J Sched (2006) 9: 153–176
DOI 10.1007/s10951-006-7186-9

Algorithms for the car sequencing and the level scheduling problem

Andreas Drexl · Alf Kimms · Lars Matthießen

Abstract This paper deals with two most important problems arising in sequencing mixed-model assembly lines. One problem is to keep the line's workstations loads as constant as possible (the 'car sequencing problem') while the other is to keep the usage rate of all parts fed into the final assembly as constant as possible (the 'level scheduling problem'). The first problem is a difficult constraint-satisfaction problem while the second requires to optimize a nonlinear objective function. The contribution of this paper is twofold: First, we describe a branching scheme and bounding algorithms for the computation of feasible sequences for the car sequencing problem. Second, we present an algorithm which can optimize a level scheduling objective while taking care of the car sequencing constraints. Computational results are presented which show that feasible sequences can be obtained quickly for large problem instances.

Keywords Car sequencing · Level scheduling · Branching · Bounding · Computational results

1. Introduction

In many assembly systems, products are mounted on a conveyor belt. Operators or installation teams move along with the belt while working on a product. In general, an operator can work on a product only when it is at his station. If the operator does not finish work on a product before it leaves his station, there are two alternative approaches for completing what so far has not been done. Usually, in the United States, utility workers are employed to finish the work left undone by

A. Drexl (✉)
Department of Business Administration, University of Kiel, 24098 Kiel, Germany
e-mail: andreas.drexl@bwl.uni-kiel.de

A. Kimms
Department of Technology and Operations Management, University of Duisburg-Essen,
47048 Duisburg, Germany
e-mail: alf.kimms@uni-duisburg-essen.de

L. Matthießen
Department of Computer Science, University of Kiel, 24098 Kiel, Germany

the primary operator. In Japan, the operator pushes a stop button whenever he is unable to finish his work. Clearly, the management philosophy behind such distinct approaches is quite different.

Mixed-model assembly lines with negligible change-over between the products enable diversified small-lot production. Just-in-Time (JIT) production methods of the 'pull' variety can be used to control such systems. By the use of JIT methods it becomes possible to satisfy customers' demands for several products without holding large inventories and without incurring large shortages (see, e.g., Kubiak, 1993; Steiner and Yeomans, 1993).

When several models of the same general product have to be assembled on one common line the underlying design problem has two components: (i) a long-term planning problem called line balancing and (ii) a short-term planning problem known as model sequencing. In the line balancing problem, the tasks required to assemble the product have to be allocated to workstations. Each station has to be designed, tooled and equipped with respect to the tasks assigned to it and, hence, the allocation is based more on strategic than on operational issues. A discussion of these topics can be found in, for instance, Scholl (1999). In the model sequencing problem, the shop floor is the focus of attention where we have to decide about the specific order in which the different models have to be launched onto the line. Usually, the demand rate of the models varies and the problem must be solved periodically.

Because of the pull nature of JIT systems, once the sequence of the models is fixed at the final assembly level, the production schedules at all preceding levels are also inherently fixed. Therefore, the major problem to be solved as a prerequisite for the effective utilization of JIT systems is to determine the sequence in which different models have to be scheduled at the final assembly level. In general, the final assembly level consists of several stations where each is serviced by a part feeder or one feeder provides parts for several stations (see, e.g., Sumichrast and Clayton, 1996). In such production environments, we have to take care about the station loads and about the part usages, both of which are a function of the sequence.

In practice, usually subsequences consisting of, for instance, six copies are used in a cyclic manner. These subsequences work 'reasonable good' and they evolve by experience, not by analysis. However, problems arise frequently because of negligence of interdependencies between consecutive subsequences. The methods developed in this paper allow to evaluate subsequences consisting of about 50 copies. Hence, the contribution of our work shall be to reduce the number and the amount of problems arising because of not considering interdependencies between consecutive subsequences.

The exposition of our work is as follows: First, we address in Sections 2 and 3 the question whether a feasible sequence exists, that is, if the car sequencing problem has a feasible solution or not. To this end, Section 2 introduces the car sequencing problem mathematically and by means of an example. Then, Section 3 provides the branching scheme along with bounding rules for pruning large parts of the search tree. Second, in Section 4, the optimization problem is dealt with, that is, techniques for computing optimal level schedules taking into account the car sequencing constraints are proposed. Finally, computational results are presented in Section 5.

2. The car sequencing problem

The car sequencing problem can be explained easily in the context of car manufacturing. Assume that some variants of a car have to be produced. Usually, each of the variants requires a specific set of options such as 'sun roof', etc. For each variant, the customer demand is known, that is, we assume that the number of cars (identical copies) of each variant that have to be manufactured is given.

Table 1 Alphabetical list of notations

α	:	branching step or b-step
D	:	$\{1,\ldots,V\} \to \mathbb{N}$; function D maps a variant type v to the demand of that type, i.e. the number of copies of v requested within the planning horizon T
$D(v)$	:	number of copies (demand) to be produced of variant v
δ^*	:	schedule with minimal objective value
edd	:	earliest due date (left interval limit)
$H_j : N_j$	:	at most H_j of N_j successively sequenced copies may require option j
ldd	:	latest due date (right interval limit)
M	:	$O \times T$ matrix $(m_{j,t})$
O	:	number of options, index j
$occ(j)$	:	number of occurrences of option j in σ
opt	:	$\{1,\ldots,V\} \to 2^{\{1,\ldots,o\}}$; injective function $j \in opt(v) \overset{\text{def}}{\Leftrightarrow}$ variant v needs option j
σ	:	sequence
$\sigma^{-1}(v)$	:	set of periods, in which variant v is scheduled
T	:	total production volume (periods, cycles), i.e. $T = \sum_{v=1}^{V} D(v)$, index t
U_ξ	:	$\sigma^{-1}(0)$, the set of non-assigned periods
$\bar{v}$	:	$\{1,\ldots,V\} \to \{-1, 1\}$, where $\bar{v}(v)_j \overset{\text{def}}{=} \begin{cases} 1 \text{ if } j \in opt(v) \\ -1 \text{ otherwise} \end{cases}$
V	:	number of variants, index v
ξ	:	$\langle \sigma_\xi, M_\xi \rangle$, *CS*-state
Ξ	:	set of all *CS*-states

The core problem shall now be explained by means of an example: Assume that 60% of the cars manufactured on the line require the option 'sun roof'. Moreover, assume that five cars pass the station where the sun roofs are installed during the time for the installation of a single copy. Then, three operators (installation teams) are necessary for the installation of sun roofs. Hence, the capacity constraint of the final assembly line for the option 'sun roof' is three out of five in a sequence, or '3:5' for short.

Table 1 summarizes our notation. Formulation 1 defines the car sequencing problem mathematically while Example 1 illustrates the problem under consideration.

Formulation 1. The car sequencing problem is to find a sequence $\sigma : \{,\ldots,T\} \to \{1,\ldots,V\}$ which satisfies conditions (1) and (2):

$$|\sigma^{-1}(v)| = D(v) \qquad 1 \le v \le V \tag{1}$$

$$\sum_{\tau=t}^{t+N_j-1} \max\{0, \bar{v}(\sigma(\tau))_j\} \le H_j \qquad 1 \le j \le O, 1 \le t \le T - N_j + 1 \tag{2}$$

Equations (1) ensure the production of the required number of copies of each variant, while inequalities (2) restrict the sequences to be feasible only if the '$H_j : N_j$' constraints are fulfilled. Additionally, the property of a being a function ensures that one variant is assigned to each cycle.

Example 1. Consider an instance with $V = 7$ variants and $O = 4$ options, the number of requested copies and the option requirements given in Table 2, and the option restrictions

Table 2 Number of requested copies and option requirements

v	1	2	3	4	5	6	7
$opt(v)$	$\{2,3,4\}$	$\{3\}$	$\{4\}$	$\emptyset$	$\{1,4\}$	$\{1,3\}$	$\{3,4\}$
$D(v)$	1	3	1	3	2	1	1

Table 3 Feasible car sequence

period	1	2	3	4	5	6	7	8	9	10	11	12
variant	7	2	4	5	4	1	2	5	4	3	2	6
1:4				×				×				×
1:6						×						
2:5	×	×				×	×				×	×
1:2	×			×		×		×		×		

{1:4, 1:6, 2:5, 1:2}. For this instance, a feasible solution is given in Table 3 where a '×' shows an option occurrence to be installed to the variant.

The car sequencing model pays attention to the work contents of the products. Similarly, the part usage rate can be controlled within the same framework of constraints imposed on the number of options which are required consecutively. Hence, this approach uniquely addresses two fundamental aspects of mixed-model assembly lines which usually are dealt with separately (see, e.g., the recent papers by Zeramdini, Aigbedo and Monden, 2000; Zhang et al., 2000).

The car sequencing model does not require to define upstream and downstream station limits explicitely. In other words, it is not necessary to state whether the problem setting confines to what is called an open or closed station. This is advantageous because, in practice, there is generally some degree of freedom in this aspect which sometimes makes a clear distinction between open and closed difficult. Furthermore, it is not necessary to model the upstream and downstream movements of the operators explicitely in order to control the risk of conveyor stoppage or the cost for utility work (see, e.g., Yano and Rachamadugu, 1991; Bard, Shtub and Josh, 1994; Tsai, 1995; Bolat, 1997; Kim, Hyun, and Kim, 1996).

So far, only few papers deal with the car sequencing model which is known to be NP-hard in the strong sense (see Kis, 2004). Constrained logic programming approaches have been proposed by Parello, Kabat and Wos (1986), Dincbas, Simonis and van Hentenryck (1988), Parello (1988) and (Drexl and Jordan, 1995). Unfortunately, the performance of these approaches in general is totally disappointing. Drexl and Jordan (1995) provide limited computational results on the basis of an implementation in the constraint programming language, CHARME. The result is that even very small instances might take minutes on a workstation. As shown in this reference also, to use standard MIP-solvers is impractical too. Drexl and Kimms (2001) show how to combine the car sequencing and level scheduling problem within an integrated model (see also Section 4).

3. An algorithm for solving the car sequencing problem

In this section, we present an algorithm for deciding whether a feasible solution exists for a given instance of the car sequencing problem (*CS* for short) and how it looks like. To this end we describe in Section 3.1 a branching mechanism for the construction of a (feasible) sequence. Then we present in Section 3.2 bounding techniques which are able to prune large parts of the search tree in early stages of the branching process.

3.1. Branching

The basic idea for the construction of a sequence σ is to assign successively each period $t \in \{1, \ldots, T\}$ to a variant copy of type v. Doing so, we handle σ as a function $\sigma : \{1, \ldots, T\} \to \{1, \ldots, V\} \cup \{0\}$. The zero is added to avoid partial sequences by the assumption that any yet non-assigned period is mapped onto the void variant 0. This simplifies further definitions without constraining them.

According to the construction of σ, the idea for branching the problem *CS* considered here is to find an assignment for the least non-assigned period t to a variant copy of type v, i.e. to the leftmost period t where $\sigma(t) = 0$. Because there exist at most V possible assignments for such a period t, the problem is branched into at most V subproblems, each with domain's cardinality decreased by one.

All subproblems not being fully explored are kept in the *candidate list*. The well-known last-in-first-out (LIFO) principle is used for guiding the search. By definition, a subproblem is explored, if (i) it has an empty solution space or if (ii) it represents a feasible solution. (An additional condition (iii) which relates to the calculation of bounds for the objective function is described in Section 4.2.)

Any subproblem is a problem *CS* like in Formulation 1, additionally restricted by the current state of construction of a sequence σ arising from branching. Because, in that way, any subproblem corresponds to a partial sequence σ, it is fully described by the definition of σ. A formal description of a subproblem as well as of branching a subproblem is given in the definitions given later.

A sequence σ induces an $O \times T$-matrix M which represents the periods of the planning horizon T, and each row of M is incident with an option j. The interpretation of M is as follows:

$$m_{j,t} = \begin{cases} 1 & \text{if option } j \text{ is planned in period } t, \text{ respecting restriction } H_j : N_j \\ 0 & \text{if option } j \text{ may be planned in period } t \\ -1 & \text{if option } j \text{ is not planned in period } t \text{ or, if planned, it would} \\ & \text{violate restriction } H_j : N_j \end{cases}$$

Because of the requirement of feasibility we derive the following.

Definition 1. A row r_j of matrix M is called admissible, if

$$\sum_{i=t}^{i+N_j-1} \max\{0, m_{j,i}\} \le H_j \quad \text{for all } 1 \le t \le T - N_j + 1.$$

A row r_j of matrix M is said to be complete, if r_j is admissible and

$$\sum_{t=1}^{T} \max\{0, m_{j,t}\} = \sum_{\{v \in \{1,\ldots,V\} | j \in opt(v)\}} D(v)$$

additionally holds.

The next definition contains the description of a subproblem, consisting of its corresponding sequence σ and the matrix M induced by σ.

Definition 2. Let s and r denote a column and a row of M, respectively. A state of construction of a sequence σ, referred to as CS-state, is given by a tuple $\xi = \langle \sigma_\xi, M_\xi \rangle$, where

(i) $\forall t \in \{1, \ldots T\}\big(\sigma_\xi(t) \neq 0 \Rightarrow s_t^{(\xi)} = \bar{v}(\sigma_\xi(t))\big)$ and

(ii) $\forall j \in \{1, \ldots, O\} \left(\sum_{t=1}^{T} \max\{0, m_{j,t}^{(\xi)}\} \leq \sum_{\{v \in \{1,\ldots,V\} | j \in opt(v)\}} D(v) \right)$.

Let the set of all CS-states be denoted by Ξ.

As already mentioned, a *CS*-state is identical to a subproblem of *CS*, hence we will refer to *CS*-states instead of subproblems of *CS*. In fact, this definition allows a *CS*-state to contain more information than that we can obtain from simple branching, because $m_{j,t}^{(\xi)} = 0$ in case of $\sigma_\xi(t) = 0$ is not demanded.

Definition 3. Let $\xi, \xi' \in \Xi$. Both CS-states are equivalent if and only if the corresponding subproblems have the same solution space.

Remark 1. In particular, the CS-state $\xi_I = \langle \sigma, M \rangle$ with $\sigma(t) = 0$, $1 \leq t \leq T$, and $m_{j,t} = 0$, $1 \leq j \leq O$, $1 \leq t \leq T$, is called initial state.

Obviously, as a consequence of the general description of a *CS*-state ξ, it is allowed that some coefficients of a certain column $s_t^{(\xi)}$ of M_ξ with $\sigma_\xi(t) = 0$ are fixed to 1, others to -1, and some are not fixed at all, i.e. they have value 0. For that reason, the choice of an assignment of the period corresponding to column $s_t^{(\xi)}$ is not arbitrary. The next definition points out which variants are allowed to be assigned to column $s_t^{(\xi)}$.

Definition 4. Let $\xi \in \Xi$. A variant v is called compatible with a column $s_t^{(\xi)}$ of M_ξ, denoted by $v \Delta s_t^\xi$, by

$$\forall j \in \{1, \ldots, O\}\big(\bar{v}(v)_j = 1 \Rightarrow \big(s_t^{(\xi)}\big)_j \neq -1 \ \vee \ \bar{v}(v)_j = -1 \Rightarrow \big(s_t^{(\xi)}\big)_j \neq 1\big).$$

Furthermore, the definition of *CS*-states does not prevent any option restriction $H_j : N_j$ to be violated. Therefore, we introduce

Definition 5. A CS-state ξ is said to be allowed if the following conditions hold:

(i) All rows of M are admissible.
(ii) $\forall t \in \{1, \ldots, T\}(\sigma_\xi(t) = 0 \Rightarrow \big\{v \in V \mid v \Delta s_t^{(\xi)}\big\} \cap \{v \in V | \ |\sigma_\xi^{-1}(v)| < D(v)\} \neq \emptyset)$.

While the first condition relates to feasibility, the second one ensures that at least one subproblem exists for any non-assigned period t. Conversely, a *CS*-state being not allowed provides a very weak condition for the corresponding subproblem to be fully explored.

In case the problem *CS* has a non-empty solution space, the following is obvious:

Remark 2. The initial state is allowed.

There are several possible definitions for a *CS*-state ξ to be a solution. As they are all equivalent, we will spot the following

Definition 6. A problem ξ_S is a solution if all rows of M_ξ are complete and $\sigma_\xi^{-1}(0) = \emptyset$ holds.

Finally, we give a formal definition of branching with respect to the notation of *CS*-states.

Definition 7. Let $\xi, \xi' \in \Xi, \xi$ be allowed. Branching the *CS*-state ξ is performed by a transformation

$\alpha : \Xi \times \{1, \ldots, T\} \times \{1, \ldots, V\} \to \Xi, \quad (\xi, t_0, v) \mapsto \xi'$

where $v \Delta s_{t_0}^{(\xi)}$ and $t_0 = \min\{1 \le t \le T | \sigma_\xi(t) = 0\}$.

In analogy of branching a subproblem, we will refer to a transformation α as branching step or *b-step*. Instead of $\alpha(\xi, t, v) = \xi'$ we write $\xi \vec{v} \xi'$ and $\xi \to \xi'$ in case of unknown v. Then $\xi \overset{\rightarrow}{*} \xi'$ denotes a sequence of *b*-steps of arbitrary length.

As seen in the next example, a *b*-step $\xi \to \xi'$ might lead to a *CS*-state ξ' which is not allowed. In connection with the construction of feasible sequences, those steps shall be called *destructive*. To be more general, we define

Definition 8. Let $\xi, \xi' \in \Xi, \xi$ be allowed. A *b*-step $\xi \to \xi'$ is destructive, if there does not exist any solution ξ_S which can be obtained by $\xi' \overset{\rightarrow}{*} \xi_S$.

Example 2. Recall Example 1 and assume that a sequence of *b*-steps $\xi_I \overset{\rightarrow}{*} \xi \overset{\rightarrow}{6} \xi'$ has produced the *CS*-state given in Table 4. By definition, ξ' is not allowed, because none of the variant types $\{2, 6, 7\}$, which are the only ones with non-assigned copies, offers a feasible assignment for period 10. $\xi \overset{\rightarrow}{6} \xi'$ is a destructive step.

Identifying *b*-steps to be destructive as early as possible is of major significance in view of the vast search space. To reduce this search space, in Section 3.2 we concentrate on detecting destructive steps in early stages of the construction of a sequence σ.

3.2. Pruning the search tree

The techniques presented now are based on information which can be deduced from a *CS*-state ξ. To this end, first we present in Section 3.2.1 a polynomial algorithm for the specification of the matrix M_ξ using state-inherent information. Second, a necessary condition for the construction

Table 4 Destructive step

Period	1	2	3	4	5	6	7	8	9	10	11	12
Variant	2	4	5	2	3	4	1	4	6	0	0	0
1:4			×						×			
1:6							×					
2:5	×			×			×		×			
1:2			×		×		×					

of an allowed completion of σ is given in Section 3.2.2. Third, we show in Section 3.2.3 how to learn from failure steps. Overall this information will be used to design an algorithm which explores a whole subtree in polynomial time. The following definitions are prerequisites of what follows.

Definition 9. Let $\xi, \xi' \in \Xi$. A matrix M'_ξ is said to be a specification of M_ξ, if

$$\forall j \in \{1, \ldots, O\}, \quad \forall t \in \{1, \ldots, T\}\big(m_{j,t}^{(\xi)} \neq m_{j,t}^{(\xi')} \Rightarrow m_{j,t}^{(\xi)} = 0\big).$$

If M'_ξ is a specification of M_ξ, we say that M'_ξ is more specified than is M_ξ and denote this by $M_\xi \preceq_s M'_\xi$.

Definition 10. Let $\xi \in \Xi$. We say that there exists a completion of ξ, if $\xi \overset{\rightarrow}{*} \xi_S$ and ξ_S is a solution. If completion of ξ exists, we denote this by the predicate *compl*(ξ).

Each of the three steps outlined above is explained in the subsequent sections.

3.2.1. Specification method

Each allowed *CS*-state ξ carries some state-inherent information. This information can be used to formulate a method for the specification of M_ξ. The core functions of such a method arise from the fact that all the occurrences of an option j have to take place in a row r_j of M_ξ without violating the restriction $H_j : N_j$. More precisely, for each option occurrence, a range of periods exists within which it must be sequenced.

In a totally unspecified matrix M_{ξ_I}, an option range is an interval of the planning horizon with an earliest due date as left limit and a latest due date as right limit. Let

$$occ(j) = \sum_{\{v \in V \mid j \in opt(v)\}} D(v)$$

denote the number of occurrences of option j in the sequence. Then the earliest due date is calculated by the function

$$edd : \{1, \ldots, O\} \times \mathbb{N} \to \mathbb{N} \quad \text{where} \tag{3}$$

$$(j, i) \mapsto \begin{cases} \lfloor \dfrac{i}{H_j} \rfloor \cdot N_j + (i \bmod H_j) & \text{if } i \bmod H_j \neq 0 \text{ and } i \leq occ(j) \\ \lfloor \dfrac{i}{H_j} - 1 \rfloor \cdot N_j + H_j & \text{if } i \bmod H_j = 0 \text{ and } \leq occ(j) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Because of symmetry the right interval limit is calculated by

$$ldd : \{1, \ldots, O\} \times \mathbb{N} \to \mathbb{N} \text{ where} \tag{4}$$

$$(j, i) \mapsto \begin{cases} T - edd(occ(j) - i + 1) + 1 & \text{if } i \leq occ(j) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Remark 3 follows immediately.

Remark 3. An empty interval $[edd(j,i), ldd(j,i)]$ verifies non-existence of a solution.

Definition 11. The allowed option ranges are determined for an arbitrary *CS*-state ξ by

$$range : \{1,\ldots,O\} \times \mathbb{N} \times \Xi \longrightarrow \mathbb{N}^{\{1,\ldots,T\}} \quad \text{where} \tag{5}$$

$$(j,i,\xi) \mapsto \Bigg\{ t \in [edd(j,i), ldd(j,i)] \,|\, \sigma(t) = 0 \wedge m_{j,t} \neq -1 \wedge \forall \max\{1, t - N_j + 1\} \leq t' \leq t : \sum_{\tau=t'}^{t'+N_j-1} \max\{0, m_{j,\tau}\} < H_j \Bigg\}.$$

Remark 4. Whenever range$(j,i,\xi) = \theta$, the solution space of ξ is empty.

The function *range* is a reduction of the interval given by the earliest and the latest due dates to a subset of periods in which an option occurrence can be planned without violating the option restriction while paying attention to the current specification of M_ξ. It is easy to see that *range* is a monotonous function in the sense of

Remark 5. Let $\xi, \xi' \in \Xi$. Then

$$M_\xi \preceq_s M'_\xi \Rightarrow range(j,i,\xi) \supseteq range(j,i,\xi')$$

holds for all $1 \leq j \leq O, 1 \leq i \leq occ(j)$.

By means of option ranges, first a specification of M_ξ is performed by the following rules:

R1 if $range(j,i,\xi)$ contains exactly one element t_0 with $\sigma_\xi(t_0) = 0$ (i.e. an occurrence i of option j must be installed in period t_0), mark this in M_ξ by setting $m^{\xi}_{j,t_0} = 1$;

R2 if $\sigma_\xi(t_0) = 0 \wedge t_0 \notin \bigcup_{1 \leq i \leq occ(j)} range(j,i,\xi)$ holds for a period $1 \leq t_0 \leq T$ (i.e. an occurrence i of option j planned in period t_0 would violate the option restriction), mark this in M_ξ by setting $m^{(\xi)}_{j,t_0} = -1$;

R3 for a remaining period t_0 with $\sigma(t_0) = 0$ we set $m^{(\xi)}_{j,t} = 0$.

Such a specification of M_ξ effects the possible assignments of all non-assigned periods to variant copies, and consequently the possibilities of branching not only in state ξ, but also in the whole subtree with root ξ. Because the number of copies of each variant type v is limited by its demand $D(v)$, we can calculate these possibilities by

$$possV : \{1,\ldots,T\} \times \Xi \longrightarrow 2^{\{1,\ldots,V\}} \quad \text{where} \tag{6}$$

$$(t,\xi) \mapsto \begin{cases} possV(t,\xi') & \text{if } \sigma_\xi(t) \neq 0 \\ \left\{ v \in V \,\middle|\, v\Delta s^{(\xi)}_t \wedge \left|\sigma^{-1}_\xi(v)\right| < D(v) \right\} & \text{otherwise} \end{cases}$$

Here it is assumed that the *CS*-state ξ has been reached by a *b*-step $\xi' \to \xi$. Whenever $\alpha_\xi(t) \neq 0 \wedge |possV(t,\xi) > 1$ holds for a *CS*-state ξ, the construction of σ_ξ can proceed alternatively. Therefore, the elements of $possV(t,\xi)$ shall be called *alternatives*.

Of course, the more M_ξ is specified, the more the sets of possible assignments are restricted. It is easy to see that a further specification of M_ξ monotonously decreases the cardinality of the sets of alternatives obtained from $possV$, formally stated as

Remark 6. Let $\xi, \xi' \in \Xi$ Then

$$M_\xi \preceq_s M'_\xi \Rightarrow possV(t,\xi) \supseteq possV(t,\xi')$$

holds for all $1 \leq t \leq T$.

Second the monotonicity of $possV$ justifies to continue the specification of M_ξ by the following rules:

R4 if $possV(t_0,\xi)$ contains exactly one variant type compatible to column $s_{t_0}^{(\xi)}$, assign a copy of this variant type to period t_0:

$$\forall t_0 \in \{1,\ldots,T\}\big(|possV(t_0,\xi)| = 1 \wedge \sigma_\xi(t_0) = 0 \Rightarrow s_{t_0}^{(\xi)} = \bar{v}(v)\big)$$

R5 if all variant types in $possV(t_0,\xi)$ need an option of type j to be installed, insert this option in period t_0:

$$\forall t_0 \in \{1,\ldots,T\}(\exists j_0 \in \{1,\ldots,O\}(\forall v \in possV(t_0,\xi))$$
$$\big(j \in opt(v) \wedge \sigma_\xi(t_0) = 0 \Rightarrow m_{j_0,t_0}^{(\xi)} = 1\big),$$

R6 vice versa, if none of the variant types in $possV(t_0,\xi)$ need an option of type j to be installed, reject this option in period t_0:

$$\forall t_0 \in \{1,\ldots,T\}(\exists\, j_0 \in \{1,\ldots,O\}(\forall\, v \in possV(t_0,\xi))$$
$$\big(j \notin opt(v) \wedge \sigma_\xi(t_0) = 0 \Rightarrow m_{j_0,t_0}^{(\xi)} = -1\big),$$

R7 if there are exactly n non-assigned periods which variant type v_0 can be assigned to, and there are exactly n copies of v_0 left-over for assignment, then assign those periods to variant type v_0:

$$\text{let } A_v := \{t \in \{1,\ldots,T\} | \sigma(t) = 0 \wedge v_0 \in possV(t,\xi)\}; \text{then}$$
$$\exists v_0 \in \{1,\ldots,V\}\big(|A_{v0}| = D(v_0) - \big|\sigma_\xi^{-1}(v_0)\big| \Rightarrow (\forall t \in A_{v0})\big(s_t^{(\xi)} = \bar{v}(v_0)\big)\big).$$

The specification rules R4–R7 can be formulated as polynomial operations on a matrix M_ξ. Applied to M_ξ, the operations may lead to a more specified $M'_\xi \succeq_s M_\xi$ which incurs the necessity of a repeated application of *range* and $possV$. Consequently, the specification of M_ξ is an iterative process which can be expressed in pseudo code as follows:

```
1 repeat
2   calculate function range
3   apply rules R1–R3
4   calculate possV
```

Table 5 Value of the function *range* in the initial *CS*-state

	Occurrence ranges					
Option	1	2	3	4	5	6
1	[1:4]	[5:8]	[9:12]			
2	[1:12]					
3	[1:1]	[2:2]	[6:6]	[7:7]	[11:11]	[12:12]
4	[1:4]	[3:6]	[5:8]	[7:10]	[9:12]	

Table 6 First-level specification of the initial state

t	1	2	3	4	5	6	7	8	9	10	11	12
$\sigma(t)$	0	0	0	0	0	0	0	0	0	0	0	0
possV	1,2	1,2	3,4	3,4	3,4	1,2	1,2	3,4	3,4	3,4	1,2	1,2
	6,7	6,7	5	5	5	6,7	6,7	5	5	5	6,7	6,7
1:4												
1:6												
2:5	×	×	–	–	–	×	×	–	–	–	×	×
1:2												

5 apply rules R4–R7
6 until none of the rules R4–R7 effects M_ξ

This iteration runs until none of the matrix elements changed its value, i.e. some kind of fix point M_ξ^* has been reached. Because, as aforementioned, *possV* decreases monotonously in the cardinality of its image by specification of M_ξ, such a fix point exists.

A *CS*-state $\xi = \langle \sigma_\xi, M_\xi \rangle$ is equivalent to the state $\xi' = \langle \sigma_{\xi'}, M_\xi^* \rangle$ obtained from a specification of ξ. However, ξ' provides a smaller search space than ξ.

Example 3. The effect of rules R1–R7 can be seen in Example 1. First, we calculate the option ranges in the initial state ξ_I, according to (5) (see Table 5). From this we derive the possible assignments for each period according to (6), and we obtain the sequence provided in Table 6. The row representing the third option type is already fixed, according to rules R1 and R2. Having performed branching $\xi_I \vec{2} \xi_1 \vec{1} \xi_2$ which produces a sequence with initial part $\sigma(1) = 2$ and $\sigma(2) = 1$, the iterative process continues (see Table 7). After branching only twice and three iterations of the specification method a solution is found. From periods 3 to 12, all assignments leading to destructive steps are skipped.

3.2.2. A condition on compl(ξ)

A *CS*-state ξ with M_ξ^* being obtained from the specification method can be used in order to decide whether it is worth continuing σ_ξ constructed so far or to drop ξ from further consideration. To do so, we use a condition $\Im(\xi)$ which is necessary for the completion of σ_ξ, that is,

$$compl(\xi) \Rightarrow \Im(\xi). \tag{7}$$

The idea is that, once $\neg\Im(\xi)$ holds, by contradiction no solution ξ_S exists which can be obtained by further exploration of ξ. Being not allowed, as a predicate on *CS*-states, is a rather poor

Table 7 Problem solved by specification

it	*t*	1	2	3	4	5	6	7	8	9	10	11	12
	$\sigma(t)$	2	1	4	0	0	0	0	0	0	0	0	0
	possV	1,2	1,2	4	3,4	3,4	2,6	2,6	3,4	3,4	3,4	2,6	2,6
		6,7	6,7		5	5	7	7	5	5	5	7	7
1	1:4	–	–	–									
	1:6	–	×	–	–	–	–	–	–	–	–	–	–
	2:5	×	×	–	–	–	×	×	–	–	–	×	×
	1:2	–	×	–									
	$\sigma(t)$	2	1	4	5	0	0	0	5	0	0	0	6
	possV	1,2	1,2	4	5	3,4	2,7	2,7	5	3,4	3,4	2,7	6
		6,7	6,7										
2	1:4	–	–	–	×	–	–	–	×	–	–	–	×
	1:6	–	×	–	–	–	–	–	–	–	–	–	–
	2:5	×	×	–	–	–	×	×	–	–	–	×	×
	1:2	–	×	–	×				×				–
	$\sigma(t)$	2	1	4	5	4	7	2	5	4	3	2	6
	possV	1,2	1,2	4	5	4	7	2	5	4	3	2	6
		6,7	6,7										
3	1:4	–	–	–	×	–	–	–	×	–	–	–	×
	1:6	×	–	–	–	–	–	–	–	–	–	–	–
	2:5	×	×	–	–	–	×	×	–	–	–	×	×
	1:2	–	×	–	×	–	×	–	×	–	×	–	–

example for such a condition. The problem is to find a condition $\Im(\xi)$ strong enough to predict an empty solution space in early stages of the construction of σ. This is the topic of what follows.

From Remarks 3 and 4 and the definition of allowed *CS*-states we know that a solution of σ cannot be obtained from state ξ, if one of the following two conditions is valid:

C1 $(\exists j \in \{1, \ldots, O\})(\exists i \in \{1, \ldots, occ(j)\})(range(j, i, \xi) = \emptyset)$
there exists an option occurrence of type j which cannot be inserted into M without violating the $H_j : N_j$ constraint,

C2 $(\exists t \in \{1, \ldots, T\})(possV(t, \xi) = \emptyset)$
there is a yet non-assigned period no variant type is compatible with.

C2 is only a consequence of the fact that there were less periods a variant type v could be assigned to than the number of copies of type v still to assign in some state ξ', where $\xi \rightarrow \xi$. So we modify this condition to

C2′ $(\exists v \in \{1, \ldots, V\})$

$$\left(D(v) - \left|\sigma_\xi^{-1}(v)\right| > |\{t \in \{1, \ldots, T\} | \sigma_\xi(t) = 0 \wedge v \in possV(t, \xi)\}|\right)$$

which is able to predict occurrences of C2.

First, we derive a condition $\Im^{(1)}(\xi)$ as follows: Let $r(v, \xi)$ be used in order to abbreviate $D(v) - |\sigma_\xi^{-1}(v)|$ and let $\#_{j,\xi}$ denote the number of occurrences already inserted into M_ξ, that is,

$$\#_{j,\xi} = \sum_{t=1}^{T} \max\{0, m_{j,t}\}.$$

Then we define

$$\begin{aligned}\Im^{(1)}(\xi) = (\forall v \in \{1,\ldots,V\})\\ (|\{t \in \{1,\ldots,T\} | \sigma_\xi(t) = 0 \wedge v \in possV(t,\xi)\}| \geq r(v,\xi))\\ \wedge\ (\forall j \in \{1,\ldots,O\})(\forall \#_{j,\xi} + 1 \leq i \leq occ(j))\\ (\#_{j,\xi} \leq occ(j) \Rightarrow range(j,i,\xi) \neq \emptyset).\end{aligned}$$

To evaluate $\Im^{(1)}(\xi)$, we introduce the set of non-assigned periods

$$U_\xi = \sigma^{-1}(0)$$

in state ξ and the sets of non-assigned periods

$$U_\xi^v = \{t \in U_\sigma | v \in possV(t,\xi)\}, \quad 1 \leq v \leq V$$

in which a variant copy of type v is an alternative.

Example 4. Consider a state ξ with $U_\xi \supset \{t_{i_1}, t_{i_2}, t_{i_3}, t_{i_4}\}$,

$$possV(t_{i_1},\xi) \supseteq \{1,2\}, \quad possV(t_{i_2},\xi) \supseteq \{1,2\},$$

$$possV(t_{i_3},\xi) \supseteq \{1,2\}, \quad possV(t_{i_4},\xi) \supseteq \{2\},$$

and $\forall t \notin \{t_{i_1}, t_{i_2}, t_{i_3}, t_{i_4}\} : 1 \notin possV(t,\xi), 2 \notin possV(t,\xi)$. Moreover, let $r(1,\xi) = 3$ and $r(2,\xi) = 2$.

Provided the fact that C1 does not hold, the condition $\Im^{(1)}(\xi)$ is valid, because

$$U_\xi^1 = 3 \geq 3 = r(1,\xi) \wedge U_\xi^2 = 4 \geq 2 = r(2,\xi),$$

though there does not exist any completion of the sequence. In the worst case, having scheduled $\sigma(t_{i_1}) = \sigma(t_{i_2}) = \sigma(t_{i_3}) = 1$, $\Im^{(1)}(\xi')$ becomes invalid after having performed three more b-steps, reaching state ξ'. Consequently, $\Im^{(1)}(\xi)$ is too weak.

In Example 4, the invalidity of $compl(\sigma_\xi)$ could have been noticed in an earlier state by calculating

$$r(1,\xi) + r(2,\xi) = 5 \wedge \left|U_\xi^1 \cup U_\xi^2\right| = 4,$$

showing that the set of non-assigned periods $U_\xi^1 \cup U_\xi^2$ shared by both variants is not capable to pick up the remaining copies of those variants. In view of the monotonicity of *possV*, this capability is necessary for all subsets of variants, so C2′ is modified to

$$C2''\left(\exists K \in 2^{\{1,\ldots,V\}}\right)\left(\left|\bigcup_{k\in K} U_\xi^k\right| < \sum_{k\in K} r(k,\xi)\right)$$

and, second, we strengthen the condition $\Im$ by the negation of C2″

$$\Im^{(2)}(\xi) = \left(\forall K \in 2^{\{v \in V \mid r(v,\xi)>0\}}\right) \tag{8}$$

$$(K \neq \emptyset \Rightarrow \left|\bigcup_{k\in K} U_{\xi}^{k}\right| \geq \sum_{k\in K} r(k,\xi)) \tag{9}$$

$$\wedge\ (\forall j \in \{1,\ldots,O\})(\forall i \in \{\#_{j,\xi}+1,\ldots,occ(j)\}) \tag{10}$$

$$(\#_{j,\xi} \leq occ(j) \Rightarrow range(j,i,\xi) \neq \emptyset). \tag{11}$$

Once $\neg\Im^{(2)}(\xi)$ holds, ξ is fully explored, even in case further branching is possible. Unfortunately, validating $\Im^{(2)}(\xi)$ requires exponential effort. Nevertheless, $\Im^{(2)}(\xi)$ can be used partially to predict destructive steps. For example, $\Im^{(2)}(\xi)$ can be evaluated for singletons and pairs of variants with complexity $O(V)$ and $O(V^2)$, respectively. The question whether $\Im^{(2)}(\xi)$ is also sufficient for $compl(\xi)$ is answered negative, as shown in the following example.

Example 5. Consider an instance of CS with $T = 5$, $V = 3$, $D(v) = (3, 1, 1)$ and $O = 3$. Furthermore, we have the option restrictions $\{1:3; 1:3; 1:4\}$ and the option sets $opt(1) = \{1\}$, $opt(2) = \{2, 3\}$ and $opt(3) = 2$. Then the option ranges are

$range(1,1) = [1]$, $range(2,1) = [1,2]$,

$range(1,2) = [3]$, $range(2,2) = [4,5]$,

$range(1,3) = [5]$, $range(3,1) = [1,5]$.

It follows

$possV(1,\xi_I) = 1$, $possV(3,\xi_I) = 1$, $possV(5,\xi_I) = 1$,

$possV(2,\xi_I) = 2,4$, $possV(4,\xi_I) = 2,4$,

and it is easy to see that

$$\forall K \in 2^{\{1,\ldots,3\}} \left|\bigcup_{k\in K} U_{\xi_I}^{k}\right| \geq \sum_{k\in K} D(k)$$

holds. However, there exists no solution, because the copies of variant 1 must be scheduled in periods 1, 3 and 5, and variants 2 and 3 must be scheduled in periods 2 and 4 (violating the option restriction 1:3, because both request option 2).

Hence, we see this instance to be unsolvable by one application of the specification method.

3.2.3. Learning from failure steps

The methods presented so far help to detect and exclude destructive steps during the construction of σ. Additionally, learning from failure steps can be used in order to avoid repeat failures, once they have been done. Whenever $\Im^{(2)}(\xi)$ helps to identify that the construction runs into a

destructive step, such that a completion of σ cannot be obtained from state ξ, this state can be dropped and we track back according to the LIFO principle.

Consider a state ξ for which $\Im^{(2)}(\xi)$ does not hold, and assume an occurrence of option j with $range(j, i, \xi) = \emptyset$. Further, let ξ' be the latest state with at least one alternative not yet explored. The set of non-explored alternatives in ξ' may be denoted by A and the assignment performed in ξ' by $\sigma(t') = v$.

Now we have to differentiate between two cases:

(1) $j \in opt(v)$
Though an occurrence of option j has been inserted into M_ξ in period t', an empty option range for another occurrence of j results from specification of $M_{\xi'}$; hence, it is useless to try an alternative in period t', since any of the alternatives would lead to the same result because of the monotonicity of *range*. Therefore, we set $possV(t', \xi') = \emptyset$ and choose the next step from the candidate list immediately.

(2) $j \notin opt(v)$
Then there may be a solution only when inserting an occurrence of option j in period t'; therefore, we set $possV(t', \xi') = possV(t', \xi') - \{v \in V | j \notin opt(v)\}$ and proceed from ξ' with a variant of the updated smaller set of alternatives.

Here, the incapability to schedule an option occurrence is utilized to cut branches of the search tree which, otherwise, would have been recognized as destructive only later.

4. Computing optimal level schedules

The topic of leveling scheduling is to keep the quantity of each product manufactured per time unit as close as possible to the demand for that product. Monden (1998) attached to his description of Toyota's production system, published in 1983, two scheduling algorithms named *Goal Chasing I* and *Goal Chasing II*. The first one considers the minimal mean squared deviation between expected and actual accumulated component usage while the second one is a simplification of the first one aiming at reducing computation time. Miltenburg (1989) presents a nonlinear integer programming formulation with the objective of minimizing the deviation between current and desired production rates. Kubiak and Sethi (1991) introduce a variant, which reduces the scheduling algorithm to an assignment problem. Inman and Bulfin (1991) give a formulation of the problem as a minimization problem in which the objective is to minimize the sum of deviations between current positions of the copies from the ideal ones in the sequence.

Usually, leveling scheduling does not take care of the line's workstations loads, that is, the constraints of the car sequencing problem are not taken into account. By contrast, it is shown in the following how to efficiently compute optimal level schedules without relaxing the car sequencing constraints.

4.1. Objective function

In the sequel, we will adopt the approach of Inman and Bulfin (1991). They proposed a formulation which can be solved to optimality in polynomial time—at the price of relaxing the car sequencing constraints—by ordering the copies according to the earliest due date (EDD) rule.

Formally, the formulation of Inman and Bulfin (1991) can be described by the following:

Formulation 2. Let the ideal position be given by a function

$$dd : \{1, \ldots, V\} \times \mathbb{N} \to \mathbb{R} \quad where$$

$$(v, i) \mapsto \begin{cases} \dfrac{(i - \frac{1}{2}) \cdot T}{D(v)} & \text{if} \quad i \leq D(v) \\ \infty & \text{otherwise.} \end{cases}$$

Then the level scheduling model is

$$\min \quad \sum_{v=1}^{V} \sum_{i=1}^{D(v)} \|\delta^{-1}(v, i) - dd(v, i)\|$$

$$\text{s.t.} \quad \delta : \{1, \ldots, T\} \mapsto \{1, \ldots, V\} \times N.$$

Apparently, this formulation is different from, but equivalent to the formulation given by Inman and Bulfin (1991). The objective minimizes the sum of deviations of the scheduled periods from the optimal ones given by dd, using an arbitrary l_p-norm. If not mentioned otherwise, we will consider the absolute norm. Hence, it is clear that in an optimal solution (function δ) none of the periods of the planning horizon is mapped onto a variant copy with index greater than $D(v)$. As mentioned previously, there exists a construction method for an optimal solution which is performed in polynomial time. This method is explained now.

4.2. Lower bound

In addition to conditions (i) and (ii) introduced in Section 3.1 a subproblem is explored, if (iii) the expected value of the objective function is worse than the best one known so far. To estimate the objective value of any feasible completion of σ_ξ in an arbitrary *CS*-state ξ, lower bounds are calculated for a partial sequence σ_ξ by using a relaxation of *CS*. The relaxation considered here is to drop constraints (2). The objective value of any feasible sequence σ_{ξ_S} is an upper bound for the optimal objective. We calculate lower bounds $lb(\xi)$ in a *CS*-state ξ by solving the level scheduling problem given in Formulation 2 with input data restricted to $T' = \sigma_\xi^{-1}(0)$ and $D'(v) = D(v) - |\sigma_\xi^{-1}(v)|$ for all $v \in V$, and add the obtained value to the current objective of σ_ξ

$$lb(\xi) = \sum_{t=1}^{T} \|t - dd(v_0, i)\|$$

where for each period $t v_0$ is either obtained from σ_ξ in case $\sigma_\xi(t) \neq 0$, or v_0 is the variant determined by the method of Inman and Bulfin (1991), otherwise. Now, we present the construction method for the level scheduling model.

Copy (v, i) causes objective cost only if it is produced at a period t which is not equal to the ideal position calculated by $dd(v, i)$. Let us consider the difference between ideal position and planning period as cost measure, that is

$$c : \{1, \ldots, T\} \times \{1, \ldots, V\} \times N \longrightarrow \mathbb{R} \cup \{\infty\} \quad \text{where}$$

$$(t, v, i) \mapsto \begin{cases} dd(v, i) - t & \text{if } dd(v, i) \neq \infty \\ \infty & \text{otherwise} \end{cases}$$

For illustrative purposes, let us look at the following.

Example 6. Recall Example 1, where the demand $D(v)$ for the variants is given by (1, 3, 1, 3, 2, 1, 1) for $v = 1, \ldots, 7$. Because one copy is produced in each period, the overall demand equals the amount of periods within the planning horizon, that is,

$$T = \sum_{v=1}^{V} D(v) = 12.$$

The ideal positions are

$$dd : (v, i) \mapsto \begin{pmatrix} 6 & \infty & \infty & \infty & \ldots \\ 2 & 6 & 10 & \infty & \ldots \\ 6 & \infty & \infty & \infty & \ldots \\ 2 & 6 & 10 & \infty & \ldots \\ 3 & 9 & \infty & \infty & \ldots \\ 6 & \infty & \infty & \infty & \ldots \\ 6 & \infty & \infty & \infty & \ldots \end{pmatrix}$$

According to the construction method, we obtain the sequence given in Table 8.

The construction of δ^* as the schedule with minimal objective value is as follows: map each period t consecutively onto that copy of a variant with least c-value. Formally, it is denoted by

$$\delta^*(t) := (v_0, i_o), \quad \text{where}$$

$$c(t, v_0, i_0) = \min\{c(t, v, i) | (v, i) \notin \delta([1, \ldots, t-1]).$$

Let Δ denote the set of all sequences $\{1, \ldots, T\} \longrightarrow \{1, \ldots, V\} \times N$. The objective value of a sequence is the sum of the norm of the c-values, i.e.

$$C_\Delta : \Delta \to \mathbb{R} \cup \{\infty\}, \quad \delta \mapsto \sum_{t=1}^{T} \|c(t, \delta(t))\|.$$

Similar to the study by Inman and Bulfin (1991), optimality of δ^* can be shown by proving the nonexistence of another sequence δ' the objective value of which is less than the objective of δ^*.

Table 8 Sequence obtained from level scheduling

Period	1	2	3	4	5	6	7	8	9	10	11	12
Variant	2	4	5	1	2	3	4	6	7	5	2	4
Deviation	1	0	0	2	1	0	1	2	3	1	1	2
objective function value: 14.00												

4.3. Overall algorithm

The performance of the algorithm for computing optimal sequences on the one hand depends on how fast a feasible solution is found, and on the other hand, on the quality of its objective value related to the optimal sequence's objective. As part of the specification method for a *CS*-state ξ, the function *possV* calculates the set of alternatives ξ can be branched into. The suitable choice of an alternative to proceed from ξ allows to lead the construction of σ into a certain direction, for instance in order to obtain a feasible solution fast. Therefore, we need an evaluation of the alternatives for ordering them into a priority queue before adding them to the candidate list.

Three priority rules are suggested here. Assume ξ to be an allowed *CS*-state with least non-assigned period t; then let $possV(t, \xi) = \{v_{i_1}, \ldots, v_{i_k}\}$ be the set of alternatives which can be assigned to period t; $p(v)$ denotes the priority value of v.

PR-1 The first idea for ordering the alternatives $\{v_{i_1}, \ldots, v_{i_k}\}$ arises from the goal to minimize the objective value of σ. Consequently, the next variant copy to assign should be the one which increases the overall objective value of σ least, as done by the level scheduling method by Inman and Bulfin (1991). Therefore, the priority values of the alternatives are calculated by

$$\forall 1 \leq q \leq k : p\left(v_{i_q}\right) = \left\| t - \left(\left(|\sigma^{-1}\left(v_{i_q}\right)|_{\{1,\ldots,t-1\}}| + 1\right) - \frac{1}{2} \right) \cdot \frac{T}{D\left(v_{i_q}\right)} \right\|.$$

A mapping of t to $v^*_{i_q}$ with minimal value $p(v_{i_q})$ takes care that branching the state ξ results in lowest objective cost. Nevertheless, because of the restriction to the variant set given by *possV* this assignment is to be considered as locally cheapest; there may be a permutation of σ with $\sigma(t) \neq v^*_{i_q}$ and smaller sequence objective.

Intending to find a feasible solution fast, the following priority rules are derived from condition $\Im^{(2)}_{\xi}$.

PR-2 The rule is based on the idea to insert option occurrences as early as possible, so the risk for an occurrence to have an empty range is reduced. This is done by

$$\forall 1 \leq q \leq k : p\left(v_{i_q}\right) = \left|opt\left(v_{i_q}\right)\right|.$$

The next variant to assign is $v^*_{i_q}$ with maximal value $p(v_{i_q})$. The rule corresponds to the second clause of condition $\Im^{(2)}(\xi)$.

PR-3 The third rule evaluates an alternative v by the relation between the number of columns v that can be assigned to and the number of its copies to be assigned yet:

$$\forall 1 \leq q \leq k : p\left(v_{i_q}\right) = \frac{\left|\left\{t \leq t' \leq T \,|\, v_{i_q} \in possV(t', \xi)\right\}\right|}{D\left(v_{i_q}\right) - \left|\sigma^{-1}\left(v_{i_q}\right)\right|}$$

If $\Im^{(2)}(\xi)$ holds, then $p(_{i_q}) \geq 1$. Choose $v^*_{i_q}$ to be assigned to t with minimal value $p(v_{i_q})$. This rule corresponds to the first clause of condition $\Im^{(2)}(\xi)$, reducing the risk to run in a *CS*-state where there are lesser columns a variant type v can be assigned to than the number of copies of v still to be scheduled.

While the first rule is expected to lead to a solution of reasonable good quality, the other branching criteria may prevent unnecessary branching which may provide a feasible solution to be found faster. A description of the algorithm can be given as follows:

1 $\xi^* := \langle\rangle$
2 candidateList:= $\langle \xi_I \rangle$
3 while candidateList $\neq \langle\rangle$ do
4 ξ:= getFirstCandidateFromList
5 apply specification method
6 if $\neg\Im(\xi)$ or lb(ξ) $\geq$ value ($\xi*$) then
7 drop ξ from candidateList
8 else
9 if $\sigma^{-1}(0) = \emptyset$ then
10 if $lb(\xi) <$ value (ξ^*) or $\xi^* = \langle\rangle$ then
11 $\xi^* = \xi$
12 endif
13 else
14 $t_0 = \min \sigma^{-1}(0)$
15 A_ξ = possV(t_0, ξ)
16 sort $A_\xi \rightarrow (v_{i_l}, \ldots, v_{i_k})$
17 candidateList:= $\langle \xi \overrightarrow{v_{i_1}} \xi_{i_1}, \ldots, \xi \overrightarrow{v_{i_k}} \xi_{i_k}$; candidateList$\rangle$
18 endif
19 endif
20 end while

When initiating the computation, the candidate list of yet non-explored *CS*-states consists of the initial state only. The best solution known so far is denoted by ξ^*. As long as there are *CS*-states still to explore, we consider the first one from the candidate list and specify it by the method described in Section 3.2. In case the condition $\Im(\xi)$ predicts that the solution space of ξ is empty, or in case ξ can be bounded by its objective value, the CS-state is dropped from further consideration. If ξ is not dropped, then it may be a solution and is eventually stored as the best new solution, otherwise the alternatives of ξ are sorted by their priority values and, according to the LIFO principle, added to the candidate list as being the next *CS*-states to explore. The algorithm stops when every branch of the search tree either is explored or cut off by bounding. If a feasible solution exists, ξ^* contains the solution with minimal objective value.

5. Computational results

The computational evaluation has been performed on a PC with an Intel-350 MHz processor, 64 MB RAM, and operating system Windows™ 98. The algorithm has been implemented by the use of the Delphi™ environment, developer's edition, version 2.01.

To evaluate the algorithms presented previously we used the testbed provided in Drexl and Kimms (2001). The instances have the characteristics shown in Table 9. Note that because of the

Table 9 Characteristics of the testbed

T	*O*	*easy*	*hard*
10	3	1:2,2:5,7:8	1:8,1:7,2:8
	5	1:2,2:5,7:8,3:4,6:7	1:8,1:7,2:8,1:6,2:7
	7	1:2,2:5,7:8,3:4,6:7,2:3,5:6	1:8,1:7,2:8,1:6,2:7,3:8,1:5
15	3	1:2,2:5,7:8	1:8,1:7,2:8
	5	1:2,2:5,7:8,3:4,6:7	1:8,1:7,2:8,1:6,2:7
	7	1:2,2:5,7:8,3:4,6:7,2:3,5:6	1:8,1:7,2:8,1:6,2:7,3:8,1:5
20	3	1:2,2:5,7:8	1:8,1:7,2:8
	5	1:2,2:5,7:8,3:4,6:7	1:8,1:7,2:8,1:6,2:7
	7	1:2,2:5,7:8,3:4,6:7,2:3,5:6	1:8,1:7,2:8,1:6,2:7,3:8,1:5
30	3	1:2,2:5,7:8	1:8,1:7,2:8
	5	1:2,2:5,7:8,3:4,6:7	1:8,1:7,2:8,1:6,2:7
	7	1:2,2:5,7:8,3:4,6:7,2:3,5:6	1:8,1:7,2:8,1:6,2:7,3:8,1:5
40	3	1:2,2:5,7:8	1:8,1:7,2:8
	5	1:2,2:5,7:8,3:4,6:7	1:8,1:7,2:8,1:6,2:7
	7	1:2,2:5,7:8,3:4,6:7,2:3,5:6	1:8,1:7,2:8,1:6,2:7,3:8,1:5
50	3	1:2,2:5,7:8	1:8,1:7,2:8
	5	1:2,2:5,7:8,3:4,6:7	1:8,1:7,2:8,1:6,2:7
	7	1:2,2:5,7:8,3:4,6:7,2:3,5:6	1:8,1:7,2:8,1:6,2:7,3:8,1:5

construction scheme each of the (non-trivial) instances has at least one feasible solution. Various planning horizons T were considered each with 3, 5 and 7 different option types and with two different hardness categories w.r.t. satisfying the restrictions, namely *easy* and *hard*.

The algorithm has been applied three times to all problem instances, each with a different priority rule PR-1, PR-2 and PR-3. As indicated the l_p-norm has been used. The performance of the algorithms has been measured in terms of

- the run-time needed to find a feasible solution, satisfying Formulation 1, together with the number of failure steps (i.e. the number of boundings induced by an empty solution space of a subproblem) until such a solution was found;
- the run-time needed to find an optimal solution, together with the total number of failure steps until the search tree was fully explored;
- the percentage deviation between the objective value of the first solution and the optimal objective value.

The computational results are given in Tables 10 and 11. The tables cover the results for the different priority rules distinctly; the first entry in columns *find* and *opt(imize)* represents the number of failure steps while the second denotes the run-time in *hrs:min:sec*. Entries in the column % denote the deviation between the first and the best solution found. An entry *not comp* indicates that the computation has been aborted because of exceeding the time limit of 3 h.

At first sight, the distinction between *easy* and *hard* problem instances is confirmed when comparing the results, especially when comparing the run-times needed for optimization. This can be explained by the fact that the instance generator tends to produce less variety of variant types with higher demand rates when using *easy* restrictions than it does by the use of *hard*

Table 10 Computational results—easy instances

T	*O*		PR-1			PR-2			PR-3	
10	3	0	0		0	1		0	0	
		0:00:00	0:00:00	0.00	0:00:00	0:00:00	30.77	0:00:00	0:00:00	0.00
	5	0	1		0	0		0	1	
		0:00:00	0:00:00	0.00	0:00:00	0:00:00	18.75	0:00:00	0:00:00	18.75
	7	0	0		0	0		0	0	
		0:00:00	0:00:00	0.00	0:00:00	0:00:00	4.76	0:00:00	0:00:00	0.00
15	3	0	0		0	1		0	1	
		0:00:00	0:00:00	0.00	0:00:00	0:00:00	52.00	0:00:00	0:00:00	52.00
	5	0	33		0	36		0	32	
		0:00:00	0:00:01	32.14	0:00:00	0:00:01	44.12	0:00:00	0:00:01	26.92
	7	3	19		0	7		2	28	
		0:00:00	0:00:00	15.00	0:00:00	0:00:00	26.09	0:00:00	0:00:00	29.17
20	3	0	6		0	4		0	4	
		0:00:00	0:00:00	28.00	0:00:00	0:00:00	21.74	0:00:00	0:00:00	14.29
	5	0	58		0	52		0	68	
		0:00:00	0:00:01	0.00	0:00:00	0:00:01	28.57	0:00:00	0:00:01	26.83
	7	7	141		0	151		1	104	
		0:00:00	0:00:02	6.35	0:00:00	0:00:02	20.27	0:00:00	0:00:01	1.64
30	3	0	73		0	57		2	83	
		0:00:00	0:00:01	17.95	0:00:00	0:00:00	3.03	0:00:00	0:00:01	44.83
	5	0	23		0	23		0	28	
		0:00:00	0:00:01	34.21	0:00:00	0:00:01	25.37	0:00:00	0:00:01	31.51
	7	4	101		0	96		7	89	
		0:00:00	0:00:03	22.40	0:00:00	0:00:03	23.62	0:00:00	0:00:02	16.38
40	3	0	47		0	19		0	31	
		0:00:00	0:00:02	27.27	0:00:00	0:00:02	19.62	0:00:00	0:00:02	39.62
	5	1	788		0	975		4	914	
		0:00:00	0:00:24	4.40	0:00:00	0:00:24	30.40	0:00:00	0:00:24	5.43
	7	55	520		0	381		2	456	
		0:00:01	0:00:11	16.27	0:00:00	0:00:10	22.60	0:00:00	0:00:11	11.41
50	3	0	223		0	165		0	189	
		0:00:00	0:00:11	20.34	0:00:00	0:00:10	24.19	0:00:00	0:00:10	18.97
	5	7	1144		0	1338		0	1199	
		0:00:00	0:01:27	29.10	0:00:00	0:01:34	29.10	0:00:00	0:01:28	46.02
	7	2	4303		0	15782		2	9867	
		0:00:00	0:03:15	34.59	0:00:00	0:11:06	32.97	0:00:00	0:08:05	33.92
task		*find*	*opt*	*%*	*find*	*opt*	*%*	*find*	*opt*	*%*

Table 11 Computational results – hard instances

T	O		PR-1			PR-2			PR-3	
10	3	0	0		0	0		0	0	
		0:00:00	0:00:00	0.00	0:00:00	0:00:00	60.67	0:00:00	0:00:00	60.00
	5	0	0		0	0		0	1	
		0:00:00	0:00:00	0.00	0:00:00	0:00:00	43.48	0:00:00	0:00:00	43.48
	7	0	1		2	3		0	1	
		0:00:00	0:00:00	0.00	0:00:00	0:00:00	19.23	0:00:00	0:00:00	19.23
15	3	0	0		0	0		0	0	
		0:00:00	0:00:00	0.00	0:00:00	0:00:0	68.75	0:00:00	0:00:00	64.29
	5	0	0		0	0		0	3	
		0:00:00	0:00:00	33.33	0:00:00	0:00:01	52.63	0:00:00	0:00:00	53.85
	7	0	19		0	24		0	38	
		0:00:00	0:00:00	25.00	0:00:00	0:00:01	40.00	0:00:00	0:00:01	35.71
20	3	0	0		0	0		0	0	
		0:00:00	0:00:00	0.00	0:00:00	0:00:00	72.73	0:00:00	0:00:00	68.42
	5	0	0		0	80		1	17	
		0:00:00	0:00:01	31.43	0:00:00	0:00:05	51.02	0:00:00	0:00:01	47.83
	7	1	3382		0	5169		4	2084	
		0:00:00	0:00:58	21.43	0:00:00	0:01:01	43.59	0:00:00	0:00:57	33.33
30	3	0	0		0	0		0	0	
		0:00:00	0:00:00	0.00	0:00:00	0:00:01	69.84	0:00:00	0:00:01	65.45
	5	0	15765		0	73164		7	9669	
		0:00:00	0:04:02	25.71	0:00:00	0:16:02	44.68	0:00:00	0:04:07	35.00
	7	4	390611		0	not		530	not	
		0:00:00	2:37:54	15.52	0:00:00	comp		0:00:07	comp	28.37
40	3	0	6		0	0		0	112	
		0:00:00	0:00:00	0.00	0:00:00	0:00:01	57.50	0:00:00	0:00:05	52.78
	5	1	310066		0	126966		22	not	
		0:00:00	1:58:15	23.23	0:00:00	1:46:16	44.79	0:00:00	comp	
	7	408	552160		0	not		1728	not	
		0:00:10	4:32:37	16.84	0:00:00	comp		0:00:45	comp	
50	3	0	14271		0	5963		0	9608	
		0:00:00	0:03:53	23.94	0:00:00	0:05:47		0:00:00	0:03:55	
	5	0	415981		0	not		0	not	
		0:00:00	2:52:44	37.82	0:00:00	comp		0:00:00	comp	
	7	28862	not		33	not		11864	not	
		0:10:16	comp		0:00:01	comp		0:05:26	comp	
task		*find*	*opt*	%	*find*	*opt*	%	*find*	*opt*	%

restrictions which typically result in a broad variety of variant types with small demand rates. In view of the huge number of potential sequences, problems arising from *easy* instances induce a search tree with smaller size. Implicitly, the characteristics of the option restrictions have great influence on the run-time.

It is generally noticed that a feasible solution is found quite fast when using the priority rule PR-2, even for larger-sized problems (compare the results of $T = 50$, $O = 7$, category hard). Here the effect the criterion aimed at has been reached, in contrast to the rule PR-3 which failed to fulfill the expectations.

With respect to optimizing the sequence the priority rule PR-1, choosing an alternative with locally cheapest value is of some advantage compared to the others. Here the deviation between the first and the best solution value is rarely worse than 33%, while the other rules result in much greater deviation. On the basis of these observations it can be concluded that by using PR-1, finding a feasible sequence fast through the algorithm is fast and the value of the first sequence is an upper bound close to the optimal objective value.

Apparently, in general, the run-time gap between the first and the best solution is large for larger problems. This seems to be due to a poor lower bound on partly constructed sequences. A bound regarding the variant ranges given by the function *possV* would probably result in a much better run-time performance. However, a comparison between the amount of failure steps and the huge number of potential sequences shows that the solution procedure is quite powerful in decreasing the search space. This is due to the specification method which succeeds in bounding the problem in early stages of the construction of a sequence σ.

Acknowledgment The authors are indebted to two anonymous referees for their comments and suggestions which greatly improved the readibility of the paper. Thanks also to Nils Boysen for pointing to some flaws in a previous version of the manuscript.

References

Bard, J. F., A. Shtub, and S. B. Josh, "Sequencing mixed-model assembly lines to level parts usage and minimize line length," *International Journal of Production Research*, **32**, 2431–2454 (1994).

Bolat, A., "Efficient methods for sequencing minimum job sets on mixed model assembly lines," *Naval Research Logistics*, **44**, 419–437 (1997).

Dincbas, M., H. Simonis, and P. van Hentenryck, "Solving the car-sequencing problem in constraint logic programming," pp. 290–295, in *Proceedings of the European Conference on Artificial Intelligence (ECAI-88)*, München, (1988).

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

Drexl, A. and A. Kimms, "Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints," *Management Science*, **47**, 489–491 (2001).

Inman, R. R. and R. L. Bulfin, "Sequencing JIT mixed-model assembly lines," *Management Science*, **37**, 901–904 (1991).

Kim, Y. K., C. J. Hyun, and Y. Kim, "Sequencing in mixed model assembly lines: A genetic algorithm approach," *Computers & Operations Research*, **23**, 1131–1145 (1996).

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

Kubiak, W., "Minimizing variation of production rates in just-in-time systems: a survey," *European Journal of Operational Research*, **66**, 259–271 (1993).

Kubiak, W., S. Sethi, "A note on 'Level schedules for mixed-model assembly lines in just-in-time production systems'," *Management Science*, **37**, 121–122 (1991).

Miltenburg, G. J., "Level schedules for mixed-model assembly lines in just-in-time production systems," *Management Science*, **35**, 192–207 (1989).

Monden, Y., *Toyota Production System—An Integrated Approach to Just-in-Time*. Chapman & Hall, Cheriton House/UK, 3rd edition (1998).

Parello, B. D. "CAR WARS: The (almost) birth of an expert system," *AI Expert*, **3**, 60–64 (1988).

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

Scholl, A. *Balancing and Sequencing of Assembly Lines*. Physica, Heidelberg, 2. edition, 1999.

Steiner, G.and J. S. Yeomans, "Level schedules for mixed-model, just-in-time processes," *Management Science*, **39**, 728–735 (1993).

Sumichrast, R. T. and E. R. Clayton, "Evaluating sequences for paced, mixed-model assembly lines with JIT component fabrication," *International Journal of Production Research*, **34**, 3125–3143 (1996).

Tsai, L.-H. "Mixed-model sequencing to minimize utility work and the risk of conveyor stopping," *Management Science*, **41**, 485–495 (1995).

Yano, C. A. and R. M. V. Rachamadugu, "Sequencing to minimize work overload in assembly lines with product options," *Management Science*, **37**, 572–586 (1991).

Zeramdini, W., H. Aigbedo, and Y. Monden, "Bicriteria sequencing for just-in-time mixed-model assembly lines," *International Journal of Production Research*, **38**, 3451–3470 (2000).

Zhang, Y., P. B. Luh, K. Yoneda, T. Kano, and Y. Kyoya, "Mixed-model assembly line scheduling using the Lagrangian relaxation technique," *IIE Transactions*, **32**, 125–134 (2000).