Computers & Operations Research 40 (2013) 98–108

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A decomposition approach for the car resequencing problem with selectivity banks

Nils Boysen *, Michael Zenker

Friedrich-Schiller-Universität Jena, Lehrstuhl für Operations Management, Carl-Zeiß-Straße 3, D-07743 Jena, Germany

ARTICLE INFO

Available online 7 June 2012

Keywords:
Mixed-model assembly line
Car sequencing
Resequencing
Selectivity bank
Graph search
Ant colony optimization

ABSTRACT

An important decision problem when mass-producing customized product to order is the sequencing problem, which decides on the succession of models launched down a mixed-model assembly line. To avoid work overload of workforce the car sequencing problem restricts the maximum occurrence of labor-intensive options, e.g., a sunroof, in a subsequence of a certain length by applying sequencing rules. In the real-world, frequently perturbations occur stirring up an initially planed sequence, so that a resequencing is required. This paper treats the car resequencing problem where a selectivity bank, which is a special form of buffer organization consisting of parallel line segments without assembly operations, is applied to reshuffle a given initial sequence and rule violations are to be minimized. The problem is formalized and suited heuristic solution procedures are presented and tested. Furthermore, the impact of differently sized mix-banks on resequencing flexibility is investigated.

1. Introduction

Mixed-model assembly lines like they are applied, e.g., in automobile or electronics industry, require the solution of a short-term sequencing problem, which determines the succession of product models launched down the line. A widespread approach for this decision task is the car sequencing problem (CSP), which is based on a set of sequencing rules (see, e.g., [22]). These rules of kind $H_o : N_o$ restrict the occurrence of a labor-intensive option o, e.g., a sun-roof, to at most H_o within any subsequence of N_o successive models and CSP aims at model sequences, which minimize rule violations. Since its first formulation by Parrello et al. [20] the CSP received widespread attention in practical applications and research. A recent review paper of Boysen et al. [3] surveys more than three dozens of papers introducing different solution procedures for CSP and also reviews alternative sequencing approaches for mixed-model assembly lines.

The vast majority of these papers on sequencing mixed-model assembly lines treats initial sequence planning, where a desirable production sequence (with all degrees of freedom) is determined and communicated to part suppliers. However, in real-world applications the resequencing problem, where a given sequence is to be reshuffled with the help of a resequencing buffer, is often equally essential. On the one hand, typically multiple departments, e.g., body-shop, paint-shop, and final assembly in automobile production, having different sequencing objectives participate in production. Then, a resequencing between these departments allows for an individual sequence reshuffled with regard to each shop's individual objective instead of producing one joint and unchanged compromise sequence. On the other hand, disturbances like material shortages, machine breakdowns or workpiece defects mixing up the initially planned sequence might occur. In automobile production, especially the paint-shop is a major source of (unplanned) sequence alterations due to rework of paint defects (see [4]). Again, resequencing buffers can be applied to regain a desirable model sequence.

There exist different forms of organizing resequencing buffers, which all show individual resequencing flexibility (for a detailed survey see [5]):

- *Pull-off tables*: With this type of buffer organization a model can be pulled off-line into a pull-off table, so that succeeding models are brought forward until the model is inserted again into the final sequence at a later sequence position. This way, a model can be shifted to any later position in the sequence, whereas forward shifting is limited by the number of pull-off tables available. Existing research especially treats pull-off tables if applied for paint-batching (see [15–17]). Only recently, Boysen et al. [4] investigated the CSP if pull-off tables are available for resequencing.
- A typical automated storage and retrieval system (*AS*/*RS*) in automobile industry consists of hundreds of buffer places and is located prior to final assembly (see [13]). Each buffer place can individually be accessed, so that a facultative sequence (of those models in buffer) can be generated and resequencing flexibility is only limited by the total number of buffer places.

* Corresponding author.
E-mail addresses: nils.boysen@uni-jena.de (N. Boysen), michael.zenker@uni-jena.de (M. Zenker).

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.05.012

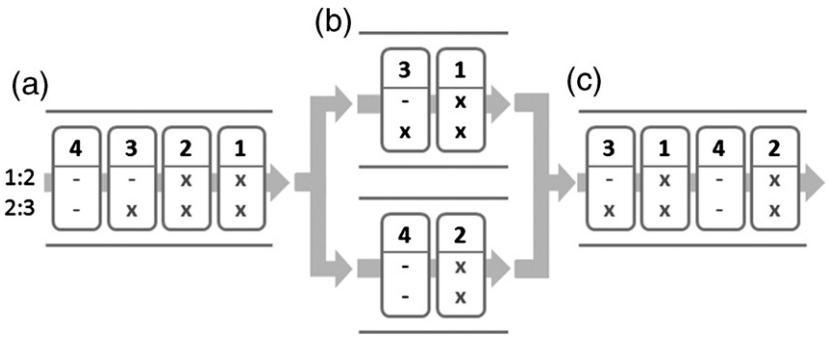

Fig. 1. Example with four models and a mix-bank with two lanes.

- A *selectivity bank* (also denoted as mix bank or parallel line buffer) consists of multiple parallel line segments or lanes (without assembly operations), where car bodies are queued and the first workpiece of each lane can be released into the final sequence. Existing research on this type of buffer organization especially treats paint-batching prior to the paint-shop [6,23].

This paper is the first to couple the resequencing version of car sequencing (CRSP) with selectivity banks. Thus, we aim at a reshuffled model sequence, which minimizes the violations of given sequencing rules, where an initial sequence can be reshuffled by applying a mix bank with a given number of lanes and capacity. An illustrative example for this decision task is given in Fig. 1.

Consider an initial sequence of four models ordered from number $i=1,\ldots,4$ according to their initial sequence position. These models require two options constrained by a 1:2 and a 2:3-sequencing rule, respectively, where "x" and "-" denote whether or not a model requires the respective option. Fig. 1(a) depicts the initial sequence, which would result in two rule violations. For instance, option 1 is required in cycles 1 and 2, which violates the 1:2-rule. This initial sequence can be reshuffled by partitioning models among the two lanes of the mix-bank each having a capacity for two models as is shown in Fig. 1(b). Then, by pulling models out off selectivity bank the final sequence results, which shows no rule violations (Fig. 1(c)).

The remainder of the paper is organized as follows. Section 2 gives a detailed description of the CRSP and presents a mathematical model. Then, a solution approach is presented, which divides the solution process into two steps. First, the fill subproblem allocates models to buffer lanes. Then, the final sequence is determined by releasing models out of the mix bank. For a matter of convenience, we describe both problems in reverted order, so that Section 3 describes different graph approaches for the release subproblem. In Section 4, a priority rule based approach and an ant colony procedure for the solution of the fill problem are presented. A comprehensive computational study in Section 5 tests the computational performance of our solution procedures. Furthermore, by varying the number of lanes the impact of varying resequencing flexibility is investigated, so that the practitioner receives some decision support for dimensioning selectivity banks. Finally, Section 6 concludes the paper.

2. Detailed problem description and mathematical program

Consider a given initial sequence consisting of T different models, which are w.l.o.g. assumed to be numbered according to their initial sequence position: $i=1,\ldots,T$. Each model represents a specific workpiece, which is to be assembled according to a customer specification defining whether or not a specific option $o\in O$ is required. This specification is represented by demand coefficients a_{oi}, which receive a value of one (zero), if option o is (not) required within model i. With regard to these options sequencing rules are defined of kind $H_o:N_o$, which restrict the maximum occurrence of option o in any subsequence of N_o successive models to at most H_o. Typically, an initial sequence causes violations of these rules leading to work overload of the assembly workforce. Thus, a selectivity bank consisting of L parallel lanes ($L>1$) each having a capacity for at most C models can be applied, so that the initial sequence is reshuffled into a final sequence to be fed into the successive line segment (or department). Resequencing flexibility is restricted by the selectivity bank in such a way, that models can be moved into a facultative lane (as long as the lane's capacity is not exceeded), while only the first model of each lane is accessible to be pulled into the final sequence. With these restrictions on hand, CRSP aims at a final sequence minimizing rule violations. Additionally, the following simplifying assumptions are presupposed:

- As is typically given in real-world buffer implementations it is assumed that all parallel lanes show identical capacity to store at most C workpieces. However, it would be easily possible to extend all our solution approaches to integrate lane specific capacities.
- It is assumed that, initially, the mix-bank is empty. This situation is quite unrealistic in real-world implementations as, typically, the CRSP is executed in a rolling horizon. The steady stream of cars is decomposed into multiple smaller CRSPs with in each case only the first x cars being finally released into final sequence. For instance, the major German car manufacturer we supported with our research applies a planning horizon of $T=30$ with only the first car ($x=1$) being finally released. Thus, in a rolling horizon the buffer is partly filled with cars left over from a previous planning run. However, it is easily possible to initialize all our solution procedure with a partly filled buffer, so that for a matter of conciseness we abstain from a detailed description.
- Furthermore, it is assumed that the buffer is large enough to intermediately store the complete initial sequence, so that $T\leq L\cdot C$ holds. This assumption allows for a decomposition of the problem into a separate fill subproblem and a release subproblem. However, with regard to real-world implementations this assumption seems not very restrictive. On the one hand, in the real world, typically, the complete initial sequence is broken down into comparatively small CRSPs (as was described above). On the other hand, typically mix-banks in automobile industry are fairly large. For instance, Choi and Shin [6] report on a mix-bank with $L=25$ buffer lanes each having capacity $C=5$. The buffer utilized by our OEM (original equipment manufacturer) has a dimension of $L=10$ and $C=7$.

According to this problem description and the notation summarized in Table 1 the CRSP can be defined as a mathematical program consisting of objective function (1) and constraints (2) to (8)

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

subject to

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

$$\sum_{t=1}^{T} x_{it} = 1 \quad \forall i = 1,\ldots,T \tag{3}$$

$$\sum_{i=1}^{T} \sum_{\tau=t}^{\min\{t+N_o-1;T\}} x_{i\tau} \cdot a_{oi} - \left(1 - \sum_{i=1}^{T} a_{oi} \cdot x_{it}\right) \cdot BI \le H_o + BI \cdot z_{ot} \quad \forall o \in O;\ t = 1,\ldots,T \tag{4}$$

$$\sum_{l=1}^{L} y_{il} = 1 \quad \forall i = 1,\ldots,T \tag{5}$$

$$\sum_{i=1}^{T} y_{il} \le C \quad \forall l = 1,\ldots,L \tag{6}$$

$$\sum_{t=1}^{T} x_{it} \cdot t - \sum_{t=1}^{T} x_{jt} \cdot t \le BI \cdot (2 - y_{il} - y_{jl}) \quad \forall i = 1,\ldots,T-1;\ j = i+1,\ldots,T;\ l = 1,\ldots,L \tag{7}$$

$$x_{it},\ y_{il},\ z_{ot} \in \{0,1\} \quad \forall i,\ t = 1,\ldots,T;\ o \in O;\ l = 1,\ldots,L \tag{8}$$

Objective function (1) minimizes the number of rule violations summarized over all options $o \in O$ and productions cycles $t = 1,\ldots,T$, where each violation is indicated by binary variable $z_{ot} = 1$. Constraints (2) and (3) ensure that each production cycle of the final sequence receives exactly one model and each model is assigned to exactly one cycle, respectively. The assignment of models to positions (cycles) of the final sequence is represented by binary variables x_{it}, which receive value $x_{it} = 1$, whenever model i is assembled in cycle t (zero otherwise). Whether or not a rule violation occurs is checked by (4). Here, the number of option occurrences, which actually lead to a violation, are counted (see [9]). However, our model can easily be adapted to integrate other approaches as well, like the sliding-window technique (see [12,10]). Eq. (5) ensures that each model i is assigned to exactly one lane l, with the respective assignment of model i to lane l being indicated by binary variable y_{il} receiving value one (zero otherwise). Furthermore, it is to be ensured that each lane's capacity C is not exceeded (constraints (6)). Finally, limited resequencing flexibility according to the mix bank setting is ensured by constraints (7). Clearly, for any two models i and j with $i < j$ it must hold that if assigned to the same lane, i must precede j both in their buffer lane and in the final sequence. Note that parameter BI represents a big integer value, which, e.g., can be chosen as $BI = T$.

Table 1
Notation.

T	Number of production cycles (and models) (indices t (i resp.))
O	Set of options (index o)
L	Number of buffer lanes (index l)
C	Capacity of each lane
a_{oi}	Demand coefficient: 1, if model i requires option o, 0 otherwise
$H_o : N_o$	Sequencing rule: at most H_o out of N_o successively sequenced models may require option o
x_{it}	Binary variable: 1, if model i is assigned to cycle t, 0 otherwise
y_{il}	Binary variable: 1, if model i is buffered in lane l, 0 otherwise
z_{ot}	Binary variable: 1, if a rule violation of option o occurrences exists in window starting in cycle t
BI	Big integer

The CRSP with facultative number of lanes is NP-hard in the strong sense. With $L \ge T$, CRSP is not restricted in its assignment decision of models to cycles and the original CSP arises. CSP was shown to be NP-hard in the strong sense by Kis [14].

In the following a decomposition procedure for solving CRSP is presented. The problem is separated into a fill subproblem, which decides on lane assignment of each model and on each lane's model sequence. Then, the release subproblem decides on how to pull models out of a given buffer arrangement into the final sequence. For both subproblems different solutions procedure are described in the following two sections, where – for a matter of convenience – both problems are described in reverted order.

3. The release subproblem

Given a selectivity bank filled with workpieces, the release subproblem decides on pulling models out of buffer and releasing them into final sequence, so that the number of rule violations is minimized. First, we describe a general graph structure, which transforms the CRSP into a shortest path problem. Then, this graph is applied in different algorithmic settings.

3.1. Graph structure

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

At each decision point it is to be decided among the models being first in lane, which one to release into the final sequence. Thus, starting from an initial buffer setting (as defined by the fill subproblem) any decision point can successively be constructed by a simple pointer structure, where one pointer per lane refers to the currently accessible model. Furthermore it is to be checked whether the current decision violates one of the sequencing rules. Therefore the occurrences of each option o among the last $N_o - 1$ fixed positions have to be known as only these assignments have an influence whether the actual decision leads to a violation or not. Fliedner and Boysen [9] label the last $N_o - 1$ option occurrences of all $o \in O$ options as the "active sequence". Thus, a node $[P,act]$ representing a decision point is exactly defined by pointer vector P defining the next not yet fixed model for each lane, where p_l denotes the pointer of lane l, and active sequence act. Additionally, a unique start node is introduced, which is defined as $[P^*,act^*]$, where P^* and act^* denote the initial pointer vector with $p_l^* = 1 \forall l = 1,\ldots,L$ and the initial active sequence invariably filled with zeros, respectively. Note, that a target node $[P,act]$ of the final stage is reached, whenever $\sum_{l=1}^{L} p_l = T + L$ holds.

Example. Consider our example given in Fig. 1. If models 2 and 4, which were intermediately stored in lane 2, have already been released into the final sequence, the current decision point is represented by node $[\{1,3\};\{\{0\},\{0,1\}\}]$.

Arcs of set E connect adjacent nodes and thus represent a transition between two decision points. Specifically, two nodes $[P,act]$ and $[Q,act']$ are connected by an arc whenever $\sum_{l=1}^{L} p_l + 1 = \sum_{l=1}^{L} q_l$ and $p_l \le q_l \ \forall l = 1,\ldots,L$ holds. Furthermore, with each arc the model finally released into the final sequence (as represented by the respective arc) is to be stored. This is the one model for which $p_l < q_l$ holds. Clearly, whenever identical nodes are

generated, duplicate nodes can be deleted and arcs are to be redirected to the persistent node.

Example. Consider our example given in Fig. 1 and only model 2 being finally released. Then, two arcs are to be inserted pointing from current node [{1,2};{{1},{1,0}}] to successive nodes [{2,2};{{1},{1,1}}] (model 1 is released) and [{1,3};{{0},{0,1}}] (model 4 is released).

Finally, an arc weight $f: E \rightarrow \mathbb{N}$ is assigned with each arc, which stores the contribution of the current sequencing decision to the overall objective value. An arc weight amounts to the number of sequencing rule violations, if a model i is finally fixed in the sequence

$$f = \sum_{o \in O} \Theta\left(\left(\sum_{t=1}^{N_o-1} act_{ot} + a_{oi}\right) \cdot a_{oi} - H_o\right)$$

with

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

being the Heaviside step function and *act* being the active sequence of the source node.

With this graph on hand solving CRSP reduces to finding the shortest path from the unique start node to a target node. Such a graph search can be applied in multiple algorithmic settings, which are described in the following.

3.2. Exhaustive search

In an exhaustive search (ES) setting the complete graph is constructed and all nodes per stage are branched. Note that such a stage-wise graph structure allows for a simultaneous construction and shortest path evaluation, so that no additional shortest path algorithm needs to be applied after constructing the complete graph. Compared to a full enumeration of all possible sequences the computational effort is already considerably reduced, however, the number of nodes per stage raises exponentially in the number L of lanes. Thus, a further speed-up, e.g., obtained by integrating upper and lower bounds, is desirable. ES can be extended by additionally computing node specific lower bounds *LB* on the remaining path length to a target node. Furthermore, a global upper bound *UB* can be determined upfront by some heuristic procedure (e.g., by those described in subsequent sections). Whenever *LB* plus the shortest path to the current node is equal to or exceeds upper bound *UB*, the respective node can be fathomed as it cannot be part of a solution with a better objective value than the incumbent solution. In the following it is shown how to determine a simple lower bound argument for any node.

This bound is based on the relaxation of the limited resequencing flexibility, so that a lower bound of the traditional CSP can be applied. Fliedner and Boysen [9] showed for the CSP that the maximum number of cycles D_{ot} in a sequence of t models, which may contain an option o without violating a given $H_o : N_o$-rule, amounts to: $D_{ot} = \lfloor t/N_o \rfloor \cdot H_o + \min\{H_o; t \bmod N_o\}$. This maximum number of option occurrences can be compared to remaining options not yet scheduled, so that a simple lower bound (LB) for a current node [*P*,*act*] is readily available

$$LB = \sum_{o \in O} \max\left\{0; \sum_{l=1}^{L} \sum_{\tau=p_l}^{T_l} a_{o\delta(l,\tau)} - D_{or}\right\} \quad (9)$$

where T_l ($T_l \leq C \; \forall \, l = 1, \ldots, L$) defines the number of models that are initially queued in lane l, so that $\sum_{l=1}^{L} T_l = T$. To determine the number of options to be scheduled (first term), option occurrences $a_{o\delta(l,\tau)}$ of models waiting in buffer need to be summarized over all lanes $l = 1, \ldots, L$ and not yet scheduled buffer positions $\tau = p_l, \ldots, T_l$, where $\delta(l,\tau)$ denotes the model waiting in lane l on position τ. The maximum number D_{or} of option occurrences allowed (second term) are to be distributed among the $r = \sum_{l=1}^{L} (T_l + 1 - p_l)$ cycles not yet fixed. The additional max-function prevents that negative violations, i.e., excessive production of options, can compensate for violations of another. The bound can be calculated very quickly in $O(|O|)$, which allows for a fast inspection of subproblems.

Example. Given an upper bound of $UB = 1$ the resulting graph for our example is depicted in Fig. 2.

For a further acceleration of the search an additional dominance rule can be applied. A dominance rule allows for a fathoming of a considered node by showing that another subproblem, which already has been inspected, can only possess an equal or better objective value. For specifying such a dominance rule first the following definition is introduced, which is an adoption of the concepts developed by Fliedner and Boysen [9] for the CSP:

Definition. An active sequence *act* of a node is *less or equally restrictive* than the active sequence *act′* of another node, if it holds that $act_{ot} \leq act'_{ot} \forall o \in O; t = 1, \ldots, N_o - 1$.

This definition is applied within the following dominance rule:

Dominance rule: A node $s = [P, act]$ with already fixed violations $f(s)$ (length of shortest path to s) dominates another node $s' = [Q, act']$ with $f(s')$, if it holds that $f(s) \leq f(s')$, $p_l = q_l \forall l = 1, \ldots, L$, and *act* is less or equally restrictive than *act′*. This means a node can be discarded if another node (dominator) exists with less or equal fixed rule violations, equal pointer vector and a less or equally restrictive active sequence.

Example. Consider a mix bank with $L = 3$ and $C = 2$ and an initial sequence of six car models facing two options for which a 1:2 and a 2:3-rule holds, respectively. Fig. 3 depicts two decision points and their respective nodes s and s'. Here, s dominates s', because

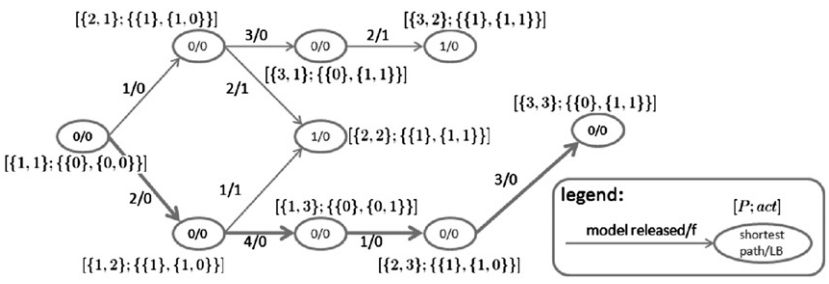

Fig. 2. Resulting graph for the example.

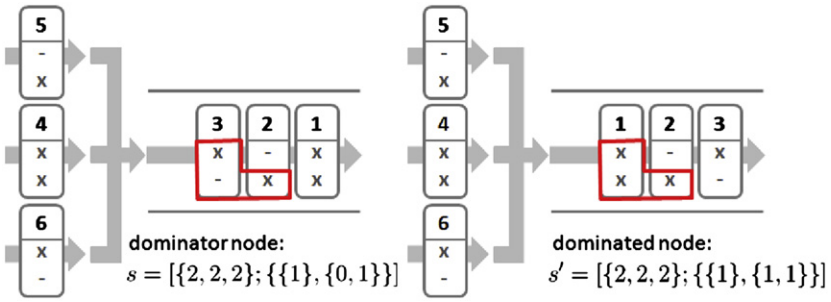

Fig. 3. Example for dominance rule.

of (i) $f(s)=f(s')=0$, (ii) identical models not yet scheduled and (iii) the active sequence of s being less restrictive than that of s'.

3.3. Beam search

Beam search (BS) is a truncated graph search heuristic and was first applied to speech recognition systems by Lowerre [18]. Ow and Morton [19] were the first to systematically study the performance of BS compared to other well-known 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 [25] or hybridization with other meta-heuristics [2], 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. [21].

Like other graph search procedures BS is based on a tree representation of the solution process. However, unlike a breadth-first version of Branch&Bound, BS restricts the number of nodes per stage to be further branched to a promising subset and thus might exclude nodes, which contain optimal solutions. The size of the subset is determined by the beam width *BW* in advance. The nodes to include in the subset are identified by heuristic choice in a filtering process.

Starting with the root node of stage 0, all nodes of stage 1 are constructed. Then, the filtering process of BS starts to identify promising nodes among all nodes of stage 1. The filtering can be obtained by a simple priority value, cost function, etc. Moreover, a multi-stage filtering can be applied, where filtering procedures are ordered according to their degree of detail [21]. The *BW* best nodes found by the filtering form the promising subset of stage 1. Only these nodes are further expanded and branched to build the set of nodes in stage 2. Then again the filtering is applied to identify the promising subset of stage 2. These steps are repeated until the final stage is reached and the best solution out of the final set is returned as the result of BS. Note that BS only constructs the complete graph and therefore equals ES, if *BW* is large enough. Otherwise only a subgraph is explored, so that merely the shortest path within the resulting subgraph is employed. Additionally an upper bound *UB* and the aforementioned dominance rule can be used to further reduce the number of nodes to be branched.

To apply BS in a problem-specific domain two components must be predefined: (i) graph structure and (ii) filtering. In the following, we describe these specifications for CRSP.

Graph structure: The aforementioned acyclic digraph $G(V,E,f)$, as described above, can directly be used for BS.

Filtering: To select *BW* nodes out of the set of branched nodes in each state, we calculate the objective value of the current partial solution and add the lower bound argument of Eq. (9). This value amounts to the sum of arc weights along the shortest path leading to the current node plus the estimated remaining path up to the target node.

Algorithm 1. Iterated beam search (IBS).

```
Input: Ordered list ϒ of increasing beam widths BW;
Initialization: Set UB = ∞;
for i = 1 to length of ϒ do
    Solve Beam Search : Execute BS(UB,BW) with BW = ϒ_i;
    Update upper bound : Set UB = result of BS;
return model sequence;
```

Clearly, the performance of BS profits from a tight upper bound argument *UB*. Thus, instead of executing BS only once (typically, with a comparatively large beam width *BW* and a poor upper bound *UB*) an iterative execution of BS while increasing *BW* and concurrently improving *UB* could increase performance. For the original car sequencing problem such an iterated beam search (IBS) was shown to lead to promising results by Golle et al. [11]. Algorithm 1 defines an IBS scheme for the CRSP. The algorithm is initialized with an ordered list of increasing beam widths *BW*. First, BS is executed with a relative small *BW* for quickly generating an initial *UB*. The result of this initial BS run builds the *UB* for the next iteration of BS executed with a larger *BW* and so on. By successively tightening *UB*, larger parts of the graph are pruned, so that an increasing *BW* does not excessively increase computational times.

4. The fill subproblem

The fill subproblem decides on the assignment of models to lanes and on the model succession per lane. It is to be executed before subsequently the release problem can be started. First, a simple priority rule based procedure is described, which is, then, extended to an ant colony approach.

4.1. A priority rule based start heuristic

In order to compute a first feasible solution, we propose a greedy algorithm based on a dynamic priority rule. The buffer is filled position-by-position starting with the first model of the initial sequence. In each iteration $i=1,\ldots,T$, the lane with highest priority and residual capacity is selected and the current model is queued up in the respective lane. Note that EMP_i defines the set of empty lanes in iteration i. Determining the priority rule is based on the following hierarchy:

1. If there are enough empty lanes, so that any remaining model can receive an exclusive lane ($T-i+1 \leq |EMP_i|$), then choose the lane $l \in EMP_i$ with the lowest index.
2. Select the lane l which has residual capacity and causes the least number of additional rule violations over all options if the

models of the respective lane would be released in direct succession into the final sequence. Clearly, the active sequence act^l of each lane l decides on the number of rule violations caused by a current model i, so that the lane l is chosen which minimizes

$$w(i,l) = \sum_{o \in O} \Theta\left(\left(\sum_{t=1}^{N_o-1} act_{ot}^{l} + a_{oi}\right) \cdot a_{oi} - H_o\right) \tag{10}$$

with $\Theta(x)$ again being the heaviside step function.

3. First-level ties are broken in favor of the lane whose active sequence shows most option occurrences, i.e., the lane l which maximizes $\sum_{t=1}^{N_o-1} act_{ot}^{l}$.
4. Second-level ties are broken in favor of the lane with lower index l.

Example. Consider a selectivity bank, sequencing rules and an initial sequence as defined in Fig. 1. First, our heuristic assigns model 1 to lane 1 because of the lane's lowest index (according to step 4). Then, model 2 would cause a rule violation of the 1:2-rule on lane 1 and is thus assigned to lane 2 (according to step 2). Model 3 causes no rule violation (step 2) in neither lane, both lanes have identical model at the succeeding position (step 3), so that model 3 is queued up in lane 1 because of the lower index (step 4) and, finally, model 4 receives the only remaining buffer place on lane 2, so that exactly the buffer allocation as depicted in Fig. 1 results from our priority rule based procedure.

4.2. An ant colony approach

In an ant colony approach (e.g., see [7]), solutions are constructed repetitively by software agents (artificial ants), which typically base their decisions on some local heuristic measure and a so-called pheromone matrix, which stores the aggregated experiences of former ants. The search process of an individual ant resembles that of our simple priority rule based heuristic, such that at each sequence position i a single lane is chosen out of the set POS_i of possible alternatives (lanes not yet fully occupied). An ant's buffer is hence successively filled. However, the choices of an ant are not deterministic, but stochastic according to a weighted probability scheme which is repetitively calculated at each decision point (sequencing position). In this scheme, each model receives a choice probability proportionally to its *fitness*, which is myopically calculated according to the respective heuristic measure and the pheromone value. The probability $Prob(i,l)$ that model i is assigned to lane l is then determined on the basis of its priority value $w(i,l)$ and the intensity of the pheromone $\tau_{ij(l)}$ with respect to its alternatives, where $j(l)$ denotes the currently last product queued in lane l

$$Prob(i,l) = \begin{cases} 0 & \text{if } T-i+1 \le |EMP_i| \wedge l \notin EMP_i \\ \dfrac{(\tau_{ij(l)})^{\alpha} \cdot \left(\frac{1}{w(i,l)+1}\right)^{\beta}}{\sum_{l' \in POS_i} (\tau_{ij(l')})^{\alpha} \cdot \left(\frac{1}{w(i,l')+1}\right)^{\beta}} & \text{otherwise} \end{cases} \qquad \forall i = 1,\ldots,T;\ l \in POS_i \tag{11}$$

Analogously to our priority rule, an empty lane $l \in EMP_i$ is randomly chosen if each remaining models can receive its individual empty lanes ($T-i+1 \le |EMP_i|$). As priority value $w(i,l)$ we simply apply the second-level priority value as was already utilized for our priority rule based start heuristic. Note that $w(i,l)$ might be zero, so that the division by zero is avoided by adding $\epsilon = 1$. Further note, that additional computational tests indicated that varying parameter ϵ has merely negligible impact on the solution quality. Pheromone value $\tau_{ij(l)}$ is determined between predecessor model $j(l)$ on lane l and current model i. Additionally, a virtual model is defined which is queued in any empty lane, so that pheromone is stored in a $T \times T+1$-matrix. Parameters α and β control the relative importance of the pheromone versus the priority values. Because of experiences with other sequencing problems reported in the literature, these parameters are set to $\alpha = 1$ and $\beta = 2$ (see [24]).

In this way, all ants belonging to the current iteration k construct their individual buffer allocation, which is, then, passed over to the second stage of our decomposition procedure, where the release problem is solved for the respective buffer setting and minimum rule violations are determined. The buffer allocation $\pi(k)$ leading to the least number of rule violations $D^*(\pi(k))$ of current iteration k is applied to update the pheromone trail. The formula incorporates two mechanisms for guiding the search. Older pheromone is constantly reduced (evaporation) which strengthens the influence of more recent solutions and new pheromone is assigned to all model successions, which are part of the solution, in proportion to the respective objective value. The parameter ρ, which is set to 0.5, controls the relative importance of these two components. Thus, the pheromone value $\tau_{ij}(k)$ in iteration k is calculated as follows:

$$\tau_{ij}(k) = \tau_{ij}(k-1) \cdot (1-\rho) + \rho \cdot \begin{cases} \dfrac{1}{D^*(\pi(k))} & \text{if } i \text{ directly follows } j \text{ on a lane in } \pi(k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall i = 1,\ldots,T;\ j = 1,\ldots,T+1. \tag{12}$$

Note that the pheromone matrix has to be initialized with starting values, i.e.,

$$\tau_{ij}(0) = \frac{1}{D^*(\pi_{start})} \forall i = 1,\ldots,T;\ j = 1,\ldots,T,$$

where π_{start} represents a first, randomly drawn buffer allocation and $D^*(x)$ is the minimum rule violation as determined by beam search BS with beam with $BW = 20$. In the current implementation 20 ants are employed to construct solutions in any iteration. After 10 iterations, the algorithm terminates and the best solution found is returned. Note that the aforementioned parameter values have been chosen by thoroughly evaluating different parameter constellations. An excerpt of these tests is presented in Section 5.2.

5. Computational study

The computational study seeks to explore the algorithmic performance of both our graph approaches dedicated to the release subproblem (Section 5.1) and the decomposition approach for the CRSP (Section 5.2). Furthermore, the impact of differently sized selectivity banks and, thus, varying resequencing flexibility is explored in Section 5.3. Note that the solution procedures described above have been implemented in C#.NET and run on an Intel (R) Core(TM) i7 1,73 GHz CPU PC, with 4 GB of memory. When we obtain optimal solutions in order to evaluate our approaches we do so by implementing optimization model CRSP in off-the-shelf solver XPress-Optimizer (version 21.01.06).

Our computational study is based on the well-known testbed for the original CSP provided by Fliedner and Boysen [9]. It contains 18 basic instances where the number of models and options vary from 10 to 50, i.e., $T \in \{10, 15, 20, 30, 40, 50\}$, and three to seven, i.e., $|O| \in \{3, 5, 7\}$, respectively. In any instance rule violations are inevitable, so that for all objective values $Z > 0$ holds. We have adapted these instances by randomly drawing an initial sequence and defining a buffer setting. The latter task is done by randomly drawing the number of lanes L from interval $[2; \lfloor T/2 \rfloor]$ with equal distribution and setting lane capacity to $C = \lceil T/L \rceil$. Per basic instance the generation process is repeated 10 times, so that in total 180 instances were obtained.

5.1. Algorithmic performance for the release subproblem

For evaluating the computational performance of the solution procedures dedicated to the release subproblem, the mix-bank of each instance is initially filled with models according to our priority rule based procedure (see Section 4.1). First, we explore the best configuration of our exhaustive search procedure (ES; see Section 3.2) and its ability to solve the release problem to optimality. Note that due to the exponential increase of the runtime we only investigate the small instances with $T \in \{10, 15, 20\}$. For ES it has to be decided up to which branching level the dominance rule should be applied and whether our bounding strategy shows successful.

With regard to the dominance rule, dominance tests at earlier graph levels seem especially promising. On the one hand, the earlier a node is fathomed the more successive nodes are excluded from the search. On the other hand, the node comparisons required by the dominance rule are quite time consuming, so that at earlier levels where fewer nodes exist a reduced number of operations are required. Fig. 4 relates the average CPU-time (a), and the average number of nodes generated in the search graph (b) to the level up to which the dominance rule is applied. Note that a level of zero represents ES without applying the dominance rule. Further note that at level one no dominance can occur.

The results show a considerable reduction in the total number of nodes generated when increasing the graph level for dominance tests. However, with regard to CPU times this effect is outweighed by the increasing number of node comparisons required. Compared to ES without dominance rule (level zero) there is a (marginal) improvement up to level three and from there on a strong increase of the runtime occurs. It can be concluded that the dominance rule has no considerable influence on the performance of ES and should, thus, be discarded.

A much better effect results from extending ES with upper and lower bounds (see Table 2). Here, the average CPU seconds and the average number of nodes generated are listed for solving the instances to optimality with ES and a varying upper bound quality. Upper bounds are generated with beam search (BS) initialized with varying beam widths as listed in column (*BW*). Note that if BS is applied with a beam width of zero no node is branched, so that ES receives $UB = \infty$ as its upper bound. With an increasing beam width, larger parts of the total graph are explored, so that the solution quality increases and upper bounds become tighter. Clearly, there is a trade-off between investing more time into the initial BS for tightening the upper bound and the number of nodes fathomed during the successive ES when comparing upper and lower bounds. In Table 2 minimum runtime and the minimum numbers of nodes generated per instance group are highlighted in bold. With an increasing instance size (as indicated by the number of models T) an increasing effort on tight upper bounds should be spent. For instance, for $T=20$ the average runtime reduces from a maximum ($BW=1$) of 68 CPU seconds to merely 2.4 CPU seconds when initializing BS with a much tighter upper bound gained with $BW=1000$.

Due to the exponential increase of the runtime and the huge memory requirements larger test instances need to be solved heuristically. In this context the impact of an increasing beam width *BW* on the performance of beam search (BS) is depicted in Fig. 5. The results show that the positive impact of increasing *BW*, which leads to larger parts of the solution graph being explored, on the solution quality quickly diminishes (a). With a beam width of $BW=2000$ the average absolute (relative) deviation from our benchmark is only 0.22 rule violations (2%). Fig. 5(b) depicts the linear increase of runtime when increasing *BW*.

When comparing the solution performance of beam search BS and iterative beam search IBS in different parameter settings (see Table 3), we differentiate between small ($T \in \{10, 15, 20\}$) and large ($T \in \{30, 40, 50\}$) instances. While the small instances are solved to optimality with ES, for the large instances optimal solutions remain unknown. Thus, performance measure *sol** in Table 3 indicates the number of optimal (best) solutions found by

Table 2
Performance of ES with beam search (BS) and varying beam widths (*BW*) applied for determining upper bounds.

BW	$T=10$		$T=15$		$T=20$	
	Time (s)	Nodes	Time (s)	Nodes	Time (s)	Nodes
0	0.057	6165	1.558	144,381	66.108	5,912,486
1	0.023	2065	1.095	82,992	68.032	3,970,853
5	**0.01**	974	0.688	50,397	34.837	2,341,838
20	0.011	710	0.343	25,363	12.642	888,089
50	0.012	383	0.27	19,740	6.158	460,437
100	0.178	307	0.273	18,001	4.557	350,182
200	0.03	**279**	**0.269**	15,209	2.527	189,541
500	0.044	**279**	0.29	**9906**	2.514	180,679
1000	0.063	**279**	0.416	**9906**	**2.368**	**129,468**
2000	0.073	**279**	0.626	**9906**	3.039	**129,468**
5000	0.077	**279**	1.06	**9906**	4.831	**129,468**

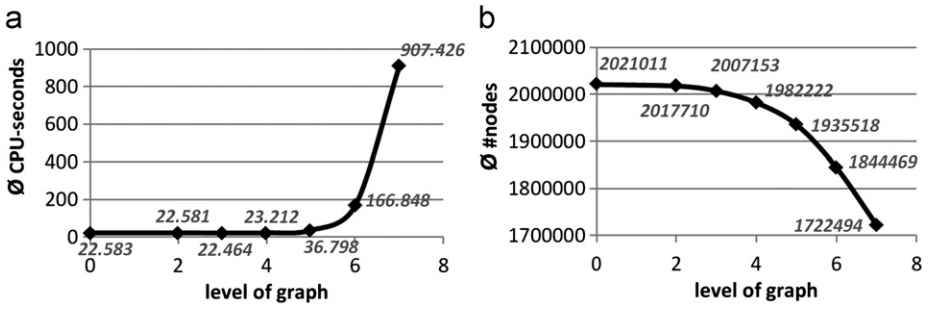

Fig. 4. Effect of the graph level up to which the dominance rule is applied on the performance of ES.

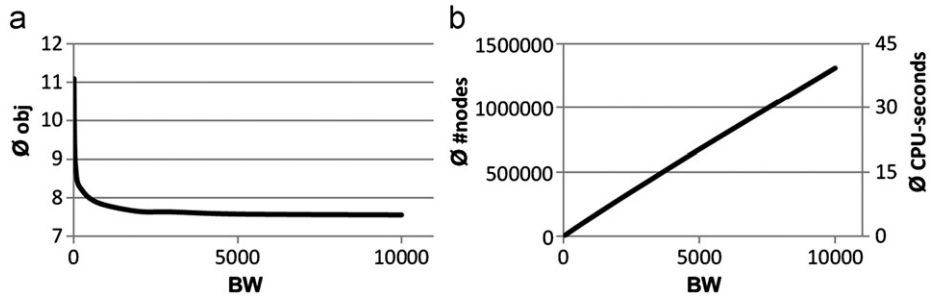

Fig. 5. Impact of beam width *BW* on the performance of beam search BS for all instances.

Table 3
Comparison between BS and IBS with different parameters.

		BS_1 ($BW=5$)	BS_2 ($BW=1000$)	BS_3 ($BW=2000$)	BS_4 ($BW=5000$)	IBS_1 ($BW \in \{1,5,10,2000\}$)	IBS_2 ($BW \in \{5,20,2000\}$)	IBS_3 ($BW \in \{5,25,100,200,2000\}$)
Small inst.	*sol**	27	90	90	90	90	90	90
	gap^{abs}_{avg}	1.71	0	0	0	0	0	0
	gap^{abs}_{max}	9	0	0	0	0	0	0
	gap^{rel}_{avg}	0.42	0	0	0	0	0	0
	gap^{rel}_{max}	2	0	0	0	0	0	0
	time (s)	<0.01	0.39	0.7	1.47	0.47	0.44	0.43
Large inst.	*sol**	5	48	56	69	56	56	56
	gap^{abs}_{avg}	5.63	0.76	0.44	0.31	0.44	0.44	0.44
	gap^{abs}_{max}	17	4	3	3	3	3	3
	gap^{rel}_{avg}	0.6	0.08	0.04	0.03	0.04	0.04	0.04
	gap^{rel}_{max}	2	0.75	0.25	0.25	0.25	0.25	0.25
	time (s)	0.05	7.8	15.6	38.75	15.09	15.5	17.83

the respective solution approach for the small (large) instances. Furthermore, we report the average (maximum) absolute gap gap^{abs}_{avg} (gap^{abs}_{max}), and the average (maximum) relative gap gap^{rel}_{avg} (gap^{rel}_{max}), where optimal and best solutions are applied as a benchmark for the small and large instances, respectively. It turns out that all heuristic procedures (except for BS_1 with $BW=5$) solve all small instances to optimality while requiring less than a second solution time (except for BS_4 with $BW=5000$ requiring less than 1.5 s). Among the large instances BS_4 with $BW=5000$ leads to the best solution quality and produces an average absolute gap of only 0.31 rule violations. Note that the comparatively high relative deviations are caused by objective values with just a very few rule violations, so that only one additional rule violation leads to a considerable gap. Among those solutions procedures with a maximum beam width of $BW=2000$ it is interesting to see that IBS slightly outperforms BS. While all procedures (BS_3, IBS_1, IBS_2, and IBS_3) deliver the same solution quality, IBS_1 with $BW \in \{1,5,10,2000\}$ requires only 15.09 CPU seconds on average over all large instances. Thus, the reduction of the graph size by stepwise increasing the bound quality justifies the additional effort of multiple BS iterations.

5.2. Algorithmic performance for the CRSP

This section investigates the solution quality of our decomposition approach when solving the CRSP consisting of both the fill and the release subproblem. For solving the fill problem we compare our priority rule based procedure (PRIO) and the ant colony approach (ANT) as described in Section 4. The solution of the successive release problem is gained by applying IBS with $BW \in \{1,5,10,2000\}$, because this parameter constellation showed a good compromise between solution quality and runtime in the previous section. The results of this comparison are summarized in Table 4. Note that due to its stochastic nature five repetitions of ANT have been executed per instance.

As a benchmark we apply off-the-shelf solver Xpress delivering optimal solution values, which are averaged over all 10 instances per group and labeled as *obj**. Alternatively, if the solver was not able to return the optimal solution of an instance within a given time frame of 900 s, we report the best known objective value for the original CSP as stated in Boysen et al. [4]. Note that this solution value is only a lower bound (denoted as LB in Table 4) to our CRSP, because it remains unknown whether the respective objective value is actually reachable given the limited resequencing flexibility of CRSP. For each solution approach (PRIO and ANT) we report the average objective value per instance group (*obj*), the average runtime in CPU seconds (time (s)), the maximum absolute and relative deviation from our benchmark (gap^{abs}_{max} and gap^{rel}_{max}), and the number of solutions being as good as our benchmark (*sol**/*sol*). Furthermore, we highlight the best heuristic approach for each instance group in bold.

Standard solver XPRESS was only able to solve the test instances with $T=10$ models to optimality. Any larger instance ended with a timeout, whereas our heuristic approaches solved all instances in reasonable time. Clearly, PRIO is much faster than ANT, which required an average of 126.94 CPU seconds for the largest test instances with $T=50$. However, spending about 2 min computational time for reshuffling a model sequencing seems acceptable even if the CRSP is iteratively solved in a rolling horizon.

With regard to the solution quality ANT outperforms PRIO. ANT solved 50% of all instances (450 out of 900 repetitions) as good as our benchmark (compared to 25% reached by PRIO). Recall that for 150 instances (750 repetitions of ANT) an LB serves as a benchmark, so that we conjectured quite a few additional optimal solutions. Thus, our decomposition approach consisting of an ant colony approach ANT for solving the fill problem and iterated beam search IBS for solving the release problem seems well suited for solving real-world CRSP instances.

To further assure the conclusion that ANT is a well-suited algorithmic setting for solving CRSP, we provide an additional

Table 4
Solution performance of the decomposition approach for the CRSP.

Instance group	XPRESS		PRIO				ANT			
	*obj** (LB)	Time (s)	*obj*	Time (s)	gap^{abs}_{max} (gap^{rel}_{max})	*sol**/*sol*	*obj*	Time (s)	gap^{abs}_{max} (gap^{rel}_{max})	*sol**/*sol*
CAR_3_10	1 (1)	0.28	1.3	0.01	2 (2)	8/10	**1**	0.98	**0 (0)**	50/50
CAR_5_10	1.3 (1)	4.59	2.6	0	3 (3)	3/10	**1.36**	1.35	**1 (1)**	47/50
CAR_7_10	2.6 (2)	166.81	5.4	0.02	4 (2)	0/10	**2.9**	1.72	**2 (0.5)**	38/50
CAR_3_15	−(2)	> 900	2.7	0.09	3 (1.5)	6/10	**2.2**	2.23	**1 (0.5)**	40/50
CAR_5_15	−(2)	> 900	4.2	0.09	5 (2.5)	3/10	**2.98**	3.11	**4 (2)**	25/50
CAR_7_15	−(4)	> 900	10	0.45	13 (3.25)	0/10	**6.6**	3.87	**8 (2)**	9/50
CAR_3_20	−(3)	> 900	3.9	0.34	4 (1.33)	4/10	**3.18**	3.86	**1 (0.33)**	41/50
CAR_5_20	−(3)	> 900	5.4	1.86	9 (3)	0/10	**3.82**	7.85	**5 (1.67)**	34/50
CAR_7_20	−(3)	> 900	9.2	1.41	14 (4.67)	0/10	**6.32**	7.59	**9 (3)**	0/50
CAR_3_30	−(4)	> 900	5.1	5.21	7 (1.75)	5/10	**4.32**	19.91	**4 (1)**	45/50
CAR_5_30	−(3)	> 900	7.1	5.11	10 (3.33)	0/10	**5.06**	18.88	**7 (2.33)**	16/50
CAR_7_30	−(4)	> 900	11.4	9.08	17 (4.25)	0/10	**8.26**	31.91	**12 (3)**	0/50
CAR_3_40	−(5)	> 900	5.4	9.13	2 (0.4)	7/10	**5.08**	31.89	**1 (0.2)**	46/50
CAR_5_40	−(5)	> 900	9.8	12.37	8 (1.6)	0/10	**7.52**	42.83	**6 (1.2)**	4/50
CAR_7_40	−(7)	> 900	16.3	15.11	28 (4)	0/10	**14.58**	51.89	**27 (3.86)**	0/50
CAR_3_50	−(6)	> 900	**6.1**	19.35	**1 (0.17)**	9/10	**6.1**	63.44	**1 (0.17)**	45/50
CAR_5_50	−(8)	> 900	13.3	23.19	14 (1.75)	0/10	**12.14**	75.69	**11 (1.38)**	0/50
CAR_7_50	−(11)	> 900	18.9	39.37	36 (3.27)	0/10	**16.96**	126.94	**34 (3.09)**	1/50
Total	−(4.11)	–	7.67	7.9	36 (4.67)	45/180	**6.13**	27.55	**34 (3.86)**	450/900

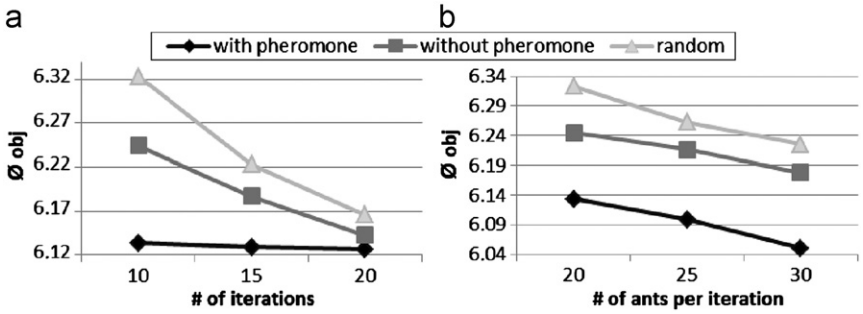

Fig. 6. Comparing ANT to other multi-pass construction heuristics. (a) Impact of iterations. (b) Impact of ants.

test, which investigates the quite uncommon parameter constellation of ANT. Indeed, applying only 10 iterations within an ant algorithm leaves the question, if these few iterations allow for a significant impact on the pheromone trail. For investigating this question, we compare ANT (denoted as *with pheromone* in Fig. 6) with two related algorithmic settings. Alternative 1 (denoted as *without pheromone* in Fig. 6) alters our ANT approach by neglecting pheromone. It exclusively utilizes the priority values for deriving the choice probabilities and, thus, resembles some multi-pass randomized priority rule approach. Alternative 2 (denoted as *random* in Fig. 6) applies a pure random choice and successively fills the buffer by choosing among all available models with equal probability. Note that all three alternatives apply the same algorithmic setting in stage 2, i.e., IBS with $BW \in \{1,5,10,2000\}$, and generate an identical total number of solutions. The solution quality of all three alternatives aggregated over all 180 instances is summarized in Fig. 6.

Fig. 6(a) relates the absolute solution value averaged over all instances in relation to an increasing number of iterations, whereas in Fig. 6(b) the number of ants per iteration is varied (while all other parameter values remain unchanged). The results reveal that our ANT approach outperforms both alternatives and, thus, integrating the pheromone feedback seems a valuable extension, even if applied with only a few iterations. Note that restricting the number of iterations seems inevitable because of the relatively large effort for solving the release subproblem on stage 2. Further note that the additional computational time of integrating the pheromone compared to both competitors is negligible, so that ANT seems well suited for solving the CRSP. Finally, it can be concluded that increasing the number of ants per iteration has a larger impact on solution quality than increasing the number of iterations.

5.3. The impact of varying resequencing flexibility

Clearly, there exists a trade-off between the investment costs for realizing a specific buffer setting and its resulting resequencing flexibility. With an increasing number of lanes the cost for installing a mix-bank increases while leading to additional flexibility, i.e., more possible model sequences, and vice versa. For providing some information for an experienced line manager when weighting investment cost against resequencing flexibility, we relate the resulting number of rule violations to different buffer constellations. For this purpose, we heuristically solve 90 instances, i.e., five per instance group, with our ant colony approach (including IBS in the parameter constellation as derived above) while successively increasing the number of lanes: $L = 1, \ldots, \lfloor T/2 \rfloor$, with $C = \lceil T/L \rceil$. Fig. 7 displays the average objective value (averaged over all five instances and five repetitions of the solution process) when increasing the number of buffer lanes available for all six instance groups. Serving as reference points the dotted lines indicate the best know objective values (as stated as LB in Table 4) averaged over all five instances.

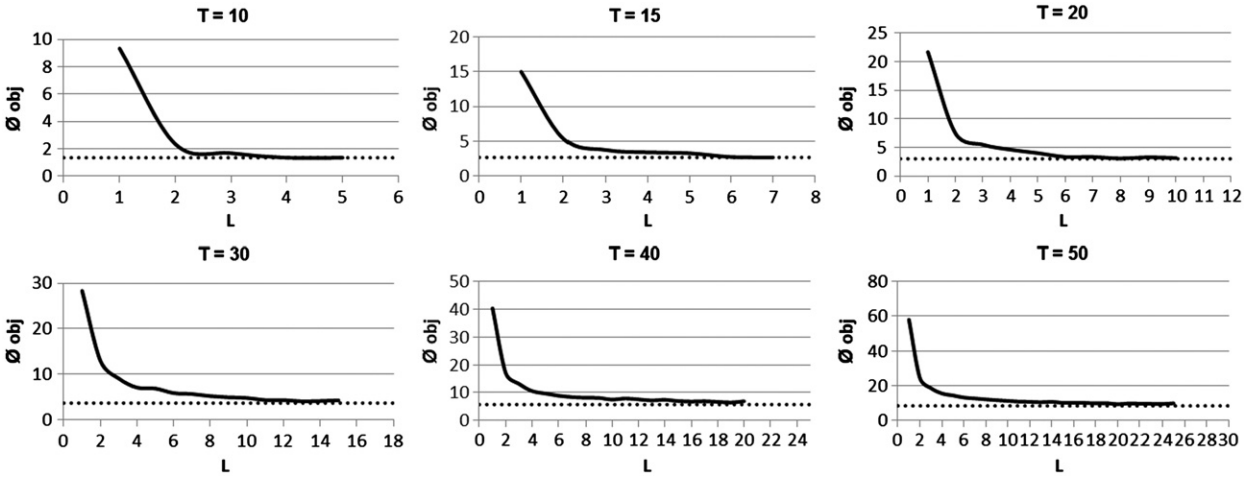

Fig. 7. Impact of the number of lanes on resequencing flexibility.

The results show that the positive effect of additional lanes quickly diminishes, so that already a few lanes are sufficient to nearly reach the results possible with full resequencing flexibility. Clearly, full flexibility is guaranteed if each model of the initial sequence can be assigned to its individual lane. However, the high investment cost associated with such a solution is unnecessary because already a considerable smaller number of lanes leads to a comparable flexibility level. The same conclusion holds true for the buffer setting ($T=30$ and $L=10$) at the OEM we supported with our research, which is sufficient to nearly reduce the number of rule violations to the benchmark with full resequencing flexibility.

6. Conclusion

This paper treats the car-resequencing problem with selectivity banks. A given initial sequence is to be reshuffled, so that violations of given sequencing rules are minimized and limited resequencing flexibility of a given buffer constellation is considered. For this problem setting a two-stage decomposition approach is introduced, which consists of different heuristic procedures for solving the fill subproblem of assigning models to lanes (stage one) and the release subproblem (stage two). The latter problem decides on the succession of models removed from the buffer lanes in the final assembly sequence. A comprehensive computational study shows computational efficiency of the decomposition approach. Our best algorithmic configuration, i.e., an ant algorithm (stage one) is coupled with a beam search approach (stage two), clearly outperforms an off-the-shelf solver with regard to solution time and quality. Furthermore, our study reveals that the impact of additional lanes quickly diminishes, so that already integrating just a few buffer lanes leads to nearly full resequencing flexibility.

Although designed for a specific problem setting of the automobile industry our solution approach is flexible enough to be customized for a wide range of related real-world resequencing settings:

- In real-world assembly lines sequencing rules are often subdivided into hard constraints, e.g., no two large trucks fit into two successive stations, and soft constraints, e.g., a successive assembly of laborious options causes a work overload at a station, which is to be balanced by additional utility workers [22]. Our graph approach can be readily adjusted to minimize violations of soft sequencing rules while observing hard rules with certainty. Any arc associated with a violation of hard rules simply needs to be deleted, while for calculating weights of the remaining arcs only violations of soft sequencing rules are summarized. Additionally, nodes can be deleted whenever the lower bound (9) for the set of hard sequencing rules indicates a value greater than 0.
- Another possible problem setting is investigated by Drexl and Kimms [8]. Here, sequencing rules are considered as hard constraints while a level scheduling objective function is pursued, which aims to evenly balance part demands induced by the production sequence over time. While Drexl and Kimms [8] consider initial sequence planning our graph approach can straightforwardly be adapted to transfer their problem setting into our resequencing environment. Again, arcs violating strict sequencing rules need to be deleted, while the arc weighting function is simply replaced by the leveling objective. Plenty lower bounds for diverse level scheduling objectives are introduced in the literature (e.g. see [1]), which can directly be applied, e.g., for filtering nodes within a modified beam search procedure.
- Finally, in many automobile assembly lines continuously paint defects in paint shop lead to stirred car sequences. Therefore, post-paint shop buffers are applied, so that the initial (stirred) sequence can be rearranged to approximate a reference sequence, i.e., the one originally released into paint shop. Such a reshuffling aims to reduce logistics effort at the stations of final assembly for rearranging material, which has already been delivered Just-in-Sequence as defined by the reference sequence, while hard car sequencing constraints need to be considered (see [4]). For coupling this problem setting with selectivity banks, again, only invalid arcs need to be deleted and the modified objective function is to be applied for calculating arc weights within our graph approach.

However, there remain some future research challenges in the field of resequencing mixed-model assembly lines. On the one hand, other efficient exact solution procedures, for instance, branch-and-bound procedures, should be developed for our problem setting, so that efficiency of heuristic procedures can be quantified for even larger test instances. On the other hand, either coupling the car sequencing objective with other buffer settings or joining our buffer setting with other objective functions for sequencing mixed-model assembly systems would be a valuable contribution.

References

[1] Bautista J, Companys R, Corominas A. Heuristics and exact algorithms for solving the monden problem. European Journal of Operational Research 1996;88:101–13.

[2] Blum C. Beam-ACO-hybridizing ant colony optimization with beam search: an application to open shop scheduling. Computers & Operations Research 2005;32:1565–91.

[3] Boysen N, Fliedner M, Scholl A. Sequencing mixed-model assembly lines: survey, classification and model critique. European Journal of Operational Research 2009;192(2):349–73.

[4] Boysen N, Golle U, Rothlauf F. The car resequencing problem with pull-off tables. BuR—Business Research 2011;4:276–92.

[5] Boysen N, Scholl A, Wopperer N. Resequencing of mixed-model assembly lines: survey and research agenda. European Journal of Operational Research 2012;216(3):594–604.

[6] Choi W, Shin H. A real-time sequence control system for the level production of the automobile assembly line. Computers & Industrial Engineering 1997;33:769–72.

[7] Dorigo M, Caro GD, Gambardella LM. Ant algorithms for discrete optimization. Artificial Life 1999;5:137–72.

[8] Drexl A, Kimms A. Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints. Management Science 2001;47:480–91.

[9] Fliedner M, Boysen N. Solving the car sequencing problem via branch & bound. European Journal of Operational Research 2008;191(3):1023–42.

[10] Gagné C, Gravel M, Price WL. Solving real car sequencing problems with ant colony optimization. European Journal of Operational Research 2006;174(3): 1427–48.

[11] Golle U, Rothlauf F, Boysen N. Iterative beam search for car sequencing. Technical report 02-2011. Department of Information Systems and Business Administration, Johannes Gutenberg-University, Mainz; 2011.

[12] Gottlieb J, Puchta M, Solnon C. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In: Proceedings of the 2003 international conference on applications of evolutionary computing. EvoWorkshops'03. Berlin, Heidelberg: Springer-Verlag; 2003. pp. 246–57.

[13] Inman R. ASRS sizing for recreating automotive assembly sequences. International Journal of Production Research 2003;41:847–63.

[14] Kis T. On the complexity of the car sequencing problem. Operations Research Letters 2004;32:331–5.

[15] Lahmar M, Benjaafar S. Sequencing with limited flexibility. IIE Transactions 2007;39:937–55.

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

[17] Lim A, Xu Z. Searching optimal resequencing and feature assignment on an automated assembly line. Journal of the Operational Research Society 2009;60:361–71.

[18] Lowerre B. The harpy speech recognition system. PhD thesis. Carnegie Mellon University; 1976.

[19] Ow PS, Morton T. Filtered beam search in scheduling. International Journal of Production Research 1988;26:35–62.

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

[21] Sabuncuoglu I, Gocgun Y, Erel E. Backtracking and exchange of information: methods to enhance a beam search algorithm for assembly line scheduling. European Journal of Operational Research 2008;186:915–30.

[22] Solnon C, Cung V, Nguyen A, Artigues C. 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 2008;191(3):912–27.

[23] Spieckermann S, Gutenschwager K, Voß S. A sequential ordering problem in automotive paint shops. International Journal of Production Research 2004;42:1865–78.

[24] Stützle T, Dorigo M. ACO algorithms for the quadratic assignment problem. Maidenhead, England, UK: McGraw-Hill Ltd.; 1999 pp. 33–50.

[25] Wang F, Lim A. A stochastic beam search for the berth allocation problem. Decision Support Systems 2007;42:2186–96.