JOHANNES GUTENBERG UNIVERSITÄT MAINZ

The Car Resequencing Problem

Nils Boysen, Uli Golle, Franz Rothlauf

Working Paper 01/2010
June 2010

Working Papers in Information Systems and Business Administration

Johannes Gutenberg-University Mainz
Department of Information Systems and Business Administration
D-55128 Mainz/Germany
Phone +49 6131 39 22734, Fax +49 6131 39 22185
E-Mail: sekretariat[at]wi.bwl.uni-mainz.de
Internet: `http://wi.bwl.uni-mainz.de`

The Car Resequencing Problem

Nils Boysen[a], Uli Golle[b], Franz Rothlauf[b]

[a]Friedrich-Schiller-Universität Jena, Lehrstuhl für Operations Management,
Carl-Zeiß-Straße 3, D-07743 Jena, Germany, nils.boysen@uni-jena.de

[b]Johannes Gutenberg-Universität Mainz, Lehrstuhl für Wirtschaftsinformatik und BWL,
Jakob-Welder-Weg 9, D-55128 Mainz, Germany, {golle,rothlauf}@uni-mainz.de

June 10, 2010

Abstract

The car sequencing problem is a widespread short-term decision problem, in which sequences of different car models launched down a mixed-model assembly line are to be determined. To avoid work overloads of workforce, car sequencing restricts the maximum occurrence of labor-intensive options, e.g., a sunroof, in a subsequence of a certain length by applying sequencing rules. Existing research invariably assumes that the model sequence can be planned with all degrees of freedom. However, in real-world, the sequence of cars in each department can not be arbitrarily changed but depends on the sequence in previous departments and disturbances like machine breakdowns, rush orders, or material shortages. Therefore, in reality the sequencing problem often turns into a resequencing problem. Here, a given model sequence has to be reshuffled with the help of resequencing buffers (denoted as pull-off tables). This paper formulates the car resequencing problem, where pull-off tables can be used to reshuffle a given initial sequence and rule violations are minimized. The problem is formalized and problem-specific exact and heuristic solution procedures are developed and studied. To speed up search, a lower bound as well as a dominance rule are introduced which both reduce the running time of the solution procedures. Finally, a real-world case study is presented. In comparison to the currently used real-world scheduling approach, the resequencing approach can improve solution quality by on average about 30%.

Keywords: Mixed-model assembly line; Car sequencing; Resequencing

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 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

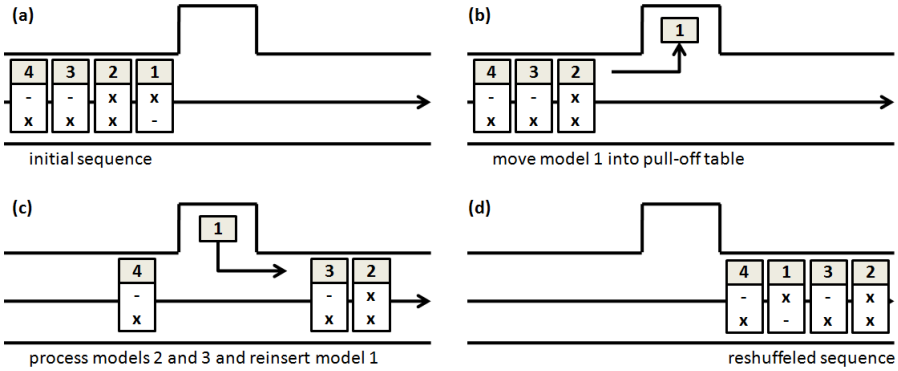


Figure 1: Example on the use of a pull-off table of size one

et al. (1986) and received wide attention both in research and practical application (see 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 can not be arbitrarily changed but depends on the sequence in the previous department. This results into 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 into disordered model sequences.

To be able to change the order of models in a sequence, car manufactures use resequencing buffers. With the help of such buffers, models can be removed from the sequence, stored for a while, and reinserted into the sequence. Buffers can reshuffle a sequence according to a department's individual objectives or reconstruct desired model sequences after disturbances. A common and widespread form of resequencing buffers are off-line buffers, which are also known as pull-off tables (Lahmar et al., 2003). Here, buffers are directly accessible and 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 1 gives an example on 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 1(a) depicts the initial sequence, which would result in one violation of the 1:2-sequencing rule and one of the the 2:3-sequencing rule. The initial sequence can be reshuffled by pulling the model

at position 1 into a single pull-off table (Figure 1(b)). Then, the models at positions 2 and 3 can be processed. After reinserting the model from pull-off table (Figure 1(c)), the rearranged sequence $< 2, 3, 1, 4]$ of Figure 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). 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 treat 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 on how to use pull-off tables to minimize violations of sequencing rules in the resulting sequence. We develop a graph transformation for the problem and present various solution approaches for the problem. In addition, a case study is presented, which demonstrates the advantage of the proposed solution approaches.

The paper is organized as follows. Section 2 models the CRSP as a mathematical program. In Section 3, we develop a graph transformation, which strongly reduces the size of the solution space. With this graph transformation on hand, Section 4 presents different exact and heuristic solution approaches, which are tested in a comprehensive computational study (Section 5). To demonstrate the applicability of the approach, Section 6 presents a real-world case requiring only a few simple modifications of our solution approach. The paper ends with concluding remarks.

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

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
z_{ot}	binary variable: 1, if sequencing rule defined for option o is violated in window starting in cycle t
BI	Big Integer

Table 1: Notation

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 1, we can formulate it as a binary linear program:

$$\text{CRSP: Minimize } Z(X,Y) = \sum_{o=1}^{O}\sum_{t=1}^{T} z_{ot} \tag{1}$$

subject to

$$\sum_{i=1}^{T}\sum_{m=1}^{M} x_{itm} = 1 \quad \forall\, t = 1,\ldots,T \tag{2}$$

$$\sum_{t=1}^{T}\sum_{m=1}^{M} m \cdot x_{itm} = \pi_0(i) \quad \forall\, i = 1,\ldots,T \tag{3}$$

$$\sum_{i=1}^{T}\sum_{\tau=t}^{t+N_o-1}\sum_{m=1}^{M} x_{i\tau m}\cdot a_{om} - (1 - \sum_{i=1}^{T}\sum_{m=1}^{M} a_{om}\cdot x_{itm})\cdot BI \le H_o + BI\cdot z_{ot} \quad \forall\, o = 1,\ldots,O;\ t = 1,\ldots,T-N_o+1 \tag{4}$$

$$x_{itm} = 0 \quad \forall\, m = 1,\ldots,M; i,t = 1,\ldots,T; i-t > P \tag{5}$$

$$x_{itm} \in \{0,\,1\} \quad \forall\, m = 1,\ldots,M; i,t = 1,\ldots,T \tag{6}$$

$$z_{ot} \in \{0,\,1\} \quad \forall\, o = 1,\ldots,O;\ t = 1,\ldots,T \tag{7}$$

For an option o, the binary variable z_{ot} indicates whether the sequencing rule $H_o : N_o$ is violated in the window starting at cycle t. The objective function (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 (2) and (3) enforce that at each cycle t exactly one model is processed and each car of the initial sequence π_0 is assigned to a cycle. (4) 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. However, our model can easily be adapted to other approaches like the sliding-window technique (Gravel et al., 2005). (5) ensures that there is a maximum of P pull-off table and, therefore, a model at position i in the initial sequence can not be shifted to an earlier sequence position than $i - P$.

Kis (2004) showed that the CSP is NP-hard in the strong sense. Since for $P \geq T - 1$ the CRSP is equivalent to the CSP, CRSP is also NP-hard in the strong sense.

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 $\pi_0(i+1) \dots \pi_0(i+P)$). 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. 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 modelled 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 $f : E \rightarrow \mathbb{N}$.

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 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 1(c). The production of the model at position 1 in 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. At decision point $i = 2$, we have two active sequences $act_2^1 = \{0\}$ and $act_2^2 = \{1, 1\}$. The state before the production of 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, see Figure 1(a)); the (artificial) target node is defined as $[T + 1, \emptyset, ACT^0]$.

Proposition: The number of states in V is at most $O(TOM^P)$.

Proof: Overall there are T decision points 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.

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

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.

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 into 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 1. We start with the start node $[0, \emptyset, \{\{0\}, \{0, 0\}\}]$ (Figure 1 (a)). By pulling model 1 into the pull-off table, we branch into node $[0, \{1\}, \{\{0\}, \{0, 0\}\}]$ (Figure 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 1 which processes model 2. The scheduling decision branches node $[0, \{1\}, \{\{0\}, \{0, 0\}\}]$ (Figure 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 1(c). We reinsert model 1 from the pull-off table and processes it. This operation branches node $[2, \{1\}, \{\{0\}, \{1, 1\}\}]$ (Figure 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 $f : E \rightarrow \mathbb{N}$ 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 can not violate a sequencing rule. Therefore, we assign an arc weight of zero to all transitions of type 1. For the transition of type 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.

4 Search Algorithms for the CRSP

For finding the shortest path in the graph, exact and heuristic search strategies can be used. We propose breadth-first search, beam search, iterative beam search, and an A* search.

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 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 occurences of option o in the first $t \bmod N_o$ 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:

$$LB = \sum_{o=1}^{O} \max \left\{ 0; \sum_{j=i}^{T} a_{o\pi(j)} + \sum_{m \in K} a_{om} - D_{o,T-i} \right\}. \tag{8}$$

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 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: We start with the initial state depicted in Figure 1(a). 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 $[1, \emptyset, \{\{1\}, \{0, 0\}\}]$.

We want to further speed up search by defining dominance rules. Dominance rules allow fathoming of nodes if other nodes, which already have 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 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 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 less or at most the same number of rule violations than ACT_i'. Since $f(s) \leq f(s')$, s leads to a better or equal solution than s'.

(Second part) Deleting option occurrences from a sequence of remaining models (for example, by storing models in pull-off tables) leads to less or at most the same num-

ber 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 less or equal number of rule violations than s'. With $f(s) \leq f(s')$, s results into a better or equal solution than 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 2 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'.

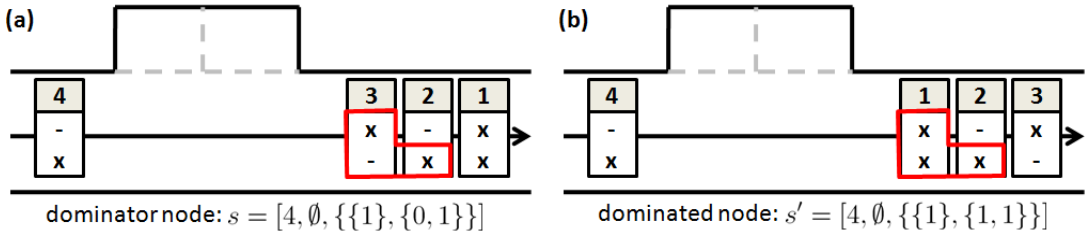


Figure 2: Example for dominance rule

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 on 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 4.1

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 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 (8), 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.

4.3 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 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 A* Search

Unlike BFS or BS, A* search (Hart et al., 1968) does not perform a stage-wise exploration of a decision tree but traverses the tree along the nodes that appear to be most likely on the shortest path from the start to a target node. For each node, we calculate the path weight from the start node to the current node and add a heuristic function h, which estimates the remaining path costs from the current to a target node. When using the lower bound argument (8) for the heuristic function h, A* is optimal, since h is admissible, hence never overestimates the remaining costs to the target node.

A* requires a large number of nodes to be stored during the search process since all nodes that are explored during search 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 4.1 can also be used to reduce the number of stored nodes.

5 Computational Study

We study the performance of the search approaches and the influence of the number of pull-off tables for a set of car sequencing instances.

5.1 Experimental Setup

To apply CRSP on a car sequencing instance, an initial sequence π_0 has to be constructed for each instance which is then resequenced using P pull-off tables. We use the set of

test instances provided by Fliedner and Boysen (2008), which consists of 18 test problems with 10-50 production cycles, 3-7 options, and 5-28 models. All our experiments run on a Pentium 2.5 Ghz processor with 2GB RAM.

5.2 Algorithmic Performance

We compare the performance of the different search algorithms. For each instance of the problem set, 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 all of the ten sequences to 6000 seconds (on average six hundred seconds per sequence) and stop an algorithm after this limit. For BFS, IBS and A*, we use the dominance rule from Section 4. IBS is conducted with four iterations and beam widths $BW_1 = 10$, $BW_2 = 100$, $BW_3 = 1000$, $BW_4 = \infty$. BW_4 returns the optimal solution. For BS, we set $BW = 100$.

Table 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 and the number of options O. For the different search algorithms, we list the average objective value obj, average CPU time consumed, and the average number of explored nodes. BFS, IBS, and A* return the optimal solution if the search does not exceed the time limit. Since the time and nodes needed by BFS to find the optimum solution are worse compared to IBS and A*, we do not consider BFS for further experiments. Comparing A* and IBS, A* is slightly in favor for small problem instances; for large problem instances, IBS shows better performance. Although BS is a heuristic and we have no guarantee to find the optimal solution, the found solutions are close to the optimum, whereas it explores only a fraction of nodes and, thus, requires less CPU time compared to the optimal algorithms.

We study how the performance of A*, BS, and IBS depends on the number of pull-off tables P and the number of production cycles T. Figures 3(a) and 3(b) show the average number of nodes explored and average time consumed over P, respectively. BS performs best since the number of nodes and the time required are low and increase approximately linear. In contrast to BS, the number of nodes and solution time grow about exponentially for A* and IBS.

To study how algorithm performance depends on T, we increase the number of cars of each model by a factor $\lambda \in \{1, \ldots, 4\}$. All other parameters of the problem instances remain unaffected and the number of pull-off tables is set to $P = 3$. Figures 3(c) and 3(d) show the average number of nodes branched and the average time over the average number of production cycles T and λ. T increases linearly with λ. When modelling the CRSP as a graph problem (see Section 3), the number of nodes and the time necessary for the algorithms increases about linear with T.

We study the influence of the beam width BW on the performance of BS. We set $P = 10$ and varied BW between 10 and 200 in steps of 10. Figures 4(a) and 4(b) show the number of explored nodes and the time consumed over BW. Both increase about linearly with BW. Figure 4(c) shows the average absolute deviation $\Delta obj = obj^{BS} - obj^{*}$ between the objective values obj^{BS} produced by BS and the best known objective value obj^{*} of the CSP (see Fliedner and Boysen, 2008). Δobj measures the number of additional rule violations in comparison to the sequence obtained when solving the original CSP. Solution quality of BS slightly increases with larger BW. Since a larger beam width ($BW \geq 100$)

only has a minor effect on solution quality, we see a beam width of 100 sufficient for the studied problem instances.

Problem	T	O	UB	BFS			IBS			BS			A*		
				obj	time	nodes	obj	time	nodes	obj	time	nodes	obj	time	nodes
CAR_3_10	10	3	5.7	1	0.06	1080.8	1	0.04	437	1	0.06	2264	1	<0.01	83
CAR_5_10	10	5	8.3	1.2	0.42	3565	1.2	0.05	379	1.2	0.10	2516	1.2	<0.01	75
CAR_7_10	10	7	11.9	2.4	4.04	9726	2.4	0.05	665	2.4	0.17	2723	2.4	0.02	216
CAR_3_15	15	3	10.2	2.4	0.27	3490	2.4	0.08	885	2.4	0.13	4516	2.4	<0.01	246
CAR_5_15	15	5	15	3.4	6.70	18843	3.4	0.08	1202	3.4	0.27	5028	3.4	0.04	421
CAR_7_15	15	7	21.7	6.6	209.73	86719	6.6	1.25	8391	6.7	0.39	5233	6.6	2.20	4032
CAR_3_20	20	3	14.9	3.8	0.77	6912	3.8	0.14	2882	3.8	0.23	7114	3.8	0.08	1084
CAR_5_20	20	5	20.8	6.1	19.27	39556	6.1	1.03	10212	6.1	0.40	7504	6.1	0.82	3668
CAR_7_20	20	7	25.8	-	>600	-	7.1	8.39	27802	7.3	0.62	7712	7.1	14.66	14812
CAR_3_30	30	3	21.6	6.1	11.64	290462	6.1	0.47	7410	6.3	0.40	11703	6.1	0.23	2517
CAR_5_30	30	5	30.4	8.8	588.01	6509516	8.8	35.03	73278	9.2	0.77	12568	8.8	51.46	43992
CAR_7_30	30	7	45.6	-	>600	-	14.4	309.68	195874	15.3	1.23	12731	-	>600	-
CAR_3_40	40	3	31.7	11.9	31.82	725502	11.9	3.52	31081	12.4	0.62	16405	11.9	2.47	2910
CAR_5_40	40	5	41.9	-	>600	-	13.8	215.42	213772	14.8	1.15	17486	13.8	307.63	143799
CAR_7_40	40	7	57.5	-	>600	-	-	>600	-	21.7	1.80	17702	-	>600	-
CAR_3_50	50	3	40.8	17.2	48.70	1079206	17.2	11.13	64211	17.4	0.80	21102	17.2	8.02	28446
CAR_5_50	50	5	54	-	>600	-	21	577.27	397594	22.5	1.52	22381	-	>600	-
CAR_7_50	50	7	79.4	-	>600	-	-	>600	-	33.2	2.38	22648	-	>600	-

Table 2: Average performances of one run ($P = 4$)

problem	T	O	obj*	BS			
				obj	P'	time	nodes
CAR_3_10	10	3	1	1	4	0.06	2264
CAR_5_10	10	5	1	1	5	0.13	2836
CAR_7_10	10	7	2	2	6	0.26	3345
CAR_3_15	15	3	2	2	6	0.22	6029
CAR_5_15	15	5	2	2	7	0.49	7250
CAR_7_15	15	7	4	4	8	0.91	8115
CAR_3_20	20	3	3	3	7	0.48	10839
CAR_5_20	20	5	3	3	10	1.16	14099
CAR_7_20	20	7	3	3.1	15	3.05	17567
CAR_3_30	30	3	4	4	10	1.31	24141
CAR_5_30	30	5	3	3	15	4.06	33329
CAR_7_30	30	7	4	4.9	19	9.73	38306
CAR_3_40	40	3	5	5	18	3.94	54581
CAR_5_40	40	5	5	5.6	20	9.23	59964
CAR_7_40	40	7	9	7.9*	25	22.29	68294
CAR_3_50	50	3	6	6	18	5.38	73056
CAR_5_50	50	5	8	9.3	21	14.19	83745
CAR_7_50	50	7	12	12.4	30	46.57	104864

Table 3: average best results of BS

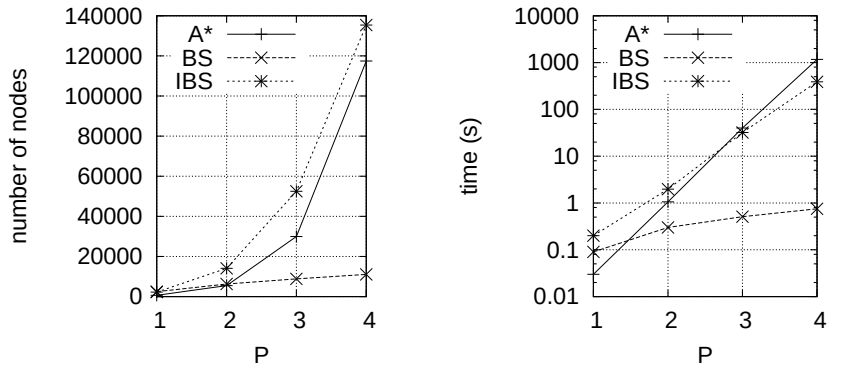
In summary, BS performs well in comparison to the exact search approaches. BS finds solutions close to optimal solutions, but explores considerably less nodes and, therefore, requires less CPU time.

5.3 Resequencing Flexibility

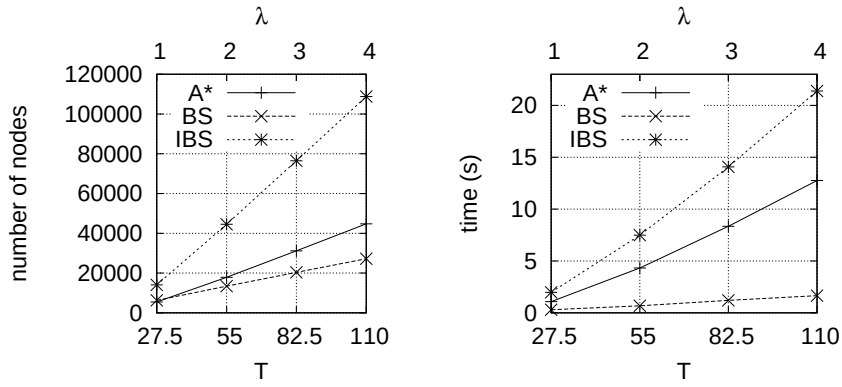
We focus on BS and study how solution quality depends on the number of pull-off tables P. With increasing P, planning flexibility increases, we are less dependent on the initial sequence π_0, and we are able to build better sequences more similar to solutions of the CSP. For our study, we construct ten random initial sequences π_0 per instance and set $BW = 100$. The number P of pull-off tables varies between 1 and 30.

Figure 5 shows the average absolute deviation $\Delta obj = obj^{BS} - obj^*$ between the solution found by BS and the best known objective value of the CSP. 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 less 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 increases solution quality. Adding pull-off tables give us more flexibility and allow us finding better sequences. For example, for $P = 20$, BS finds sequences that violate on average $\Delta obj = 0.21$ 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. The remaining deviation from the optimal solution is low (≈ 0.2 violations) and comes from the heuristic character of BS.

Table 3 lists the best average results found for each instance and the minimum number

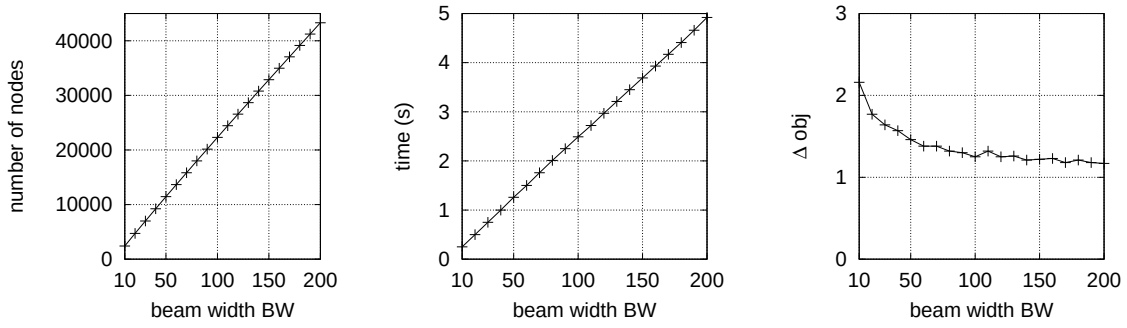


(a) average number of nodes over P (b) average time over P



(c) average number of nodes over T (d) average time over T

Figure 3: Performance of search algorithms over number of pull-off tables P and number of production cycles T



(a) number of nodes over BW (b) time over BW (c) deviation from optimum over BW

Figure 4: Performance of BS over beam width BW

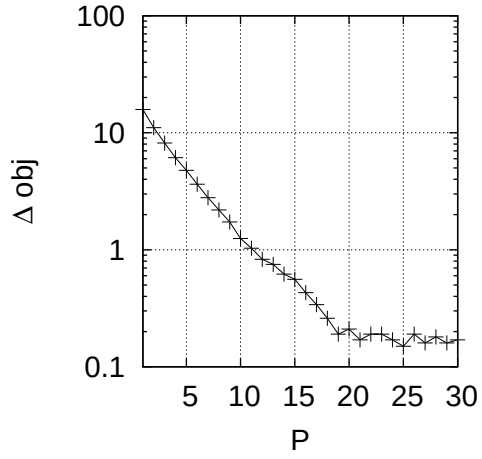


Figure 5: deviation from optimum over P

P' of pull-off tables leading to this result. Note that for instance CAR_7_40 (marked with * in Table 3), we were able to find a new best solution with only seven 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 (see Boysen et al., 2009). However, our results reveal that the number of pull-off tables to be installed heavily depends on the number O of options 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 seems not advisable, since on average over all instances $P' = 0.482T$ leads to (nearly) full resequencing flexibility.

6 A Case Study

Typically, 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 (see Inman, 2003). The situation is different for the production of commercial vehicles like trucks, buses, or construction vehicles, where investment costs for AS/RS are very high and only a few random access buffers are installed. 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 a production cycle $t = 1, \ldots, T$ within original sequence π_{-1} as follows:

$$r_{mt} = \sum_{o=1}^{O} |a_{o\pi_{-1}(t)} - a_{om}| \quad \forall\, m = 1, \ldots, M;\, t = 1, \ldots, T\,. \tag{9}$$

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 graph approach can be used for determining a complete (reshuffled) model sequence, which minimizes re-shuffling effort and avoids sequencing rule violations. Some modifications of our graph approach are required:

- Only those nodes s are feasible that cause no sequencing rule violation ($f(s) = 0$). The lower bound (8) 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

problem instance	decision rule r	BS r	BS improv.
1	78.6	57.6	26.7%
2	52.2	34.4	34.1%
3	78.8	66.4	15.7%
4	134	87.2	34.9%
5	91.6	64.6	29.5%
6	74.8	58.2	22.2%
7	56.8	51.2	9.9%
8	29.4	17.6	40.1%
9	55	24	56.4%
10	63.8	43.2	32.3%
avg	71.5	50.44	29.5%

Table 4: average overall material deviations r

simply 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 4.1) can not be used since it can exclude optimal solutions.

To compare our resequencing approach with the real-world (human) decision rule, we construct ten instances each with 50 production cycles, 20 options, and 15 buffers (pull-off tables). For each instance, we construct an optimal sequence of models which contains no rule violations as initially planned sequence π_{-1}. To simulate rework in the paint shop, this sequence is modified by randomly pulling models out of the sequence (on average 85% of the models) and re-insert them into the sequence on a random position between their original position and the 20 following positions. For every instance, we create ten mixed-up sequences π_0 leading to 100 test problems overall. For resequencing, we apply BS with $BW = 100$ on the graph modified according to the aforementioned suggestions.

For each instance, Table 4 compares the average material deviation r of the solution found by BS with the solution using the real-world decision rule. We also show the improvement (in %). Clearly, BS finds better sequences and considerably reduces the number of material deviations. On average, BS overcomes about 30% of the currently necessary effort for material reshuffling.

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.

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 and develop exact and heuristic solution approaches. To 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 breadth-first search, iterated beam search, A* search, and a beam search heuristic. The computational effort of breadth-first search is higher than iterated beam search or A* search. A* needs less effort for small problem instances than iterated beam search, whereas iterated beam search is faster than A* for larger problem instances. The heuristic beam search approach finds solution very close to the optimal solution 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 20 pull-off tables are sufficient to reach full resequencing flexibility for the considered problem instances.

Finally, we present a real-world case study from a major German truck producer and illustrate how the approach can be applied to such a setting. In comparison to the currently used real-world scheduling approach, our new resequencing approach can improve solution quality by on average about 30%.

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

References

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

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

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

Ding, F., 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, 1525–1543.

Epping, T., Hochstättler, W., Oertel, P., 2004. Complexity results on a paint shop problem. Discrete Applied Mathematics 136, 217–226.

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

Gravel, M., Gagne, C., 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, 1287–1295.

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

Hart, P., Nilsson, N., 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, 847–863.

Inman, R., Schmeling, D., 2003. Algorithm for agile assembling-to-order in the automotive industry. International Journal of Production Research 41, 3831–3848.

Kis, T., 2004. On the complexity of the car sequencing problem. Operations Research Letters 32, 331–335.

Kuhn, H. W., 1955. The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2, 83–87.

Lahmar, M., Benjaafar, S., 2007. Sequencing with limited flexibility. IIE Transactions 39, 937–955.

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

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

Lowerre, B., 1976. The harpy speech recognition system. Ph.D. thesis, Carnegie Mellon University.

Ow, P., Morton, T., 1988. Filtered beam search in scheduling. International Journal of Production Research 26, 35–62.

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

Sabuncuoglu, I., Gocgun, Y., 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, 915–930.

Solnon, C., Cung, V., Nguyen, A., 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., Voß, S., 2004. A sequential ordering problem in automotive paint shops. International Journal of Production Research 42, 1865–1878.

Wang, F., Lim, A., 2007. A stochastic beam search for the berth allocation problem. Decision Support Systems 42, 2186–2196.