

Matthießen, Lars; Drexl, Andreas; Kimms, Alf

Working Paper — Digitized Version

Constraint propagation algorithms for the car sequencing problem

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 531

Provided in Cooperation with:

Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Matthießen, Lars; Drexl, Andreas; Kimms, Alf (2000) : Constraint propagation algorithms for the car sequencing problem, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 531, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:

<http://hdl.handle.net/10419/147610>

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

**Constraint Propagation Algorithms
for the Car Sequencing Problem**

Lars Matthießen, Andreas Drexl, Alf Kimms

December 2000



Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 531

**Constraint Propagation Algorithms
for the Car Sequencing Problem**

Lars Matthießen, Andreas Drexl, Alf Kimms

December 2000

© Do not copy, publish or distribute without authors' permission.

Lars Matthießen, Andreas Drexl, Alf Kimms
Institut für Betriebswirtschaftslehre, Lehrstuhl für Produktion und Logistik
Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24118 Kiel
email: {Drexl, Kimms}@bwl.uni-kiel.de
URL: <http://www.wiso.uni-kiel.de/bwlinstitute/Prod>

Abstract

The car sequencing problem is to find feasible sequences of product variants requiring specific options while taking care of sequencing constraints. First, a branching scheme and constraint propagation algorithms for the computation of feasible sequences are provided. Second, an algorithm is presented 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.

KEY WORDS: Car Sequencing, Level Scheduling, Constraint Propagation Algorithms, 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 U.S., utility workers are employed to finish work left undone by 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; cp. e.g., Kubiak [9] and Steiner and Yeomans [16].

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 operational issues. A discussion of these topics can be found in, for instance, Scholl [15]. 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.

Due to 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 inherently fixed also. Therefore, the major problem to be solved as a prerequisite for the effective utilization of JIT systems is to determine the sequence in which the 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 (cp., e.g., Sumichrast and Clayton [17]). In such production environments we have to take care about the station loads and about the part usages which both 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, frequently problems arise because of neglecting 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: In Section 2 the car sequencing problem is introduced. Section 3 provides the branching scheme along with constraint propagation algorithms in order to prune large parts of the search tree. Section 4 proposes techniques for computing optimal level schedules. The computational evaluation is presented in Section 5. Finally, Section 6 provides a summary.

2 The Car Sequencing Problem

The subject of the so-called car sequencing approach (cp. Parello, Kabat, and Wos [14] and Parello [13]) is to derive schedules for the final assembly line. The central idea shall be explained by means of an example (which is related to car manufacturing): Assume that 60% of the cars manufactured on the line require the option ‘sun roof’. Moreover, assume that five cars (copies) 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.

In order to describe the car sequencing model formally we use the following notation:

- V : number of variants, index v
- $D(v)$: number of copies (demand) to be produced of variant v
- T : total production volume (periods, cycles), i.e. $T = \sum_{v=1}^V D(v)$, index t
- O : number of options, index j
- $a_{v,j}$: equals 1, if variant v requires option j (0, otherwise)
- $H_j : N_j$: at most H_j of N_j successively sequenced copies may require option j
- $x_{v,t}$: equals 1, if variant v is assigned to period t (0, otherwise)

Formulation 1 *The car sequencing model can be stated as a constraint satisfaction problem by expressions (1)-(4).*

$$\sum_{v=1}^V x_{v,t} = 1 \quad 1 \leq t \leq T \quad (1)$$

$$\sum_{t=1}^T x_{v,t} = D(v) \quad 1 \leq v \leq V \quad (2)$$

$$\sum_{v=1}^V \sum_{\tau=t}^{t+N_j-1} a_{v,j} x_{v,\tau} \leq H_j \quad 1 \leq j \leq O, 1 \leq t \leq T - N_j + 1 \quad (3)$$

$$x_{v,t} \in \{0, 1\} \quad 1 \leq v \leq V, 1 \leq t \leq T \quad (4)$$

Equations (1) are assignment constraints, i.e. exactly one copy has to be produced in each period. Constraints (2) require to manufacture the prespecified number of copies of each

product. Inequalities (3) restrict sequences to be feasible only if the “ $H_j : N_j$ ” constraints are fulfilled. Finally, constraints (4) define the decision variables to be binary.

The car sequencing model (1)-(4) 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; cp., e.g., the recent papers by Zeramdini, Aigbedo, Monden [20] and Zhang, Luh, Yoneda, Kano, Kyoya [21].

The car sequencing model does not require to define upstream and downstream station limits explicitly. In other words, it is not necessary to state whether the problem setting confines to what is called open or closed station. This is of advantage because in practice there is in general 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 explicitly in order to control the risk of conveyor stoppage or the cost for utility work; cp., e.g., Yano and Rachamadugu [19], Bard, Shtub, and Josh [1], Tsai [18], Bolat [2], and Kim, Hyun, and Kim [8].

So far, only very few papers are dedicated to the car sequencing model. Constrained logic programming approaches have been proposed by Parello, Kabat, and Wos [14], Dincbas, Simonis, and van Hentenryck [4], Parello [13], and Drexel and Jordan [5]. Unfortunately, the performance of these approaches in general is totally disappointing. Drexel and Jordan [5] provide limited computational results based on 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. Drexel and Kimms [6] show how to combine the car sequencing and level scheduling problem within an integrated model (cp. Section 4 also).

The constraint propagation algorithm provided in Section 3 is based on Formulation 2 of the car sequencing problem, which uses the following notation:

$D : \{1, \dots, V\} \rightarrow \mathbb{N}$; 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

$opt : \{1, \dots, V\} \rightarrow 2^{\{1, \dots, O\}}$; injective function $j \in opt(v) \stackrel{def}{\Leftrightarrow}$ variant v needs option j

$\bar{v} : \{1, \dots, V\} \rightarrow \{-1, 1\}$, where

$$\bar{v}(v)_j \stackrel{def}{=} \begin{cases} 1 & \text{if } j \in opt(v) \\ -1 & \text{otherwise} \end{cases}$$

Formulation 2 *The subject of the car sequencing problem is to find a sequence $\sigma : \{1, \dots, T\} \rightarrow \{1, \dots, V\}$ which satisfies conditions (5) and (6).*

$$|\sigma^{-1}(v)| = D(v) \quad 1 \leq v \leq V \quad (5)$$

$$\sum_{\tau=t}^{t+N_j-1} \max\{0, \bar{v}(\sigma(\tau))_j\} \leq H_j \quad 1 \leq j \leq O, 1 \leq t \leq T - N_j + 1 \quad (6)$$

Note that the property of σ being a function ensures that equations (1) and (4) are fulfilled. Additionally, condition (5) corresponds to equation (2), and condition (6) covers inequality (3).

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 1, and the option restrictions $\{1:4, 1:6, 2:5, 1:2\}$. For this instance a feasible solution is given in Table 2 where an ‘ \times ’ shows an option occurrence to be installed to the variant.

Table 1: 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 2: 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				\times				\times				\times
1:6						\times						
2:5	\times	\times				\times	\times				\times	\times
1:2	\times			\times		\times		\times		\times		

3 Constraint Propagation Algorithms

In the following we are going to present algorithms the aim of which is to decide whether a feasible solution exists at all for a given problem instance and how it looks like. For short, the problem of car sequencing will be abbreviated as *CS*. The efficiency of these algorithms stems from the fact that we propagate constraints in early stages of the branching process, that is, we devise constraint propagation algorithms. For a recent survey on constraint propagation algorithms cp., e.g., Brailsford, Potts, and Smith [3].

3.1 Branching

The basic idea for the construction of a sequence σ is to assign successively each period $t \in \{1, \dots, T\}$ to a variant copy of type v . During the construction we handle σ as a function

$$\sigma : \{1, \dots, T\} \longrightarrow \{1, \dots, V\} \cup \{0\} \quad (7)$$

in contrast to Formulation 2. 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 $\sigma(t) = 0$. Since there exist at most V possible assignments for such a period t , the problem is branched into at most V subproblems, each with a domain’s cardinality decreased by one.

For an estimation of the width of the search tree note that the number of all possible sequences is calculated by

$$P_\sigma = \frac{T!}{\prod_{v=1}^V (D(v))!} \quad (8)$$

which is an upper bound for the number of all feasible sequences σ . Hence, the width of the search tree depends on the relation between the planning horizon T and the demand rate $D(v)$ of all different variants $v \in V$.

In general, a very small percentage of the P_σ sequences yields solutions of CS according to Formulation 2. For the instance in Example 1 there are actually only 13 solutions. Hence, we have to prevent the unnecessary exploration of subproblems. All subproblems not being fully explored put up the so-called *candidate list* from which one is to choose to continue the exploration of the search tree. For the problem CS we adapt the well-known last-in-first-out (LIFO) principle. According to LIFO, the exploration of the search tree is continued within the subproblem added last to the candidate list, leading to depth-first-search. By definition, a subproblem is explored, if (i) it has an empty solution space or if (ii) it represents a feasible solution. (One additional condition (iii) which relates to the calculation of bounds for the objective function will be issued in Section 4.2.)

Any subproblem is a problem CS like in Formulation 2, additionally restricted by the current state of construction of a sequence σ arising from branching. Since, in that way, any subproblem corresponds to a partial sequence σ , it is fully described by the definition of σ . Therefore, in the following a subproblem is identified with the state the corresponding sequence construction is in. A formal description of a subproblem as well as of branching a subproblem is given in the definitions below.

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

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

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

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

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

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

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

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

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, the above definition allows a CS-state to contain more information than we can obtain from simple branching, since it is not demanded that $m_{j,t}^{(\xi)} = 0$ in case of $\sigma_\xi(t) = 0$.

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 $v \Delta s_t^{(\xi)}$, if

$$\bar{v}(v)_j = 1 \quad \Rightarrow \quad (s_t^{(\xi)})_j = 1.$$

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) $\forall t \in \{1, \dots, T\} (\sigma_\xi(t) = 0 \Rightarrow \{v \in V \mid v \Delta s_t^{(\xi)}\} \cap \{v \in V \mid |\sigma_\xi^{-1}(v)| < D(v)\} \neq \emptyset)$.

While the first condition relates to feasibility, the second one ensures that at least one subproblem exists for any non-assigned period t which ξ can be branched into. 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. Since 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 $\xi, \xi' \in \Xi$, ξ be allowed. Branching the *CS*-state ξ is performed by a transformation*

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

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

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 $\xi \xrightarrow{v} \xi'$ and $\xi \rightarrow \xi'$ in case of unknown v . Then $\xi \xrightarrow{*} \xi'$ denotes a sequence of *b-steps* of arbitrary length.

As seen in the example below, a *b-step* $\xi \rightarrow \xi'$ 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 $\xi, \xi' \in \Xi$, ξ be allowed. A *b-step* $\xi \rightarrow \xi'$ is destructive, if there does not exist any solution ξ_S which can be obtained by $\xi' \xrightarrow{*} \xi_S$.*

Example 2 *Recall Example 1 and assume that a sequence of *b-steps* $\xi_I \xrightarrow{*} \xi \xrightarrow{6} \xi'$ has produced the *CS*-state given in Table 3. By definition, ξ' is not allowed, since none of the variant types $\{2, 6, 7\}$, which are the only ones with non-assigned copies, offers a feasible assignment for period 10. $\xi \xrightarrow{6} \xi'$ is a destructive step.*

Table 3: 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			×		×		×					

Identifying *b-steps* as being 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

Our efforts to decrease the search tree base on the utilization of information we can deduce from a CS -state ξ . This information will be used to formulate a method which supports the exploration of a whole subtree in polynomial time. The following definitions are prerequisites for what follows.

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

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

If M'_ξ is a specification of M_ξ , we say that M'_ξ is more specified than 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 $\xi \xrightarrow{*} \xi_S$ and ξ_S is a solution. If a completion of ξ exists, we denote this by the predicate $\text{compl}(\xi)$.

In order to decrease the search space, we follow the three steps (1) specify M_ξ by a polynomial method using state-inherent information, (2) validate a CS -state by the use of a condition on the construction which is necessary for an allowed completion of σ , hence for the existence of a solution, and (3) learn from failure steps. Each of these steps is issued in the subsequent sections.

3.2.1 Specification Method

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

In a totally unspecified matrix M_{ξ_t} 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

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

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

$$\text{edd} : \{1, \dots, O\} \times \mathbb{N} \longrightarrow \mathbb{N} \quad \text{where} \quad (9)$$

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

Because of symmetry the right interval limit is calculated by

$$\text{ldd} : \{1, \dots, O\} \times \mathbb{N} \longrightarrow \mathbb{N} \quad \text{where} \quad (10)$$

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

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

$$range : \{1, \dots, O\} \times \mathbb{N} \times \Xi \longrightarrow \mathbb{N}^{\{1, \dots, T\}} \quad \text{where} \quad (11)$$

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

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

The function *range* is a reduction of the interval given by earliest and latest due dates to a subset of periods in which an option occurrence can be planned without hurting 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 following sense:

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. occurrence i of option j must be installed in period t_0), mark this in M_ξ by setting $m_{j,t_0}^{(\xi)} = 1$,
- 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 hurt the option restriction), mark this in M_ξ by setting $m_{j,t_0}^{(\xi)} = -1$,
- R3 for a remaining period t_0 with $\sigma(t_0) = 0$ we set $m_{j,t}^{(\xi)} = 0$.

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 . Since the number of copies of each variant type v is limited by its demand $D(v)$, we can calculate these possibilities by

$$possV : \{1, \dots, T\} \times \Xi \longrightarrow 2^{\{1, \dots, V\}} \quad \text{where} \quad (12)$$

$$(t, \xi) \mapsto \begin{cases} possV(t, \xi') & \text{if } \sigma_\xi(t) \neq 0 \\ \{v \in V \mid v \Delta s_t^{(\xi)} \wedge |\sigma_\xi^{-1}(v)| < D(v)\} & \text{otherwise} \end{cases}$$

Here it is assumed that the CS-state ξ has been reached by a b -step $\xi' \rightarrow \xi$. Whenever $\sigma_\xi(t) \neq 0 \wedge |possV(t, \xi)| > 1$ 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*.

It is clear that the more M_ξ is specified, the more the sets of possible assignments are restricted. In fact, it is easy to see that with a further specification of M_ξ the cardinality of the sets of alternatives obtained from $possV$ decreases monotonously, formally noted by

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

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

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

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

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

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

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

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

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

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

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 :

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

The specification rules R4 to 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 to R3
- 4 calculate possV
- 5 apply rules R4 to R7
- 6 until non of the rules R4 to 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. Since, 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 to R7 can be seen in Example 1. First, we calculate the option ranges in the initial state ξ_I , according to (11); cp. Table 4. From this we derive the possible assignments for each period according to (12), and we obtain the sequence provided in Table 5. The row representing the third option type is already fixed, according to rules R1 and R2. Having performed branching $\xi_I \xrightarrow{\mathcal{Z}} \xi_1 \xrightarrow{\mathcal{V}} \xi_2$ which produces a sequence with initial part $\sigma(1) = 2$ and $\sigma(2) = 1$, the iterative process is invoked; cp. Table 6. 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.

Table 4: Value of the Function range in the Initial CS-State

option	occurrence ranges					
	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 5: 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 6,7	1,2 6,7	3,4 5	3,4 5	3,4 5	1,2 6,7	1,2 6,7	3,4 5	3,4 5	3,4 5	1,2 6,7	1,2 6,7
1:4												
1:6												
2:5	×	×	-	-	-	×	×	-	-	-	×	×
1:2												

3.2.2 A Condition on $compl(\xi)$

While the specification method supports skipping a whole subtree, in the second step a criterion is developed to analyse the CS-state ξ with M_ξ^* obtained from the specification method in view of whether it is worth continuing the so far constructed σ_ξ or to drop ξ from consideration. For this purpose, we use a condition $\mathfrak{S}(\xi)$ which is necessary for the completion of σ_ξ , hence

$$compl(\xi) \Rightarrow \mathfrak{S}(\xi). \quad (13)$$

The idea is that, once $\neg\mathfrak{S}(\xi)$ holds, it is clear by contradiction that there is no solution ξ_S available by further exploration of ξ . Being not allowed, as a predicate on CS-states, is a rather poor example for such a condition. The problem is to find a condition $\mathfrak{S}(\xi)$ strong enough to predict an empty solution space in early stages of the construction of σ . This is the subject 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:

Table 6: Problem Solved by Specification

<i>it</i>	<i>t</i>	1	2	3	4	5	6	7	8	9	10	11	12
1	$\sigma(t)$	2	1	4	0	0	0	0	0	0	0	0	0
	$possV$	1,2 6,7	1,2 6,7	4	3,4 5	3,4 5	2,6 7	2,6 7	3,4 5	3,4 5	3,4 5	2,6 7	2,6 7
	1:4	-	-	-									
	1:6	-	x	-	-	-	-	-	-	-	-	-	-
	2:5	x	x	-	-	-	x	x	-	-	-	x	x
	1:2	-	x	-									
2	$\sigma(t)$	2	1	4	5	0	0	0	5	0	0	0	6
	$possV$	1,2 6,7	1,2 6,7	4	5	3,4	2,7	2,7	5	3,4	3,4	2,7	6
	1:4	-	-	-	x	-	-	-	x	-	-	-	x
	1:6	-	x	-	-	-	-	-	-	-	-	-	-
	2:5	x	x	-	-	-	x	x	-	-	-	x	x
	1:2	-	x	-	x				x				-
3	$\sigma(t)$	2	1	4	5	4	7	2	5	4	3	2	6
	$possV$	1,2 6,7	1,2 6,7	4	5	4	7	2	5	4	3	2	6
	1:4	-	-	-	x	-	-	-	x	-	-	-	x
	1:6	-	x	-	-	-	-	-	-	-	-	-	-
	2:5	x	x	-	-	-	x	x	-	-	-	x	x
	1:2	-	x	-	x	-	x	-	x	-	x	-	-

$$C1 (\exists j \in \{1, \dots, O\}) (\exists i \in \{1, \dots, occ(j)\}) (range(j, i, \xi) = \emptyset)$$

there exists an option occurrence of type j which cannot be inserted into M without hurting the $H_j : N_j$ constraint,

$$C2 (\exists t \in \{1, \dots, T\}) (possV(t, \xi) = \emptyset)$$

there is a yet non-assigned period which 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

$$C2' (\exists v \in \{1, \dots, V\})$$

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

which is able to predict occurrences of C2.

First, we derive a condition $\mathfrak{S}^{(1)}(\xi)$ as follows: Let $r(v, \xi)$ be short for $D(v) - |\sigma_{\xi}^{-1}(v)|$, 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 define

$$\begin{aligned} \mathfrak{S}^{(1)}(\xi) = & (\forall v \in \{1, \dots, V\}) \\ & (|\{t \in \{1, \dots, T\} \mid \sigma_\xi(t) = 0 \wedge v \in \text{poss}V(t, \xi)\}| \geq r(v, \xi)) \\ \wedge & (\forall j \in \{1, \dots, O\})(\forall \#_{j, \xi} + 1 \leq i \leq \text{occ}(j)) \\ & (\#_{j, \xi} \leq \text{occ}(j) \Rightarrow \text{range}(j, i, \xi) \neq \emptyset). \end{aligned}$$

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

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

in state ξ and the sets of non-assigned periods

$$U_\xi^v = \{t \in U_\sigma \mid v \in \text{poss}V(t, \xi)\}, \quad 1 \leq v \leq V$$

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

Example 4 Consider a state ξ with $U_\xi \supset \{t_{i_1}, t_{i_2}, t_{i_3}, t_{i_4}\}$,
 $\text{poss}V(t_{i_1}, \xi) \supseteq \{1, 2\}$, $\text{poss}V(t_{i_2}, \xi) \supseteq \{1, 2\}$,
 $\text{poss}V(t_{i_3}, \xi) \supseteq \{1, 2\}$, $\text{poss}V(t_{i_4}, \xi) \supseteq \{2\}$,

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

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

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

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

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

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

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 monotony of $\text{poss}V$, this capability is necessary for all subsets of variants, so C2' is modified to

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

and, second, we strengthen the condition \mathfrak{S} by the negation of C2''

$$\begin{aligned} \mathfrak{S}^{(2)}(\xi) = & (\forall K \in 2^{\{v \in V \mid r(v, \xi) > 0\}}) \\ & (K \neq \emptyset \Rightarrow \left| \bigcup_{k \in K} U_\xi^k \right| \geq \sum_{k \in K} r(k, \xi)) \\ \wedge & (\forall j \in \{1, \dots, O\})(\forall i \in \{\#_{j, \xi} + 1, \dots, \text{occ}(j)\}) \\ & (\#_{j, \xi} \leq \text{occ}(j) \Rightarrow \text{range}(j, i, \xi) \neq \emptyset). \end{aligned}$$

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

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

$$\begin{aligned} range(1, 1) &= [1], & range(2, 1) &= [1, 2], \\ range(1, 2) &= [3], & range(2, 2) &= [4, 5], \\ range(1, 3) &= [5], & range(3, 1) &= [1, 5]. \end{aligned}$$

It follows

$$\begin{aligned} possV(1, \xi_I) &= 1, & possV(3, \xi_I) &= 1, & possV(5, \xi_I) &= 1, \\ possV(2, \xi_I) &= 2, 4, & possV(4, \xi_I) &= 2, 4, \end{aligned}$$

and it is easy to see that

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

holds. However, there exists no solution, since the copies of variant 1 must be scheduled in periods 1, 3 and 5, and variants 2 and 3 scheduled in periods 2 and 4, both requesting option 2, hurting the option restriction 1:3.

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

3.2.3 Learning from Failure Steps

In contrast to the methods presented so far which support the construction of σ by detecting and excluding destructive steps, the usage of learning from failure steps avoids to repeat them, once they were done. Whenever $\mathfrak{S}^{(2)}(\xi)$ recognizes that the construction runs into a destructive step, such that a completion of σ cannot be obtained from state ξ , ξ can be dropped from consideration, and the construction proceeds to the latest state where at least one non-explored alternative is available, according to the LIFO principle of selecting a CS-state from the candidate list.

Consider a state ξ for which $\mathfrak{S}^{(2)}(\xi)$ 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 differ 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 monotony 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 $possV(t', \xi') = possV(t', \xi') - \{v \in V \mid j \notin opt(v)\}$ 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 off the search tree which, otherwise, would have been recognized as destructive later. In this context learning from failure steps means recognizing the cause of such step.

4 Optimal Level Schedules

The algorithms of the preceding section tell us how to compute feasible sequences. If more than just one feasible sequence exists the natural question arises which one to choose. This leads to the problem of determining an optimal level schedule, the subject of what follows.

As outlined by Drexel and Kimms [6], the main idea of leveling the schedule is to keep the quantity of each product manufactured per unit time as close as possible to the demand for that product per unit time. This is underlined by the algorithmic approaches presented in the literature. Monden [12] attaches 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 [11] presents a nonlinear integer programming formulation with the objective of minimizing the deviation between actual and desired production rates. Kubiak and Sethi [10] introduce a variant, which reduces the scheduling algorithm to an assignment problem. Inman and Bulfin [7] give a formulation of the problem as a minimization problem in which the objective is to minimize the sum of deviations between actual positions of the copies from the ideal ones in the sequence.

4.1 Objective Function

In the sequel, we will adopt the approach of Inman and Bulfin [7]. Formally, it can be described by

Formulation 3 *Let the ideal position be given by a function*

$$dd : \{1, \dots, V\} \times \mathbb{N} \longrightarrow \mathbb{R} \quad \text{where}$$

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

Then the level scheduling model is

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

$$\text{s.t. } \delta : \{1, \dots, T\} \longrightarrow \{1, \dots, V\} \times \mathbb{N}.$$

Apparently, this formulation is different from, but equivalent to the formulation given by Inman and Bulfin [7]. 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 Subsection 3.1 a subproblem is explored, if (iii) the expected value of the objective function is worse than the best one known so far.

In order 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 on *CS*. The relaxation considered here is to drop constraints (3). 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 given in Formulation 3 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 [7], otherwise.

Now, we present the construction method for the level scheduling model. Then it is proven in Lemma 1 that the constructed schedule is optimal with respect to the objective.

The production of a 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

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

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

For illustrative purposes, let us have a look at the following

Example 6 Recall Example 1, where the demand $D(v)$ for the seven variants is given by $(1, 3, 1, 3, 2, 1, 1)$ for $v = 1, \dots, 7$. Since one copy is produced each period, the overall demand is equal to the amount of periods within the planning horizon, hence

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

The ideal positions are

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

According to the construction method, we obtain the sequence given in Table 7.

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

$$\delta^*(t) := (v_0, i_0), \quad \text{where}$$

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

Table 7: 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												

Let the set of all sequences $\{1, \dots, T\} \rightarrow \{1, \dots, V\} \times N$ be denoted by Δ . The objective value of a sequence is the sum of the norm of the c -values, i.e.

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

Optimality of δ^* is shown by proving the nonexistence of another sequence δ' the objective value of which is less than the objective of δ^* .

Lemma 1 $C_{\Delta}(\delta^*) = \min\{C_{\Delta}(\delta) \mid \delta \in \Delta\}$

Proof. Assume the statement to be wrong. Then there is a sequence $\delta' \in \Delta$ for which $C_{\Delta}(\delta') < C_{\Delta}(\delta^*)$ holds. Consider such a function δ' . Obviously $\delta' \neq \delta^*$, so there exists $t \in \{1, \dots, T\}$ such that $\delta'(t) \neq \delta^*(t)$. Consequently, the set $D = \{1 \leq t \leq T \mid \delta'(t) \neq \delta^*(t)\}$ is nonempty. Without loss of generality it can be assumed that $\delta^*(\{1, \dots, T\}) = \delta'(\{1, \dots, T\})$, i.e. δ' is a permutation of δ^* , otherwise there would be $t \in \{1, \dots, T\}$ with $c(t, \delta'(t)) = \infty$, and this would result in a contradiction immediately.

Set $t' := \min D$, hence $(\forall t \in \{1, \dots, t' - 1\}) (\delta^*(t) = \delta'(t))$, and there exists $t'' > t'$ such that $\delta^*(t') = \delta'(t'')$ since δ' is a permutation of δ . Let $(v', i') := \delta'(t')$, $(v'', i'') := \delta'(t'')$ and define the permutation $\pi \in \Delta$ as

$$\pi := (\delta' \setminus \{(t', (v', i')), (t'', (v'', i''))\}) \cup \{(t', (v'', i'')), (t'', (v', i'))\},$$

that is, π is obtained from δ' by changing the images at the domain values t' and t'' in such a way that π matches the image of δ^* at t' . (Recall π to be an approximation step from δ' to δ^* .) Further it is clear that $v' \neq v''$. Hence, the difference between the objective value of δ' and π is

$$\begin{aligned} d &:= C_{\Delta}(\delta') - C_{\Delta}(\pi) \\ &= C_{\Delta}(\delta' \upharpoonright_{\{t', t''\}}) - C_{\Delta}(\pi \upharpoonright_{\{t', t''\}}) \\ &= \|dd(v', i') - t'\| + \|dd(v'', i'') - t''\| - \|dd(v'', i'') - t'\| - \|dd(v', i') - t''\|. \end{aligned}$$

Now, use a' as short-hand notation for $dd(v', i')$ and a'' for $dd(v'', i'')$. We have to consider two cases:

(1) $a' = a''$; then

$$\begin{aligned} d &= \|a' - t'\| + \|a'' - t''\| - \|a'' - t'\| - \|a' - t''\| \\ &= \|a' - t'\| + \|a' - t''\| - \|a' - t'\| - \|a' - t''\| = 0. \end{aligned}$$

(2) $a' \neq a''$; according to the construction of δ^* (and consequently of π at periods t', t'') it is clear that $a'' < a'$; furthermore, we have $t' < t''$, and, hence, we have to distinguish between six subcases:

(a) $a'' \leq a' \leq t' < t''$

$$\begin{aligned} d &= (t' - a') + (t'' - a'') - (t' - a'') - (t'' - a') \\ &= t' - a' + t'' - a'' - t' + a'' - t'' + a' = 0. \end{aligned}$$

(b) $t' < a'' \leq a' \leq t''$

$$\begin{aligned} d &= (a' - t') + (t'' - a'') - (a'' - t') - (t'' - a') \\ &= a' - t' + t'' - a'' - a'' + t' - t'' + a' \\ &= 2a' - 2a'' > 0. \end{aligned}$$

(c) $t' < t'' < a'' \leq a'$

$$\begin{aligned} d &= (a' - t') + (a'' - t'') - (a'' - t') - (a' - t'') \\ &= a' - t' + a'' - t'' - a'' + t' - a' + t'' = 0. \end{aligned}$$

(d) $a'' \leq t' < a' \leq t''$

$$\begin{aligned} d &= (a' - t') + (t'' - a'') - (t' - a'') - (t'' - a') \\ &= a' - t' + t'' - a'' - t' + a'' - t'' + a' \\ &= 2a' - 2t' > 0. \end{aligned}$$

(e) $a'' \leq t' < t'' < a'$

$$\begin{aligned} d &= (a' - t') + (t'' - a'') - (t' - a'') - (a' - t'') \\ &= a' - t' + t'' - a'' - t' + a'' - a' + t'' \\ &= 2t'' - 2t' > 0. \end{aligned}$$

(f) $t' < a'' \leq t'' < a'$

$$\begin{aligned} d &= (a' - t') + (t'' - a'') - (a'' - t') - (a' - t'') \\ &= a' - t' + t'' - a'' - a'' + t' - a' + t'' \\ &= 2t'' - 2a'' > 0. \end{aligned}$$

In any case $C_\Delta(\delta') - C_\Delta(\pi) \geq 0$ holds, so an approximation of a sequence δ' to an optimal sequence δ^* leads to a stagnation or a reduction of the objective function value. Hence, vice versa, $C_\Delta(\delta') \geq C_\Delta(\delta^*)$ holds, and this yields the contradiction. \square

4.3 Overall Algorithm

The performance of the algorithm for computing optimal sequences depends on the one hand 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. Being 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 $possV(t, \xi) = \{v_{i_1}, \dots, v_{i_k}\}$ 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 $\{v_{i_1}, \dots, v_{i_k}\}$ 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 [7]. Therefore, the priority values of the alternatives are calculated by

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

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 $\mathfrak{S}_\xi^{(2)}$.

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

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

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 $\mathfrak{S}^{(2)}(\xi)$.

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:

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

If $\mathfrak{S}^{(2)}(\xi)$ holds, then $p(v_{i_q}) \geq 1$. 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 $\mathfrak{S}^{(2)}(\xi)$, reducing the risk to run in a *CS*-state where there are less 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    $\xi := \text{getFirstCandidateFromList}$ 
5   apply specification method
6   if  $\neg \mathfrak{S}(\xi)$  or  $\text{lb}(\xi) \geq \text{value}(\xi^*)$  then
7     drop  $\xi$  from candidateList
8   else
9     if  $\sigma^{-1}(0) = \emptyset$  then
10      if  $\text{lb}(\xi) < \text{value}(\xi^*)$  or  $\xi^* = \langle \rangle$  then
11         $\xi^* = \xi$ 
12      endif
13    else
14       $t_0 = \min \sigma^{-1}(0)$ 
15       $A_\xi = \text{possV}(t_0, \xi)$ 
16      sort  $A_\xi \rightarrow (v_{i_1}, \dots, v_{i_k})$ 
17      candidateList :=  $\langle \xi \overrightarrow{v_{i_1}} \xi_{i_1}, \dots, \xi \overrightarrow{v_{i_k}} \xi_{i_k}; \text{candidateList} \rangle$ 
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 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 of Section 3.2. In case the condition $\mathfrak{S}(\xi)$ 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 new best 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 equipped with an Intel-350MHz-processor, 64MB RAM, and operating system Windows™ 98. The algorithm has been implemented by the use of the Delphi™ environment, developer's edition, version 2.01.

In order to evaluate the algorithms presented above we used the testbed provided in Drexel and Kimms [6]. The instances have the characteristics shown in Table 8. Note that due to the 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*.

Table 8: Characteristics of the Testbed

T	O	<i>easy</i>	<i>hard</i>
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

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 2, together with the number of failure steps (i.e. the number of boundings invoked 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;

Table 9: Computational Results – Easy Instances

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

Table 10: Computational Results – Hard Instances

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

- the percentage deviation between the objective value of the first solution and the optimal objective value.

The computational results are given in Tables 9 and 10. 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 three hours.

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* restrictions which typically result in a broad variety of variant types with small demand rates. In view of the overall number of sequences P_σ , problems arising from *easy* instances put on 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.

Considering the task to optimize the solution 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 first and best solution value is rarely worse than 33%, while the other rules result in much greater deviation. Based on these observations it can be concluded that using PR-1 the algorithm is capable to find a feasible sequence fast, and the value of the first sequence is an upper bound close to the optimal objective value.

Apparently, the gap in run-time between the first and the best solution is very wide for larger-sized problems in general. 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 number of sequences P_σ 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 .

6 Summary

The subject of the car sequencing problem is to derive sequences for the final assembly line. More precisely, the goal is to produce a sequence for the final assembly level while taking maximal station loads (capacities) and implied part usages explicitly into account via sequencing constraints. Clearly, to look at station loads reduces the risk of stopping the conveyor while the part usage aspect avoids shortages. By contrast level scheduling is based on the assumption that the different options (i.e. parts such as engines, transmissions, accelerators, number of doors) required by the different variants do not affect station loads and that the implied part demand for the output of the feeder process has not to be taken into account.

A branching scheme and constraint propagation algorithms are provided which allow to compute a feasible solution for large-size problem instances. Furthermore, an algorithm is presented which can optimize a level schedule objective while taking care of the car sequencing constraints. Computational results are presented which show that feasible (optimal) solutions can be obtained quickly for (fairly) quite large problem instances.

References

- [1] BARD, J.F., SHTUB, A., JOSH, S.B. "Sequencing mixed-model assembly lines to level parts usage and minimize line length". *International Journal of Production Research*, 32:2431–2454, 1994.
- [2] BOLAT, A. "Efficient methods for sequencing minimum job sets on mixed model assembly lines". *Naval Research Logistics*, 44:419–437, 1997.
- [3] BRAILSFORD, S.C., POTTS, C.N., SMITH, B.M. "Constraint satisfaction problems: algorithms and applications". *European Journal of Operational Research*, 119:557–581, 1999.
- [4] DINCIBAS, M., SIMONIS, H., VAN HENTENRYCK, P. "Solving the car-sequencing problem in constraint logic programming". In *Proceedings of the European Conference on Artificial Intelligence (ECAI-88)*, pages 290–295. München, 1988.
- [5] DREXL, A., JORDAN, C. "Materialflußorientierte Produktionssteuerung bei Variantenfließfertigung". *Zeitschrift für betriebswirtschaftliche Forschung*, 47:1073–1087, 1995.
- [6] DREXL, A., KIMMS, A. "Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints", *Management Science* (to appear).
- [7] INMAN, R.R., BULFIN, R.L. "Sequencing JIT mixed-model assembly lines". *Management Science*, 37:901–904, 1991.
- [8] KIM, Y.K., HYUN, C.J., KIM, Y. "Sequencing in mixed model assembly lines: a genetic algorithm approach". *Computers & Operations Research*, 23:1131–1145, 1996.
- [9] KUBIAK, W. "Minimizing variation of production rates in just-in-time systems: a survey". *European Journal of Operational Research*, 66:259–271, 1993.
- [10] KUBIAK, W., SETHI, S. "A note on 'Level schedules for mixed-model assembly lines in just-in-time production systems' ". *Management Science*, 37:121–122, 1991.
- [11] MILTENBURG, G.J. "Level schedules for mixed-model assembly lines in just-in-time production systems". *Management Science*, 35:192–207, 1989.
- [12] MONDEN, Y. *Toyota Production System — An Integrated Approach to Just-in-Time*. Chapman & Hall, Cheriton House/UK, 3. edition, 1998.
- [13] PARELLO, B.D. "CAR WARS: The (almost) birth of an expert system". *AI Expert*, 3:60–64, 1988.

- [14] PARELLO, B.D., KABAT, W.C., WOS, L. "Job-shop scheduling using automated reasoning: a case study of the car-sequencing problem". *Journal of Automated Reasoning*, 2:1-42, 1986.
- [15] SCHOLL, A. *Balancing and Sequencing of Assembly Lines*. Physica, Heidelberg, 2. edition, 1999.
- [16] STEINER, G., YEOMANS, J.S. "Level schedules for mixed-model, just-in-time processes". *Management Science*, 39:728-735, 1993.
- [17] SUMICHRAST, R.T., CLAYTON, E.R. "Evaluating sequences for paced, mixed-model assembly lines with JIT component fabrication". *International Journal of Production Research*, 34:3125-3143, 1996.
- [18] TSAI, L.-H. "Mixed-model sequencing to minimize utility work and the risk of conveyor stopping". *Management Science*, 41:485-495, 1995.
- [19] YANO, C.A., RACHAMADUGU, R.M.V. "Sequencing to minimize work overload in assembly lines with product options". *Management Science*, 37:572-586, 1991.
- [20] ZERAMDINI, W., AIGBEDO, H., MONDEN, Y. "Bicriteria sequencing for just-in-time mixed-model assembly lines". *International Journal of Production Research*, 38:3451-3470, 2000.
- [21] ZHANG, Y., LUH, P.B., YONEDA, K., KANO, T., KYOYA, Y. "Mixed-model assembly line scheduling using the Lagrangian relaxation technique". *IIE Transactions*, 32:125-134, 2000.