BuR – Business Research
Official Open Access Journal of VHB
German Academic Association for Business Research (VHB)
Volume 4 | Issue 2 | December 2011 | 276-292

The Car Resequencing Problem with Pull-Off Tables

Nils Boysen, *Chair of Operations Management, Friedrich Schiller University Jena, Germany, E-Mail: nils.boysen@uni-jena.de*
Uli Golle, *Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Germany, E-Mail: golle@uni-mainz.de*
Franz Rothlauf, *Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Germany, E-Mail: rothlauf@uni-mainz.de*

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.

JEL-Codes: C61, M11, L62

Keywords: mixed-model assembly line, car sequencing, resequencing

Manuscript received October 27, 2010, accepted by Karl Inderfurth (Operations and Information Systems) March 11, 2011.

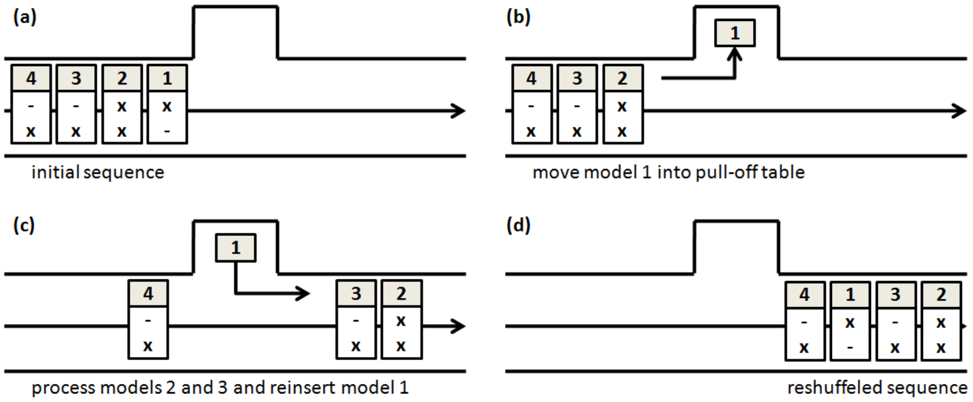
1 Introduction

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 Parello, Kabat, and Wos (1986) and received wide attention both in research and practical application (Solnon, Cung, Nguyen, and Artigues 2008; Boysen, Fliedner, and Scholl 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, Fliedner, and Scholl 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

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



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, Gutenschwager, and Voß 2004) and pull-off tables (Lahmar, Ergan, and Benjaafar 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 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,. . . ,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 2:3-sequencing rule. The initial sequence can be reshuffled by pulling the model at position 1 into the single pull-off table (Figure 1(b)). Then, the models at positions 2 and 3 can be processed. After reinserting the model from the 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, Ergan, and Benjaafar 2003; Epping, Hochstättler, and Oertel 2004; Spieckermann, Gutenschwager, and Voß 2004; Lahmar and Benjaafar 2007; Lim and Xu 2009) or in front of final assembly to level the material demand (Boysen, Fliedner, and Scholl 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, Gutenschwager, and

Voß 2004), or virtual resequencing (Inman and Schmeling 2003; Gusikhin, Caprihan, and Stecke 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 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 resequencing setting requiring only a few simple modifications of our solution approach. The paper ends with concluding remarks.

Table 1: Notation

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, 0 otherwise
BI	Big Integer

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 $m = 1, \dots, M$ requires option $o = 1, \dots, O$. 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 1, we can formulate it as a binary linear program:

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

subject to

$$(2) \quad \sum_{i=1}^{T} \sum_{m=1}^{M} x_{itm} = 1 \quad \forall\, t = 1, \dots, \mathrm{T}$$

$$(3) \quad \sum_{t=1}^{T} \sum_{m=1}^{M} x_{itm} = 1 \quad \forall\, i = 1, \dots, \mathrm{T}$$

$$(4) \quad \sum_{t=1}^{T} \sum_{m=1}^{M} m \cdot x_{itm} = \pi_0(i) \quad \forall\, i = 1, \dots, \mathrm{T}$$

$$(5) \quad \sum_{i=1}^{T} \sum_{\tau=t}^{\min\{t+N_o-1,T\}} \sum_{m=1}^{M} x_{i\tau m} \cdot a_{om}$$
$$-\left(1 - \sum_{i=1}^{T} \sum_{m=1}^{M} a_{om} \cdot x_{itm}\right) \cdot BI$$
$$\leq H_o + BI \cdot y_{ot} \quad \forall\, o = 1, \dots, O;\ t = 1, \dots, \mathrm{T}$$

(6) $x_{itm} = 0 \quad \forall m = 1, \ldots, M;$
$i,t=1, \ldots, T;\ i\text{-}t > P$

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

(8) $y_{ot} \in \{0,1\} \quad \forall o = 1, \ldots, O;\ t = 1, \ldots, T$

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 (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, resp. i, exactly one model is processed, while (4) ensures that each car of the initial sequence π_0 is assigned to a cycle. (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. (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, Gagne, and Price 2005), to distinguish between hard- and soft-sequencing rules (Solnon, Cung, Nguyen, and Artigues 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 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).

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 $f : E \to \mathbb{N}$.

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,..., 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 $\mid K \mid \leq P$) 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). 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 1(a)); the (artificial) target node is defined as $[T+1,\emptyset,ACT^0]$.

Proposition: 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.

$$T \cdot \binom{M+P-1}{P} \cdot O \cdot 2^{\max\{N_o\}-1} + 2$$

$$= T \cdot \frac{(M+P-1)\cdot(M+P-2)\ldots(M+1)\cdot M}{P!} \cdot O \cdot 2^{\max\{N_o\}-1} + 2$$

$$\leq T \cdot \frac{M^P \cdot P!}{P!} \cdot O \cdot 2^{\max\{N_o\}-1} + 2$$

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.

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 ($\mid K \mid < P$), the current model m at cycle i can be stored in a free pull-off table. Note that current model $m = \pi_0(i + \mid K \mid +1)$ 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 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 process 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 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 $k = \mid K \mid$ models are stored in a pull-off table (see Figure 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.

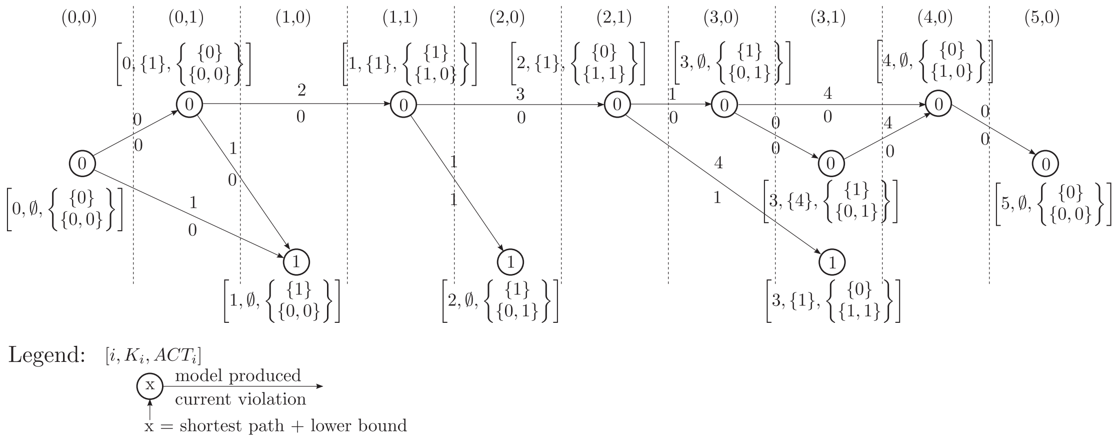
4 Search Algorithms for the CRSP

In section 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.

4.1 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 $k = \mid K \mid$ 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)

Figure 2: Example graph for BFS with *UB* = 1



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:

$$(9) \quad LB = \sum_{o=1}^{O} \max\{\, 0; \sum_{j=i}^{T} a_{o\pi(j)} + \sum_{m \in K} a_{om} - D_{o,T-i} \,\}$$

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: Figure 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 $D_{23} = \lfloor \frac{3}{3} \rfloor \cdot 2 + \min\{2; 3 \mod 3\} = 2$ 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 1: An active sequence ACT_i is *less or equally restrictive* than an active sequence ACT_j, denoted by $ACT_i \preccurlyeq ACT_j$, 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 $K_i \preccurlyeq K_j$, 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 $f(s) \leq f(s')$, $K_i \preccurlyeq K_i'$, and $ACT_i \preccurlyeq ACT_i'$.

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 $ACT_i \preccurlyeq ACT_i'$. Then, we prove that s dominates s', if $f(s) \leq f(s')$, $ACT_i = ACT_i'$, and $K_i \preccurlyeq K_i'$. 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 $ACT_i \preccurlyeq ACT_i'$, 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'.

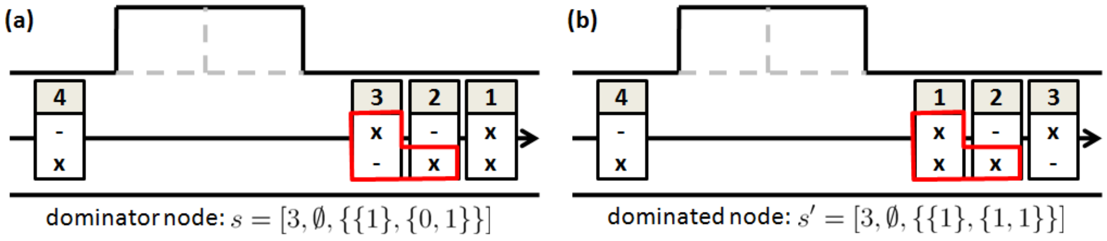
(Second part) Deleting option occurrences from a sequence of remaining models (for example, by storing models in pull-off tables) leads to fewer or at most the same number of rule violations caused by the remaining models that have to be processed. With $K_i \preccurlyeq K_i'$, we can construct for any 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 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'.

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

Figure 3: Example for dominance rule


(a)
4
-
x
3
x
-
2
-
x
1
x
x
dominator node: $s = [3, \emptyset, \{\{1\}, \{0, 1\}\}]$
(b)
4
-
x
1
x
x
2
-
x
3
x
-
dominated node: $s' = [3, \emptyset, \{\{1\}, \{1, 1\}\}]$

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, Gocgun, and Erel (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, Gocgun, and Erel 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 (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.

4.3 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 A* Search

Unlike the aforementioned algorithms, A* search (Hart, Nilsson, and Raphael 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 (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 4.1 can also be used to reduce the number of stored nodes.

5 Computational Study

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 and 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, Gagne, and Price 2005). The experiments run on a Pentium 2.5 Ghz processor with 2GB RAM.

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 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. 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 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(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.

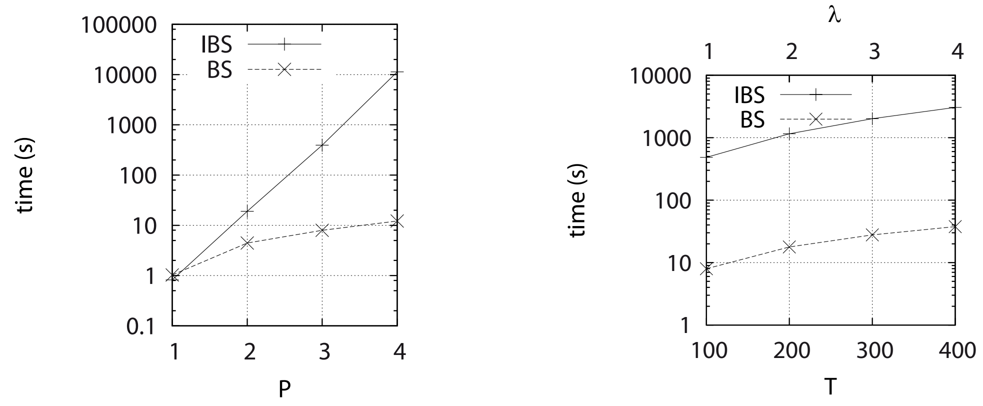
To study how algorithm performance depends on T, we simply duplicate for every instance the demand of each model by a factor $\lambda \in \{1,\dots,4\}$. Thus, for, e.g., $\lambda = 2$ each instance in set two has 200 production cycles. All other parameters of the problem instances remain unaffected and the number of pull-off tables is set to P =3. Figure 4(b) shows the average time over 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 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

Table 2: Average performances over 10 runs (P = 4)

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

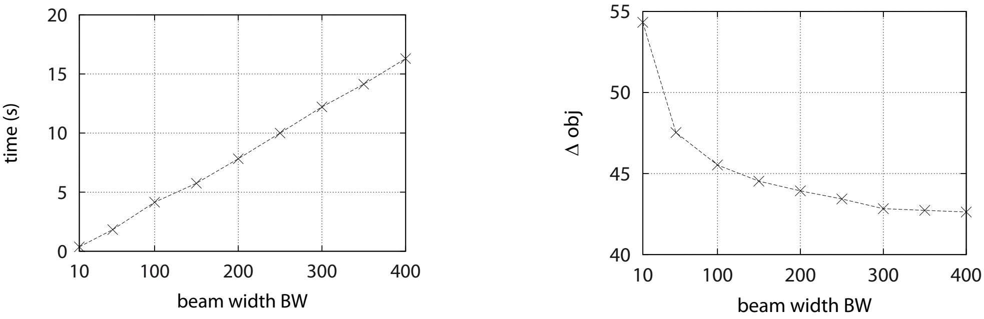
Figure 4: Performance of BS and IBS against *P* and *T*



(a) average time against *P*

(b) average time against *T*

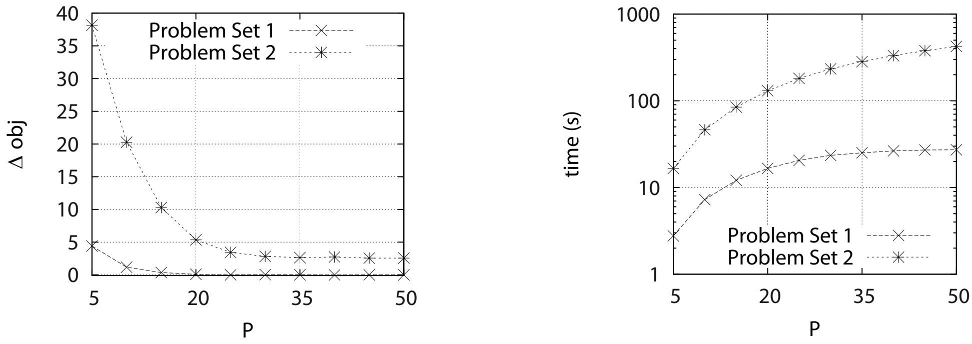
Figure 5: Performance of BS against beam width BW



(a) time against *BW*

(b) deviation from optimum against *BW*

Figure 6: Resequencing flexibility



(a) deviation from optimum against *P*

(b) time against *P*

varied BW between 10 and 400. Figure 5(a) shows the time consumed against BW, which increases approximately linearly with increasing BW. Figure 5(b) shows the average absolute deviation $\Delta obj = obj^{BS} - obj^{*}$ 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. Solution quality of BS slightly increases with larger BW. Since a larger beam width ($BW \geq 300$) only has 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.

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 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 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 Δobj = 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 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 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 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, Fliedner, and Scholl 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' =0.459 T leads to (nearly) full resequencing flexibility.

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 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

Table 3: Average best results of BS

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

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 reshuffling 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,...,T within the original sequence π_{-1} as follows:

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

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 reshuffling effort and avoids sequencing rule violations. The objective function (1) and constraint (5) of our model (see section 2) have to be modified with (11) and (12), resp.:

(11) $$\text{Minimize } Z(X) = \sum_{i=1}^{T} \sum_{m=1}^{M} \sum_{t=1}^{T} x_{itm} \cdot r_{mt}$$

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

(11) minimizes the number of material deviations and (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 (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 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) 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 $N_o = \{2 \ldots 10\}$ and $H_o = \{1 \ldots N_o\}$. 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 reinserting 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 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.

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

Table 4: average overall material deviations *r*

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 %

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, Gagne, and Price 2005), to distinguish between hard- and soft-sequencing rules (Solnon, Cung, Nguyen, and Artigues 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.

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 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 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, Gutenschwager, and Voß 2004). On the other hand, alternative sequencing objectives, like mixed-model sequencing (Boysen, Fliedner, and Scholl 2009) could be modified to cope with limited resequencing flexibility due to a given number of pull-off tables.

References

Blum, Christian (2005): Beam-ACO - Hybridizing Ant colony Optimization with Beam Search: An Application to Open Shop Scheduling, *Computers & Operations Research*, 32 (6): 1565-1591.

Boysen, Nils, Malte Fliedner, and Armin Scholl (2009): Sequencing Mixed-Model Assembly Lines: Survey, Classification and Model Critique, *European Journal of Operational Research,* 192 (2): 349-373.

Boysen, Nils, Malte Fliedner, and Armin Scholl (2010): Level Scheduling under Limited Resequencing Flexibility, *Flexible Services and Manufacturing Journal,* 22 (3-4): 236-257.

Choi, Wonjoon and Hyunoh Shin (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, Fong-Yuen and Hui Sun (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, Andreas and Alf Kimms (2001): Sequencing Jit Mixed-Model Assembly Lines under Station-Load and Part-Usage Constraints, *Management Science,* 47 (3): 480-491.

Epping, Thomas, Winfried Hochstättler, and Peter Oertel (2004): Complexity Results on a Paint Shop Problem, *Discrete Applied Mathematics,* 136 (2-3): 217-226.

Fliedner, Malte and Nil Boysen (2008): Solving the Car Sequencing Problem via Branch Bound, *European Journal of Operational Research,* 191 (3): 1023-1042.

Gravel, Marc, Caroline Gagne, and Wilson Price (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, Oleg, Rahul Caprihan, and Kathryn E. Stecke (2008): Least In-Sequence Probability Heuristic for Mixed-Volume Production Lines, *International Journal of Production Research,* 46 (3): 647-673.

Hart, Peter, Nils Nilsson, and Bertram Raphael (1968): A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions on Systems Science and Cybernetics,* 4 (2): 100-107.

Inman, Robert R. (2003): ASRS Sizing for Recreating Automotive Assembly Sequences, *International Journal of Production Research,* 41 (5): 847-863.

Inman, Robert R. and D. M. Schmeling (2003): Algorithm for Agile Assembling-To-Order in the Automotive Industry, *International Journal of Production Research,* 41 (16): 3831-3848.

Kis, Tamás (2004): On the Complexity of the Car Sequencing Problem, *Operations Research Letters,* 32 (4): 331-335.

Kuhn, Harold W. (1955): The Hungarian Method for the Assignment Problem, *Naval Research Logistics Quarterly,* 2 (1-2): 83-87.

Lahmar, Maher and Saif Benjaafar (2007): Sequencing with Limited Flexibility, *IIE Transactions,* 39 (10): 937-955.

Lahmar, Maher, Hakan Ergan, and Saif Benjaafar (2003): Resequencing and Feature Assignment on an Automated Assembly Line, *IEEE Transactions on Robotics and Automation,* 19 (1): 89-102.

Lim, Andrew and Zhou Xu (2009): Searching Optimal Resequencing and Feature Assignment on an Automated Assembly Line, *Journal of the Operational Research Society,* 60 (3): 361-371.

Lowerre, Bruce T. (1976): The Harpy Speech Recognition System, Ph.D. Thesis, Carnegie Mellon University.

Ow, Peng Si and Thomas E. Morton (1988): Filtered Beam Search in Scheduling, *International Journal of Production Research,* 26 (1): 35-62.

Parrello, Bruce D., Waldo C. Kabat, and Larry Wos (1986): Job-Shop Scheduling Using Automated Reasoning: A Case Study of the Car-Sequencing Problem, *Journal of Automated Reasoning,* 2 (1): 1-42.

Sabuncuoglu, Isan, Yasin Gocgun, and Erdal Erel (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, Christine, Van Dat Cung, Alain Nguyen, and Christian Artigues (2008): The Car Sequencing Problem: Overview of State-of-the-Art Methods and Industrial Case-Study of the ROADEF2005 Challenge Problem, *European Journal of Operational Research,* 191 (3): 912-927.

Spieckermann, Sven, Kai Gutenschwager, and Stefan Voß (2004): A Sequential Ordering Problem in Automotive Paint Shops, *International Journal of Production Research,* 42 (9): 1865-1878.

Wang, Fan and Andrew Lim (2007): A Stochastic Beam Search for the Berth Allocation Problem, *Decision Support Systems,* 42 (4): 2186-2196.

Biographies

Nils Boysen is Professor of Operations Management at the Friedrich-Schiller-University of Jena (Germany). He received a Diploma Degree and a Ph.D. in Business Administration from the University of Hamburg. During the period 2000-2002 he worked for IBM Global Services. His main research interests are production and operations management as well as optimization techniques.

Uli Golle is a research assistant at the Department of Information Systems at the University of Mainz. He graduated in information systems in summer 2007. His research interests include combinatorial optimization problems, especially scheduling problems and evolutionary algorithms.

Franz Rothlauf received a Diploma in Electrical Engineering from the University of Erlangen, a Ph.D. in Business Administration from the University of Bayreuth, a Habilitation from the University of Mannheim, in 1997, 2001, and 2007, respectively. Since 2007, he has had a chair in Information Systems at the University of Mainz. His main research interests are combinatorial optimization problems, optimization algorithms especially evolutionary algorithms, and E-Business.