against MMS, we encourage decision makers to use MMS instead of CS for future applications, at least until new developments increase the solution quality of CS. Thus, future research should address the following issues:

1. The solution quality of CS and MMS should be compared in a real-world setting. Unfortunately, at the time of this research, only practical instances for CS, which stem from Renault, are available in the literature (Solnon et al., 2008). However, these instances can not be used to derive related MMS instances in order to compare both approaches.

2. New sequencing rule generation approaches and objective functions for CS need to be developed, that better reflect the underlying goal of minimizing the work overload. However, we have doubts that both would add to a considerable improvement of the solution quality of the CS model.

3. The simple CS model need to be extended in order to incorporate more practical characteristics of the line, e.g., to consider open stations and more than merely two processing times per stations. A first approach regarding multiple processing times can be found in Section 2.5. Another idea to consider multiple processing times could be to set up a non-binary CS model, which uses the displacements (processing time minus cycle time) a model induces to the operator instead of merely binary options. A sequencing rule for this kind of problem could check, if the sum of the displacements in any window of a certain length amounts to zero. Solnon et al. (2008) present a not further elaborated idea of using cross ratio constraints instead of simple $H : N$ rules to allow for an interaction between option/station constraints and, thus, to model open stations. A different idea would be to use multidimensional sequencing rules of the kind $(H_1, H_2) : N$, where not more than H_1 occurrences of option 1 and H_2 occurrences of option 2 are allowed in any 2-dimensional window of length N incorporating both options. Maybe some insights on these issues can be drawn from the literature on system reliability (Chao et al., 1995) as well, where sequencing rules of type $H : N$ are also applied in consecutive "k-out-of-m-from-n" systems.

Bibliography

Benoist, T. (2008). Soft car sequencing with colors: Lower bounds and optimality proofs. *European Journal of Operational Research*, 191(3):957–971.

Bolat, A. and Yano, C. (1992a). A surrogate objective for utility work in paced assembly lines. *Production Planning & Control*, 3(4):406–412.

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

Boysen, N. (2005). *Variantenfließfertigung*. Gabler, Wiesbaden.

Boysen, N., Fliedner, M., and Scholl, A. (2009). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.

Boysen, N., Kiel, M., and Scholl, A. (2010). Sequencing mixed-model assembly lines to minimise the number of work overload situations. *International Journal of Production Research*, (to appear).

Cano-Belmán, J., Ríos-Mercado, R. Z., and Bautista, J. (2010). A scatter search based hyper-heuristic for sequencing a mixed-model assembly line. *Journal of Heuristics*, 16(6):749–770.

Chao, M. T., Fu, J. C., and Koutras, M. V. (1995). Survey of reliability studies of consecutive-k-out-of-n:f and related systems. *IEEE Transactions on Reliability*, 44(1):120 – 127.

Drexl, A. and Jordan, C. (1995). Materialflussorientierte Produktionssteuerung bei Variantenfließfertigung. *Zeitschrift für betriebswirtschaftliche Forschung*, 47:1073–1087.

Drexl, A. and Kimms, A. (2001). Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints. *Management Science*, 47(3):480–491.

Drexl, A., Kimms, A., and Matthießen, L. (2006). Algorithms for the car sequencing and the level scheduling problem. *Journal of Scheduling*, 9(2):153–176.

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

Gagne, C., Gravel, M., and Price, W. (2006). Solving real car sequencing problems with ant colony optimization. *European Journal of Operational Research*, 174(3):1427–1448.

Gottlieb, J., Puchta, M., and Solnon, C. (2003). A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In Cagnoni, S., Johnson, C., Cardalda, J., Marchiori, E., Corne, D., Meyer, J.-A., Gottlieb, J., Middendorf, M., Guillot, A., Raidl, G., and Hart, E., editors, *EvoWorkshops 2003*, volume 2611 of *LNCS*, pages 246–257, Berlin Heidelberg. Springer.

Miltenburg, J. (1989). A brief survey of just-in-time sequencing for mixed-model systems. *Management Science*, 35(2):192–207.

Parrello, B. D., Kabat, W. C., and Wos, L. (1986). Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2(1):1–42.

Puchta, M. and Gottlieb, J. (2002). Solving car sequencing problems by local optimization. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, volume 2279 of *LNCS*, pages 132–142, Berlin Heidelberg. Springer.

Scholl, A. (1999). *Balancing and sequencing of assembly lines*. Physica, Heidelberg, 2nd edition.

Solnon, C., Cung, V., Nguyen, A., and Artigues, C. (2008). 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.

Wester, L. and Kilbridge, M. D. (1964). The assembly line model-mix sequencing problem. In *Proceedings of the Third International Conference on Operations Research*.

Chapter 4

The car resequencing problem with pull-off tables

Nils Boysen, Uli Golle, Franz Rothlauf

Abstract

The car sequencing problem determines sequences of different car models launched down a mixed-model assembly line. To avoid work overloads of workforce, car sequencing restricts the maximum occurrence of labor-intensive options, e.g., a sunroof, by applying sequencing rules. We consider this problem in a resequencing context, where a given number of buffers (denoted as pull-off tables) is available for rearranging a stirred sequence. The problem is formalized and suited solution procedures are developed. A lower bound and a dominance rule are introduced which both reduce the running time of our graph approach. Finally, a real-world resequencing setting is investigated.

4.1 Introduction

Most car manufacturers offer their customers the possibility to tailor cars according to their individual preferences. Usually, customers are able to select from a given set of options like different types of sunroofs, engines, or colors. However, offering a variety of options makes car production more demanding. For example, when assembling cars on a mixed-model assembly line, car bodies should be scheduled in such a way that the work load of the workforce has no peaks by avoiding the cumulated succession of cars requiring work-intensive options. The car sequencing problem (CSP), which was developed by Parrello et al. (1986) and received wide attention both in research and practical application (Solnon et al., 2008; Boysen et al., 2009), returns a production schedule where work overload is avoided or minimized. It uses $H_o : N_o$-sequencing rules,

which restrict the maximum occurrence of a work-intensive option o to at most H_o out of N_o successive car models launched down the line.

Standard CSP approaches (for an overview see Boysen et al., 2009) assume that a department's production schedule can be fully determined by the planner and no unforeseen events occur. However, those assumptions are not realistic. During production, cars visit multiple departments, i.e., body and paint shop, before reaching final assembly. The sequence of cars in each department cannot be arbitrarily changed but depends on the sequence in the previous department. This results in problems since a sequence that might be optimal for the first department is usually suboptimal for the following departments. Furthermore, disturbances like machine breakdowns, rush orders, or material shortages affect the production sequence. For example, in the paint shop, small color defects make a retouch or complete repainting necessary resulting in disordered model sequences.

Therefore, automobile producers install large automated storage and retrieval systems (AS/RS) with hundreds of random access buffers to decouple their major departments: body shop, paint shop, and final assembly (Inman, 2003). With the help of AS/RS, manufacturers are able to change the order of models between these departments, allowing them to plan and reshuffle optimal sequences according to each department's individual objectives and reconstruct desired model sequences after disturbances during production. Common and widespread forms of resequencing buffers in the automobile industry are selectivity banks (Spieckermann et al., 2004) and pull-off tables (Lahmar et al., 2003). Selectivity banks consist of a set of parallel first-in-first-out lanes. Models are assigned to one of the lanes, enter the lane on, e.g., the left-hand side and move forward to the right-hand side. Only models on the right-hand side of each lane are accessible to proceed downstream. Thus, the number of models to choose from is bounded by the number of lanes. In contrast, pull-off tables are direct accessible buffers. A model in the sequence can be pulled into a free pull-off table, so that successive models can be brought forward and processed before the model is reinserted from the pull-off table back into a later sequence position.

Figure 4.1 gives an example how pull-off tables can be used for reordering a sequence in such a way that no sequencing rules are violated any more. We assume an initial sequence of four models at positions $i = 1, \ldots, 4$. There are two options for each model: "x" and "-" denote whether or not a model requires the respective option. For the two options, we assume a 1:2- and a 2:3-sequencing rule, respectively. Figure 4.1(a) depicts the initial sequence, which would result in one violation of the 1:2-sequencing rule and one of the 2:3-sequencing rule. The initial sequence can be reshuffled by pulling the model at position 1 into the single pull-off table (Figure 4.1(b)). Then, the models at positions 2 and 3 can be processed. After reinserting the model from the

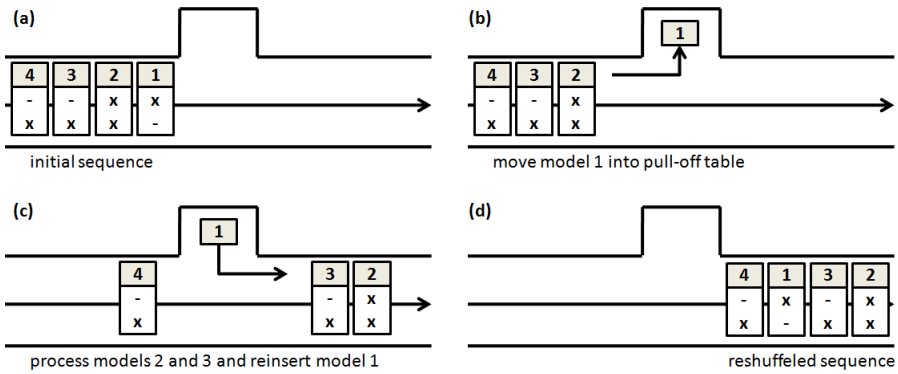


Figure 4.1: Example on the use of a pull-off table of size one

pull-off table (Figure 4.1(c)), the rearranged sequence $< 2, 3, 1, 4]$ of Figure 4.1(d) emerges, which violates no sequencing rule.

Although pull-off tables as well as car sequencing rules are widely used in industry, no approaches are available in the literature that address both aspects at the same time and return strategies for reordering car sequences in such a way that violations of sequencing rules are minimized. The use of pull-off tables is only considered in specific mixed-model assembly line settings neglecting the existence of sequencing rules. For example, a variety of papers address sequence alterations in front of the paint shop to build larger lots of identical color (e.g., by Lahmar et al., 2003; Epping et al., 2004; Spieckermann et al., 2004; Lahmar and Benjaafar, 2007; Lim and Xu, 2009) or in front of final assembly to level the material demand (Boysen et al., 2010). Other resequencing papers either deal with buffer dimensioning (Inman, 2003; Ding and Sun, 2004), alternative forms of buffer organization, e.g., mix banks (Choi and Shin, 1997; Spieckermann et al., 2004), or virtual resequencing (Inman and Schmeling, 2003; Gusikhin et al., 2008), where the physical production sequence remains unaltered and merely customer orders are reassigned to models.

This paper introduces the *car resequencing problem* (CRSP) which assumes a given model sequence and returns a strategy how to use pull-off tables to minimize violations of sequencing rules in the resulting sequence. First, we develop a graph transformation for the offline version of the problem and present various solution approaches. Note that in real-world applications resequencing is often an online problem since, for instance, models leave preceding production stages in unpredictable succession. Then, the offline problem version, where a given static initial sequence is to be reshuffled, needs to be applied in a rolling horizon. To clarify the application of CRSP in an online environment, a real-world resequencing setting is presented, which demonstrates the advantage of the proposed solution approaches.

The paper is organized as follows: Section 4.2 models the CRSP as a math-

ematical program. In Section 4.3, we develop a graph transformation, which strongly reduces the size of the solution space. With this graph transformation on hand, Section 4.4 presents different exact and heuristic solution approaches, which are tested in a comprehensive computational study (Section 4.5). To demonstrate the applicability of the approach, Section 4.6 presents a real-world resequencing setting requiring only a few simple modifications of our solution approach. The paper ends with concluding remarks.

4.2 Problem formulation

We assume an initial production sequence of length T. Since it takes one production cycle to process a car, the overall number of production cycles equals the sequence length T. Two models are different, if at least one option is different. Consequently, there are M different models with $M \leq T$. The binary demand coefficients a_{om} indicate whether model $m = 1, \ldots, M$ requires option $o = 1, \ldots, O$. Furthermore, we assume a given set of sequencing rules of type $H_o : N_o$ which restrict the maximum occurrence of option o in N_o successive cars to at most H_o. The initial sequence, which results from the ordering in the previous department or from disturbances, typically violates some of the sequencing rules.

To reorder the initial sequence, P pull-off tables can be used. Each pull-off table can store one car. When pulling a car into a pull-off table, subsequent models of the initial sequence advance by one position. Thus, by using P pull-off tables, we can shift a model at most P positions forward and an arbitrarily number of positions backward in the sequence. The CRSP returns a reshuffled production sequence that minimizes the number of violations of given car sequencing rules. With the notation from Table 4.1, we can formulate it as a binary linear program:

$$\text{CRSP: Minimize } \sum_{o=1}^{O} \sum_{t=1}^{T} y_{ot} \tag{4.1}$$

T	number of production cycles (index t or i)
M	number of models (index m)
O	number of options (index o)
P	number of pull-off tables
a_{om}	binary demand coefficient: 1, if model m requires option o, 0 otherwise
$H_o : N_o$	sequencing rule: at most H_o out of N_o successively sequenced models require option o
π_0	initial sequence before resequencing ($\pi_0(i)$ returns the number of the model that is scheduled for the ith cycle)
π_1	sequence after resequencing ($\pi_1(i)$ returns the number of the model that is processed at the ith cycle)
x_{itm}	binary variable: 1, if model number m at cycle i before resequencing is assigned to cycle t after resequencing, 0 otherwise
y_{ot}	binary variable: 1, if sequencing rule defined for option o is violated in window starting in cycle t
BI	Big Integer

Table 4.1: Notation

subject to

$$\sum_{i=1}^{T}\sum_{m=1}^{M} x_{itm} = 1 \quad \forall\, t = 1, \ldots, T \tag{4.2}$$

$$\sum_{t=1}^{T}\sum_{m=1}^{M} x_{itm} = 1 \quad \forall\, i = 1, \ldots, T \tag{4.3}$$

$$\sum_{t=1}^{T}\sum_{m=1}^{M} m \cdot x_{itm} = \pi_0(i) \quad \forall\, i = 1, \ldots, T \tag{4.4}$$

$$\sum_{i=1}^{T}\sum_{\tau=t}^{\min\{t+N_o-1,T\}}\sum_{m=1}^{M} x_{i\tau m}\cdot a_{om} - (1 - \sum_{i=1}^{T}\sum_{m=1}^{M} a_{om}\cdot x_{itm})\cdot BI \le H_o + BI\cdot y_{ot} \quad \forall\, o = 1, \ldots, O;\ t = 1, \ldots, T \tag{4.5}$$

$$x_{itm} = 0 \quad \forall\, m = 1, \ldots, M; i, t = 1, \ldots, T; i - t > P \tag{4.6}$$

$$x_{itm} \in \{0,\, 1\} \quad \forall\, m = 1, \ldots, M; i, t = 1, \ldots, T \tag{4.7}$$

$$y_{ot} \in \{0,\, 1\} \quad \forall\, o = 1, \ldots, O;\ t = 1, \ldots, T \tag{4.8}$$

For an option o, the binary variable y_{ot} indicates whether the sequencing rule $H_o : N_o$ is violated in the window starting at cycle t. The objective function (4.1) minimizes the sum of rule violations over all options o and cycles t. $\pi_0(i)$ and $\pi_1(i)$ return the number of the model that is processed at cycle i before and after resequencing, respectively. Constraints (4.2) and (4.3) enforce that at each cycle t, resp. i, exactly one model is processed, while (4.4) ensures that each car of the initial sequence π_0 is assigned to a cycle. (4.5) checks whether or not a rule violation occurs. Here, we follow Fliedner and Boysen (2008) and count the number of option occurrences that actually lead to a violation of a sequencing rule. (4.6) ensures that there is a maximum of P pull-off tables and, therefore, a model at position i in the initial sequence cannot be shifted to an earlier sequence position than $i - P$.

This basic model and the graph approach presented in the next section aim to minimize sequencing rule violations counted with the approach by Fliedner and Boysen (2008). However, our model and the graph approach can easily be adapted to other resequencing scenarios as well, e.g., to incorporate other functions for counting rule violations like the sliding-window technique (Gravel et al., 2005), to distinguish between hard- and soft-sequencing rules (Solnon et al., 2008), to pursue alternative resequencing objectives like leveling the required material (Drexl and Kimms, 2001) while avoiding rule violations or to minimize the number of material deviations between the resulting sequence and an original planned sequence. An example for such a model extension is presented in Section 4.6.

In general, CRSP is NP-hard in the strong sense, since for $P \geq T - 1$ (full resequencing flexibility) the problem is equivalent to CSP, which was shown to be NP-hard in the strong sense (Kis, 2004).

4.3 Transforming CRSP into a graph search problem

Given an initial sequence π_0 and P pull-off tables, a model at position i can be shifted arbitrarily to the back or up to P positions to the front. Thus, for each position i in the reordered sequence π_1, there are $P+1$ choices (the model $\pi_0(i)$ or one of the following models $\pi_0(i+1) \ldots \pi_0(i+P)$). Since there are T positions to decide on, the solution space is bounded by $O(P^T)$. Therefore, CRSP grows exponentially with the number T of cycles. In the following paragraphs, we transform the CRSP into a graph search problem, where the objective is to find a shortest path. The size of the resulting search space is lower than the original CRSP which reduces the effort of solution approaches. The transformation is inspired by Lim and Xu (2009), who used a related approach for solving a resequencing problem with pull-off tables for paint-shop batching. Since Lim

and Xu used another objective function, which resulted in a different solution representation, fundamental modifications of the original approach of Lim and Xu have been necessary.

The CRSP is modeled as a graph search problem, where the graph is an acyclic digraph $G(V, E, f)$ with node set V, arc set E and an arc weighting function $f : E \to \mathbb{N}$.

4.3.1 Nodes

Each node represents a state in the sequencing process. It defines the models that are in the pull-off tables and the sequence of models that have not yet been processed. Starting with the given initial sequence, in each step we have three choices (Lim and Xu, 2009):

- If an empty pull-off table exists, we can move the current model into it.
- We can process the current model and remove it from the sequence.
- If not all pull-off tables are empty, we can select an off-line model, remove it from its pull-off table, and process it.

Consequently, each step (sequencing decision) only depends on the current model at position i and K, which is defined as the set of models currently stored in the pull-off tables. At each step, the decision maker has to check whether the planned sequencing decision violates one of the sequencing rules. To perform this check, he must know how often an option o has been processed in the last $N_o - 1$ production decisions. Fliedner and Boysen (2008) defined the last $N_o - 1$ option occurrences of all $o = 1, \ldots, O$ options as the "active sequence". act_i^o denotes the active sequence of length $N_o - 1$ for option o at production cycle i. Consequently, $act_i^{o,t} \in \{0, 1\}$ is the tth position of an active sequence act_i^o. $act_i^{o,t} = 1$ indicates that at production cycle $i - t + 1$ option o has been processed.

Thus, a node $[i, K_i, ACT_i]$ is defined by the number $i \in \{1, \ldots, T\}$ of the production decision, the set K_i of models (with $|K| \leq P$) stored in the pull-off tables at production cycle i, and the set $ACT_i = \{act_i^1, act_i^2, \ldots, act_i^O\}$ of active sequences for the O different options at production cycle i.

Example: Consider the current decision point $i = 2$ depicted in Figure 4.1(c). Note that a decision point denotes a specific stage in the decision process where an accessible model is finally assigned to the next production cycle or intermediately stored in an empty pull-off table. The final release of the model at position 1 in initial sequence π_0 is the third production decision of the decision maker. Given two sequencing rules (1:2 and 2:3) of length two

and three, the active sequences have length one and two, respectively. Therefore, at decision point $i = 2$, we have the two active sequences $act_2^1 = \{0\}$ and $act_2^2 = \{1, 1\}$. The state before finally assigning the current model is defined as $[2, \{1\}, \{\{0\}, \{1, 1\}\}]$. After production of the model from the pull-off table, we have state $[3, \{\emptyset\}, \{\{1\}, \{0, 1\}\}]$.

Furthermore, we define a unique start and target node. With ACT^0 denoting a set of O active sequences all filled with zeros, the start node is defined as $[0, \emptyset, ACT^0]$ (for an example, Figure 4.1(a)); the (artificial) target node is defined as $[T + 1, \emptyset, ACT^0]$.

Proposition 4.1. *The number of states in node set V is at most $O(TOM^P)$.*

Proof: Overall there are T decision points (production cycles) and the number of possible sets K of models in the pull-off tables is $\binom{M+P-1}{P}$. The number of possible active sequences ACT_i is bounded by $O \cdot 2^{\max\{N_o\}-1}$. Thus, including the unique start and end node there are at most $T \cdot \binom{M+P-1}{P} \cdot O \cdot 2^{\max\{N_o\}-1} + 2$ nodes. Recall that T, O and M denote the number of production cycles, options and models, respectively.

$$
\begin{aligned}
&T \cdot \binom{M+P-1}{P} \cdot O \cdot 2^{\max\{N_o\}-1} + 2 \\
&\quad = T \cdot \frac{(M+P-1) \cdot (M+P-2) \ldots (M+1) \cdot M}{P!} \cdot O \cdot 2^{\max\{N_o\}-1} + 2 \\
&\quad \leq T \cdot \frac{M^P \cdot P!}{P!} \cdot O \cdot 2^{\max\{N_o\}-1} + 2
\end{aligned}
$$

which is bounded by $O(TOM^P)$.□

Hence, the size of the state space V increases exponentially with the number of pull-off tables P but only linearly with the number of production cycles T. Since the length of the graph is bounded by $O(TOM^P)$, a polynomial in the input length, the optimal solution can be found in polynomial time, provided P is a constant.

4.3.2 Arcs

Arcs connect adjacent nodes and, thus, represent a transition between two states $[i, K_i, ACT_i]$ and $[j, K_j, ACT_j]$. An arc represents either a scheduling decision or a combined scheduling and production decision. Starting with state $[i, K_i, ACT_i]$, we can distinguish three actions that can be performed:

1. If not all pull-off tables are filled ($|K| < P$), the current model m at cycle i can be stored in a free pull-off table. Note that current model

$m = \pi_0(i + |K| + 1)$ can directly be determined with the help of the information stored with any node. This scheduling decision adds model m to K and leaves the active sequences untouched resulting in node $[i, K_i \cup \{m\}, ACT_i]$. This (pure) sequencing decision does not produce a model.

For an example, we study the first sequencing decision in Figure 4.1. We start with the start node $[0, \emptyset, \{\{0\}, \{0, 0\}\}]$ (Figure 4.1 (a)). By pulling model 1 into the pull-off table, we branch into node $[0, \{1\}, \{\{0\}, \{0, 0\}\}]$ (Figure 4.1 (b)).

2. We leave the pull-off tables untouched and produce model m at cycle i. This operation modifies the active sequences as it inserts all option occurrences of model m at the first position in the active sequences. The option occurrences at position $N_o - 1$ are removed from the active sequences and all other option occurrences are shifted by one position. The resulting node is $[i + 1, K_i, ACT_{i+1}]$.

   For an example, we study the second sequencing decision in Figure 4.1 which processes model 2. The scheduling decision branches node $[0, \{1\}, \{\{0\}, \{0, 0\}\}]$ (Figure 4.1 (b)) into node $[1, \{1\}, \{\{1\}, \{1, 0\}\}]$.

3. If at least one model is stored in a pull-off table ($K \neq \emptyset$), we can pull a model from a pull-off table and produce it. This combined scheduling and production decision removes model m from the set of models in the pull-off tables and modifies the active sequences. The resulting node is $[i + 1, K_i \setminus \{m\}, ACT_{i+1}]$.

   For an example, we study the third production cycle in Figure 4.1(c). We reinsert model 1 from the pull-off table and process it. This operation branches node $[2, \{1\}, \{\{0\}, \{1, 1\}\}]$ (Figure 4.1 (c)) into the successor node $[3, \emptyset, \{\{1\}, \{0, 1\}\}]$.

In addition to these three transitions, we connect all nodes $[T, \emptyset, ACT_T]$ with the unique target node $[T + 1, \emptyset, ACT^0]$. Furthermore, we assign arc weights $f : E \to \mathbb{N}$ to each transition. The arc weights measure the influence of the transition on the overall objective value (number of violations of sequencing rules). Since transition 1 (pulling a model into a pull-off table) does not produce a model (it is a pure sequencing decision), it cannot violate a sequencing rule. Therefore, we assign an arc weight of zero to all transitions of type 1. For the transition of types two and three, which produce a model, we use the number of violations of sequencing rules as arc weights. With the Heaviside step function

$$\Theta(x) = \begin{cases} 1, \text{ if } x > 0 \\ 0, \text{ if } x \leq 0 \end{cases},$$

we can calculate the weight of an production arc from node $[i, K_i, ACT_i]$ to node $[i+1, K_{i+1}, ACT_{i+1}]$ as

$$f = \sum_{o=1}^{O} \Theta \left(a_{om} \cdot \left(\sum_{t=1}^{N_o - 1} act_i^{o,t} + a_{om} - H_o \right) \right).$$

With this graph problem formulation at hand, we can solve the CRSP by finding the shortest path from start to target node. However, instead of constructing the complete graph before computing the shortest path in two successive steps, both steps can be unified in a dynamic programming procedure. For this purpose node set V is subdivided into $T \cdot (P+1) + 2$ stages, where a stage (j, k) contains all nodes $V_{(j,k)} \subset V$, where j models are definitely fixed and $k = |K|$ models are stored in a pull-off table (see Figure 4.2). This way, a forwardly directed graph arises, which means that an arc can only point from a node of stage (j, k) to a node of stage (j', k'), if $j < j' \vee (j = j' \wedge k < k')$ holds. In particular, a node of stage (j, k) can only be connected with nodes of the following stages: $(j, k+1)$ (put current model in pull-off table), $(j+1, k-1)$ (reinsert model from pull-off) or $(j+1, k)$ (produce current model). This way, a stage-wise generation of the graph and a simultaneous evaluation of the shortest path to any node is enabled, where j and k are brought into lexicographic order. Thus, only two stages of the graph have to be stored simultaneously, because the shortest path to a node of stage $t+1$ is composed of a shortest path to a node of stage t (already determined and stored) and the connecting arc. Among all paths to a node, one with a minimal sum of arc weights is to be selected. The length-minimizing node is stored as the predecessor in the shortest path together with the length of this path. After reaching the final state in stage $(T+1, 0)$, the optimal path can be retrieved in a backward direction stage-by-stage using the stored predecessor nodes.

4.4 Search algorithms for the CRSP

In Section 4.3, we transformed CRSP into a graph search problem, where the aim is to find a shortest path from the start node to the target node. A shortest path corresponds to an optimal solution to CRSP. For finding a shortest path in a graph, different exact and heuristic search strategies are available. We propose three exact approaches, namely breadth-first search, iterative beam search, and A* search, and one heuristic beam search approach.

4.4.1 Breadth-first search

For the breadth-first search (BFS), we subdivide the node set V into $T \cdot (P+1) + 2$ different stages. For all nodes in one stage, the number j of models that

are already processed and the number $k = |K|$ of models stored in the pull-off tables are equal. Therefore, a stage (j, k) contains all nodes $V_{(j,k)} \subset V$. By subdividing V into different stages, we construct a forwardly directed graph. An arc can only point from a node of stage (j, k) to a node of stage (j', k'), if $j < j' \vee (j = j' \wedge k < k')$ holds. As outlined in Section 4.3.2, a node of stage (j, k) can only be connected with nodes of the following stages:

1. $(j, k + 1)$ (put current model in pull-off table),
2. $(j + 1, k)$ (produce current model), or
3. $(j + 1, k - 1)$ (reinsert model from pull-off table and produce it).

If we bring j and k into lexicographic order, a stage-wise generation of the graph and a simultaneous evaluation of the shortest path to any node is enabled. Starting with the start node $[0, \emptyset, ACT^0]$ in stage $(0, 0)$, we step-wise construct all nodes per stage until we reach the target node $[T + 1, \emptyset, ACT^0]$ in stage $(T + 1, 0)$. We obtain the reshuffled sequence of models by a simple backward recursion along the shortest path.

In comparison to a full enumeration of all possible sequences, this BFS approach considerably reduces the computational effort. We can obtain a further speed-up by using upper and lower bounds. For each node, we can determine a lower bound LB on the length of the remaining path to the target node. Furthermore, a global upper bound UB can be determined upfront by, for example, a heuristic. A node can be fathomed, if LB plus the length of the shortest path to the node is equal to or exceeds the UB.

We determine a simple lower bound based on the relaxation of the limited resequencing flexibility. Fliedner and Boysen (2008) showed for the CSP that in a sequence of t remaining cycles the maximum number of cycles D_{ot}, which may contain an option o without violating a given $H_o : N_o$-rule, can be calculated as $D_{ot} = \lfloor \frac{t}{N_o} \rfloor \cdot H_o + \min\{\max\{H_o - occ_t(act_i^o), 0\};\ t \bmod N_o\}$, where $occ_t(act_i^o)$ is the number of occurrences of option o in the first $t \mod N_o$ positions of act_i^o. Consequently, D_{ot} is a lower bound on the remaining options not yet scheduled. With $\pi_0(j)$, where $j = i, \ldots, T$, denoting the model at position j in the initial sequence, we obtain for each node $[i, K, act]$ a lower bound on the number of violations of sequencing rules caused by the not-yet-produced models:

$$LB = \sum_{o=1}^{O} \max \left\{ 0;\ \sum_{j=i}^{T} a_{o\pi(j)} + \sum_{m \in K} a_{om} - D_{o,T-i} \right\}. \tag{4.9}$$

The first term $(\sum_{j=i}^{T} a_{o\pi(j)})$ counts the options necessary for the remaining models not yet scheduled; the second one $(\sum_{m \in K} a_{om})$ counts the options necessary for the models stored in the pull-off tables. The sum of both terms

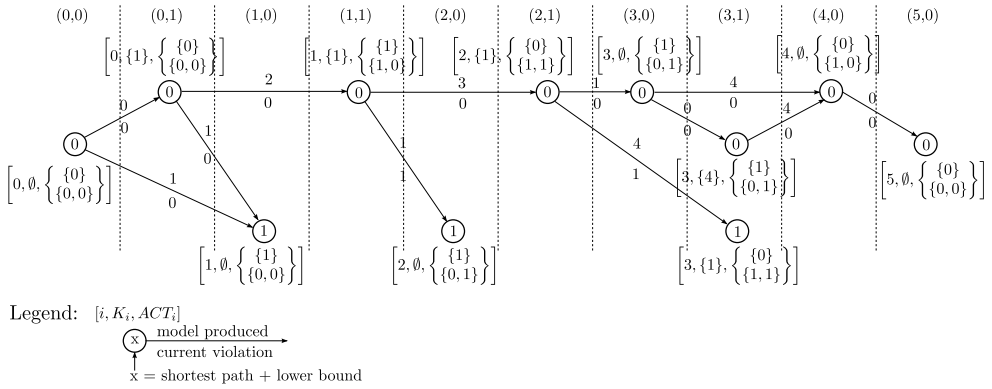


Figure 4.2: Example graph for BFS with $UB = 1$

should be smaller than the maximum number D_{ot} of option occurrences that are allowed for the remaining $t = T - i$ production cycles. To avoid that negative violations of one option, i.e., excessive production of a particular option, compensates violations of sequencing rules for a different option, we use an additional max function. The bound sums up the rule violations over all available options. The bound can be calculated very fast in $O(O)$.

Example: Figure 4.2 shows the resulting graph for our aforementioned example, when BFS is applied with $UB = 1$. We start with the initial state $[0, \emptyset, \{\{0\}, \{0, 0\}\}]$. If model 1 is produced (instead of pulling it into the pull-off table), we would reach node $[1, \emptyset, \{\{1\}, \{0, 0\}\}]$. Then, with regard to option 2, three option occurrences need to be scheduled in the remaining three production cycles. However, since only $D_{23} = \lfloor \frac{3}{3} \rfloor \cdot 2 + \min\{2; 3 \bmod 3\} = 2$ options can be scheduled, one rule violation is inevitable and the lower bound on the number of rule violations caused by the remaining models becomes $LB = 1$ for node $[1, \emptyset, \{\{1\}, \{0, 0\}\}]$. Therefore, it is optimal to put model 1 into the pull-off table and end up in node $[0, \{1\}, \{\{0\}, \{0, 0\}\}]$. Then, model 2 is finally released and stage $(1, 1)$ is reached. Here, pulling model 1 online from pull-off table would cause a rule violation, so that the given upper bound cannot be improved and instead model 3 is produced. Then, producing model 4 would lead to a rule violation, so that model 1 is pulled online and the shortest path from the start to the target node is obtained by producing models 2 and 3 first, then model 1 out of the pull-off table and finally model 4.

We want to further speed up the search by defining dominance rules. Dominance rules allow fathoming of nodes if other nodes, which have already been inspected, lead to equal or better solutions. For specifying a dominance rule, we introduce two definitions, which are an adoption of the concepts developed by Fliedner and Boysen (2008) for the CSP.

Definition 4.1. An active sequence ACT_i is *less or equally restrictive* than an active sequence ACT_j, denoted by $ACT_i \leqslant ACT_j$, if it holds that $act_i^{o,t} \leq act_j^{o,t} \forall\, o = 1, \ldots, O;\ t = 1, \ldots, N_o - 1$.

Definition 4.2. The content K_i of a node's pull-off tables is *less or equally demanding* than content K_j of another node, denoted by $K_i \leqslant K_j$, if there exists a mapping between K_i and K_j (with $|K_i| = |K_j|$) such that for each pair of models $m \in K_i$ and $m' \in K_j$ of this mapping $a_{om} \leq a_{om'} \forall\, o = 1, \ldots, O$ holds.

Dominance rule: A node $s = [i, K_i, ACT_i]$ with rule violations $f(s)$ (length of shortest path to s) dominates another node $s' = [i, K_i', ACT_i']$ with $f(s')$ and $|K_i| = |K_i'|$, if it holds that $f(s) \leq f(s')$, $K_i \leqslant K_i'$, and $ACT_i \leqslant ACT_i'$.

Proof: The proof consists of two parts: First, we show that a node $s = [i, K_i, ACT_i]$ dominates another node $s' = [i, K_i', ACT_i']$, if $f(s) \leq f(s')$, $K_i = K_i'$, and $ACT_i \leqslant ACT_i'$. Then, we prove that s dominates s', if $f(s) \leq f(s')$, $ACT_i = ACT_i'$, and $K_i \leqslant K_i'$. If both parts hold, the combination of them, as defined in the dominance rule, also holds.

(First part) Since the models stored in the pull-off tables are the same for both nodes s and s' ($K_i = K_i'$), the same remaining models have to be processed. If we assume that $ACT_i \leqslant ACT_i'$, for any possible sequence of the remaining models, ACT_i leads to a lower or at most the same number of rule violations than ACT_i'. Since $f(s) \leq f(s')$, s leads to a solution better than or equal to s'.

(Second part) Deleting option occurrences from a sequence of remaining models (for example, by storing models in pull-off tables) leads to fewer or at most the same number of rule violations caused by the remaining models that have to be processed. With $K_i \leqslant K_i'$, we can construct for any sequence of remaining models, which is possible for s', a counterpart sequence for s with this condition. Therefore, starting with the same active sequence ($ACT_i = ACT_i'$), s leads to a lower or equal number of rule violations than s'. With $f(s) \leq f(s')$, s results in a solution better than or equal to s'. □

Example: Consider two pull-off tables and an initial sequence of four models. We have two options for which a 1:2- and a 2:3-rule holds, respectively. Figure 4.3 depicts two decision points and their respective nodes s and s'. s dominates s', because $f(s) = f(s') = 0$, the contents of the pull-off tables are equally demanding, and the active sequence of s is less restrictive than that of s'.

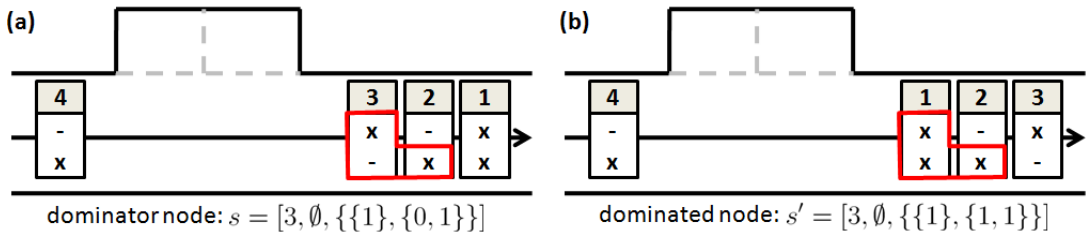


Figure 4.3: Example for dominance rule

4.4.2 Beam search

Beam search (BS) is a truncated BFS heuristic and was first applied to speech recognition systems by Lowerre (1976). Ow and Morton (1988) were the first to systematically compare the performance of BS and other heuristics for two scheduling problems. Since then, BS was applied within multiple fields of application and many extensions have been developed, e.g., stochastic node choice (Wang and Lim, 2007) or hybridization with other meta-heuristics (Blum, 2005), so that BS turns out to be a powerful meta-heuristic applicable to many real-world optimization problems. A review of these developments is provided by Sabuncuoglu et al. (2008).

Like other BFS heuristics, BS uses a graph formulation of a problem and searches for the shortest path from a start to a target node. However, unlike BFS or a breadth-first version of branch&bound, BS is not optimal since the number of nodes that are branched in each stage is bounded by the beam width BW. If BW is equal to the maximum number of nodes in a stage, BS becomes BFS. The BW nodes to be branched are identified by a heuristic in a filtering process. Starting with the root node in stage 0, all nodes of stage 1 are constructed. Then, the filtering process of BS selects all nodes in stage 1 that should be branched. Typical approaches are the use of priority values, cost functions, or multi-stage filtering, where several filtering procedures are consecutively applied (Sabuncuoglu et al., 2008). The BW best nodes found by filtering form the promising subset of stage 1. These nodes are further branched. The filtering and branching steps are iteratively applied until the target node is reached. Analogously to other tree search methods like BFS, we can use the bounding argument and dominance rule formulated in Section 4.4.1 for BS to reduce the number of nodes to be branched. To apply BS, we must define a proper graph structure and a filtering mechanism:

Graph structure: For BS, we can use the acyclic digraph $G(V, E, r)$ introduced in Section 4.3. BS examines the $T(P + 1) + 2$ stages in lexicographic order.

Filtering: For each node, we calculate the objective value (number of rule

violations) of the current partial solution, which is the sum of arc weights along the shortest path from the root node to the current node, and add the lower bound argument of equation (4.9), which is the estimated path weight from the current node to a target node. Then, we order the nodes according to the estimated overall cost and select the BW nodes with lowest cost.

4.4.3 Iterative beam search

Iterative beam search (IBS) is conducted by applying a series of n BS iterations with gradually increasing beam width BW. With larger beam width, more nodes are explored, which increases the probability of finding the optimal path. If the beam width of the nth iteration is equal to the maximum number of nodes in a stage, a BFS is conducted and IBS is optimal. To speed up the search, the solution found by the ith Beam Search BS_i, with $i = 1, \ldots, n-1$, is used as an upper bound for the subsequent Beam Search BS_{i+1}.

4.4.4 A* search

Unlike the aforementioned algorithms, A* search (Hart et al., 1968) does not perform a stage-wise exploration of the decision tree but traverses the tree along the nodes that appear to be most likely on the shortest path from the start to the target node. For each explored node during the search process, we calculate the costs g of the path from the start node to the current node and add a heuristic function h, which estimates the remaining path costs from the current to the target node. The search examines the nodes in ascending order, according to their overall cost value $g + h$, and returns the first solution found. When using the lower bound argument (4.9) for the heuristic function h, h is admissible, hence never overestimates the remaining costs to the target node. Therefore, A* becomes an exact algorithm and returns the global best solution.

A* requires a large number of nodes to be stored during the search process since all explored nodes have to be stored in a list ordered with respect to the estimated overall cost. We can reduce this number by removing all nodes from the list whose overall cost is equal to or exceeds a global upper bound UB. In addition, the dominance rule introduced in Section 4.4.1 can also be used to reduce the number of stored nodes.

4.5 Computational study

4.5.1 Experimental setup

We study the performance of the search approaches and the influence of the number of pull-off tables with two sets of car sequencing instances. To apply CRSP on a car sequencing instance, an initial sequence π_0 is constructed randomly which is then resequenced using P pull-off tables. As a first set, we use the test instances provided by Fliedner and Boysen (2008). These 18 instances have 10-50 production cycles, 3-7 options, and 5-28 models. The second set stems from the CSPLib and contains 9 larger instances with 100 production cycles, 5 options each, and 19-26 models. All algorithms were implemented with VB.Net. Throughout the experiments, violations of instances in the first set are measured using the approach by Fliedner and Boysen (2008) and violations of the second set are counted with the sliding-window approach (Gravel et al., 2005). The experiments run on a Pentium 2.5 Ghz processor with 2 GB RAM.

4.5.2 Algorithmic performance

First we compare the performances of the proposed search algorithms on both problem sets. We also apply the commercial standard solver CPLEX 12.2 with our binary linear model from Section 4.2 as a benchmark, to better assess the solution qualities of the algorithms. For each instance of the problem sets, ten different initial sequences are constructed randomly. The number of rule violations of each initial sequence serves as an initial global upper bound UB. We use a fixed number of pull-off tables $P = 4$. We set the time limit to solve each of the ten sequences to 600 seconds and stop an algorithm if it exceeds this limit. BFS, IBS and A*, are applied with the dominance rule from Section 4.4. IBS is conducted with four iterations and beam widths $BW_1 = 10$, $BW_2 = 100$, $BW_3 = 1000$, $BW_4 = \infty$. Since $BW_4 = \infty$, IBS returns the optimal solution. For BS, we set $BW = 300$.

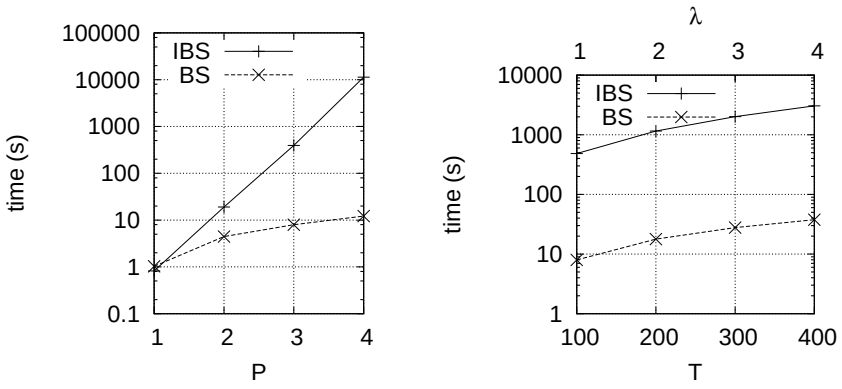
Table 4.2 compares the average performance of the search algorithms for solving one problem instance. An instance is characterized by the number of production cycles T, the number of options O and the number of models M. For the different search algorithms, we list the average objective value obj, average CPU time consumed in seconds, and the average number of explored nodes. BFS, A*, IBS and CPLEX return the optimal solution, if the search does not exceed the time limit, otherwise BFS and A* return no solution at all. Comparing the optimal algorithms, A* is slightly favored for small problem instances; for large problem instances, IBS shows the best results and outperforms BFS, A* and CPLEX in terms of solution quality and runtime.

Although BS is a heuristic and we have no guarantee of finding the optimal solution, the solutions found are close to the optimum, whereas it explores only a fraction of nodes and, thus, requires less CPU time compared to the optimal algorithms. Since the performances of BFS, A* and CPLEX are worse compared to IBS and BS, we do not consider these algorithms for further experiments.

On problem set two we study how the performance of BS and IBS depends on the number of pull-off tables P and the number of production cycles T. Figure 4.4(a) shows the average time consumed with varying P. BS performs best since the time required is low and increases approximately linearly with increasing P. In contrast to BS, the solution time grows approximately exponentially for IBS.

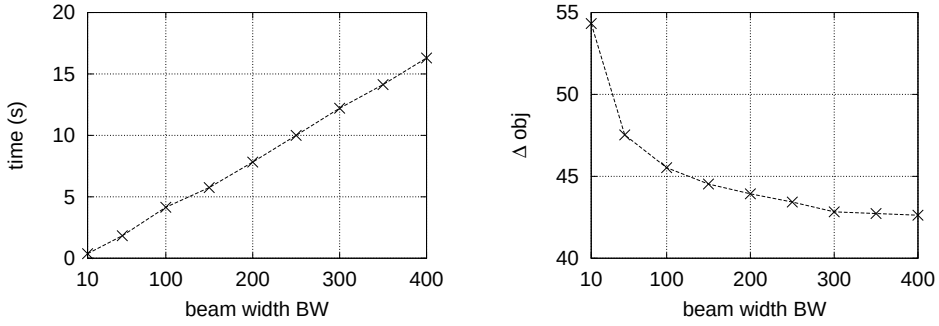
Problem	T	O	M	UB	BFS			A*			IBS			BS			CPLEX		
					obj	time(s)	nodes	obj	time(s)	nodes	obj	time(s)	nodes	obj	time(s)	nodes	obj	time(s)	nodes
CAR_3_10	10	3	5	5.7	1.0	0.06	1081	1.0	<0.01	83	1.0	0.04	437	1.0	0.13	4707	1.0	0.04	20
CAR_5_10	10	5	5	8.3	1.2	0.42	3565	1.2	<0.01	75	1.2	0.05	379	1.2	0.25	5923	1.2	0.06	17
CAR_7_10	10	7	9	11.9	2.4	4.04	9726	2.4	0.02	216	2.4	0.06	665	2.4	0.42	6872	2.4	0.11	23
CAR_3_15	15	3	5	10.2	2.4	0.27	3490	2.4	<0.01	246	2.4	0.10	885	2.4	0.34	11308	2.4	0.25	238
CAR_5_15	15	5	7	15.0	3.4	6.7	18843	3.4	0.04	421	3.4	0.10	1202	3.4	0.68	13592	3.4	0.90	419
CAR_7_15	15	7	13	21.7	6.6	209.73	86719	6.6	2.2	4032	6.6	1.20	8391	6.6	1.05	14260	6.6	2.06	737
CAR_3_20	20	3	6	14.9	3.8	0.77	6912	3.8	0.08	1084	3.8	0.16	2882	3.8	0.63	18943	3.8	1.22	818
CAR_5_20	20	5	7	20.8	6.1	19.27	39556	6.1	0.82	3668	6.1	0.98	10212	6.1	1.08	20819	6.1	3.74	1267
CAR_7_20	20	7	15	25.8	-	>600	-	7.1	14.66	14812	7.1	8.31	27802	7.1	1.78	21653	7.1	16.95	4261
CAR_3_30	30	3	6	21.6	6.1	11.64	290462	6.1	0.23	2517	6.1	0.45	7410	6.1	1.08	31724	6.1	15.32	4879
CAR_5_30	30	5	11	30.4	8.8	588.01	6509516	8.8	51.46	43992	8.8	34.47	73278	8.8	2.13	35704	8.8	276.69	47648
CAR_7_30	30	7	23	45.6	-	>600	-	-	>600	-	14.4	323.94	195874	14.5	3.47	36424	14.8	432.75	54011
CAR_3_40	40	3	7	31.7	11.9	31.82	725502	11.9	2.47	2910	11.9	3.30	31081	12.2	1.64	45145	11.9	38.52	9697
CAR_5_40	40	5	13	41.9	-	>600	-	13.8	307.63	143799	13.8	215.42	213772	14.4	3.17	50277	14.4	599.38	59748
CAR_7_40	40	7	26	57.5	-	>600	-	-	>600	-	19.6	>600	-	20.4	5.15	51244	21.7	>600	38162
CAR_3_50	50	3	7	40.8	17.2	48.7	1079206	17.2	8.02	28446	17.2	11.13	64211	17.2	2.17	58818	17.2	136.48	20228
CAR_5_50	50	5	14	54.0	-	>600	-	-	>600	-	21.0	577.27	397594	21.7	4.22	64638	22.7	>600	40186
CAR_7_50	50	7	28	79.4	-	>600	-	-	>600	-	30.9	>600	-	32.5	6.87	65891	34.0	>600	30504
4-72	100	5	22	131.1	-	>600	-	-	>600	-	37.8	>600	-	40.9	12.22	137977	41.7	>600	8107
6-76	100	5	22	118.8	-	>600	-	-	>600	-	27.5	>600	-	30.0	12.10	136812	31.2	>600	7585
10-93	100	5	25	156.0	-	>600	-	-	>600	-	53.7	>600	-	57.7	12.90	138447	58.2	>600	7552
16-81	100	5	26	139.6	-	>600	-	-	>600	-	43.2	>600	-	47.4	13.22	138608	49.4	>600	7091
19-71	100	5	23	151.4	-	>600	-	-	>600	-	47.2	>600	-	51.7	11.93	136509	53.3	>600	8979
21-90	100	5	23	139.6	-	>600	-	-	>600	-	42.8	>600	-	46.0	12.33	137376	46.2	>600	8869
36-92	100	5	22	145.6	-	>600	-	-	>600	-	46.7	>600	-	49.9	11.85	137209	50.3	>600	8805
41-66	100	5	19	119.8	-	>600	-	-	>600	-	35.5	>600	-	37.7	11.03	136916	38.0	>600	7085
26-82	100	5	24	134.3	-	>600	-	-	>600	-	37.0	>600	-	39.6	12.28	137779	41.0	>600	7599

Table 4.2: Average performances over 10 runs ($P = 4$)



(a) average time against P (b) average time against T

Figure 4.4: Performance of BS and IBS against P and T

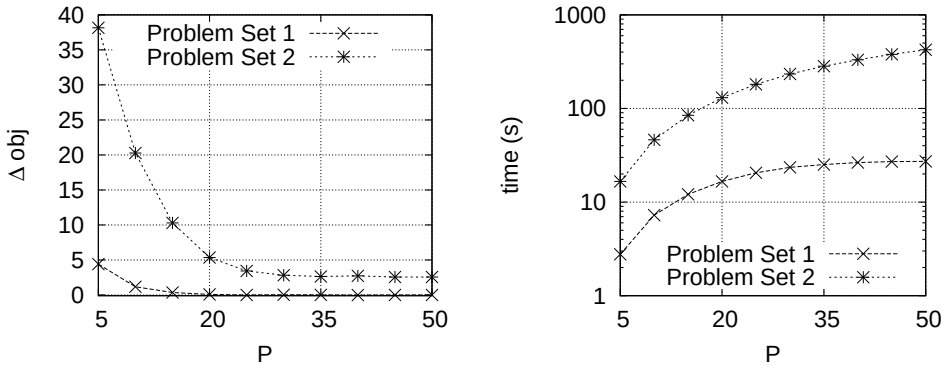


(a) time against BW (b) deviation from optimum against BW

Figure 4.5: Performance of BS against beam width BW

To study how algorithm performance depends on T, we simply duplicate for every instance the demand of each model by a factor $\lambda \in \{1, \ldots, 4\}$. Thus, for, e.g., $\lambda = 2$ each instance in set two has 200 production cycles. All other parameters of the problem instances remain unaffected and the number of pull-off tables is set to $P = 3$. Figure 4.4(b) shows the average time against the number of production cycles T derived from λ. Certainly, T increases linearly with λ. When modeling the CRSP as a graph problem (see Section 4.3), the time necessary for the algorithms increases roughly linearly with T.

Finally, we study the influence of the beam width BW on the performance of BS. We set $P = 4$ and varied BW between 10 and 400. Figure 4.5(a) shows the time consumed against BW, which increases approximately linearly with increasing BW. Figure 4.5(b) shows the average absolute deviation



(a) deviation from optimum against P (b) time against P

Figure 4.6: Resequencing flexibility

$\Delta obj = obj^{BS} - obj^*$ between the objective values obj^{BS} produced by BS and the best known objective values obj^* as stated in the CSPLib. Therefore, Δobj measures the number of additional rule violations in comparison to the sequence obtained when solving the original CSP. The solution quality of BS slightly increases with larger BW. Since a larger beam width ($BW \geq 300$) has only a minor effect on the solution quality, we see a beam width of 300 sufficient for the studied problem instances.

In summary, BS performs well in comparison to the exact search approaches. BS finds solutions close to optimal solutions, but explores considerably fewer nodes and, therefore, requires less CPU time.

4.5.3 Resequencing flexibility

We focus on BS and study on both problem sets how solution quality depends on the number of pull-off tables P. With increasing P, planning flexibility increases, we are less dependent on the initial sequence π_0, and we are able to build better sequences more similar to solutions of the CSP. For our study, we construct ten random initial sequences π_0 per instance and set $BW = 300$. The number P of pull-off tables varies between 5 and 50 in steps of 5.

Figure 4.6(a) shows the average absolute deviations $\Delta obj = obj^{BS} - obj^*$ between the solution found by BS and the best known objective value of the CSP for both problem sets. For low P, a randomly created initial sequence π_0 has a large impact on the resulting sequence and the deviation is high. With increasing P, π_0 has a lower impact and we can construct a better new sequence with fewer rules violations by using the pull-off tables. For larger P, the resulting sequences become more similar to the optimal sequence of the CSP. The plot shows that increasing P up to approximately 20 for set one increases

problem	T	O	M	obj*	BS			
					obj	P'	time	nodes
CAR_3_10	10	3	5	1	1	5	0.16	5422
CAR_5_10	10	5	5	1	1	5	0.30	6714
CAR_7_10	10	7	9	2	2	10	0.72	9055
CAR_3_15	15	3	5	2	2	10	0.80	22315
CAR_5_15	15	5	7	2	2	10	1.73	23384
CAR_7_15	15	7	13	4	4	10	3.06	24754
CAR_3_20	20	3	6	3	3	10	1.86	37349
CAR_5_20	20	5	7	3	3	10	3.17	39602
CAR_7_20	20	7	15	3	3	20	9.08	51027
CAR_3_30	30	3	6	4	4	10	3.57	67906
CAR_5_30	30	5	11	3	3	15	11.77	96315
CAR_7_30	30	7	23	4	4.7	25	36.57	123211
CAR_3_40	40	3	7	5	5	15	9.39	138435
CAR_5_40	40	5	13	5	5	25	33.36	198632
CAR_7_40	40	7	26	9	7.7*	20	52.72	176986
CAR_3_50	50	3	7	6	6	20	17.38	229299
CAR_5_50	50	5	14	8	9	25	51.29	276400
CAR_7_50	50	7	28	12	11.4*	40	149.07	350858
4-72	100	5	22	0	3	50	416.05	1124836
6-76	100	5	22	6	6.1	45	355.56	1046226
10-93	100	5	25	3	6.7	50	452.48	1125511
16-81	100	5	26	0	3.1	45	426.51	1048913
19-71	100	5	23	2	6.5	30	218.97	772178
21-90	100	5	23	2	3.8	45	384.86	1049156
36-92	100	5	22	2	3.6	35	288.51	871502
41-66	100	5	19	0	0.5	25	156.78	665434
26-82	100	5	24	0	3.6	30	246.20	774043

Table 4.3: Average best results of BS

solution quality. Adding pull-off tables give us more flexibility and allow us finding better sequences. For example, for $P = 20$, BS finds for problem set one sequences that violate on average $\Delta obj = 0.09$ more sequencing rules than the best known sequence of the CSP. Using larger number of pull-off tables ($P > 20$) does not improve solution quality on set one. For problem set two, an increase of the solution quality can be observed up to $P = 30$ where on avg. an additional 2.83 sequencing rules are violated compared to the optimum. For larger $P(> 30)$, the solution quality can only be slightly further improved. The remaining deviation from the optimal solution comes from the heuristic character of BS. Regarding Figure 4.6(b), all instances can be solved within

the 600-seconds time limit. For example, on set 2 with $P = 50$, BS requires 424.3 seconds on avg. Again, it can be observed that the solution time of BS increases about linearly with increasing P.

Table 4.3 lists the best average results found for each instance and the minimum number P' of pull-off tables leading to this result. Note that for instances CAR_7_40 and CAR_7_50 (marked with * in Table 4.3), we were able to find a new best solution with only 7, resp. 11, rule violations. Clearly, the practitioner when deciding on an appropriate buffer dimension has to balance the elementary trade-off between investment cost for pull-off table installation and the gains of additional resequencing flexibility. Especially, the latter effect is hard to quantify, since determining an appropriate option-specific cost factor for a sequencing rule violation is hardly possible (Boysen et al., 2009). However, our results reveal that the number of pull-off tables to be installed heavily depends on the number M of models to be considered and number T of production cycles. Especially, it can be concluded that adding more than $P' = \frac{T}{2}$ pull-off tables does not seem advisable, since on average over all instances $P' = 0.459T$ leads to (nearly) full resequencing flexibility.

4.6 Comparison with a real-world scheduling rule

For the production of large commercial vehicles like trucks, buses, or construction vehicles, investment costs for conventional AS/RS having multiple buffer places (see Section 4.1) are prohibitively high; therefore only a few random access buffers are installed, e.g., to decouple paint shop and final assembly. In this context, the following resequencing setting is taken from a major German truck manufacturer.

To regain a desirable model sequence after paint shop and before final assembly, the manufacturer installed a resequencing system consisting of 118 buffer places. Since quality defects in the paint shop cause a rework rate of about 85%, sequences are heavily stirred up and a resequencing is inevitable. As many buffer places are occupied over a longer period by driving cabs waiting for critical parts not yet delivered, only 10 to 20 of these buffer places are actually available for resequencing. Typically, there are about 50 cabs in the overlap area between paint shop and final assembly.

As the model variation is large (for example, trucks have a different number of aisles which makes some trucks more than twice as long as others), production times of different models are very heterogeneous. To deal with the variation, the manufacturer considers 20 sequencing rules as hard constraints which have to be considered for resequencing. However, resequencing also affects material supply. The originally planned sequence π_{-1}, which was

disordered to initial sequence π_0 within paint shop, was propagated to the suppliers and material supply was organized on the basis of the planned sequence. Therefore, parts are stored next to the line according to the planned sequence of trucks (just-in-sequence). Creating a reordered sequence π_1 that strongly differs from the originally planned sequence π_{-1}, makes a reordering of these parts to be executed by additional logistics workers necessary. To minimize the effort for material re-shuffling resequencing aims at approximating the original material demand induced by the planned sequence as close as possible. For this purpose, a deviation measure r_{mt} can be computed, which measures the number of deviations between the material demand a_{om} of a model m and the original material demand $a_{o\pi_{-1}(t)}$ planned for the production cycle $t = 1, \ldots, T$ within the original sequence π_{-1} as follows:

$$r_{mt} = \sum_{o=1}^{O} |a_{o\pi_{-1}(t)} - a_{om}| \quad \forall\, m = 1, \ldots, M;\ t = 1, \ldots, T\,. \tag{4.10}$$

Note that other deviation measures can be employed and facultative parts can be considered, i.e., not necessarily those for which sequencing rules are defined. Up to now, the resequencing decision is made by a dispatcher who makes an online decision for the next model to be fed into final assembly. The decision process is based on the following simple rules:

- Fill strategy: The dispatcher subsequently draws cabs from the waiting queue into a buffer until all buffer places are fully occupied.
- Release strategy: The dispatcher selects a model for production from the currently available models, which violates no sequencing rule and minimizes material deviations for current production cycle.

The current selection policy suffers from the myopic choice of only a single model. Alternatively, our car resequencing approach can be adapted for determining a complete (reshuffled) model sequence, which minimizes re-shuffling effort and avoids sequencing rule violations. The objective function (4.1) and constraint (4.5) of our model (see Section 4.2) have to be modified with (4.11) and (4.12), resp.:

$$\text{Minimize} \sum_{i=1}^{T} \sum_{m=1}^{M} \sum_{t=1}^{T} x_{itm} \cdot r_{mt} \tag{4.11}$$

$$\sum_{i=1}^{T} \sum_{\tau=t}^{\min\{t+N_o-1,T\}} \sum_{m=1}^{M} x_{i\tau m} \cdot a_{om} \leq H_o \quad \forall\, o = 1, \ldots, O;\ t = 1, \ldots, T \tag{4.12}$$

(4.11) minimizes the number of material deviations and (4.12) ensures that no sequencing rule is violated. Additionally, some modifications of our graph approach are required as well:

- Only those nodes s are feasible that cause no sequencing rule violation ($f(s) = 0$). The lower bound (4.9) can still be used. Therefore, any node with $LB \geq 1$ can be fathomed.
- The weight of an arc is defined as the contribution r_{mt} of current model m chosen for production at decision point t. The shortest path from the start to the target node returns a sequence with minimum overall material deviations.
- A lower bound on the overall deviations can be determined by relaxing car-sequencing rules. In the unconstrained case, the optimal assignment of models to cycles can be determined by solving an assignment problem (see, e.g., Kuhn, 1955) minimizing realized deviation measures r_{mt}. The lower bound fathoms nodes which cannot result in a better solution value than the incumbent (upper bound) solution.
- The dominance rule (Section 4.4.1) cannot be used since it can exclude optimal solutions.

To compare our resequencing approach with the real-world (human) decision rule, we construct ten test problems each with 50 production cycles, 20 options, and 15 buffers (pull-off tables). Sequencing rules are constructed randomly with $N_o = \{2 \dots 10\}$ and $H_o = \{1 \dots N_o\}$. An optimal sequence is obtained by randomly assigning options to cycles, such that no sequencing rule is violated. The average usage rate for the options is 25%. This optimal sequence of models serves as initially planned sequence π_{-1}. To simulate rework in the paint shop, the optimal sequence is modified by randomly pulling models out of the sequence (on average 85% of the models) and re-inserting them into the sequence in a random position between their original position and the 25 following positions. This leads to the initial sequence π_0, which has to be reshuffled. For every problem, we create ten different initial sequences π_0 leading to 100 problem instances overall. For resequencing, we apply BS with $BW = 300$ on the graph modified according to the aforementioned suggestions.

For each instance, Table 4.4 compares the average material deviation *obj* of the solution found by BS with the solution using the real-world decision rule. We also show the improvement (in %) and the avg. time and avg. nodes needed for solving the instance. Clearly, BS finds better sequences and considerably reduces the number of material deviations. On average, BS overcomes about 29% of the currently necessary effort for material reshuffling. Note, that all instances can be solved within the time limit of 600 seconds.

instance	decision rule	BS			
	obj	obj	time (s)	nodes	improv.
1	41.2	26.4	553.37	164308	35.9 %
2	45	29.6	599.60	176294	34.2 %
3	25.4	14.8	557.45	170522	41.7 %
4	39.8	31	558.41	172775	22.1 %
5	11	8.4	504.19	167743	23.6 %
6	47.8	29.4	538.41	167793	38.5 %
7	46.8	35.4	548.39	164162	24.4 %
8	23.2	16.8	571.80	168192	27.6 %
9	61.4	43.4	576.91	163506	29.3 %
10	49.6	42.8	473.98	159186	13.7 %
avg.	39.12	27.8	548.25	167448	28.9 %

Table 4.4: average overall material deviations r

In the real world, our resequencing approach should be applied in a rolling horizon, where only a subset of models, e.g., the first 10, are definitely fixed and the remaining (about 40) models are reinserted into the successive planning run. In this case, our graph approach must not be started with the initial node (empty pull-off tables) but with the one representing current buffer content, when starting the planning run. Note that our basic graph structure can also be applied for solving related resequencing problems. As the adoptions are truly straightforward, e.g., to incorporate other functions for counting rule violations like the sliding-window technique (Gravel et al., 2005), to distinguish between hard- and soft-sequencing rules (Solnon et al., 2008) or to pursue alternative resequencing objectives like leveling the required material (Drexl and Kimms, 2001) while avoiding rule violations, we abstain from a detailed description.

4.7 Conclusion

This paper deals with the car resequencing problem, where a number of pull-off tables can be used to reshuffle an initial sequence of car models in such a way that violations of car sequencing rules are minimized. We transform the car resequencing problem into a graph search problem, which considerably reduces solution effort, and develop efficient exact and heuristic solution approaches. To further speed up search, we develop a lower bound as well as a dominance rule which both can be used for fathoming nodes in the search graph.

For a set of test instances, we compare the performance of problem-specific variants of BFS, IBS, A* search, and a BS heuristic. The computational effort for BFS is higher than for IBS or A* search. A* needs less effort for

small problem instances than IBS, whereas IBS is faster than A* for larger problem instances. The heuristic BS approach finds solutions very close to the optimal ones but needs much less computational effort in comparison to the exact search approaches. Studying how the number of pull-off tables influences the quality of the reshuffled sequence reveals that about $\frac{T}{2}$ pull-off tables are sufficient to reach nearly full resequencing flexibility for the considered problem instances.

Finally, we present a real-world resequencing setting from a major German truck producer and illustrate how the approach can be adapted for solving the respective problem. In comparison to the currently used real-world scheduling approach, our new resequencing approach can improve solution quality by on average about 29%.

There are several ways to build on our research. On the one hand, future research could approach the car resequencing problem applying different forms of buffer organization, e.g., mixed banks (Spieckermann et al., 2004). On the other hand, alternative sequencing objectives, like mixed-model sequencing (Boysen et al., 2009) could be modified to cope with limited resequencing flexibility due to a given number of pull-off tables.

Bibliography

Blum, C. (2005). Beam-ACO - Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591.

Boysen, N., Fliedner, M., and Scholl, A. (2009). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.

Boysen, N., Fliedner, M., and Scholl, A. (2010). Level scheduling under limited resequencing flexibility. *Flexible Services and Manufacturing Journal*, 22(3-4):236–257.

Choi, W. and Shin, H. (1997). A real-time sequence control system for the level production of the automobile assembly line. *Computers & Industrial Engineering*, 33(3-4):769–772.

Ding, F. Y. and Sun, H. (2004). Sequence alteration and restoration related to sequenced parts delivery on an automobile mixed-model assembly line with multiple departments. *International Journal of Production Research*, 42(8):1525–1543.

Drexl, A. and Kimms, A. (2001). Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints. *Management Science*, 47(3):480–491.

Epping, T., Hochstättler, W., and Oertel, P. (2004). Complexity results on a paint shop problem. *Discrete Applied Mathematics*, 136(2-3):217–226.

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

Gravel, M., Gagne, C., and Price, W. L. (2005). Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56(11):1287–1295.

Gusikhin, O., Caprihan, R., and Stecke, K. (2008). Least in-sequence probability heuristic for mixed-volume production lines. *International Journal of Production Research*, 46(3):647–673.

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Inman, R. (2003). ASRS sizing for recreating automotive assembly sequences. *International Journal of Production Research*, 41(5):847–863.

Inman, R. R. and Schmeling, D. M. (2003). Algorithm for agile assembling-to-order in the automotive industry. *International Journal of Production Research*, 41(16):3831–3848.

Kis, T. (2004). On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335.

Kuhn, H. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–87.

Lahmar, M. and Benjaafar, S. (2007). Sequencing with limited flexibility. *IIE Transactions*, 39(10):937–955.

Lahmar, M., Ergan, H., and Benjaafar, S. (2003). Resequencing and feature assignment on an automated assembly line. *IEEE Transactions on Robotics and Automation*, 19(1):89–102.

Lim, A. and Xu, Z. (2009). Searching optimal resequencing and feature assignment on an automated assembly line. *Journal of the Operational Research Society*, 60(3):361–371.

Lowerre, B. (1976). *The harpy speech recognition system*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA.

Ow, P. S. and Morton, T. (1988). Filtered beam search in scheduling. *International Journal of Production Research*, 26(1):35–62.

Parrello, B. D., Kabat, W. C., and Wos, L. (1986). Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2(1):1–42.

Sabuncuoglu, I., Gocgun, Y., and Erel, E. (2008). Backtracking and exchange of information: Methods to enhance a beam search algorithm for assembly line scheduling. *European Journal of Operational Research*, 186(3):915–930.

Solnon, C., Cung, V., Nguyen, A., and Artigues, C. (2008). 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.

Spieckermann, S., Gutenschwager, K., and Voß, S. (2004). A sequential ordering problem in automotive paint shops. *International Journal of Production Research*, 42(9):1865–1878.

Wang, F. and Lim, A. (2007). A stochastic beam search for the berth allocation problem. *Decision Support Systems*, 42(4):2186–2196.

Chapter 5

Iterative beam search for car sequencing

Uli Golle, Franz Rothlauf, Nils Boysen

Abstract

The car sequencing problem seeks a production sequence of different car models launched down a mixed-model assembly line. The models can be distinguished by selected options, e.g., sun roof yes/no. For every option, car sequencing applies a so-called sequencing rule to avoid that consecutive models requiring this option lead to a work overload of the respective assembly operators. The aim is to find a sequence with minimum number of sequencing rule violations. This paper presents a graph representation of the problem and develops an exact solution approach based on iterative beam search. Furthermore, existing lower bounds are improved and applied. The experimental results reveal, that our solution approach is superior compared to the currently best known exact solution procedure. Our algorithm can even be applied as an efficient heuristic on problems of real-world size with up to 400 cars, where it shows competitive results compared to the current best known solutions.

5.1 Introduction

The car sequencing (CS) problem (Parrello et al., 1986) is a NP-hard combinatorial optimization problem seeking a production sequence of various car models which are jointly produced on a mixed-model assembly line. The car models are distinguished by different binary options, e.g., having an air-conditioning or not. An accumulation of models with the same option in a sequence could lead to work overload of the operators on the line as they cannot accomplish their work within the available station limits. Such work overload would have to

be compensated by costly strategies like employing additional utility workers or stopping the line. To avoid such scenarios, CS applies so-called sequencing rules, which restrict the number of car models having an option in any subsequence of defined length. The aim is to find a sequence of car models, which satisfies all sequencing rules (constraint satisfaction problem) or a sequence with minimum number of rule violations (optimization problem).

Iterative beam search (IBS) is a truncated breadth-first search heuristic, which is iteratively conducted with ever increasing search width. If the search width is chosen large enough to cover the entire search space, IBS becomes an exact solution approach and returns the global best solution. Otherwise, a heuristic search is performed. Like other breadth-first search procedures, IBS works on graph representations of a problem, where it seeks a shortest path (with minimum costs) from the root node to a leaf node.

In this paper, we propose an IBS approach for solving CS considered as an optimization problem. Our procedure is based on a new graph representation of CS, which was inspired by Section 4.3, where a related approach for solving the resequencing version of CS has been developed. Here, so-called pull-off tables are applied to pull models offline and reinsert them in later production cycles in order to reshuffle the sequence before entering a successive production department. We modify this graph representation to the traditional CS problem. Additionally, we define and apply improved lower bounds for CS. Experimental results reveal, that our algorithm clearly outperforms the currently best known exact solution approach for CS, a scattered branch & bound algorithm (SB&B) (Fliedner and Boysen, 2008). We also show the superiority of our new lower bound arguments, as they significantly reduce the amount of evaluated nodes for IBS.

The outline of the paper is as follows. The next section reviews relevant literature and formulates CS as a mathematical model. In Section 5.3, we show how to develop an IBS approach for CS, based on a new graph representation and improved lower bounds. Several experiments are conducted in Section 5.4, that show the superiority of our algorithm to a recent solution approach in the literature. Section 5.5 gives concluding remarks and a brief outlook on future research.

5.2 Model formulation and literature

The CS problem was first introduced by Parrello et al. (1986). Given a pool of various car models, which can be distinguished by their required options (such as air conditioning or sun roof yes/no), CS aims to find a production sequence with minimum work overload. Therefore, it uses $H_o : N_o$ sequencing rules, which restrict the occurrences of option o in any subsequence of N_o succeeding cars to at most H_o. The objective of CS is to minimize the number of rule

T	number of production slots (index t)
M	number of models (index m)
O	number of options (index o)
d_m	demand for model m
a_{om}	binary demand coefficient: 1, if model m requires option o, 0 otherwise
$H_o : N_o$	sequencing rule: at most H_o out of N_o successively sequenced models require option o
x_{mt}	binary variable: 1, if model m is produced in slot t, 0 otherwise
y_{ot}	binary variable: 1, if sequencing rule defined for option o is violated in window starting in cycle t
BI	Big Integer

Table 5.1: Notation

violations. Thus, in contrast to the related mixed-model sequencing approach (Wester and Kilbridge, 1964), CS does only implicitly minimize the resulting work overload. With the notations from Table 5.1, we can formulate the CS problem as a mathematical model:

$$\text{CS: Minimize} \quad \sum_{o \in O} \sum_{t=1}^{T-N_o+1} y_{ot} \tag{5.1}$$

$$\sum_{t=1}^{T} x_{mt} = d_m \quad \forall m \in M \tag{5.2}$$

$$\sum_{m \in M} x_{mt} = 1 \quad \forall t = 1, \ldots, T \tag{5.3}$$

$$\sum_{t'=t}^{t+N_o-1} \sum_{m \in M} x_{mt'} \cdot a_{mo} \leq H_o + y_{ot} \cdot BI \quad \forall o \in O; t = 1, \ldots, T - N_o + 1 \tag{5.4}$$

$$x_{mt} \in \{0, 1\} \quad \forall m \in M; t = 1, \ldots, T \tag{5.5}$$

$$y_{ot} \in \{0, 1\} \quad \forall o \in O; t = 1, \ldots, T \tag{5.6}$$

Variable y_{ot} indicates whether a rule violation with regard to option o occurs in a subsequence starting at position t. Objective function (5.1) minimizes the number of rule violations. Constraints (5.2) assure that the produced models meet the required demand, whereas (5.3) enforces that exactly one model is produced in each slot t. Constraints (5.4) check whether a rule violation occurs. Finally, (5.5) and (5.6) ensure that variables x_{mt} and y_{ot} take only binary

values. The literature deals with various approaches on assessing the number of rule violations. Here, we state the widely-used sliding-window technique (SW) (Gravel et al., 2005). SW counts all complete subsequences of length N_o, in which a rule violation occurs. As SW tends to double count some rule violations and weights them differently depending on their position in the sequence, an alternative approach was introduced by Fliedner and Boysen (2008) (FB). FB counts all option occurrences leading to a rule violation, so that (5.4) is replaced with:

$$\sum_{t'=t}^{\min\{t+N_o-1,T\}} \sum_{m\in M} x_{mt'} \cdot a_{mo} - (1 - \sum_{m=1}^{M} a_{om} \cdot x_{mt}) \cdot BI \leq H_o + y_{ot} \cdot BI \quad \forall o \in O; t = 1, \ldots, T. \tag{5.7}$$

A detailed discussion of both objective functions is provided by Fliedner and Boysen (2008).

CS is known to be NP-hard in the strong sense (Kis, 2004). Different exact and heuristic solution approaches have been proposed in the literature. Among the exact approaches are integer linear program (ILP) formulations (Gravel et al., 2005; Prandtstetter and Raidl, 2008) and a branch & bound algorithm (Fliedner and Boysen, 2008). Gravel et al. (2005) proposed an ILP avoiding symmetries by grouping cars with the same options. Prandtstetter and Raidl (2008) take additional paint shop constraints into account in their ILP and focus on the assignment of option/colors instead of models. This formulation shows superior results compared to the ILP of Gravel et al. (2005). The branch & bound algorithm developed by Fliedner and Boysen (2008) is currently the state-of-the-art exact solution algorithm. It is based on a scattered branch & bound scheme (Klein and Scholl, 1999). The authors introduce and apply new lower bounds and dominance rules for CS in order to speed up the search.

Beside exact solution approaches, various heuristics are available in the literature. According to experimental results, mainly variable local search procedures which use more than one neighborhood seem to be suitable for CS (Puchta and Gottlieb, 2002; Gottlieb et al., 2003; Perron and Shaw, 2004; Estellon et al., 2008; Prandtstetter and Raidl, 2008). Various neighborhoods can be identified in the literature, the most promising are swap (exchanging the positions of two car models) and lin2Opt (inverting a subsequence of car models). Estellon et al. (2008) and Prandtstetter and Raidl (2008) showed that local search based methods with variable neighborhoods can efficiently solve large-scale industrial problem instances with more than 1000 cars. A first beam search (BS) approach for CS was introduced by Bautista et al. (2008). However, their approach is different from ours as they use a different objective

function based on the over- and under-assignment of options and also a different graph representation of CS. Other heuristic algorithms were applied on CS as well, e.g., ant colony optimization (Gottlieb et al., 2003; Gravel et al., 2005), genetic algorithms (Warwick and Tsang, 1995; Zinflou et al., 2007) and hybrid approaches (Prandtstetter and Raidl, 2008; Jaszkiewicz et al., 2004). For a complete overview of the various solution approaches to CS, see Boysen et al. (2009) and Solnon et al. (2008).

5.3 An iterative beam search approach

5.3.1 Search procedure

IBS is a repeatedly conducted BS heuristic (Lowerre, 1976). BS is a truncated breadth-first search and is based on a graph representation of the problem to be solved. The graph (typically a tree) consists of a single root node, the starting point for the search, inner nodes, which can be considered as partial solutions to the problem, leaf nodes, which represent overall solutions and arcs by which the nodes are connected. BS seeks to find a shortest path from the root node to a leaf node. It branches through the graph stagewise, where a stage consists of related nodes, e.g., inner nodes representing partial solutions of the same size. Thereby, BS gradually expands the BW (beam width) best partial solutions of the same stage. This results in a pool of partial solutions of the next stage from which again the BW best nodes are considered to be further extended. This is repeated until reaching the leafs of the tree and the best leaf node is returned as the result of BS. Since the number of nodes at each stage to be further branched is restricted by BW, BS unlikely finds the global best solution. For selecting the BW best nodes at each stage, a problem-specific heuristic is applied, which allows to order the nodes according to a relevant value, e.g., estimated objective value. Additional upper bounds (lower bounds for maximization problems) can be used, which allow to delete nodes during the search that cannot lead to better solutions.

An IBS procedure consists of n subsequent BS iterations, which are applied with gradually increasing beam widths BW. The result of the ith BS will be used as upper bound UB (lower bound for maximization problems) for the next $(i+1)$th BS. Therefore $UB_{i+1} = BS_i(UB_i, BW_i)$, with $UB_1 = \infty$ and $BW_{i+1} > BW_i$ for $i = 1 \dots n-1$. If the beam width of the nth BS is chosen large enough to cover all nodes at each stage and, therefore, the entire search space, a breadth-first search is applied and IBS returns the global best solution.

5.3.2 Graph representation of CS

CS is modeled as a directed acyclic digraph $G(V, E, f)$ with node set V, arc set E and an arc weighting function $f : E \to \mathbb{N}$. Each node $v \in V$ represents a partial solution to CS with a decision point i. At each decision point, $i-1$ models have already been assigned to the first $i-1$ slots and for current slot i a model has to be chosen out of the set of remaining demands D_i. Furthermore, it has to be known whether or not the processing of this model induces any rule violations. Therefore, the subsequence act_i^o, called active sequence, at decision point i contains the last $N_o - 1$ occurrences of option o from positions $i-N_o+1$ to $i-1$. Thus, $act_i^{o,t} = 1$ indicates that at production cycle $i-t$ option o has been processed, with $act_i^{o,t} \in \{0, 1\}$ being the tth position of act_i^o for all $t = 1, \ldots, N_o - 1$. Consequently with $ACT_i = \{act_i^1, \ldots, act_i^O\}$, each node v is defined by $[i, D_i, ACT_i]$. Due to this representation, the number of nodes to be generated is considerably reduced compared to an explicit enumeration of any possible subsequence.

Arcs connect adjacent nodes and represent a sequencing decision of a current model. At each decision point i, $|D_i'|$ distinct decisions can be made with $D_i' := \{d_m \in D_i | d_m > 0\}$ since every model m, where the remaining demand $d_m \in D_i$ is larger than 0, can be produced in the current slot i. If a model m' is assigned to a production slot, the number of produced models amounts to i and the new remaining demand D_{i+1} is obtained by decreasing $d_{m'} \in D_i$ by one copy of m'. Furthermore, the active sequences are adapted by removing the option occurrences at positions $N_o - 1$, shifting all other option occurrences by one position to the end and inserting all option occurrences of the current model m' at the first position of the active sequences. Therefore, an arc connects a node $[i, D_i, ACT_i]$ with its subsequent node $[i + 1, D_{i+1}, ACT_{i+1}]$.

Additionally, we assign weights $f : E \to \mathbb{N}$ to each arc, which represent the contribution of producing model m' in slot i to the overall objective value. Here, the weights measure the number of sequencing rule violations caused by producing model m'. Depending on the test set in the experimental Section 5.4, we apply two different weighting functions f_1 and f_2:

$$f_1 = \begin{cases} \sum_{o \in O} \min\{1; \max\{\sum_{t=1}^{N_o-1} act_i^{o,t} + a_{om'} - H_o; 0\}\}, & \text{if } i \geq N_o \\ 0, & \text{otherwise} \end{cases} \tag{5.8}$$

represents the sliding window objective function (SW). And

$$f_2 = \sum_{o \in O} \min\{a_{om'}; \max\{\sum_{t=1}^{N_o-1} act_i^{o,t} + a_{om'} - H_o; 0\}\} \tag{5.9}$$

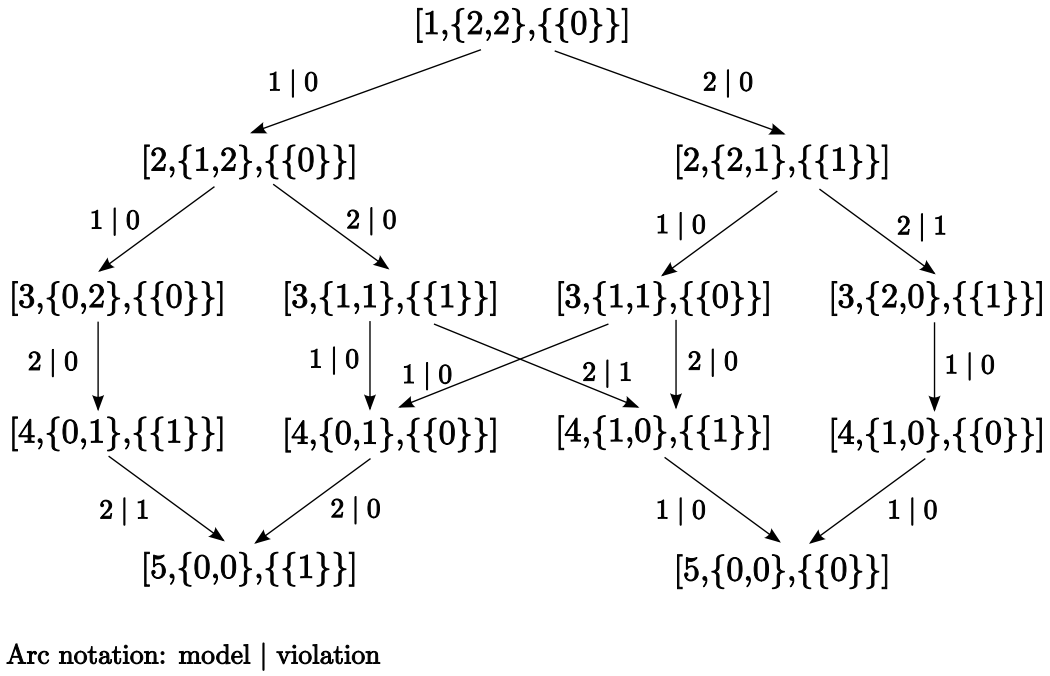


Figure 5.1: Example graph

for the objective function applied by Fliedner and Boysen (2008) (FB).

Example: Consider an example with $T = 4$ production slots, $O = 1$ option and $M = 2$ models. The single option is subject to a sequencing rule of 1:2. Both models have a demand of 2 copies, while model 2 requires the option and model 1 doesn't. Figure 5.1 presents the resulting graph for this example. Each arc is labeled with the currently produced model and the number of additional violations this causes. Note that in this example, the SW and FB objective function results in the same number of violations. We can observe for our graph that not any possible subsequence is represented by an unique node, but some subsequences point to the same node (e.g., $[4, \{0, 1\}, \{\{0\}\}]$ represents the subsequences 121 and 211). Thus, the overall number of nodes is reduced compared to a complete enumeration of subsequences. Our example contains three shortest paths from the root node to one of the leaf nodes, as the sequences 1212, 2112 and 2121 each lead to zero violations.

5.3.3 Lower bounds

On the one hand, within our IBS procedure lower bounds are applied to prune the graph. Specifically, nodes whose cumulated rule violations along the short-

est path plus a lower bound on the remaining rule violations exceed an upper bound, e.g., generated by a previous BS run with smaller BW, are fathomed. On the other hand, lower bounds are utilized to guide the search into promising regions of the solution space (see Section 5.3.4). There are some lower bounds available in the literature on CS, which consider the option occurrences decoupled from their actual assignment to models (Fliedner and Boysen, 2008; Benoist, 2008). Thus, for every option o, a lower bound can be computed regardless of other options. The sum of the lower bounds on every option lead to an overall lower bound. Benoist (2008) considers a different objective function for CS than we do and, therefore, his lower bound argument can not be applied in this paper. The lower bound by Fliedner and Boysen (2008) neglects the used objective function at all. Their bound is stated by $\sum_{o \in O} d_o^{\alpha} - a_o^{max,T}$, the difference between the actual demand d_o^{α} of models containing option o and the maximum number of available slots $a_o^{max,T} = \lfloor \frac{T}{N_o} \rfloor \cdot H_o + \min\{H_o; T \bmod N_o\}$ to produce models with option o in a sequence of length T without any violation, summed up over all options. Therefore, every option occurrence which exceeds the number of available slots leads to a sequencing rule violation of one. This lower bound can be tightened by incorporating the used objective function as exceeding option occurrences could actually lead to more than one violation. In the following, we will present lower bounds for CS incorporating the SW and FB objective function, respectively.

For the case of a single option o, Benoist (2008) states an optimal sequence construction algorithm. A sequence with minimum number of violations can be constructed from left to right by assigning a model containing the option (referred to as *optional model*) to a slot if no violation would occur and a *basic model* without the option otherwise. This rule is repeated for every slot until reaching the end of the sequence or only optional or basic models are left. In this case, the remaining slots are filled with the leftovers. The resulting delay of violations to the latest possible sequence positions leads to a minimum number of rule violations.

Example: Consider a single option o subject to a sequencing rule of $2 : 4$. The sequence length T is 13. The demand for optional models d_o^{α} is 8 and the demand for basic models d_o^{β} is 5. Applying the aforementioned construction algorithm leads to the optimal sequence depicted in Figure 5.2. The first two slots are filled with optional models, the next two slots contain basic models, and this pattern is repeated until at slot 12 only optional models are left. Considering either the SW or FB objective function, this sequence leads to an identical result of two violations induced by the subsequences starting at slots 9 and 10, respectively. Note, that the lower bound by Fliedner and Boysen (2008) would return only one violation as $d_o^{\alpha} - a_o^{max,T} = 8 - 7$.

t	1	2	3	4	5	6	7	8	9	10	11	12	13
2:4	1	1	0	0	1	1	0	0	1	1	0	1	1
	π^o												

Figure 5.2: Example result of construction algorithm

The construction algorithm generates optimal sequences consisting of a repeated pattern π^o of size N_o with H_o optional models at the beginning of the pattern followed by $N_o - H_o$ basic models, until either the required optional or basic models are exhausted. In the resulting sequence in Figure 5.2, π^o contains four slots with two optional models and two basic models and is processed twice up to slot 8.

Proposition 5.1. *Given the demand of basic models d_o^β for an option o with sequencing rule $H_o : N_o$, the maximum number of slots t_o^{max} that can be processed without violation amounts to:*

$$t_o^{max} = \left(\left\lfloor \frac{d_o^\beta}{N_o - H_o} \right\rfloor + 1 \right) \cdot H_o + d_o^\beta. \tag{5.10}$$

We apply the optimal construction algorithm with d_o^β basic models. At the beginning of the sequence, H_o optional models are sequenced. After every $N_o - H_o$ basic models, another H_o optional models can be processed. Thus, $(\lfloor \frac{d_o^\beta}{N_o - H_o} \rfloor + 1) \cdot H_o$ returns the maximum number of optional models and, therefore, t_o^{max} the overall number of models/slots that can be processed without violation.

Proposition 5.2. *Given an option o with sequencing rule $H_o : N_o$ and t_o^{max}. A lower bound LB^{SW} on the number of violations for the SW objective function amounts to:*

$$LB^{SW} = \max\{\min\{T - N_o + 1, T - t_o^{max}\}, 0\}. \tag{5.11}$$

If $t_o^{max} > T$, more than T slots can be processed without violation. Therefore, no violation occurs and LB^{SW} returns 0, which is ensured by the max-function. If $t_{max} < T$, violations are inevitable as in slot $t_o^{max} + 1$ a basic model is required, but only optional models are left. Thus, processing an optional model will lead to a violation of the window ending at slot $t_o^{max} + 1$. Since at that time only optional models are left, all remaining sliding windows from

$t_o^{max}+1$ to T will also be violated. Thus, we have an overall number of $T-t_o^{max}$ violated windows. The SW approach considers only complete windows of size N_o, such that there are exactly $T-N_o+1$ complete windows in a sequence of length T. The first window which can be violated ends at slot N_o. If $t_o^{max} < N_o$, $T - t_o^{max}$ would lead to more than $T - N_o + 1$ violated windows. Thus, the min-function bounds the number of violations by the maximum number of windows $T - N_o + 1$.

Proposition 5.3. *Given an option o with sequencing rule $H_o : N_o$ and t_o^{max}. A lower bound LB^{FB} on the number of violations for the FB objective function amounts to:*

$$LB^{FB} = \max\{T - t_o^{max}, 0\}. \tag{5.12}$$

The reasoning is almost identical to that for the SW objective function above, except that FB also considers windows of size less than N_o. Thus, in contrast to SW, the min-function is omitted.

Example (cont.): Consider the example of Figure 5.2. By applying (5.10), we receive $t_o^{max} = 11$ and, therefore, both lower bounds LB^{SW} and LB^{FB} result in the correct number of two violations.

Clearly, equation (5.10) computes t_o^{max} correctly, unless a part of the sequence is already fixed. If some slots are already processed, the models in the active sequence have an influence on the repeating pattern π^o generated by the construction algorithm. Thus, in the following t_o^{max} is determined under consideration of the option occurrences in the active sequence.

Let π_i^o be the repeating pattern for option o at decision point i and $\pi_i^o(j)$ the jth position in the pattern. Given an active sequence act_i^o at i, π_i^o is constructed by

$$\pi_i^o(j) = \begin{cases} 1, & \text{if } \sum_{t=1}^{N_o-j} act_i^{o,t} + \sum_{k=1}^{j-1} \pi_i^o(k) < H_o \\ 0, & \text{else} \end{cases} \tag{5.13}$$
$$j = 1 \dots N_o$$

Example: Consider the example in Figure 5.3. It is the same example as before, except for the first three slots that are already being filled with models. The active sequence at decision point 4 is $act_4^o = \{1, 0, 0\}$, thus in slots 1 and 2 basic models were processed and an optional model in slot 3. The result-

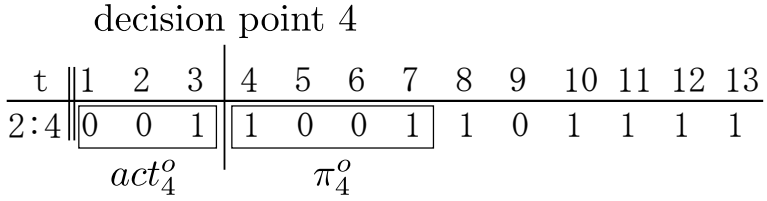


Figure 5.3: Example result of construction algorithm with already fixed slots

ing repeating pattern obtained from (5.13) is $\pi_4^o = \{1, 0, 0, 1\}$. Note, that the repeating pattern still consists of H_o optional models and $N_o - H_o$ basic models.

Let $f_{\pi_i^o}(j)$ be a function that returns the position of the jth basic model in pattern π_i^o with $j = 1 \dots N_o - H_o$. In our example $f_{\pi_4^o}(2) = 3$.

Proposition 5.4. *Given decision point i and the remaining demand of basic models $d_{o,i}^{\beta}$ for an option o. The maximum number of slots $t_{o,i}^{max}$ starting at i that can be processed without violations amounts to:*

$$t_{o,i}^{max} = \left(\left\lfloor \frac{d_{o,i}^{\beta}}{N_o - H_o} \right\rfloor \right) \cdot N_o + f_{\pi_i^o}(d_{o,i}^{\beta} \bmod (N_o - H_o) + 1) - 1. \tag{5.14}$$

A complete pattern π_i^o of size N_o can be processed with each $N_o - H_o$ basic models (first term). By doing so, $d_{o,i}^{\beta} \bmod (N_o - H_o)$ basic models remain. With these basic models, we can still process a fraction of the pattern π_i^o up to position $f_{\pi_i^o}(d_{o,i}^{\beta} \bmod (N_o - H_o) + 1) - 1$ (second term). Therefore, we can construct a sequence of length $t_{o,i}^{max}$ starting at decision point i without any violation. Adapting formulas (5.11) and (5.12) we get:

$$LB^{SW} = v_{\text{fix}} + \max\{\min\{T - N_o + 1, T - (i + t_{o,i}^{max} - 1)\}, 0\} \tag{5.15}$$

and

$$LB^{FB} = v_{\text{fix}} + \max\{T - (i + t_{o,i}^{max} - 1), 0\}, \tag{5.16}$$

where v_{fix} are the violations in the fixed part of the sequence.

Example (cont.): Again, we take a look at the example sequence of Figure 5.3, subject to sequencing rule $2 : 4$. At decision point $i = 4$, we have three

basic models left ($d^{\beta}_{o,4} = 3$). With $f_{\pi^o_4}(2) = 3$ and (5.14), we get $t^{max}_{o,4} = 6$. And finally, with (5.15) and (5.16), both lower bounds result in the correct number of 4 violations.

5.3.4 Node selection

In order to select the BW best nodes at each stage of the graph to be further branched, we apply a three-stage filtering process. First, the nodes are ordered in ascending order according to their lower bound values. Second, for nodes with identical lower bounds, we apply a utilization rate $U_1 := \sum_{o \in O} \sum_{m \in D_i} a_{mo} \cdot d_m$, which counts the number of remaining options still to be processed, where nodes with less remaining options are favored. If there are still nodes with the same lower bound and identical value for U_1, in a third step, we apply a second utilization rate $U_2 = \max\{r_o | r_o = \sum_{m \in D_i} a_{mo} \cdot d_m / a^{max,T}_o, o \in O\}$. U_2 is defined by the maximum ratio r_o among all options, where r_o reflects the actual demand for o divided by the maximum number of occurrences $a^{max,T}_o$ of o, that can be processed without violation. The smaller this ratio, the higher is the degree of freedom to process the option. Again, nodes are ordered in ascending order, such that nodes with a small maximum ratio are favored.

5.4 Computational study

5.4.1 Experimental setup

Three sets of problem instances are applied to evaluate our IBS approach. Set 1 was introduced by Fliedner and Boysen (2008) and consists of 18 test instances with 10-50 production slots, 3-7 options and 5-28 models. Throughout the experiments, the FB objective function is used to evaluate the instances within this set. Set 2 stems from the CSPLib, an online library, which provides instances for various constraint satisfaction problems. It contains 9 instances all with 100 production slots, 5 options and 19-26 models. Set 3 is provided by Gravel et al. (2005), which is also available in the CSPLib. It consists of 30 problem instances with 200-400 production slots, 5 options and 19-26 models. The two latter sets are evaluated using the SW objective function.

IBS was implemented in Java. All experiments run on a Pentium Dual Core 2.5 GHz with 2 GB RAM.

5.4.2 Results

We apply IBS in two different modes. IBS_1 utilizes the lower bound introduced by Fliedner and Boysen (2008), while IBS_2 uses our new lower bound (see Section 5.3.3). Both, IBS_1 and IBS_2, are performed with nine iterations and beam widths $BW = \{5, 10, 25, 50, 100, 500, 1000, 1500, \infty\}$. Since the last beam width $BW_9 = \infty$, IBS_1 and IBS_2 become exact solution approaches and return the global best solution. We compare both algorithms with the currently best known exact solution approach, a SB&B algorithm (Fliedner and Boysen, 2008). The SB&B is applied with $\eta = 100, \Delta = 10000, \alpha = 0.75$, dominance rules 2,3 and 6 and the lower bound by Fliedner and Boysen (for details see Fliedner and Boysen, 2008). For each instance, the time limit for solving the instance is set to 600 seconds, which seems an appropriate choice for a short-term planning problem like CS. An algorithm is aborted, if the given time limit is exceeded.

Table 5.2 shows the results for problem set 1, according to the FB objective function. Label 'obj*' denotes the currently best known solution for each instance as stated in Section 4.5.3. The columns 'obj', 'time' and 'nodes' state the best objective value obtained by each algorithm and the time and number of evaluated nodes required to find this value. Note, that 'obj' contains the global best solution, if an instance could be solved within the time limit of 600 seconds.

						SB&B			IBS_1			IBS_2		
	Instance	T	O	M	obj*	obj	time (s)	nodes	obj	time (s)	nodes	obj	time (s)	nodes
Set 1	CAR_3_10	10	3	5	1	1	0.20	16	1	0.03	226	1	0.02	226
	CAR_5_10	10	5	5	1	1	0.16	2	1	0.01	82	1	0.01	82
	CAR_7_10	10	7	9	2	2	0.25	75	2	0.08	810	2	0.06	663
	CAR_3_15	15	3	5	2	2	0.30	39	2	0.03	614	2	0.03	644
	CAR_5_15	15	5	7	2	2	0.34	73	2	0.03	738	2	0.05	776
	CAR_7_15	15	7	13	4	4	0.56	2,486	4	1.11	14,280	4	0.53	6,341
	CAR_3_20	20	3	6	3	3	0.47	1005	3	0.23	9,950	3	0.17	7,173
	CAR_5_20	20	5	7	3	3	0.42	326	3	0.11	2,954	3	0.08	2,014
	CAR_7_20	20	7	15	3	3	1.47	13,194	3	5.05	47,819	3	4.00	36,674
	CAR_3_30	30	3	6	4	4	1.31	10,006	4	1.81	67,022	4	1.52	55,285
	CAR_5_30	30	5	11	3	3	2.94	37,302	3	7.02	107,084	3	6.50	97,087
	CAR_7_30	30	7	23	4	4	468.75	3,049,761	4	495.81	2,246,516	4	275.59	1,252,970
	CAR_3_40	40	3	7	5	5	10.77	92,332	5	11.44	321,562	5	8.45	237,023
	CAR_5_40	40	5	13	5	5	423.17	3,295,946	5	349.53	3,587,007	5	196.78	2,040,611
	CAR_7_40	40	7	26	7	7	>600	-	8	>600	-	7	>600	-
	CAR_3_50	50	3	7	6	6	45.95	388,351	6	38.67	991,297	6	24.14	613,061
	CAR_5_50	50	5	14	8	8	>600	-	8	>600	-	8	>600	-
	CAR_7_50	50	7	28	11	12	>600	-	12	>600	-	11	>600	-

Table 5.2: Results for problem set 1 (FB objective function)

The results returned by IBS_1 and SB&B are comparable (excepting instance CAR_7_40, where SB&B finds a better solution than IBS_1). Regarding the solution time, IBS_1 is slightly in favor on 9 instances, while SB&B requires

less time than IBS_1 on 6 instances. Overall, IBS_1 and IBS_2 can evaluate nodes much faster than SB&B, mainly because no dominance rules are applied. Due to the new lower bound, IBS_2 is able to reduce the amount of evaluated nodes considerably compared to IBS_1. Therefore, it is able to outperform IBS_1 on all instances and SB&B on 15 out of 18 instances. Only IBS_2 finds the best known solutions for all instances of set 1.

In order to compare the performances of the algorithms on larger problem instances, we use problem sets 2 and 3 consisting of instances with up to 400 car models. The results are listed in Table 5.3 and correspond to the SW objective function. Clearly, IBS_1 and IBS_2 outperform SB&B on all instances. Regarding set 2, IBS_1 and IBS_2 are able to solve all satisfiable instances in less time than SB&B and return the best known solution for all other instances. In contrast, SB&B is not able to reproduce the best known solutions for instances 10-93, 19-71 and 36-92. On set 3, the objective values returned by IBS_1 and IBS_2 are significantly better than the results of SB&B. They are even close to the best known solutions, obtained by a genetic algorithm with a subsequent local search (Zinflou et al., 2007). Furthermore, IBS_1 and IBS_2 solve all of the satisfiable instances in a few seconds, except instance 200_01. Regarding the solution times on these instances, IBS_1 is slightly in favor compared to IBS_2, since the computation of the new bound requires little more time. The objective values returned by IBS_1 and IBS_2 are similar, except for instances 200_03, 300_05, 400_02 and 400_07, where IBS_2 finds a better solution than IBS_1. However, both, IBS_1 and IBS_2, are not able to find an optimal solution for any of the unsatisfiable instances in sets 2 and 3 within the time limit of 600 seconds, since the number of production slots T and models M are too large to be solved to optimality.

5.4.3 Optimality proof

For instances 6-76, 10-93, 19-71, 21-90 and 36-92 of problem set 2, where the current best solutions found contain sequencing rule violations, the optimality of these solutions was only proven for instance 19-71 by Gent (1998), who reasoned a lower bound with the same number of two violations as the currently best solution. For the remaining instances, the optimality of the current best solution still needs to be verified. For this reason, we relaxed these instances by merely considering two out of five options concurrently. Considering each combination of two options, we receive $\binom{5}{2} = 10$ subinstances for every instance. Note, that any optimal solution of one of the subinstances serves as a lower bound for the original problem instance. We solved each combination with our IBS_2 approach from Section 5.4.2. Table 5.4 shows the respective results. Thereby, we were able to proof the optimality of the current best solution for instances 6-76, 10-93 and 36-92. For instance 6-76 the subinstance just

						SB&B			IBS_1			IBS_2		
	Instance	T	O	M	obj*	obj	time (s)	nodes	obj	time (s)	nodes	obj	time (s)	nodes
Set 2	4-72	100	5	22	0	0	3.53	201	0	0.27	1,470	0	0.28	1,470
	6-76	100	5	22	6	6	>600	-	6	>600	-	6	>600	-
	10-93	100	5	25	3	5	>600	-	3	>600	-	3	>600	-
	16-81	100	5	26	0	0	5.08	6,502	0	1.64	8,431	0	1.70	8,431
	19-71	100	5	23	2	3	>600	-	2	>600	-	2	>600	-
	21-90	100	5	23	2	2	>600	-	2	>600	-	2	>600	-
	36-92	100	5	22	2	4	>600	-	2	>600	-	2	>600	-
	41-66	100	5	19	0	0	3.13	90	0	0.94	8,799	0	1.86	17,073
	26-82	100	5	24	0	0	3.78	22	0	0.58	3,857	0	0.61	3,857
Set 3	200_01	200	5	25	0	5	>600	-	1	>600	-	1	>600	-
	200_02	200	5	25	2	7	>600	-	3	>600	-	3	>600	-
	200_03	200	5	25	4	14	>600	-	9	>600	-	8	>600	-
	200_04	200	5	24	7	16	>600	-	8	>600	-	8	>600	-
	200_05	200	5	23	6	14	>600	-	8	>600	-	8	>600	-
	200_06	200	5	23	6	8	>600	-	7	>600	-	7	>600	-
	200_07	200	5	23	0	1	>600	-	0	0.562	2,943	0	0.578	2,943
	200_08	200	5	20	8	11	>600	-	9	>600	-	9	>600	-
	200_09	200	5	24	10	15	>600	-	10	>600	-	10	>600	-
	200_10	200	5	19	19	27	>600	-	20	>600	-	20	>600	-
	300_01	300	5	25	0	7	>600	-	0	5.891	26,767	0	6.047	26,768
	300_02	300	5	25	12	24	>600	-	12	>600	-	12	>600	-
	300_03	300	5	25	13	25	>600	-	14	>600	-	14	>600	-
	300_04	300	5	24	7	21	>600	-	10	>600	-	10	>600	-
	300_05	300	5	20	29	52	>600	-	33	>600	-	32	>600	-
	300_06	300	5	25	2	9	>600	-	6	>600	-	6	>600	-
	300_07	300	5	24	0	14	>600	-	0	5.156	25,636	0	5.344	25,636
	300_08	300	5	23	8	17	>600	-	9	>600	-	9	>600	-
	300_09	300	5	21	7	16	>600	-	7	>600	-	7	>600	-
	300_10	300	5	19	21	56	>600	-	25	>600	-	25	>600	-
	400_01	400	5	25	1	9	>600	-	3	>600	-	3	>600	-
	400_02	400	5	22	16	35	>600	-	20	>600	-	19	>600	-
	400_03	400	5	23	9	19	>600	-	12	>600	-	12	>600	-
	400_04	400	5	26	19	29	>600	-	20	>600	-	20	>600	-
	400_05	400	5	20	0	23	>600	-	0	0.359	2,008	0	0.375	2,008
	400_06	400	5	23	0	2	>600	-	0	7.047	35,745	0	7.281	35,747
	400_07	400	5	23	4	15	>600	-	5	>600	-	4	>600	-
	400_08	400	5	21	4	28	>600	-	10	>600	-	10	>600	-
	400_09	400	5	24	5	29	>600	-	11	>600	-	11	>600	-
	400_10	400	5	25	0	23	>600	-	0	3.031	15,573	0	3.125	15,573

Table 5.3: Results for problem sets 2 and 3 (SW objective function)

containing options 1 and 3 leads to 6 violations, for 10-93 the subinstance with options 1 and 2 leads to 3 violations and for instance 36-92 the subinstance composed of options 2 and 4 amounts to 2 violations. Only for instance 21-90,

the optimality proof remains open, as all subinstances return 0 violations.

Options	Instance			
	6-76	10-93	21-90	36-92
1 & 2	0	3	0	0
1 & 3	6	0	0	0
1 & 4	0	0	0	0
1 & 5	0	0	0	0
2 & 3	0	0	0	0
2 & 4	0	0	0	2
2 & 5	0	0	0	0
3 & 4	0	0	0	0
3 & 5	0	0	0	0
4 & 5	0	0	0	0
max	6	3	0	2

Table 5.4: Resulting sequencing rule violations for subinstances with two options (SW objective function)

5.5 Conclusion

This paper presents an exact IBS algorithm for the CS problem. Therefore, we model CS as a directed acyclic digraph. We improve existing lower bounds by incorporating the applied objective function. As objective function, we use either the sliding-window technique or the objective function by Fliedner and Boysen (2008). Experimental evaluation of our approach is conducted on three sets of widely-used small to medium-sized instances with up to 400 car models.

The results on smaller, and therefore exact solvable, instances reveal that our new lower bound argument significantly reduces the number of evaluated nodes and, therefore, speeds up the search process of IBS. Our IBS algorithm, applied with the new lower bound, is superior to the currently best known exact solution approach, a SB&B algorithm (Fliedner and Boysen, 2008). It finds better solutions containing fewer sequencing rule violations in less time. The performance gap between IBS and SB&B even increases for larger instances with 100-400 cars, where IBS leads to competitive results, in terms of solution quality, compared to an existing genetic algorithm in conjunction with a local search. Additionally, we proof for three instances of the CSPLib the optimality of the current best known solution by computing lower bounds with the same objective value.

Future research on CS should deal with the development of superior lower bounds. For instance, more than merely a single option at a time could

be taken into account, so that dependencies between options can be incorporated into improved lower bounds. Furthermore, alternative CS objective functions (sliding-window, objective function by Fliedner and Boysen (2008), etc.) should be addressed in order to evaluate which objective function actually leads to the underlying goal of minimizing the work overload.

Bibliography

Bautista, J., Pereira, J., and Adenso-Diaz, B. (2008). A beam search approach for the optimization version of the car sequencing problem. *Annals of Operations Research*, 159:233–244.

Benoist, T. (2008). Soft car sequencing with colors: Lower bounds and optimality proofs. *European Journal of Operational Research*, 191(3):957–971.

Boysen, N., Fliedner, M., and Scholl, A. (2009). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.

Estellon, B., Gardi, F., and Nouioua, K. (2008). Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research*, 191(3):928–944.

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

Gent, I. P. (1998). Two results on car-sequencing problems. APES research report 02-1998, Department of Computer Science, University of Strathclyde, Glasgow.

Gottlieb, J., Puchta, M., and Solnon, C. (2003). A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In Cagnoni, S., Johnson, C., Cardalda, J., Marchiori, E., Corne, D., Meyer, J.-A., Gottlieb, J., Middendorf, M., Guillot, A., Raidl, G., and Hart, E., editors, *EvoWorkshops 2003*, volume 2611 of *LNCS*, pages 246–257, Berlin Heidelberg. Springer.

Gravel, M., Gagne, C., and Price, W. L. (2005). Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56(11):1287–1295.

Jaszkiewicz, A., Kominek, P., and Kubiak, M. (2004). Adaptation of the genetic local search algorithm to a car sequencing problem. In *7th National*

Conference on Evolutionary Algorithms and Global Optimization, pages 67–74, Kazimierz Dolny, Poland.

Kis, T. (2004). On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335.

Klein, R. and Scholl, A. (1999). Scattered branch and bound: An adaptive search strategy applied to resource-constrained project scheduling. *Central European Journal of Operations Research*, 7:177–201.

Lowerre, B. (1976). *The harpy speech recognition system*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA.

Parrello, B. D., Kabat, W. C., and Wos, L. (1986). Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2(1):1–42.

Perron, L. and Shaw, P. (2004). Combining forces to solve the car sequencing problem. In Regin, J.-C. and Rueher, M., editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3011 of *LNCS*, pages 225–239, Berlin Heidelberg. Springer.

Prandtstetter, M. and Raidl, G. (2008). 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.

Puchta, M. and Gottlieb, J. (2002). Solving car sequencing problems by local optimization. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, volume 2279 of *LNCS*, pages 132–142, Berlin Heidelberg. Springer.

Solnon, C., Cung, V., Nguyen, A., and Artigues, C. (2008). 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.

Warwick, T. and Tsang, E. (1995). Tackling car sequencing problems using a generic genetic algorithm. *Evolutionary Computation*, 3(3):267–298.

Wester, L. and Kilbridge, M. D. (1964). The assembly line model-mix sequencing problem. In *Proceedings of the Third International Conference on Operations Research*.

Zinflou, A., Gagne, C., and Gravel, M. (2007). Crossover operators for the car sequencing problem. In Cotta, C. and van Hemert, J., editors, *Proceedings*

of the 7th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2007, volume 4446 of *LNCS*, pages 229–239, Valencia, Spain.

Chapter 6

Fitness landscape analysis and design of metaheuristics for car sequencing

Uli Golle

Abstract

We study the car sequencing (CS) problem, an NP-hard combinatorial optimization problem which aims at finding a production sequence of different models launched down a mixed-model assembly line. The models are differentiated by a number of selected options. For each option, a so-called sequencing rule is applied which restricts the option occurrences in the sequence in order to minimize the work overload of the respective line operators. In this paper, we perform a fitness landscape analysis of several instances of the CS problem, where we examine the autocorrelation as well as fitness-distance correlation (FDC) induced by four neighborhood operators and three distance metrics. These results are subsequently employed for the design of two metaheuristics for CS, a variable neighborhood search (VNS) and a memetic algorithm (MA) evaluated with three different crossover operators. The VNS shows superior performance and even improves currently best known solutions of instances in the CSPLib. Although the results of the MA algorithms are inferior compared to VNS, applying crossover operators that respect the adjacency distance metric lead to a better solution quality than using other crossovers.

6.1 Introduction

The car sequencing (CS) problem (Parrello et al., 1986) is a combinatorial optimization problem seeking a production sequence of various models launched down a mixed-model assembly line. The models are derived from a common

base product and differ in a number of selected options. Nevertheless, all models are jointly manufactured on the same mixed-model assembly line with a lot size of one. Since the processing times of the models can vary and the line is balanced to an average model, a sequence of consecutive work-intensive models may lead to work overload of the respective line operators. Work overload occurs, whenever an operator can not finish his assigned tasks within the available station limits. These work overload scenarios have to be compensated by other strategies, such as employing additional utility workers or stopping the line. To minimize the total amount of work overload, CS uses a so-called sequencing rule of type $H : N$ for each selected option, which restricts the occurrence of models having this option to at most H, in any subsequence of N consecutive models. The aim is to find a sequence, which meets the demand for each model and satisfies all sequencing rules (constraint satisfaction problem) or minimizes the number of sequencing rule violations (optimization problem). CS belongs to the class of NP-hard problems (Kis, 2004).

The concept of landscapes is an intuitive notion of the search space, the set of all solutions to a combinatorial optimization problem. A search algorithm navigates through the landscape in order to find the best solution, usually the highest peak or the lowest valley. Therefore, all solutions have to be connected by a certain distance measure and the quality of each solution is assessed using a so-called fitness function. Thus, landscapes are often referred to as fitness landscapes. The analysis of fitness landscapes can give valuable insights into the characteristics of the search space. These informations can be employed to design better search algorithms that incorporate more problem-specific knowledge, which is crucial for effective optimization algorithms.

In this paper, we study the resulting landscapes of four neighborhood operators and three distance metrics for a set of CS instances. We analyze local and global properties of the landscapes by performing an autocorrelation and fitness distance correlation analysis. The results of the landscape analysis are employed to design two metaheuristics for CS, a simple variable neighborhood search (VNS) as well as a memetic algorithm (MA). Thereby, we propose a new heuristic crossover operator for CS. The performance of the VNS and the MA with the new crossover operator as well as two existing ones is evaluated in experiments using widely applied CS problem instances from the CSPLib. The main findings are:

1. Three out of the four neighborhood operators lead to a smooth landscape for CS in terms of the normalized correlation length.

2. All neighborhood operators and the adjacency distance metric show a high fitness-distance correlation on all instances and, thus according to Jones and Forrest (1995), result in an 'easy' problem for evolutionary algorithms.

T	number of production slots (index t)
M	number of models (index m)
O	number of options (index o)
d_m	demand for model m
a_{om}	binary demand coefficient: 1, if model m requires option o, 0 otherwise
$H_o : N_o$	sequencing rule: at most H_o out of N_o successively sequenced models require option o
x_{mt}	binary variable: 1, if model m is produced in slot t, 0 otherwise
y_{ot}	binary variable: 1, if sequencing rule defined for option o is violated in window starting in cycle t
BI	Big Integer

Table 6.1: Notations

3. In case of the adjacency distance metric, a 'big valley' (Boese et al., 1994) structure for CS is identified.

4. Incorporating the findings of the landscape analysis into operators for the VNS and the MA leads to high quality metaheuristics that find the best-known sequences for all test instances and even improve the currently best solution of two instances.

The remainder of the paper is organized as follows. In Section 6.2, we review relevant literature on CS and state its optimization model. Section 6.3 presents four neighborhood operators proposed for CS and studies characteristics of the fitness landscapes resulting from these operators. The insights gained from Section 6.3 are used in Section 6.4 to design a VNS as well as a MA for CS. The performance of the metaheuristics is evaluated in experiments in Section 6.5. Section 6.6 concludes the paper.

6.2 The car sequencing problem

The CS problem stems from applications in the automobile industry and was first introduced by Parrello et al. (1986). As its related approach of mixed-model sequencing (MMS) (Wester and Kilbridge, 1964), it aims at finding a sequence of different models to be produced at a mixed-model assembly line with minimum work overload. Work overload occurs, whenever a line operator can not finish his assigned assembly tasks due to a consecutive order of workload intensive models in the sequence. In contrast to MMS, CS applies a surrogate objective for the minimization of work overload. Given a pool of

different models, which can be distinguished by selected binary options (such as having an air conditioning or not in case of car models), CS restricts the occurrences of each option $o \in O$ in the sequence by using so-called sequencing rules $H_o : N_o$. A sequencing rule allows only H_o models having option o in any subsequence of N_o consecutive models. For instance, a sequencing rule of 3:5 for the option 'sunroof' requires that at most 3 out of 5 consecutive models contain a sunroof. If more models have a sunroof, a violation of the respective sequencing rule occurs. The aim of CS is to find a production sequence containing all models in the pool and inducing the minimum overall number of rule violations. It is assumed that the minimization of sequencing rule violations simultaneously leads to the underlying goal of minimizing the total amount of work overload. With notations from Table 6.1, the CS problem is formulated as an integer linear program as follows Gravel et al. (2005):

$$\text{CS: Minimize} \quad \sum_{o \in O} \sum_{t=1}^{T-N_o+1} y_{ot} \tag{6.1}$$

$$\sum_{t=1}^{T} x_{mt} = d_m \quad \forall m \in M \tag{6.2}$$

$$\sum_{m \in M} x_{mt} = 1 \quad \forall t = 1, \ldots, T \tag{6.3}$$

$$\sum_{t'=t}^{t+N_o-1} \sum_{m \in M} x_{mt'} \cdot a_{mo} \leq H_o + y_{ot} \cdot BI \quad \forall o \in O; t = 1, \ldots, T - N_o + 1 \tag{6.4}$$

$$x_{mt} \in \{0, 1\} \quad \forall m \in M; t = 1, \ldots, T \tag{6.5}$$

$$y_{ot} \in \{0, 1\} \quad \forall o \in O; t = 1, \ldots, T \tag{6.6}$$

y_{ot} indicates whether a rule violation of option o occurs in a subsequence starting at position t. The objective function (6.1) minimizes the overall number of rule violations. The resulting sequence has to meet the required demand d_m for each model $m \in M$ (6.2) and is allowed to contain exactly one model at each production slot t (6.3). Constraints (6.4) check for rule violations. We apply the sliding-window approach, where a violation of a subsequence is always assessed with one violation in the objective function Gravel et al. (2005). Finally, (6.5) and (6.6) ensure variables x_{mt} and y_{ot} to be binary.

Since CS is NP-hard in the strong sense (Kis, 2004), various exact, heuristic and hybrid search procedures have been developed in order to solve the problem (Boysen et al., 2009; Solnon et al., 2008). Among the exact approaches are

integer linear programming (ILP) formulations. While Gravel et al. (2005) present the aforementioned basic ILP, the ILP by Prandtstetter and Raidl (2008) focuses on option assignments to the sequence instead of models and also includes constraints of the preceding paint shop. Fliedner and Boysen (2008) propose a scattered branch & bound algorithm introducing new lower bounds and dominance rules for CS. Their bounds are further improved in Chapter 5, where a graph representation of CS is solved with an exact iterative beam search (IBS) algorithm.

As for heuristics and metaheuristics, different approaches including construction heuristics, local searches, evolutionary algorithms and ant colony optimizations have been developed. In Gottlieb et al. (2003), different greedy heuristics for the optimization version of CS are studied and applied in local search as well as ant colony algorithms. In Butaru and Habbas (2005), the constraint satisfaction version of CS is addressed and different value ordering heuristics to construct an optimal sequence are proposed. Different local search procedures are introduced in Puchta and Gottlieb (2002) including greedy as well as threshold accepting algorithms that employ a set of various neighborhood operators. A large neighborhood search is also introduced in Perron and Shaw (2004) using different move operators and search strategies. In Estellon et al. (2008) and Cordeau et al. (2008) an industrial version of CS with instances having more than 1000 models and including paint shop constraints and the distinction between hard and soft constraints, is solved by local search applying variable neighborhoods. Among evolutionary approaches, a genetic algorithm (GA) is developed in Warwick and Tsang (1995) using a version of adaptive template type crossover and performing hill climbing as mutation operator. For the industrial version of CS, a genetic local search procedure is designed in Jaszkiewicz et al. (2004). The authors introduce a crossover operator that preserves common subsequences in both parents and apply a local search using a shift neighborhood after each recombination. In Terada et al. (2006), a standard GA is combined with squeaky-wheel optimization, where sequences are iteratively constructed and improved by adaptive priority rules. A series of new crossover operators for GAs is presented in Zinflou et al. (2007), showing good results on instances of the CSPLib. These results are further improved by applying a subsequent local search. Furthermore, ant colony optimizations are presented in Gottlieb et al. (2003); Gravel et al. (2005); Gagne et al. (2006); Solnon (2008) applying different pheromone trails and transition rules. In summary, local search procedures applying a set of neighborhood operators show the overall best results on CS instances.

Few authors provide hybrid algorithms for CS, combining exact and heuristic approaches. In Prandtstetter and Raidl (2008), an ILP is incorporated into a variable local search procedure producing competitive results for the industrial version of CS. In Zinflou et al. (2008), the authors previous GA approach is

extended by incorporating ILP formulations into the crossover operator. The solution quality of the resulting GA on the CSPLib instances is increased, but at the cost of a very high runtime.

6.3 Fitness landscape analysis of car sequencing

In general, a fitness landscape $(\mathcal{X}, f, d)$ of an instance of a combinatorial optimization problem consists of a set of solutions $\mathcal{X}$, a fitness function (objective function) $f : \mathcal{X} \to \mathbb{R}$, which assigns an objective value to each of the solutions in $\mathcal{X}$, and a distance measure d, which defines the spatial structure of the landscape. For the notion of distances, a so-called neighborhood operator $\mathcal{N}$ is applied. $\mathcal{N}$ converts a solution $x \in \mathcal{X}$ into a new solution $x' \in \mathcal{N}(x) \subseteq \mathcal{X}$ by changing the composition of x. With the neighborhood operator $\mathcal{N}$, the search space can be interpreted as an undirected graph $\mathcal{G}_{\mathcal{N}} = (\mathcal{X}, E)$ with $\mathcal{X}$ being the vertices of the graph and edge set $E = \{(x, x') \in \mathcal{X} \times \mathcal{X} \mid x' \in \mathcal{N}(x)\}$ introducing an edge between x and x', if x' can be reached from x by one application of $\mathcal{N}$ or vice versa. The distance $d(x, y)$ between two solution x and y is then defined as the length of the shortest path from x to y in $\mathcal{G}_{\mathcal{N}}$, which equals the minimum number of applications of $\mathcal{N}$ to transform x into y or vice versa.

For CS, we analyze the fitness landscapes for a set of nine widely applied problem instances in the literature. The instances are available in the CSPLib, an online library for constraint satisfaction problems. Each instance has $T = 100$ production slots, $M = 19$-26 models with $O = 5$ options to be sequenced. For every instance, we generate different fitness landscapes using equation (6.1) as fitness function and four neighborhood operators found in the literature (Puchta and Gottlieb, 2002). We study the autocorrelation of the resulting landscapes in terms of a random walk analysis and perform a fitness-distance correlation analysis.

6.3.1 Representation and neighborhood operators

A solution for CS is represented by a permutation with repetitions. Thus, a sequence is encoded as a vector of length T, where a value $m \in M$ at position $t = 1, \ldots, T$ indicates that a copy of model m is produced at the tth production slot. We only consider feasible solutions, where the sum of occurrences of each model $m \in M$ in the sequence corresponds to the models demand d_m. This is ensured by an appropriate initialization of solutions and neighborhood operators that maintain feasibility.

For the fitness landscape analysis of CS, we consider four different neighborhood operators $\mathcal{N}$ (Puchta and Gottlieb, 2002), that are currently applied in the literature. The operators are along the line of operators for the traveling salesman problem:

- Swap neighborhood $\mathcal{N}_{\text{sw}}$: Two models in the sequences exchange their positions.
- Adjacent swap neighborhood $\mathcal{N}_{\text{ad}}$: Two adjacent models in the sequences exchange their positions.
- Shift neighborhood $\mathcal{N}_{\text{sh}}$: A model is forward or backward shifted a certain number of positions in the sequence.
- Reverse neighborhood $\mathcal{N}_{\text{re}}$: The order of a subsequence of models is reversed.

6.3.2 Autocorrelation

The autocorrelation analysis studies the ruggedness of a landscape, which is important for local search procedures. A fitness landscape is said to be rugged, if no correlation between the distance of solutions and their fitness values exists. Thus, a small distance between two solutions can imply a large difference in their objective values. Rugged landscapes are difficult to search for guided local search methods, which traverse the search space by iteratively sampling new solutions in the neighborhood of a current solution. Thereby, the fitness value of the current solution is employed to decide whether the search proceeds with one neighboring solution or not. The search is guided from low quality solutions to higher quality solutions in order to find the global optimum. However, on a rugged landscape, the fitness values of neighboring solutions are not correlated and can, therefore, not be used to guide the search process, which results in a merely random search. In contrast, if a correlation between the distance and the fitness value of solutions exists, the respective landscape is said to be smooth and is adequate for guided local search methods.

In Merz and Freisleben (2000), the autocorrelation function is introduced to compute the ruggedness of a landscape. The approach requires to evaluate the entire search space of a problem instance, thus, all solutions in $\mathcal{X}$. Since for many optimization problems, including CS, the number of solutions increases exponentially depending on some input factor, Weinberger (1990) proposes to use a random walk to estimate the autocorrelation. Beginning with an arbitrary solution, a random walk picks a random solution in the neighborhood of the current solution and proceeds the walk with the new solution. This move is repeated until a maximum number m of walking steps is reached. The fitness

$\mathcal{N}_{\text{sw}}$	$T-1$
$\mathcal{N}_{\text{ad}}$	$\frac{T(T-1)}{2}$
$\mathcal{N}_{\text{sh}}$	$T-1$
$\mathcal{N}_{\text{re}}$	$T-1$

Table 6.2: Diameters of neighborhood operators

values $f(x)$ of all solutions x visited during the walk are used to compute the random walk correlation function $r(s)$ as follows

$$r(s) = \frac{1}{\sigma^2(f)(m-s)} \sum_{t=1}^{m-s} (f(x_t) - \overline{f})(f(x_{t+s}) - \overline{f}) \tag{6.7}$$

With $\sigma^2(f)$ being the variance of the fitness values, $r(s)$ computes the correlation of all solutions that are s steps away along the random walk of length m. Based on the nearest-neighbor correlation of the landscape $r(1)$, which is the correlation of neighboring solutions, the correlation length l of the landscape is defined as (Stadler, 1996):

$$l = \begin{cases} 0, & \text{if } r(1) = 0 \\ -\frac{1}{\ln(|r(1)|)}, & \text{if } r(1) \neq 0 \end{cases} \tag{6.8}$$

l reflects the ruggedness of the landscape. The lower the correlation length the more rugged the landscape. Since the correlation length depends on the applied neighborhood operator as well as on the size of the instance, different instances and/or neighborhood operators should be compared by the normalized correlation length l' (Hoos and Stützle, 2005, p. 229):

$$l' = \frac{l}{diam(\mathcal{G}_{\mathcal{N}})} \tag{6.9}$$

with $diam(\mathcal{G}_{\mathcal{N}})$ being the diameter of the search space, which is the maximum distance of any two solutions in $\mathcal{G}_{\mathcal{N}}$. The diameters of the considered neighborhood operators are listed in Table 6.2.

For each CS instance, we get four different fitness landscapes by applying the neighborhood operators of Section 6.3.1 together with equation (6.1) as fitness function. To measure the autocorrelation of the resulting landscapes, we perform a random walk with 100,000 steps on each landscape and determine the random walk correlation coefficient of neighboring solutions $r(1)$ and the normalized correlation length l'. Table 6.3 shows the respective results.

instance	r(1)				l'			
	$\mathcal{N}_{\text{sw}}$	$\mathcal{N}_{\text{ad}}$	$\mathcal{N}_{\text{sh}}$	$\mathcal{N}_{\text{re}}$	$\mathcal{N}_{\text{sw}}$	$\mathcal{N}_{\text{ad}}$	$\mathcal{N}_{\text{sh}}$	$\mathcal{N}_{\text{re}}$
4-72	0.9512	0.9880	0.9648	0.9621	0.2019	0.0167	0.2817	0.2614
6-76	0.9543	0.9878	0.9671	0.9646	0.2162	0.0165	0.3015	0.2805
10-93	0.9526	0.9877	0.9654	0.9631	0.2082	0.0163	0.2867	0.2683
16-81	0.9516	0.9876	0.9642	0.9621	0.2038	0.0162	0.2775	0.2613
19-71	0.9533	0.9880	0.9654	0.9637	0.2111	0.0168	0.2867	0.2732
21-90	0.9550	0.9879	0.9673	0.9648	0.2195	0.0166	0.3040	0.2821
26-92	0.9540	0.9886	0.9668	0.9641	0.2147	0.0176	0.2994	0.2765
41-66	0.9540	0.9887	0.9677	0.9648	0.2143	0.0178	0.3076	0.2820
26-82	0.9537	0.9885	0.9658	0.9642	0.2130	0.0175	0.2902	0.2767

Table 6.3: Random walk correlation coefficients and normalized correlation lengths

The results of the correlation coefficients $r(1)$ suggest a smooth landscape for all instances and neighborhood operators, especially for $\mathcal{N}_{\text{ad}}$. However, since $\mathcal{N}_{\text{ad}}$ has a larger diameter as compared to the other operators, the normalized correlation lengths l', suggests $\mathcal{N}_{\text{sh}}, \mathcal{N}_{\text{re}}$ and $\mathcal{N}_{\text{sw}}$ to be favorable for local search procedures.

6.3.3 Fitness-distance-correlation

The fitness-distance-correlation (FDC) (Jones and Forrest, 1995) is a measure for problem difficulty for evolutionary algorithms. The FDC measures the correlation between the fitness differences of a solution and the global best solution and their distances in the search space. Thus, with a sample of solutions, the FDC coefficient ϱ is defined as

$$\varrho(f, d_{\text{opt}}) = \frac{\text{cov}(f, d_{\text{opt}})}{\sigma(f)\sigma(d_{\text{opt}})} \tag{6.10}$$

with d_{opt} being a solutions distance to the nearest optimal solution and $\text{cov}(f, d_{\text{opt}})$ the covariance of f and d_{opt}. A value of $\varrho = 1$ indicates a perfect correlation between fitness and distance to the optimum. The more both values are correlated, the easier the resulting problem is for selection-based algorithms as a path of solutions with increasing fitness values leads to the optimum (Merz and Freisleben, 2000). The problem difficulty can be classified (Jones and Forrest, 1995) according to ϱ. A value of $\varrho \geq 0.15$ suggests a straightforward minimization problem for evolutionary algorithms, while lower values of ϱ indicate an uncorrelated or even misleading landscape. The FDC

coefficient can be computed for random solutions as well as locally optimal solutions. The usage of locally optimal solutions can give further insights into the global structure of the search space (Merz and Freisleben, 2000). Additionally, fitness-distance plots are suitable for the interpretation of the results.

To compute the FDC coefficient, the distances between solutions have to be known. The actual distance between two solutions is the minimum number of a specific neighborhood operation in order to transform one solution into another. However, the computation of the distances is not straightforward for every neighborhood operator in Section 6.3.1. For instance, no polynomial algorithm is available to compute the distances of the $\mathcal{N}_{\text{re}}$ operator (Schiavinotto and Stützle, 2007). Thus, in order to allow a comparison of the resulting FDC coefficients for different neighborhood operators, we apply the following surrogate distance metrics (Reeves, 1999):

- Adjacency distance metric d_{adj}: The bidirectional adjacency distance computes how often a pair of models is adjacent in both sequences
- Precedence distance metric d_{prec}: The precedence distance computes how often a model m is preceded by a model m' in both sequences.
- Absolute position distance metric d_{abs}: The absolute position distance computes the number of times the position of models is identical in both sequences.

The combination of the four neighborhood operators with the three distance approximations leads to twelve fitness landscapes overall for each instance. For each neighborhood operator, we determine 1000 local optima by applying a steepest ascent hill climbing algorithm using the respective operator which starts from a random solution and stops if a local optimum is reached. The global optimum for each instance is known, except for instance 19-71, where we use the best known solution instead. The global optima are obtained by the IBS algorithm of Chapter 5. The FDC coefficients are calculated for the distance to the global optimum (ϱ_{global}) as well as for the average distances to all other local optima (ϱ_{local}). Table 6.4 presents the respective results.

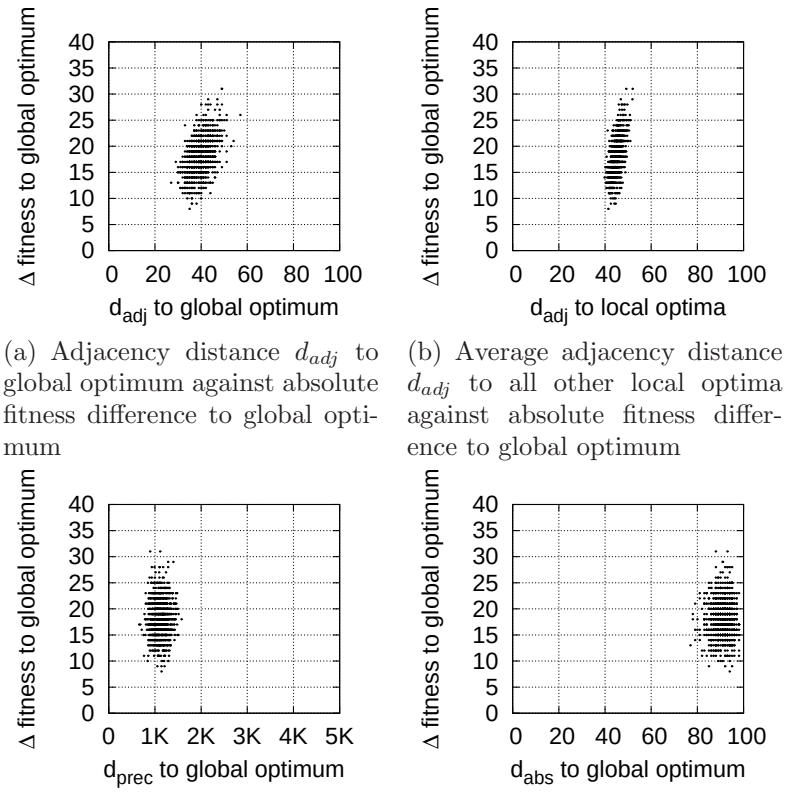
The results suggest that the adjacency distance d_{adj} is the best metric to describe structural properties of local optimal solutions, as the resulting correlation coefficients are higher compared to the other metrics. Thus, a certain amount of adjacent model pairs are shared by high quality solutions. According to Jones and Forrest (1995), $\varrho_{global} \geq 0.15$ and $\varrho_{local} \geq 0.15$ indicate that the resulting landscapes for all instances and neighborhood operators are suitable for EAs using the adjacency distance metric. A value of $\varrho_{global} \geq 0.15$ means, that the fitness and the distance to the global best solution are positively correlated since the smaller the distance to the global best solution the

instance		$\mathcal{N}_{\text{sw}}$			$\mathcal{N}_{\text{ad}}$			$\mathcal{N}_{\text{sh}}$			$\mathcal{N}_{\text{re}}$		
		d_{adj}	d_{prec}	d_{abs}	d_{adj}	d_{prec}	d_{abs}	d_{adj}	d_{prec}	d_{abs}	d_{adj}	d_{prec}	d_{abs}
4-72	ϱ_{global}	0.385	0.041	0.110	0.419	0.302	0.023	0.404	0.001	0.015	0.504	0.106	0.117
	ϱ_{local}	0.343	0.025	0.194	0.319	-0.085	-0.416	0.451	0.074	-0.069	0.510	0.127	0.245
6-76	ϱ_{global}	0.256	-0.024	0.015	0.388	0.054	-0.109	0.283	0.006	-0.062	0.319	0.048	0.102
	ϱ_{local}	0.365	0.032	0.076	0.284	-0.077	-0.387	0.429	0.073	-0.019	0.466	0.031	0.126
10-93	ϱ_{global}	0.260	-0.062	-0.005	0.575	0.276	-0.012	0.350	-0.039	-0.015	0.504	-0.001	0.063
	ϱ_{local}	0.340	-0.027	0.009	0.237	-0.146	-0.441	0.436	-0.065	-0.112	0.579	0.019	0.094
16-81	ϱ_{global}	0.300	0.056	0.044	0.451	0.256	0.077	0.357	0.061	0.060	0.470	0.102	0.103
	ϱ_{local}	0.425	-0.038	0.136	0.288	-0.054	-0.264	0.419	0.020	-0.026	0.557	0.070	0.158
19-71	ϱ_{global}	0.202	0.030	0.045	0.565	0.282	-0.061	0.317	0.036	0.039	0.336	0.082	0.042
	ϱ_{local}	0.354	0.026	0.076	0.165	-0.156	-0.507	0.451	-0.035	-0.140	0.541	0.112	0.148
21-90	ϱ_{global}	0.306	-0.010	0.043	0.541	0.286	0.003	0.309	0.006	0.021	0.364	0.045	0.063
	ϱ_{local}	0.375	0.038	0.062	0.268	-0.067	-0.372	0.469	-0.001	-0.034	0.573	-0.004	0.158
36-92	ϱ_{global}	0.191	0.068	0.041	0.610	0.138	-0.168	0.317	0.006	-0.039	0.384	0.022	0.071
	ϱ_{local}	0.359	0.034	0.111	0.135	-0.139	-0.476	0.465	0.011	-0.121	0.559	0.114	0.208
41-66	ϱ_{global}	0.112	0.024	0.002	0.427	0.048	-0.062	0.187	-0.189	-0.077	0.199	0.030	0.081
	ϱ_{local}	0.256	0.044	0.105	0.222	-0.175	-0.368	0.406	-0.086	-0.187	0.384	0.004	0.102
26-82	ϱ_{global}	0.289	0.010	0.090	0.525	0.316	-0.015	0.322	-0.068	0.025	0.403	0.088	0.024
	ϱ_{local}	0.367	0.052	0.194	0.242	-0.108	-0.512	0.413	-0.022	0.041	0.503	0.085	0.164

Table 6.4: Fitness-distance correlation coefficients ϱ

lower the fitness value of a solution. Note, that in our case solutions with a low fitness are favored, since CS is a minimization problem. $\varrho_{local} \geq 0.15$ indicates that the lower the fitness value of a solution the smaller the average distance to all other local optima. Thus, high quality local optima lie in the center of other local optima.

In Figure 6.1, we show some scatter plots for a representative example using instance 10-93 and $\mathcal{N}_{\text{re}}$. Figures 6.1(a) and 6.1(b) plot for each local optimum its adjacency distance to the global optimum and its average adjacency distance to all other local optima, respectively, against its absolute fitness deviation to the global optimum. We can observe the aforementioned positive correlation between the adjacency distance and the fitness of a solution. Furthermore, all local optima are clustered in a small region of the search space as the maximum distance between any two local optima is 12,15 (see Figure 6.1(b)). Figures 6.1(c) and 6.1(d) present the scatter plots for the precedence distance and absolute distance, respectively, of each solution to the global optimum against its absolute fitness deviation to the global optimum. Both distance metrics seem to have a low correlation which confirms the results of Table 6.4. Again, for d_{prec}, the local optima appear to be grouped in a small



(a) Adjacency distance d_{adj} to global optimum against absolute fitness difference to global optimum

(b) Average adjacency distance d_{adj} to all other local optima against absolute fitness difference to global optimum

(c) Precedence distance d_{prec} to global optimum against absolute fitness difference to global optimum

(d) Absolute distance d_{abs} to global optimum against absolute fitness difference to global optimum

Figure 6.1: Scatter plots for instance 10-93 and $\mathcal{N}_{\text{re}}$ based on 1000 local optima.

region around the global optimum, whereas, for d_{abs}, the distances to the global optimum are close to the diameter of the search space which suggests that the local optima are evenly distributed in the landscape. Interpreting the plots, we assume the existence of a 'big valley' structure (Boese et al., 1994) for d_{adj}, since local optima are accumulated in a small region of the search space with high quality solutions being in the center of the local optima.

6.4 Metaheuristics for car sequencing

In this section, we describe two metaheuristics for CS, a variable neighborhood search (VNS) and a memetic algorithm (MA), whose designs are based on the results of the preceding landscape analysis.

```
Input: initial solution x
Output: best solution found
  Initialize probabilities ρ
  Evaluate(x)
  for i := 1 to N_VNS do
    N ← SelectNeighborhoodOperator(ρ)
    x' ← N(x)
    Evaluate(x')
    if x'.fitness ≤ x.fitness then
      x ← x'
    end if
    i ← i + 1
  end for
```

Figure 6.2: Outline of VNS algorithm

6.4.1 Variable neighborhood search

Local search (LS) algorithms explore the fitness landscape of a problem instance by iteratively moving from one solution to a new neighboring solution with a better fitness, until a local optimum is reached. In order to avoid being trapped in the first local optimum found, several metaheuristics based on LS have been developed, like simulated annealing (Kirkpatrick et al., 1983), which accepts worse solutions with a certain probability, or tabu search (Glover, 1989), where the last visited solutions are stored in a tabu list so that they are not revisited repeatedly. A different metaheuristic, called variable neighborhood search (VNS) was proposed by Mladenovic and Hansen (1997), where instead of merely one neighborhood operator, a set of neighborhoods is applied. Thus, VNS can escape a local optimum in one landscape, by using a different neighborhood operator and, thus, changing the landscape. The original scheme of VNS arranges all neighborhood operators in a certain sequence with increasing neighborhood size. If VNS experiences a local optimum using one neighborhood operator, the search proceeds with the next operator in the list until a certain stopping criterion is reached.

In order to decide if the current solution is a local optimum, a systematic exploration of its entire neighborhood is required. Since for large neighborhoods this can be computational demanding, we use a different approach of VNS for the CS problem, inspired by Puchta and Gottlieb (2002). Instead of applying the neighborhood operators in a predefined sequence, we choose a neighborhood operator with a certain probability p at each step of the search. The operator is then applied to the current solution and the search proceeds with the new found neighboring solution if its fitness value is not worse compared to the current solution. Allowing the search to continue with solutions having the same fitness value as the current solution is useful to traverse plateaus of equal fitness in the landscape. The search is stopped after a maximum number

$\mathcal{N}_{\text{swa}}$	$\begin{cases} \frac{T}{2} & \text{, for even } T \\ \frac{T-1}{2} & \text{, for odd } T \end{cases}$
$\mathcal{N}_{\text{adj}}$	$\frac{T(T-1)}{2}$
$\mathcal{N}_{\text{ins}}$	$T-1$
$\mathcal{N}_{\text{rev}}$	1

Table 6.5: Distances between redundant CS solutions

of steps N_{VNS} is reached. The algorithm is outlined in Figure 6.2.

The probabilities p of the different neighborhood operators are obtained using the results of Section 6.3.2, where the autocorrelation of the landscape induced by each operator is analyzed. It is assumed that the smoother the landscape, the better the performance of LS (Hoos and Stützle, 2005). Regarding the normalized correlation lengths l' in Table 6.3, the shift operator $\mathcal{N}_{\text{sh}}$ leads to the smoothest landscape, followed by $\mathcal{N}_{\text{re}}$ and $\mathcal{N}_{\text{sw}}$. However, we also have to consider the locality induced by our representation and neighborhood operators (Rothlauf, 2006; Choi and Moon, 2008). Locality describes how well the genotypic neighborhood corresponds to the phenotypic neighborhood of a solution (Rothlauf, 2003). Our representation of solutions leads to redundant solutions in the genotypic search space, as for each sequence a symmetric solution with equal fitness can be found by inverting the sequence. A neighborhood operator which induces a large distance between both, the sequence and its inverted counterpart, results in a low locality, as two actually identical areas in the genotypic landscape are far away from each other. Thus, high quality solutions are also clustered in two regions of the search space, from which merely one is explored by a LS algorithm. In contrast, LS benefits from a small distance between both redundant solutions as the search can be intensified in one small area of the search space where high quality solutions are concentrated. For each neighborhood operator, the distance between a sequence and its inverted counterpart is shown in Table 6.5.

For the $\mathcal{N}_{\text{ad}}$ and $\mathcal{N}_{\text{sh}}$ operator, the distance between two redundant solutions equals the diameter of the respective landscapes (compare with Table 6.2), thus, both solutions are maximally away from each other in the search space. For the $\mathcal{N}_{\text{sw}}$ operator, the distance between two redundant sequences is half the diameter of its landscape. Using $\mathcal{N}_{\text{re}}$, the inverted solution can be reached in one operation as both solutions are neighbors. In summary, the $\mathcal{N}_{\text{re}}$ neighborhood operator seems most promising for LS, since it results in a smooth landscape and induces a high locality of redundant solutions. Followed by $\mathcal{N}_{\text{sw}}$ and $\mathcal{N}_{\text{sh}}$, where $\mathcal{N}_{\text{sh}}$ leads to a smoother landscape compared to $\mathcal{N}_{\text{sw}}$, but $\mathcal{N}_{\text{sw}}$ results in a higher locality of redundant solutions. Thus, we base our VNS on these three operators and assign $\mathcal{N}_{\text{re}}$ a probability of 60% to be chosen

at each step and $\mathcal{N}_{\text{sw}}$ and $\mathcal{N}_{\text{sh}}$ each a probability of 20%. We ignore the $\mathcal{N}_{\text{ad}}$ operator since its normalized correlation length l' is low and it induces a low locality of redundant solutions.

6.4.2 Memetic algorithm

Memetic algorithms (MA) (Moscato, 1989) are a conjunction of evolutionary algorithms (EA) and local search (LS). Thus, they combine the concept of population-based evolution with individual learning. In the literature, MAs are also referred to as hybrid genetic algorithms or genetic local search. They are similar to EAs in that a population of individual solutions to a problem is iteratively modified by evolutionary recombination and mutation operators, in order to explore the search space and guide the search to promising areas. In contrast to EAs, an additional local search is performed at each generation, which locally improves individuals of the population to intensify the search in promising areas. MAs have been successfully applied to other scheduling problems like parallel-machine scheduling, job-shop scheduling or flow-shop scheduling (Cotta and Fernandez, 2007; Burke and Landa Silva, 2005). Comprehensive introductions to MAs can be found in Moscato and Cotta (2003) and Krasnogor and Smith (2005). Algorithm 6.3 outlines the general structure of our MA for the CS problem, which is described in detail in the following.

A population P in EAs consists of a set of individual solutions. The size of the population $|P|$ in MAs is usually much smaller than in traditional EAs due to the complexity of the local search, which inhibits the evolution of large populations (Merz and Freisleben, 2000). The initial population can be set up randomly or obtained by a heuristic procedure. Using heuristics for the initialization phase usually improves the performance of an EA as fewer generations are needed to guide the population to promising areas in the search space. Thus, we initialize our population using the heuristic IBS algorithm of Chapter 5 with beam width ω.

The population-based evolution in MAs is achieved by selection, recombination and mutation. At each generation of our MA approach, two solutions, named parents, are selected for crossover by a binary tournament selection. The crossover operator combines parts of both solutions in order to derive two offspring solutions. This corresponds to the reproduction in biological evolution. In EAs, the crossover operator is used to intensify the search in promising areas of the landscape, thus, existing similarities of parent solutions should be preserved in the offspring. This characteristic is called respectfulness (Radcliffe, 1991) and is necessary for successful evolutionary search (Sywerda, 1989). Especially in the presence of a big valley, respectful crossover is likely to perform well, as high quality solutions can be found in the vicinity of other good solutions. The preceding FDC analysis of Section 6.3.3 suggests that

```
Output: best solution found
  InitialPopulation P ← IBS(ω)
  Evaluate(P)
  Sort(P)
  for i := 1 to N_MA do
    Parents p_1, p_2 ← TournamentSelection(size = 2)
    Offspring o_1, o_2 ← Crossover(p_1, p_2)
    Evaluate(o_1, o_2)
    if o_1.fitness ≤ o_2.fitness then
      j ← 1
    else
      j ← 2
    end if
    o_j ← VNS(o_j)
    x ← o_j with x being the last element in P
    for all x ∈ P do
      if RandomNumber(0,1) ≤ p_m then
        x ← Mutate(x)
      end if
    end for
    for all x ∈ P do
      if RandomNumber(0,1) ≤ p_ls then
        x ← VNS(x)
      end if
    end for
    Sort(P)
    i ← i + 1
  end for
```

Figure 6.3: Outline of MA

the adjacency distance metric is suited to describe similarities between high quality solutions for CS and even leads to a big valley. Therefore, the crossover operator should preserve adjacent relationships of models existing in both parents.

We consider three recombination operators for CS, order crossover (OX) (Davis, 1985) and propose a heuristic variant of OX (hOX), as well as non conflict position crossover (NCPX), which showed good results in previous experiments (Zinflou et al., 2007). OX, outlined in Figure 6.4, randomly selects a subsequence of one parent and transfers it to the corresponding slots of one offspring. Beginning at the second crossover site of the offspring, the remaining models are allocated according to their occurrence in the second parent. OX preserves the absolute positions of a subsequence in the first parent and the relative order of models in the alternative parent. It is called a blind recombination operator as it uses no problem-specific knowledge. Additionally, we propose a heuristic variant of OX (hOX), where we choose the subsequence of parent 1 such that it is delimited by violated slots. A slot is violated, if the

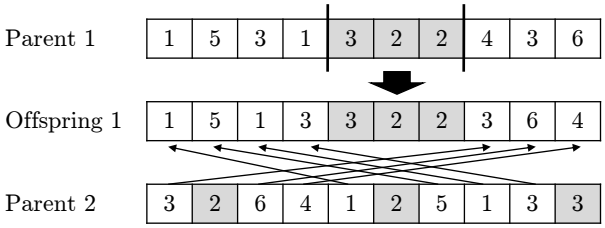


Figure 6.4: Order crossover (OX)

respective model at this slot induces at least one sequencing rule violation. If a sequence contains only one violated slot, the second crossover site is chosen randomly. The remaining models are assigned from the second parent the same way as in OX, but starting at the first slot of the offspring. To apply hOX, a vector of length T indicating the violated slots has to be stored and updated during the search. The third recombination operator NCPX is also a heuristic operator and was proposed for CS in Zinflou et al. (2007). It is outlined for an example sequence in Figure 6.5. First a random number $\mathrm{nb_g}$ between 0 and the number of non-violated slots is chosen. Our example has 7 non-violated slots and $\mathrm{nb_g}$ amounts to 5. Then, a random starting point $\mathrm{Pos_d}$ is selected from which $\mathrm{nb_g}$ models at non-violated slots are copied from parent one to the corresponding slots at the offspring. If the end of the sequence is meanwhile reached, the copy process proceeds at the beginning of the sequence. After $\mathrm{nb_g}$ models are copied, the remaining models form a list of interest L. Another random position *Pos* is chosen, from which the models in L are assigned to the empty slots according to a heuristic function which considers the induced number of violations by a model as well as a so-called utilization rate (Gottlieb et al., 2003). If models are tied in the resulting number of violations as well as in the utilization rate and one of these model occupies the same slot in the second parent, this model is selected. Alternatively, ties are broken randomly. Note, that only in case of ties a second parent is considered by NCPX and an offspring is created by merely a single parent, otherwise. NCPX preserves absolute positions of $\mathrm{nb_g}$ models of the parent solution.

We examine the resulting similarity of parents and offspring according to the adjacency distance metric d_{adj} for the different crossover operators. Therefore, 1000 recombinations are performed with randomly selected parents for each crossover operator and the adjacency distance between both parents is compared with the resulting average adjacency distance between both offspring and parents. Figure 6.6 plots the results for the representative instance 4-72. We can observe, that OX and hOX produce offspring with a lower average distance to both parents compared to NCPX. For both operators, the distance between offspring and parents becomes smaller with decreasing distance be-

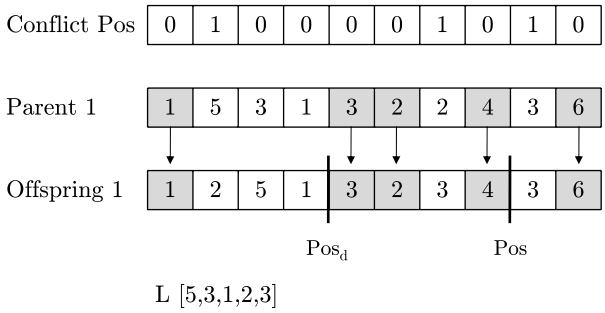


Figure 6.5: Non conflict position crossover (NCPX)

tween both parents. Thus, offspring are likely to be produced in the vicinity of both parent solutions. In contrast, NCPX doesn't seem to preserve existing adjacency relations as the average distances of the offspring are independent of the distance between both parents.

After recombination, the offspring with the least number of violations is locally improved using the VNS algorithm of Section 6.4.1. VNS introduces individual learning to the MA. The resulting solution replaces the current worst solution in the population P. Thus, we perform a so-called steady-state selection scheme (Whitley and Kauth, 1988), where one individual in the population is replaced at each generation.

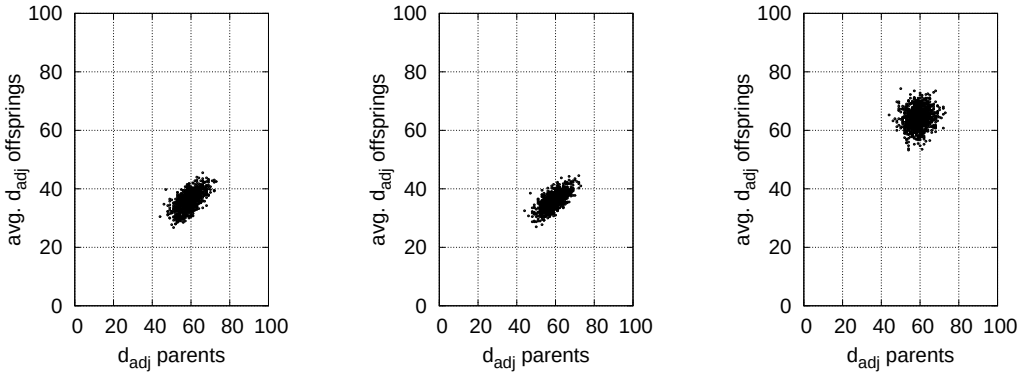
The mutation operator is used for diversification in population-based EAs. It randomly changes individuals in the population in order to explore new regions in the search space and prevent an early convergence to a single local optimum. In our MA, individuals are mutated by a single application of $\mathcal{N}_{\text{re}}$, thus, a random subsequence of a solution is reversed. We perform mutation with probability p_m after recombination and the local improvement of the best offspring. After mutation, each generation concludes with another application of VNS. The VNS is applied to each individual in the population depending on a certain probability p_{ls}.

The MA is stopped after reaching the maximum number of generations N_{MA}.

6.5 Experiments

6.5.1 Experimental setup

We evaluate the proposed algorithms on the problem instances for CS available in the CSPLib, an online library of constraint satisfaction problems. The instances are divided in two sets. The first set consists of the aforementioned nine instances, all with a sequence length $T = 100$, 5 options and 19-26 models. The second set is composed of 30 larger problem instances with 200-400



(a) Order Crossover (OX) (b) Heuristic Order Crossover (OX^h) (c) Non Conflict Position Crossover (NCPX)

Figure 6.6: Adjacency distance d_{adj} between random parents against the average adjacency distance d_{adj} between the resulting offspring and their parents for different crossover operators (based on instance 4-72).

production slots, 5 options and 19-26 models.

The algorithms are implemented in JAVA. All experiments run on an Intel Xeon X5570 with 2.93 GHz using 4 GB RAM.

6.5.2 Results

The algorithms are applied with parameters from Table 6.6. We perform two versions of VNS with different neighborhood operators and probabilities for selecting neighborhoods. VNS_1 corresponds to Section 6.4.1 using a set of three neighborhood operators with probabilities $\{\mathcal{N}_{\text{re}}, \mathcal{N}_{\text{sw}}, \mathcal{N}_{\text{sh}}\} = \{0.6, 0.2, 0.2\}$. VNS_2 applies all neighborhood operators of Section 6.3.1 with equal probabilities, thus, $\{\mathcal{N}_{\text{sw}}, \mathcal{N}_{\text{ad}}, \mathcal{N}_{\text{sh}}, \mathcal{N}_{\text{re}}\} = \{0.25, 0.25, 0.25, 0.25\}$. The initial solution for both VNS' is obtained by the aforementioned IBS algorithm with beam width $\omega = 5$. Starting from the initial solution, VNS_1 and VNS_2 are applied with a maximum of 50,000*T moves, where T is the number of production slots in the sequence and the best sequence found is returned.

Furthermore, we consider three MA algorithms each with a different crossover operator. Thus, MA_{OX} uses the order crossover, MA_{hOX} the heuristic order crossover and MA_{NCPX} the non conflict position crossover. All MAs have a population size $|P|$ of 20 individuals and the initial population is set up with the best 20 individuals found by the IBS algorithm with $\omega = 100$. Within the MAs, VNS_1 is applied with at most 500*T moves. The mutation and local search probabilities are set to 0.05 and 0.1, respectively. Table 6.7 shows the

Table 6.6: Parameters of algorithms in the experiments

Parameter		VNS	MA
IBS beam width	ω	5	100
Number of MA generations	N_{MA}	-	100
Number of VNS moves	N_{VNS}	50,000*T	500*T
Population size	$\|P\|$	-	20
Mutation probability	p_m	-	0.05
Local Search probability	p_ls	-	0.1

performance results of the five considered algorithms on both sets of problem instances. The results are obtained by 10 independent runs of each algorithm on each instance. obj* states the currently best known solutions as obtained by a genetic algorithm with a subsequent local search (Zinflou et al., 2007). For each algorithm and instance, we present the number of sequencing rules of the best solution found (column 'best obj.'), the resulting average number of sequencing rules (column 'avg. obj.), as well as the average time (column avg. time) in seconds required for the 10 runs. To exemplify the results, the best average objective values are highlighted in gray and new best solutions found are marked with an asterisk.

Among the MAs, MA_{hOX} results in the best average objective values on all but two instances. Furthermore, it finds the currently best known solution for each instance, except instance 400-09. However, due to the heuristic crossover operator which requires to maintain the information about violated slots during the search, the solution time of MA_{hOX} as well as MA_{NCPX} is considerably larger than MA_{OX}. Compared to MA_{NCPX}, MA_{OX} leads to better average objective values on 5 instances and inferior values on 3 instances. Thus, maintaining adjacency relationships between parent solutions during crossover, as in MA_{OX} and MA_{hOX}, seems promising. Given that the differences in the solution quality of all MAs are not very large, the main contribution to the results is assumed to stem from the VNS operator and not the recombination operator. This is also confirmed by the results of the VNS_1 algorithm, which shows the overall best performance. It requires considerably less time than the MA algorithms and finds the best known solutions for all instances and even improved solutions with 3 and 27 violations for instances 200-03 and 300-05, respectively. Considering the average objective values, VNS_1 leads to the best results on 38 out of 40 instances. When changing the applied neighborhood operators and their probabilities, as in VNS_2, the solution quality on the instances of set two decreases.

instance	obj*	VNS_1			VNS_2			MA_{OX}			MA_{hOX}			MA_{NCPX}		
		best obj.	avg. obj.	avg. time[s]	best obj.	avg. obj.	avg. time[s]	best obj.	avg. obj.	avg. time[s]	best obj.	avg. obj.	avg. time[s]	best obj.	avg. obj.	avg. time[s]
4-72	0	0	0.0	0.61	0	0.0	1.28	0	0.0	1.21	0	0.0	1.07	0	0.0	1.10
6-76	6	6	6.0	13.01	6	6.0	12.49	6	6.0	41.11	6	6.0	67.16	6	6.0	67.87
10-93	3	3	3.0	12.98	3	3.0	12.76	3	3.0	41.51	3	3.0	69.13	3	3.0	69.33
16-81	0	0	0.0	2.69	0	0.0	4.17	0	0.0	1.43	0	0.0	1.44	0	0.0	1.46
19-71	2	2	2.0	12.91	2	2.0	12.78	2	2.0	41.03	2	2.0	62.09	2	2.0	62.59
21-90	2	2	2.0	12.85	2	2.0	12.77	2	2.0	41.30	2	2.0	61.58	2	2.0	62.05
36-92	2	2	2.0	12.87	2	2.0	12.76	2	2.0	41.18	2	2.0	66.38	2	2.0	66.30
41-66	0	0	0.0	0.10	0	0.0	0.09	0	0.0	0.73	0	0.0	0.76	0	0.0	0.80
26-82	0	0	0.0	0.66	0	0.0	1.74	0	0.0	1.10	0	0.0	1.11	0	0.0	1.14
200-01	0	0	0.0	10.17	0	0.1	11.17	0	0.1	47.25	0	0.1	39.17	0	0.1	65.43
200-02	2	2	2.0	31.88	2	2.0	30.53	2	2.0	96.22	2	2.0	167.65	2	2.0	168.97
200-03	4	3*	4.4	32.03	4	4.7	30.72	4	5.1	96.41	4	4.7	188.92	4	5.0	193.63
200-04	7	7	7.0	32.06	7	7.0	30.42	7	7.0	95.96	7	7.0	191.35	7	7.0	193.58
200-05	6	6	6.0	31.74	6	6.0	30.59	6	6.0	95.82	6	6.0	178.32	6	6.0	182.28
200-06	6	6	6.0	31.37	6	6.0	30.66	6	6.0	95.78	6	6.0	179.57	6	6.0	180.31
200-07	0	0	0.0	0.29	0	0.0	0.45	0	0.0	2.54	0	0.0	2.64	0	0.0	2.57
200-08	8	8	8.0	31.11	8	8.0	30.77	8	8.0	94.66	8	8.0	190.78	8	8.0	191.45
200-09	10	10	10.0	31.27	10	10.0	30.70	10	10.0	95.89	10	10.0	199.32	10	10.0	198.95
200-10	19	19	19.0	31.12	19	19.1	30.57	19	19.0	94.62	19	19.0	223.98	19	19.0	224.70
300-01	0	0	0.0	18.82	0	0.4	39.84	0	0.6	121.15	0	0.4	207.64	0	0.4	212.76
300-02	12	12	12.0	54.75	12	12.0	52.25	12	12.0	168.08	12	12.0	351.28	12	12.0	350.90
300-03	13	13	13.0	53.61	13	13.0	52.30	13	13.0	163.45	13	13.0	351.87	13	13.0	356.48
300-04	7	7	7.2	53.83	7	7.4	52.17	7	7.1	163.74	7	7.1	329.10	7	7.1	340.02
300-05	29	27*	29.2	54.72	29	29.9	52.02	29	29.7	162.74	29	29.7	398.45	29	29.8	404.93
300-06	2	2	2.0	55.63	2	2.2	51.92	2	3.1	164.27	2	3.2	308.74	3	3.3	312.04
300-07	0	0	0.0	3.63	0	0.0	6.80	0	0.0	15.16	0	0.0	19.55	0	0.0	20.63
300-08	8	8	8.0	53.87	8	8.0	52.02	8	8.0	163.97	8	8.0	333.13	8	8.0	336.66
300-09	7	7	7.0	54.96	7	7.3	52.09	7	7.0	162.38	7	7.0	345.58	7	7.0	345.86
300-10	21	21	21.0	55.27	21	21.1	52.11	21	21.0	159.86	21	21.0	390.76	21	21.0	400.17
400-01	1	1	1.1	83.29	1	1.3	77.44	1	1.8	242.35	1	1.7	432.35	1	1.9	435.24
400-02	15	15	15.4	83.19	15	15.6	78.04	16	16.2	238.09	15	15.8	530.02	16	16.4	535.58
400-03	9	9	9.1	83.38	9	9.2	77.23	9	9.0	243.58	9	9.0	507.20	9	9.0	517.08
400-04	19	19	19.0	82.80	19	19.0	77.64	19	19.0	243.77	19	19.0	564.87	19	19.0	566.98
400-05	0	0	0.0	0.25	0	0.0	0.20	0	0.0	5.33	0	0.0	11.73	0	0.0	4.44
400-06	0	0	0.0	6.01	0	0.0	7.76	0	0.0	18.31	0	0.0	38.60	0	0.0	38.60
400-07	4	4	4.0	82.43	4	4.0	77.80	4	4.3	244.32	4	4.0	470.11	4	4.1	474.97
400-08	4	4	4.0	81.69	4	4.0	77.84	4	4.0	243.84	4	4.0	466.17	4	4.0	468.50
400-09	5	5	6.6	81.64	6	6.9	78.09	6	6.8	242.96	6	6.9	519.20	6	6.9	524.94
400-10	0	0	0.0	6.55	0	0.0	11.98	0	0.0	7.04	0	0.0	11.68	0	0.0	10.79

Table 6.7: Results on problem sets 1 and 2

6.6 Conclusions

We analyze the fitness landscapes of a set of CS instances by measuring the autocorrelation as well as fitness-distance correlation when four different neighborhood operators and three distances are applied. The results show a smooth landscape in terms of the normalized correlation length for the reverse, swap and shift neighborhood. The adjacency distance is suitable to describe structural relations between solutions as it leads to a high fitness-distance correlation. Furthermore, a big valley structure can be identified when using the adjacency distance metric. The findings are included in two metaheuristics for CS, a variable neighborhood search (VNS) and a memetic algorithm (MA) evaluated with three different crossover operators. In experiments, we show the superiority of the VNS algorithm as it finds and even improves currently best known solutions for instances of the CSPLib. Despite that the performances of the MA algorithms are inferior compared to VNS, MAs with a crossover operator that respects the adjacency distance metric have a better solution quality than MAs without.

The findings of the fitness-distance correlation analysis would be more meaningful if the true distances according to the applied neighborhood operators could be used. Therefore, future research should address efficient algorithms or at least good approximations to determine these distances. Further insights should also be gained as to why local search in general leads to better results than EAs for the CS problem. We assume redundant solutions to have a negative effect on the solution quality of EAs. Other representations for CS or the normalization of CS solutions prior to recombination should be analyzed and incorporated in EAs for CS.

Bibliography

Boese, K. D., Kanhg, A. B., and Muddu, S. (1994). A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2):101–113.

Boysen, N., Fliedner, M., and Scholl, A. (2009). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.

Burke, E. and Landa Silva, J. (2005). The design of memetic algorithms for scheduling and timetabling problems. In Hart, W., Smith, J., and Krasnogor, N., editors, *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*, pages 289–311. Springer, Berlin Heidelberg.

Butaru, M. and Habbas, Z. (2005). Solving the car-sequencing problem as a non-binary CSP. In *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *LNCS*, page 840, Berlin Heidelberg. Springer.

Choi, S.-S. and Moon, B.-R. (2008). Normalization for genetic algorithms with nonsynonymously redundant encodings. *IEEE Transactions on Evolutionary Computation*, 12(5):604–616.

Cordeau, J.-F., Laporte, G., and Pasin, F. (2008). Iterated tabu search for the car sequencing problem. *European Journal of Operational Research*, 191(3):945–956.

Cotta, C. and Fernandez, A. (2007). Memetic algorithms in planning, scheduling, and timetabling. In Dahal, K., Tan, K., and Cowling, P., editors, *Evolutionary Scheduling*, volume 49 of *Studies in Computational Intelligence*, pages 1–30. Springer, Berlin Heidelberg.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th international joint conference on Artificial intelligence*, volume 1, pages 162–164, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Estellon, B., Gardi, F., and Nouioua, K. (2008). Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research*, 191(3):928–944.

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

Gagne, C., Gravel, M., and Price, W. (2006). Solving real car sequencing problems with ant colony optimization. *European Journal of Operational Research*, 174(3):1427–1448.

Glover, F. (1989). Tabu search - Part I. *ORSA Journal on Computing*, 1(3):190–206.

Gottlieb, J., Puchta, M., and Solnon, C. (2003). A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In Cagnoni, S., Johnson, C., Cardalda, J., Marchiori, E., Corne, D., Meyer, J.-A., Gottlieb, J., Middendorf, M., Guillot, A., Raidl, G., and Hart, E., editors, *EvoWorkshops 2003*, volume 2611 of *LNCS*, pages 246–257, Berlin Heidelberg. Springer.

Gravel, M., Gagne, C., and Price, W. L. (2005). Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56(11):1287–1295.

Hoos, H. and Stützle, T. (2005). *Stochastic local search*. Elsevier, Amsterdam.

Jaszkiewicz, A., Kominek, P., and Kubiak, M. (2004). Adaptation of the genetic local search algorithm to a car sequencing problem. In *7th National Conference on Evolutionary Algorithms and Global Optimization*, pages 67–74, Kazimierz Dolny, Poland.

Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192.

Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kis, T. (2004). On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335.

Krasnogor, N. and Smith, J. (2005). A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions On Evolutionary Computation*, 9(5):474–488.

Merz, P. and Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions On Evolutionary Computation*, 4(4):337–352.

Mladenovic, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. C3p report, Caltech Concurrent Computation Program.

Moscato, P. and Cotta, C. (2003). A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 105–144. Springer.

Parrello, B. D., Kabat, W. C., and Wos, L. (1986). Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2(1):1–42.

Perron, L. and Shaw, P. (2004). Combining forces to solve the car sequencing problem. In Regin, J.-C. and Rueher, M., editors, *Integration of AI and*

OR Techniques in Constraint Programming for Combinatorial Optimization Problems, volume 3011 of *LNCS*, pages 225–239, Berlin Heidelberg. Springer.

Prandtstetter, M. and Raidl, G. (2008). 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.

Puchta, M. and Gottlieb, J. (2002). Solving car sequencing problems by local optimization. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, volume 2279 of *LNCS*, pages 132–142, Berlin Heidelberg. Springer.

Radcliffe, N. J. (1991). Forma analysis and random respectful recombination. In *Proceedings of ICGA*, pages 222–229.

Reeves, C. R. (1999). Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490.

Rothlauf, F. (2003). On the locality of representations. In Cantù-Paz, E., editor, *Genetic and Evolutionary Computation - GECCO 2003*, volume 2724 of *LNCS*, pages 1608–1609, Berlin Heidelberg. Springer.

Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms*. Springer, Berlin Heidelberg, 2nd edition.

Schiavinotto, T. and Stützle, T. (2007). A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143–3153.

Solnon, C. (2008). Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. *European Journal of Operational Research*, 191(3):1043–1055.

Solnon, C., Cung, V., Nguyen, A., and Artigues, C. (2008). 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.

Stadler, P. F. (1996). Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20(1):1–45.

Sywerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Terada, J., Vo, H., and Joslin, D. (2006). Combining genetic algorithms with squeaky-wheel optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO*, pages 1329–1336. ACM.

Warwick, T. and Tsang, E. (1995). Tackling car sequencing problems using a generic genetic algorithm. *Evolutionary Computation*, 3(3):267–298.

Weinberger, E. D. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336.

Wester, L. and Kilbridge, M. D. (1964). The assembly line model-mix sequencing problem. In *Proceedings of the Third International Conference on Operations Research*.

Whitley, L. D. and Kauth, J. (1988). GENITOR: A different genetic algorithm. In *Proc. Rocky Mountain Conf. Artificial Intel.*, pages 118–130, Denver, CO.

Zinflou, A., Gagne, C., and Gravel, M. (2007). Crossover operators for the car sequencing problem. In Cotta, C. and van Hemert, J., editors, *Proceedings of the 7th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2007*, volume 4446 of *LNCS*, pages 229–239, Valencia, Spain.

Zinflou, A., Gagne, C., and Gravel, M. (2008). Designing hybrid integrative evolutionary approaches to the car sequencing problem. In *IPDPS 2008. IEEE International Symposium on Parallel and Distributed Processing*.

Chapter 7

Summary and conclusions

The purpose of this thesis is to analyze the solution quality of the CS problem, apply it to the resequencing problem as well as present different exact and metaheuristic solution methods in order to solve it. This chapter summarizes the work and lists its major contributions.

7.1 Summary

In Chapter 2, we studied the solution quality of CS compared to MMS, when both are considered as constraint satisfaction problems. We showed that CS misclassifies a large fraction of actual optimal solutions, that contain no work overload, as unfeasible (hence, these solutions violate at least one sequencing rule) when it is applied with sequencing rules generated by the only existing rule generation approach from Bolat and Yano (1992a). The created rules were found to be too restrictive and reduce the solution space of CS compared to MMS, hence, impede the search process of CS. The fraction of misclassified sequences increases with increasing sequence length and number of line stations considered. In order to overcome this problem, a new rule generation approach, called multiple sequencing rules (MSR) approach, is proposed, which creates multiple sequencing rules per option. We proved that MSR enables CS to correctly classify all sequences as feasible and unfeasible, hence, MMS-feasibility $\leftrightarrow$ CS-feasibility. A notion of strictness is presented to remove redundant sequencing rules and reduce the overall number of generated rules by MSR. Furthermore, a new approach is developed to include options with more than two processing times at a station into CS. However, this negatively affects the solution quality of CS compared to MMS in terms of the percentage of misclassified optimal solutions.

The third chapter examines CS as a combinatorial optimization problem and investigates its solution quality compared to MMS. Therefore, different variants of CS are considered combining two sequencing rule generation approaches and three existing objective functions from the literature. Furthermore, an additional weighting factor in order to distinguish between option violations is

developed and applied with CS. The linear relationship of the objective values produced by CS and MMS is examined using Pearson's product-moment coefficient. We compare the solution quality of CS and MMS on random test instances, according to Scholl (1999). The resulting MMS and CS instances are solved by the same local search algorithm and the solution quality of both approaches is measured as the total amount of work overload induced by the best sequences found during the search. The results revealed that solutions of CS (using the MSR generation approach, an objective function by Bolat and Yano (1992a) and the additional weighting factor) contain on average 15% more work overload than corresponding solutions of MMS. The solution quality of CS decreases, if it is applied with other sequencing rules and/or objective functions. In Section 3.5, we critically discussed the usage of CS instead of MMS for sequencing mixed-model assembly lines.

Chapter 4 adapted the CS problem to the resequencing problem using pull-off tables, which are directly accessible buffers used in the automobile industry. The resulting model was called car resequencing (CRS) problem and formalized in Section 4.2. CRS was transformed into a graph problem and different exact and heuristic solution approaches were developed. Experiments on data from the CSPLib were conducted and the performance of the solution methods as well as the influence of the number of pull-off tables on the solution quality was analyzed. Furthermore, the best known solution method (Beam Search) has been applied to a real-world resequencing scenario at a major German truck manufacturer, where it was found to improve the resulting solution quality by 29% compared to the currently applied myopic scheduling policy.

Inpired by the graph approach for CRS in Chapter 4, Chapter 5 modeled the CS problem as a shortest path problem in an acyclic digraph and proposed an exact iterative beam search (IBS) algorithm in order to solve it. Existing lower bounds for CS were improved and applied by IBS as well as different utilization rates. Experiments on instances from the CSPLib revealed a superior performance of IBS compared to the currently best-known exact solution approach, a scattered branch&bound algorithm. Additionally, the optimality of three currently best-known solutions of instances from the CSPLib was proven.

Finally, Chapter 6 presented a fitness landscape analysis of different CS instances from the CSPLib, determining the resulting autocorrelation as well as fitness-distance correlation, when different neighborhood operators are applied. The analysis showed a smooth landscape for three out of four neighborhood operators in terms of the normalized correlation length. The fitness-distance correlation revealed a high correlation for the adjacency distance metric and suggests the presence of a big valley structure. The findings are used to design two metaheuristics for CS, a variable neighborhood search (VNS) and a memetic algorithm (MA). The probabilities of selecting a neighborhood in VNS

were chosen in accordance to the results of the autocorrelation analysis and an analysis of redundant solutions. The MA algorithm is applied with two already existing crossover operators and one newly introduced heuristic crossover which respects the adjacency distance metric and incorporates problem-specific knowledge. Again, the performance of the proposed algorithms was evaluated on instances from the CSPLib, where VNS showed a superior performance in terms of solution quality and runtime. It improved currently best known solutions of two instances. Among the MAs, it was found that respecting the adjacency distance during crossover leads to a superior solution quality.

7.2 Conclusions

This section summarizes the main contributions of this thesis.

Analysis of CS' solution quality. CS originally stems from applications in the automobile industry, where it is used up to now since sequencing rules are considered more intuitive to human decision makers than a detailed time schedule as it is used by MMS. Although research widely adopted the CS approach, it is mainly concerned with the development of new solution methods. For the first time, this thesis quantified in comprehensive computational studies the resulting gap in the solution quality accompanied with using CS and its surrogate objective function instead of MMS. The evaluation covered CS as constraint satisfaction problem as well as combinatorial optimization problem. For the constraint satisfaction problem, it was found, that CS is unable to identify a large fraction of actual feasible solutions (that contain no work overload) if applied with sequencing rules created by Bolat and Yano (1992a). Hence, the applied rules were found to be too restrictive and lead to a reduced solution space for CS compared to MMS. The fraction of unidentified feasible solutions even increased with increasing sequence length and number of stations of the line. CS using the rules of MSR operates on the same space of feasible solutions as MMS, thus, the solution quality of both approaches can be considered as equal. If CS is regarded as optimization problem and applied with sequencing rules by MSR, it leads to solutions that contain on average at least 15% more work overload than solutions of the corresponding MMS problem on test instances created according to Scholl (1999). Hereby, the objective function by Bolat and Yano (1992a) was found to be most appropriate for CS compared to alternative functions proposed in the literature. Furthermore, a linear correlation analysis using Pearson's product-moment coefficient of both, MMS' and CS' objective values, revealed that both objectives are positive linearly related. However, when CS is applied with different sequencing rules and/or objective functions, like the rules by Bolat and Yano (1992a) and the sliding-window objective function, the gap in the solution quality compared to MMS can even increase up to 75%. The solution time of CS depends on the

number of rules per option applied and on the objective function. CS with one sequencing rule for each option can solve instances by a factor of 1.5-2 faster than MMS. However, experiments also revealed that MMS always leads to a better solution than CS if the available time for the search is fixed. Taking the results of the computational studies in Chapters 2 and 3 as well as the advantages and disadvantages of CS into account, this work concludes that MMS is more preferable than CS for sequencing mixed-model assembly lines. If CS is nonetheless applied, we encourage to use sequencing rules created by MSR and the objective function by Bolat and Yano.

A new sequencing rule generation approach. Although CS' ability to reach the underlying goal of finding a sequence with minimum work overload largely depends on the design of adequate sequencing rules (Bolat and Yano, 1992a), the literature on CS almost always takes sequencing rules as given and gives no advice on how to generate them. With the exception of Bolat and Yano (1992a), one can merely find intuitive examples or rules of thumb for creating sequencing rules which lack mathematical foundation. Bolat and Yano present the only available analytical approach, where sequencing rules are derived by considering operational characteristics of the line affecting the occurence of work overload. However, in Chapter 2 it was shown that the hereby generated rules are very restrictive and impede CS' search process as they reduce the space of actual feasible solutions in CS. Therefore, MSR as a new sequencing rule generation approach was proposed, which also takes characteristics of the assembly line into account and creates more than one sequencing rule per option. Comprehensive experiments revealed the superiority of sequencing rules created by MSR. Considering both sequencing approaches as constraint satisfaction problems, CS now operates on the same space of feasible solutions as MMS. Likewise for the optimization version of both problems, the solution quality of CS is considerably increased when MSR sequencing rules are applied instead of rules by Bolat and Yano. Since the solution time of CS increases with a higher number of rules per option, the overall number of sequencing rules was reduced by developing a notion on strictness in order to identify and remove redundant rules created by MSR. Furthermore, it was shown that applying only the first rule created by MSR for each option still leads to a far better solution quality of CS as using sequencing rules by Bolat and Yano.

Incorporating options with more than two processing times into CS. CS aggregates from the underlying sequencing problem by considering only binary options. Thus, it implicitly assumes that each option can have merely two processing times at the respective station of the line. Variants containing the option require a different processing in the station than variants without the option. This work proposed a first approach to include multiple processing times into CS. In practice, multiple processing times per station occur when options have more than two configurations, e.g., navigation sys-

tems in cars, or more than one option is assembled at the same station. The approach allocates all processing times at a station into two groups according to the cycle time and determines (either with the Bolat and Yano or MSR approach) a sequencing rule based on minimum, average or maximum processing values of each group.

A weighting factor for CS' objective functions. The currently available objective functions for CS in the literature either neglect that violations of different options may lead to a different amount of work overload or state penalties in order to distinguish between option violations without giving advice on how to derive them (Parrello et al., 1986; Warwick and Tsang, 1995; Smith et al., 1996). Therefore, this thesis presented a weighting factor which considers the amount of work overload induced by a violation of the respective option and which can be incorporated into any available objective function for CS. The weighting factor reflects the maximum induced work overload by one violation of an option. The conducted experimental study revealed that the weighting factor increases CS' solution quality without affecting its runtime. Thus, the weighting factor should be considered in future applications of CS.

Application of CS to the resequencing problem. In practice, the minimization of sequencing rule violations is not only considered for the sequencing problem in mixed-model assembly lines but also in resequencing scenarios. Resequencing faces additional contraints as a given sequence is to be optimized using available buffers. Therefore, this work provided the adaption of the CS problem to a resequencing problem using pull-off tables. Pull-off tables are direct accessible buffers and can be found in the automotive industry. The resulting resequencing approach was called car resequencing (CRS) problem. CRS is formalized and applied in a real-world example of a major German truck manufacturer.

Design of efficient exact and metaheuristic solution methods for CS. Finally, this work presented an exact IBS algorithm based on an improved lower bound and a new graph representation of CS, which outperforms the currently best known exact SB&B algorithm for CS on widely applied problem instances of the CSPLib. IBS optimally solved instances of the benchmark set with up to 40 sequencing slots and 5 options and even found feasible sequences (if existent) for instances with 100-400 production slots and 5 options in reasonable time. Alongside providing an evolutionary MA approach, this thesis proposed a VNS metaheuristic using appropriate neighborhoods identified in a fitness landscape analysis. VNS found best known solutions on all instances of the frequently used CSPLib benchmark set and even improved solutions for some of them. According to experimental results, both IBS and VNS are currently the state-of-the-art exact and heuristic solution approaches for CS, respectively.

Bibliography

Bautista, J., Pereira, J., and Adenso-Diaz, B. (2008). A beam search approach for the optimization version of the car sequencing problem. *Annals of Operations Research*, 159:233–244.

Baybars, I. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 32(8):909–932.

Becker, C. and Scholl (2006). A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168(3):694–715.

Benoist, T. (2008). Soft car sequencing with colors: Lower bounds and optimality proofs. *European Journal of Operational Research*, 191(3):957–971.

Blum, C. (2005). Beam-ACO - Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591.

Boese, K. D., Kanhg, A. B., and Muddu, S. (1994). A new adaptive multistart technique for combinatorial global optimizations. *Operations Research Letters*, 16(2):101–113.

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

Bolat, A. and Yano, C. (1992a). A surrogate objective for utility work in paced assembly lines. *Production Planning & Control*, 3(4):406–412.

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

Boysen, N. (2005). *Variantenfließfertigung*. Gabler, Wiesbaden.

Boysen, N., Fliedner, M., and Scholl, A. (2009). Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373.

Boysen, N., Fliedner, M., and Scholl, A. (2010a). Level scheduling under limited resequencing flexibility. *Flexible Services and Manufacturing Journal*, 22(3-4):236–257.

Boysen, N., Kiel, M., and Scholl, A. (2010b). Sequencing mixed-model assembly lines to minimise the number of work overload situations. *International Journal of Production Research*, (to appear).

Burke, E. and Landa Silva, J. (2005). The design of memetic algorithms for scheduling and timetabling problems. In Hart, W., Smith, J., and Krasnogor, N., editors, *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*, pages 289–311. Springer, Berlin Heidelberg.

Butaru, M. and Habbas, Z. (2005). Solving the car-sequencing problem as a non-binary CSP. In *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *LNCS*, page 840, Berlin Heidelberg. Springer.

Cano-Belmán, J., Ríos-Mercado, R. Z., and Bautista, J. (2010). A scatter search based hyper-heuristic for sequencing a mixed-model assembly line. *Journal of Heuristics*, 16(6):749–770.

Chao, M. T., Fu, J. C., and Koutras, M. V. (1995). Survey of reliability studies of consecutive-k-out-of-n:f and related systems. *IEEE Transactions on Reliability*, 44(1):120 – 127.

Choi, S.-S. and Moon, B.-R. (2008). Normalization for genetic algorithms with nonsynonymously redundant encodings. *IEEE Transactions on Evolutionary Computation*, 12(5):604–616.

Choi, W. and Shin, H. (1997). A real-time sequence control system for the level production of the automobile assembly line. *Computers & Industrial Engineering*, 33(3-4):769–772.

Cordeau, J.-F., Laporte, G., and Pasin, F. (2008). Iterated tabu search for the car sequencing problem. *European Journal of Operational Research*, 191(3):945–956.

Cotta, C. and Fernandez, A. (2007). Memetic algorithms in planning, scheduling, and timetabling. In Dahal, K., Tan, K., and Cowling, P., editors, *Evolutionary Scheduling*, volume 49 of *Studies in Computational Intelligence*, pages 1–30. Springer, Berlin Heidelberg.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th international joint conference on Artificial intelligence*,

volume 1, pages 162–164, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Dichtl, E., Raffée, H., Beeskow, W., and Köglmayer, H. G. (1983). Faktisches Bestellverhalten als Grundlage einer optimalen Ausstattungspolitik bei Pkw-Modellen. *Zeitschrift für betriebswirtschaftliche Forschung*, 35:173–196.

Ding, F. Y. and Sun, H. (2004). Sequence alteration and restoration related to sequenced parts delivery on an automobile mixed-model assembly line with multiple departments. *International Journal of Production Research*, 42(8):1525–1543.

Drexl, A. and Jordan, C. (1995). Materialflussorientierte Produktionssteuerung bei Variantenfließfertigung. *Zeitschrift für betriebswirtschaftliche Forschung*, 47:1073–1087.

Drexl, A. and Kimms, A. (2001). Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints. *Management Science*, 47(3):480–491.

Drexl, A., Kimms, A., and Matthießen, L. (2006). Algorithms for the car sequencing and the level scheduling problem. *Journal of Scheduling*, 9(2):153–176.

Epping, T., Hochstättler, W., and Oertel, P. (2004). Complexity results on a paint shop problem. *Discrete Applied Mathematics*, 136(2-3):217–226.

Estellon, B., Gardi, F., and Nouioua, K. (2008). Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research*, 191(3):928–944.

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

Gagne, C., Gravel, M., and Price, W. (2006). Solving real car sequencing problems with ant colony optimization. *European Journal of Operational Research*, 174(3):1427–1448.

Gent, I. P. (1998). Two results on car-sequencing problems. APES research report 02-1998, Department of Computer Science, University of Strathclyde, Glasgow.

Glover, F. (1989). Tabu search - Part I. *ORSA Journal on Computing*, 1(3):190–206.

Gottlieb, J., Puchta, M., and Solnon, C. (2003). A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In Cagnoni, S., Johnson, C., Cardalda, J., Marchiori, E., Corne, D., Meyer, J.-A., Gottlieb, J., Middendorf, M., Guillot, A., Raidl, G., and Hart, E., editors, *EvoWorkshops 2003*, volume 2611 of *LNCS*, pages 246–257, Berlin Heidelberg. Springer.

Gravel, M., Gagne, C., and Price, W. L. (2005). Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56(11):1287–1295.

Gusikhin, O., Caprihan, R., and Stecke, K. (2008). Least in-sequence probability heuristic for mixed-volume production lines. *International Journal of Production Research*, 46(3):647–673.

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hindi, K. S. and Ploszajski, G. (1994). Formulation and solution of a selection and sequencing problem in car manufacture. *Computers and Industrial Engineering*, 26(1):203–211.

Hoos, H. and Stützle, T. (2005). *Stochastic local search*. Elsevier, Amsterdam.

Inman, R. (2003). ASRS sizing for recreating automotive assembly sequences. *International Journal of Production Research*, 41(5):847–863.

Inman, R. R. and Schmeling, D. M. (2003). Algorithm for agile assembling-to-order in the automotive industry. *International Journal of Production Research*, 41(16):3831–3848.

Jaszkiewicz, A., Kominek, P., and Kubiak, M. (2004). Adaptation of the genetic local search algorithm to a car sequencing problem. In *7th National Conference on Evolutionary Algorithms and Global Optimization*, pages 67–74, Kazimierz Dolny, Poland.

Jones, T. and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192.

Kirkpatrick, S., Gelatt Jr., C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kis, T. (2004). On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335.

Klein, R. and Scholl, A. (1999). Scattered branch and bound: An adaptive search strategy applied to resource-constrained project scheduling. *Central European Journal of Operations Research*, 7:177–201.

Krasnogor, N. and Smith, J. (2005). A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions On Evolutionary Computation*, 9(5):474–488.

Kuhn, H. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–87.

Lahmar, M. and Benjaafar, S. (2007). Sequencing with limited flexibility. *IIE Transactions*, 39(10):937–955.

Lahmar, M., Ergan, H., and Benjaafar, S. (2003). Resequencing and feature assignment on an automated assembly line. *IEEE Transactions on Robotics and Automation*, 19(1):89–102.

Lim, A. and Xu, Z. (2009). Searching optimal resequencing and feature assignment on an automated assembly line. *Journal of the Operational Research Society*, 60(3):361–371.

Lowerre, B. (1976). *The harpy speech recognition system*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA.

Merz, P. and Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions On Evolutionary Computation*, 4(4):337–352.

Meyr, H. (2004). Supply chain planning in the German automotive industry. *OR Spectrum*, 26(4):447–470.

Miltenburg, J. (1989). A brief survey of just-in-time sequencing for mixed-model systems. *Management Science*, 35(2):192–207.

Mladenovic, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100.

Monden, Y. (1998). *Toyota production system : An integrated approach to just-in-time*. Engineering & Management Press, Norcross, 3rd edition.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. C3p report, Caltech Concurrent Computation Program.

Moscato, P. and Cotta, C. (2003). A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 105–144. Springer.

Ow, P. S. and Morton, T. (1988). Filtered beam search in scheduling. *International Journal of Production Research*, 26(1):35–62.

Parrello, B. D., Kabat, W. C., and Wos, L. (1986). Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2(1):1–42.

Perron, L. and Shaw, P. (2004). Combining forces to solve the car sequencing problem. In Regin, J.-C. and Rueher, M., editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 3011 of *LNCS*, pages 225–239, Berlin Heidelberg. Springer.

Pine, B. J. (1993). *Mass customization: The new frontier in business competition*. Harvard Business School Press, Boston.

Prandtstetter, M. and Raidl, G. (2008). 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.

Puchta, M. and Gottlieb, J. (2002). Solving car sequencing problems by local optimization. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, volume 2279 of *LNCS*, pages 132–142, Berlin Heidelberg. Springer.

Radcliffe, N. J. (1991). Forma analysis and random respectful recombination. In *Proceedings of ICGA*, pages 222–229.

Reeves, C. R. (1999). Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490.

Rothlauf, F. (2003). On the locality of representations. In Cantù-Paz, E., editor, *Genetic and Evolutionary Computation - GECCO 2003*, volume 2724 of *LNCS*, pages 1608–1609, Berlin Heidelberg. Springer.

Rothlauf, F. (2006). *Representations for genetic and evolutionary algorithms*. Springer, Berlin Heidelberg, 2nd edition.

Sabuncuoglu, I., Gocgun, Y., and Erel, E. (2008). Backtracking and exchange of information: Methods to enhance a beam search algorithm for assembly line scheduling. *European Journal of Operational Research*, 186(3):915–930.

Schiavinotto, T. and Stützle, T. (2007). A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143–3153.

Schlott, S. (2005). Wahnsinn mit Methode. *Automobil Produktion*, 1/2005:38–42.

Scholl, A. (1999). *Balancing and sequencing of assembly lines*. Physica, Heidelberg, 2nd edition.

Smith, K., Palaniswami, M., and Krishnamoorthy, M. (1996). Traditional heuristic versus Hopfield neural network approaches to a car sequencing problem. *European Journal of Operational Research*, 93(2):300–316.

Solnon, C. (2008). Combining two pheromone structures for solving the car sequencing problem with Ant Colony Optimization. *European Journal of Operational Research*, 191(3):1043–1055.

Solnon, C., Cung, V., Nguyen, A., and Artigues, C. (2008). 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.

Spieckermann, S., Gutenschwager, K., and Voß, S. (2004). A sequential ordering problem in automotive paint shops. *International Journal of Production Research*, 42(9):1865–1878.

Stadler, P. F. (1996). Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20(1):1–45.

Sywerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Terada, J., Vo, H., and Joslin, D. (2006). Combining genetic algorithms with squeaky-wheel optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO*, pages 1329–1336. ACM.

Wang, F. and Lim, A. (2007). A stochastic beam search for the berth allocation problem. *Decision Support Systems*, 42(4):2186–2196.

Warwick, T. and Tsang, E. (1995). Tackling car sequencing problems using a generic genetic algorithm. *Evolutionary Computation*, 3(3):267–298.

Weinberger, E. D. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336.

Weiner, S. A. (1985). Perspectives on automotive manufacturing. In Kleindorfer, P. R., editor, *Management of Productivity and Technology in Manufacturing*, chapter 2, pages 57–72. Plenum Press.

Wester, L. and Kilbridge, M. D. (1964). The assembly line model-mix sequencing problem. In *Proceedings of the Third International Conference on Operations Research*.

Whitley, L. D. and Kauth, J. (1988). GENITOR: A different genetic algorithm. In *Proc. Rocky Mountain Conf. Artificial Intel.*, pages 118–130, Denver, CO.

Zinflou, A., Gagne, C., and Gravel, M. (2007). Crossover operators for the car sequencing problem. In Cotta, C. and van Hemert, J., editors, *Proceedings of the 7th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2007*, volume 4446 of *LNCS*, pages 229–239, Valencia, Spain.

Zinflou, A., Gagne, C., and Gravel, M. (2008). Designing hybrid integrative evolutionary approaches to the car sequencing problem. In *IPDPS 2008. IEEE International Symposium on Parallel and Distributed Processing*.