JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Iterative Beam Search for Car Sequencing

Uli Golle, Franz Rothlauf, Nils Boysen

Working Paper 02/2011
March 2011

Working Papers in Information Systems and Business Administration

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

Iterative Beam Search for Car Sequencing

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

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

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

June 1, 2011

Abstract

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

Keywords: Mixed-model assembly line; Car sequencing; Iterative Beam Search

1 Introduction

The car sequencing (CS) problem (Parrello et al., 1986) is a NP-hard combinatorial optimization problem seeking a production sequence of various car models which are jointly produced on a mixed-model assembly line. The car models are distinguished by different binary options, e.g., having an air-conditioning or not. An accumulation of models with the same option in a sequence could lead to work overload of the operators on the line as they cannot accomplish their work within the available station limits. Such work overload would have to be compensated by costly strategies like employing additional utility workers or stopping the line. To avoid such scenarios, CS applies so-called sequencing rules, which restrict the number of car models having an option in any subsequence of defined length. The aim is to find a sequence of car models, which satisfies all sequencing rules

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

Table 1: Notation

(constraint satisfaction problem) or a sequence with minimum number of rule violations (optimization problem).

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

In this paper, we propose an IBS approach for solving CS considered as an optimization problem. Our procedure is based on a new graph representation of CS, which was inspired by Boysen et al. (2010), who used a related approach for solving the resequencing version of CS. Here, so-called pull-off tables are applied to pull models offline and reinsert them in later production cycles in order to reshuffle the sequence before entering a successive production department. We modify their graph representation to the traditional CS problem. Additionally, we define and apply improved lower bounds for CS. Experimental results reveal, that our algorithm clearly outperforms the currently best known exact solution approach, a Scattered Branch & Bound algorithm. We also show the superiority of our new lower bound arguments, as they significantly reduce the amount of evaluated nodes for IBS.

The outline of the paper is as follows. The next section reviews relevant literature and formulates CS as a mathematical model. In Section 3, we show how to develop an IBS approach for CS, based on a new graph representation and improved lower bounds. Several experiments are conducted in Section 4, that show the superiority of our algorithm to a recent solution approach in the literature. Section 5 gives concluding remarks and a brief outlook on future research.

2 Model formulation and literature

The CS problem was first introduced by Parrello et al. (1986). Given a pool of various car models, which can be distinguished by their required options (such as air conditioning or

sun roof yes/no), CS aims to find a production sequence with minimum work overload. Therefore, it uses $H_o : N_o$ sequencing rules, which restrict the occurrences of option o in any subsequence of N_o succeeding cars to at most H_o. The objective of CS is to minimize the number of rule violations. Thus, in contrast to the related Mixed-Model Sequencing approach (Wester and Kilbridge, 1964), CS does only implicitly minimize the resulting work overload. With the notations from Table 1, we can formulate the CS problem as a mathematical model:

$$\text{CS: Minimize} \quad Z(X,Y) = \sum_{o \in O} \sum_{t=1}^{T-N_o+1} y_{ot} \tag{1}$$

$$\sum_{t=1}^{T} x_{mt} = d_m \quad \forall m \in M \tag{2}$$

$$\sum_{m \in M} x_{mt} = 1 \quad \forall t = 1, \ldots, T \tag{3}$$

$$\sum_{t'=t}^{t+N_o-1} \sum_{m \in M} x_{mt'} \cdot a_{mo} \leq H_o + y_{ot} \cdot BI \quad \forall o \in O; t = 1, \ldots, T - N_o + 1 \tag{4}$$

$$x_{mt} \in \{0,1\} \quad \forall m \in M; t = 1, \ldots, T \tag{5}$$

$$y_{ot} \in \{0,1\} \quad \forall o \in O; t = 1, \ldots, T \tag{6}$$

Variable y_{ot} indicates whether a rule violation with regard to option o occurs in a subsequence starting at position t. Objective function (1) minimizes the number of rule violations. Constraints (2) assure that the produced models meet the required demand, whereas (3) enforces that exactly one model is produced in each slot t. Constraints (4) check whether a rule violation occurs. Finally, (5) and (6) ensure that variables x_{mt} and y_{ot} take only binary values. The literature deals with various approaches about how to count the number of rule violations. Here, we state the widely-used sliding-window technique (SW) (Gravel et al., 2005). SW counts all complete subsequences of length N_o, in which a rule violation occurs. As SW tends to double count some rule violations and weights them differently depending on their position in the sequence, an alternative approach was introduced by Fliedner and Boysen (2008) (FB). FB counts all option occurrences leading to a rule violation, so that (4) is replaced with:

$$\sum_{t'=t}^{\min\{t+N_o-1,T\}} \sum_{m \in M} x_{mt'} \cdot a_{mo} - (1 - \sum_{m=1}^{M} a_{om} \cdot x_{mt}) \cdot BI \leq H_o + y_{ot} \cdot BI \quad \forall o \in O; t = 1, \ldots, T. \tag{7}$$

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

CS is known to be NP-hard in the strong sense (Kis, 2004). Different exact and heuristic solution approaches have been proposed in the literature. Among the exact

approaches are Integer Linear Program (ILP) formulations (Gravel et al., 2005; Prandtstetter and Raidl, 2008) and a branch & bound algorithm (Fliedner and Boysen, 2008). Gravel et al. (2005) proposed an ILP avoiding symmetries by grouping cars with the same options. Prandtstetter and Raidl (2008) take additional paint shop constraints into account in their ILP and focus on the assignment of option/colors instead of models. This formulation shows superior results compared to the ILP of Gravel et al. (2005). The branch & bound algorithm developed by Fliedner and Boysen (2008) is currently the state-of-the-art exact solution algorithm. It is based on a scattered branch & bound scheme. The authors introduce and apply new lower bounds and dominance rules for CS in order to speed up the search.

Beside exact solution approaches, various heuristics are available in the literature. According to experimental results, mainly variable local search procedures which use more than one neighborhood seem to be suitable for CS (Puchta and Gottlieb, 2002; Gottlieb et al., 2003; Perron and Shaw, 2004; Estellon et al., 2008; Prandtstetter and Raidl, 2008). Various neighborhoods can be identified in the literature, the most promising are Swap (exchanging the positions of two car models) and Lin2Opt (inverting a subsequence of car models). Estellon et al. (2008) and Prandtstetter and Raidl (2008) showed that local search based methods with variable neighborhoods can efficiently solve large-scale industrial problem instances with more than 1000 cars. A first Beam Search approach for CS was introduced by Bautista et al. (2008). However, their approach is different from ours as they use a different objective function based on the over- and under-assignment of options and also a different graph representation of CS. Other heuristic algorithms were applied as well on CS, e.g., Ant Colony Optimization (Gottlieb et al., 2003; Gravel et al., 2005), Genetic Algorithms (Warwick and Tsang, 1995; Zinflou et al., 2007) and hybrid approaches (Prandtstetter and Raidl, 2008; Jaszkiewicz et al., 2004). For a complete overview of the various solution approaches to CS, see Boysen et al. (2009) and Solnon et al. (2008).

3 An Iterative Beam Search approach

3.1 Search procedure

Iterative Beam Search (IBS) is a repeatedly conducted Beam Search (BS) heuristic (Lowerre, 1976). BS is a truncated breadth-first search and is based on a graph representation of the problem to be solved. The graph (typically a tree) consists of a single root node, the starting point for the search, inner nodes, which can be considered as partial solutions to the problem, leaf nodes, which represent overall solutions and arcs by which the nodes are connected. BS seeks to find a shortest path from the root node to a leaf node. It branches through the graph stagewise, where a stage consists of related nodes, e.g., inner nodes representing partial solutions of the same size. Thereby, BS gradually expands the BW (beam width) best partial solutions of the same stage. This results in a pool of partial solutions of the next stage from which again the BW best nodes are considered to be further extended. This is repeated until reaching the leafs of the tree and the best leaf node is returned as the result of Beam Search. Since the number of nodes at each stage to be further branched is restricted by BW, BS unlikely finds the global best solution. For

selecting the BW best nodes at each stage, a problem-specific heuristic is applied, which allows to order the nodes according to a relevant value, e.g., estimated objective value. Additional upper bounds (lower bounds for maximization problems) can be used, which allow to delete nodes during the search that cannot lead to better solutions.

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

3.2 Graph representation of CS

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

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

Additionally, we assign weights $f : E \to \mathbb{N}$ to each arc, which represent the contribution of producing model m' in slot i to the overall objective value. Here, the weights measure the number of sequencing rule violations caused by producing model m'. Depending on the test set in the experimental Section 4, we apply two different weighting functions f_1 and f_2:

$$f_1 = \begin{cases} \sum_{o \in O} \min\{1; \max\{\sum_{t=1}^{N_o-1} act_i^{o,t} + a_{om'} - H_o; 0\}\}, & \text{if } i \geq N_o \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

represents the sliding window objective function (SW). And

$$f_2 = \sum_{o \in O} \min\{a_{om'}; \max\{\sum_{t=1}^{N_o-1} act_i^{o,t} + a_{om'} - H_o; 0\}\} \tag{9}$$

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

3.3 Lower Bounds

On the one hand, within our IBS procedure lower bounds are applied to prune the graph. Specifically, nodes whose cumulated rule violations along the shortest path plus a lower bound on the remaining rule violations exceed an upper bound, i.e., generated by a previous BS run with smaller BW, are fathomed. On the other hand, lower bounds are utilized to guide the search into promising regions of the solution space (see Section 3.4). There are some lower bounds available in the literature on CS, which consider the option occurrences decoupled from their actual assignment to models (Fliedner and Boysen, 2008; Benoist, 2008). Thus, for every option o, a lower bound can be computed regardless of other options. The sum of the lower bounds on every option lead to an overall lower bound. In his work, Benoist (2008) considers a different objective function for CS than we do and, therefore, his lower bound argument can not be applied in this paper. The lower bound by Fliedner and Boysen (2008) neglects the used objective function. Their bound is stated by $\sum_{o \in O} d_o^\alpha - a_o^{max,T}$, the difference between the actual demand d_o^α of models containing option o and the maximum number of available slots $a_o^{max,T} = \lfloor \frac{T}{N_o} \rfloor \cdot H_o + \min\{H_o; T \bmod N_o\}$ to produce models with option o in a sequence of length T without any violation, summed up over all options. Therefore, every option occurrence which exceeds the number of available slots leads to a sequencing rule violation of one. This lower bound can be tightened by incorporating the used objective function as exceeding option occurrences could actually lead to more than one violation. In the following, we will present lower bounds for CS incorporating the SW or FB objective function.

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

Example: Consider a single option o subject to a sequencing rule of $2:4$. The sequence length T is 13. The demand for optional models d_o^α is 8 and the demand for basic models d_o^β is 5. Applying the construction algorithm leads to the optimal sequence depicted in Figure 1. The first two slots are filled with optional models, the next two slots contain basic models, and this pattern is repeated until at slot 12 only optional models are left. Considering either the SW or FB objective function, this sequence leads to an identical

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

Figure 1: Example result of construction algorithm

result of two violations in the subsequences starting at slots 9 and 10, respectively. Note, that the lower bound by Fliedner and Boysen (2008) would return only one violation as $d_o^\alpha - a_o^{max,T} = 8 - 7$.

The construction algorithm generates optimal sequences consisting of a repeated pattern π^o of size N_o with H_o optional models at the beginning of the pattern followed by $N_o - H_o$ basic models, until either the required optional or basic models are exhausted. In the resulting sequence in Figure 1, π^o contains four slots with two optional models and two basic models and is processed twice up to slot 8.

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

$$t_o^{max} = \left(\left\lfloor \frac{d_o^\beta}{N_o - H_o} \right\rfloor + 1 \right) \cdot H_o + d_o^\beta. \tag{10}$$

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

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

$$LB^{SW} = \max\{\min\{T - N_o + 1, T - t_o^{max}\}, 0\}. \tag{11}$$

If $t_o^{max} > T$, more than T slots can be processed without violation. Therefore, no violation occurs and LB^{SW} returns 0, which is ensured by the max-function. If $t_{max} < T$, violations are inevitable as in slot $t_o^{max} + 1$ a basic model is required, but only optional models are left. Thus, processing an optional model will lead to a violation of the window ending at slot $t_o^{max} + 1$. Since at that time only optional models are left, all remaining sliding windows from $t_o^{max} + 1$ to T will also be violated. Thus, we have an overall number of $T - t_o^{max}$ violated windows. The SW approach considers only complete windows of size N_o, such that there are exactly $T - N_o + 1$ complete windows in a sequence of length T. The first window which can be violated ends at slot N_o. If $t_o^{max} < N_o$, $T - t_o^{max}$ would lead to more than $T - N_o + 1$ violated windows. Thus, the min-function bounds the number

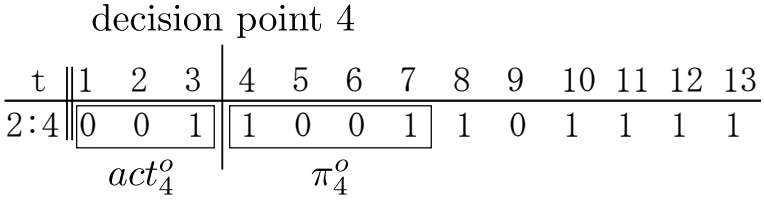

Figure 2: Example result of construction algorithm

of violations by the maximum number of windows $T - N_o + 1$.

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

$$LB^{FB} = \max\{T - t_o^{max}, 0\}. \tag{12}$$

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

Example (cont.): Consider the example of Figure 1. By applying (10), we receive $t_o^{max} = 11$ and, therefore, both lower bounds LB^{SW} and LB^{FB} result in the correct number of two violations.

Clearly, equation (10) computes t_o^{max} correctly, unless a part of the sequence is already fixed. If some slots are already processed, the models in the active sequence have an influence on the repeating pattern π^o generated by the construction algorithm. Thus, in the following t_o^{max} is determined under consideration of the option occurrences in the active sequence.

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

$$\pi_i^o(j) = \begin{cases} 1, & \text{if } \sum_{t=1}^{N_o - j} act_i^{o,t} + \sum_{k=1}^{j-1} \pi_i^o(k) < H_o \\ 0, & \text{else} \end{cases} \tag{13}$$
$$j = 1 \ldots N_o$$

Example: Consider the example in Figure 2. It is the same example as before, except for the first three slots that are already being filled with models. The active sequence at decision point 4 is $act_4^o = \{1, 0, 0\}$, thus in slots 1 and 2 basic models were processed and an optional model in slot 3. The resulting repeating pattern obtained from (13) is $\pi_4^o = \{1, 0, 0, 1\}$. Note, that the repeating pattern still consists of H_o optional models and $N_o - H_o$ basic models.

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

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

$$t_{o,i}^{max} = \left(\left\lfloor \frac{d_{o,i}^{\beta}}{N_o - H_o} \right\rfloor \right) \cdot N_o + f_{\pi_i^o}(d_{o,i}^{\beta} \bmod (N_o - H_o) + 1) - 1. \tag{14}$$

A complete pattern π_i^o of size N_o can be processed with each $N_o - H_o$ basic models (first term). By doing so, $d_{o,i}^{\beta} \bmod (N_o - H_o)$ basic models remain. With these basic models, we can still process a fraction of the pattern π_i^o up to position $f_{\pi_i^o}(d_{o,i}^{\beta} \bmod (N_o - H_o)+1)-1$ (second term). Therefore, we can construct a sequence of length $t_{o,i}^{max}$ starting at decision point i without any violation. Adapting formulas (11) and (12) we get:

$$LB^{SW} = v_{\text{fix}} + \max\{\min\{T - N_o + 1, T - (i + t_{o,i}^{max} - 1)\}, 0\} \tag{15}$$

and

$$LB^{FB} = v_{\text{fix}} + \max\{T - (i + t_{o,i}^{max} - 1), 0\}, \tag{16}$$

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

Example (cont.): Again, we take a look at the example sequence of Figure 2, subject to sequencing rule $2:4$. At decision point $i = 4$, we have three basic models left ($d_{o,4}^{\beta} = 3$). With $f_{\pi_4^o}(2) = 3$ and (14), we get $t_{o,4}^{max} = 6$. And finally, with (15) and (16), both lower bounds result in the correct number of 4 violations.

3.4 Node Selection

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

4 Computational study

4.1 Experimental setup

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

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

4.2 Results

We apply IBS in two different modes. IBS_1 utilizes the lower bound introduced by Fliedner and Boysen (2008), while IBS_2 uses our new lower bound (see Section 3.3). Both, IBS_1 and IBS_2, are performed with nine iterations and beam widths $BW = \{5, 10, 25, 50, 100, 500, 1000, 1500, \infty\}$. Since the last beam width $BW_9 = \infty$, IBS_1 and IBS_2 become exact solution approaches and return the global best solution. We compare both algorithms with the currently best known exact solution approach, a Scattered Branch & Bound algorithm (SB&B) (Fliedner and Boysen, 2008). The SB&B is applied with $\eta = 100, \Delta = 10000, \alpha = 0.75$, dominance rules 2,3 and 6 and the lower bound by Fliedner and Boysen (see Fliedner and Boysen, 2008). For each instance, the time limit for solving the instance is set to 600 seconds, which seems an appropriate choice for a short-term planning problem like CS. An algorithm is aborted, if the given time limit is exceeded.

Table 2 shows the results for Problem Set 1, according to the FB objective function. Label 'obj*' denotes the currently best known solution for each instance as stated in Boysen et al. (2010). The columns 'obj', 'time' and 'nodes' state the best objective value obtained by each algorithm and the time and number of evaluated nodes required to find this value. Note, that 'obj' contains the global best solution, if an instance could be solved within the time limit of 600 seconds.

The results returned by IBS_1 and SB&B are comparable (excepting instance CAR_7_40, where SB&B finds a better solution than IBS_1). Regarding the solution time, IBS_1 is slightly in favour on 9 instances, while SB&B requires less time than IBS_1 on 6 instances. Overall, IBS_1 and IBS_2 can evaluate nodes much faster than SB&B, mainly because no dominance rules are applied. Due to the new lower bound, IBS_2 is able to reduce the amount of evaluated nodes considerably compared to IBS_1. Therefore, it is able to outperform IBS_1 on all instances and SB&B on 15 out of 18 instances. Only IBS_2 finds the best known solutions for all instances of Set 1.

In order to compare the performances of the algorithms on larger problem instances, we use Problem Sets 2 and 3 consisting of instances with up to 400 car models. The results

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

Table 2: Results for Problem Set 1 (FB objective function)

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

4.3 Optimality Proof

For instances 6-76, 10-93, 19-71, 21-90 and 36-92 of Problem Set 2, where the current best solutions found contain sequencing rule violations, the optimality of these solutions was only proven for instance 19-71 by Gent (1998), who reasoned a lower bound with the same number of two violations as the currently best solution. For the remaining instances, the optimality of the current best solution still needs to be verified. For this reason, we relaxed these instances by merely considering two out of five options concurrently. Considering each combination of two options, we receive $\binom{5}{2} = 10$ subinstances for every instance. Note, that any optimal solution of one of the subinstances serves as a lower bound for

						SB&B			IBS_1			IBS_2		
	Instance	T	O	M	obj*	obj	time (s)	nodes	obj	time (s)	nodes	obj	time (s)	nodes
Set 2	4-72	100	5	22	0	0	3.53	201	0	0.27	1,470	0	0.28	1,470
	6-76	100	5	22	6	6	>600	-	6	>600	-	6	>600	-
	10-93	100	5	25	3	5	>600	-	3	>600	-	3	>600	-
	16-81	100	5	26	0	0	5.08	6,502	0	1.64	8,431	0	1.70	8,431
	19-71	100	5	23	2	3	>600	-	2	>600	-	2	>600	-
	21-90	100	5	23	2	2	>600	-	2	>600	-	2	>600	-
	36-92	100	5	22	2	4	>600	-	2	>600	-	2	>600	-
	41-66	100	5	19	0	0	3.13	90	0	0.94	8,799	0	1.86	17,073
	26-82	100	5	24	0	0	3.78	22	0	0.58	3,857	0	0.61	3,857
Set 3	200_01	200	5	25	0	5	>600	-	1	>600	-	1	>600	-
	200_02	200	5	25	2	7	>600	-	3	>600	-	3	>600	-
	200_03	200	5	25	4	14	>600	-	9	>600	-	8	>600	-
	200_04	200	5	24	7	16	>600	-	8	>600	-	8	>600	-
	200_05	200	5	23	6	14	>600	-	8	>600	-	8	>600	-
	200_06	200	5	23	6	8	>600	-	7	>600	-	7	>600	-
	200_07	200	5	23	0	1	>600	-	0	0.562	2943	0	0.578	2943
	200_08	200	5	20	8	11	>600	-	9	>600	-	9	>600	-
	200_09	200	5	24	10	15	>600	-	10	>600	-	10	>600	-
	200_10	200	5	19	19	27	>600	-	20	>600	-	20	>600	-
	300_01	300	5	25	0	7	>600	-	0	5.891	26767	0	6.047	26768
	300_02	300	5	25	12	24	>600	-	12	>600	-	12	>600	-
	300_03	300	5	25	13	25	>600	-	14	>600	-	14	>600	-
	300_04	300	5	24	7	21	>600	-	10	>600	-	10	>600	-
	300_05	300	5	20	29	52	>600	-	33	>600	-	32	>600	-
	300_06	300	5	25	2	9	>600	-	6	>600	-	6	>600	-
	300_07	300	5	24	0	14	>600	-	0	5.156	25636	0	5.344	25636
	300_08	300	5	23	8	17	>600	-	9	>600	-	9	>600	-
	300_09	300	5	21	7	16	>600	-	7	>600	-	7	>600	-
	300_10	300	5	19	21	56	>600	-	25	>600	-	25	>600	-
	400_01	400	5	25	1	9	>600	-	3	>600	-	3	>600	-
	400_02	400	5	22	16	35	>600	-	20	>600	-	19	>600	-
	400_03	400	5	23	9	19	>600	-	12	>600	-	12	>600	-
	400_04	400	5	26	19	29	>600	-	20	>600	-	20	>600	-
	400_05	400	5	20	0	23	>600	-	0	0.359	2008	0	0.375	2008
	400_06	400	5	23	0	2	>600	-	0	7.047	35745	0	7.281	35747
	400_07	400	5	23	4	15	>600	-	5	>600	-	4	>600	-
	400_08	400	5	21	4	28	>600	-	10	>600	-	10	>600	-
	400_09	400	5	24	5	29	>600	-	11	>600	-	11	>600	-
	400_10	400	5	25	0	23	>600	-	0	3.031	15573	0	3.125	15573

Table 3: Results for Problem Set 2 and 3 (SW objective function)

the original problem instance. We solved each combination with our IBS_2 approach from Section 4.2. Table 4 shows the respective results. Thereby, we were able to proof the optimality of the current best solution for instances 6-76, 10-93 and 36-92. For instance 6-76 the subinstance just containing options 1 and 3 leads to 6 violations, for 10-93 the subinstance with options 1 and 2 leads to 3 violations and for instance 36-92 the subinstance composed of options 2 and 4 amounts to 2 violations. Only for instance 21-90, the optimality proof remains open, as all subinstances return 0 violations.

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

Table 4: Results for subinstances (SW objective function)

5 Conclusion

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

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

Future research on CS should deal with the development of superior lower bounds. For instance, more than merely a single option at a time could be taken into account, so that dependencies between options can be incorporated into improved lower bounds. Furthermore, alternative CS objective functions (sliding-window, objective function by Fliedner and Boysen (2008), etc.) should be addressed in order to evaluate which objective function actually leads to the underlying goal of minimizing the work overload.

References

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

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

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

Boysen, N., Golle, U., and Rothlauf, F. (2010). The car resequencing problem. *Department of Information Systems and Business Administration, Johannes Gutenberg-University Mainz.*

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

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

Gent, I. (1998). Two results on car-sequencing problems. Technical report, University of Strathclyde.

Gottlieb, J., Puchta, M., and Solnon, C. (2003). A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In Cagnoni, S. e. a., editor, *EvoWorkshops 2003*, pages 246–257, Berlin Heidelberg. Springer.

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

Jaszkiewicz, A., Kominek, P., and Kubiak, M. (2004). Adaptation of the genetic local search algorithm to a car sequencing problem. In *7th National Conference on Evolutionary Computation and Global Optimization*, pages 75–82.

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

Lowerre, B. (1976). *The Harpy Speech Recognition System.* PhD thesis, Carnegie Mellon University.

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

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

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

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

Solnon, C., Cung, V., Nguyen, A., and Artigues, C. (2008). The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927.

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

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

Zinflou, A., Gagne, C., and Gravel, M. (2007). Crossover operators for the car sequencing problem. In *Evolutionary Computation in Combinatorial Optimization - Proceedings of the 7th European Conference, EvoCOP 2007*, pages 229–239.