Drexl, Andreas; Kimms, Alf

Sequencing JIT mixed-model assembly lines under station load- and part usage-constraints

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 460

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 460

Sequencing JIT Mixed–Model Assembly Lines Under Station Load– and Part Usage–Constraints

Andreas Drexl and Alf Kimms

November 1997

Paper presented at the Workshop on Scheduling in Computer– and Manufacturing–Systems, Schloß Dagstuhl/Germany, June 1997.

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
ftp://ftp.wiso.uni-kiel.de/pub/operations-research

Abstract

In the past several years, there has been growing interest in the problem of sequencing mixed–model assembly lines. So far most research concentrated on the 'level scheduling problem' while only some papers have been published on the 'car sequencing problem'. The subject of the level scheduling problem is to derive 'smooth' production schedules while the car sequencing problem takes sequencing constraints related to station loads and part usages into account. The former in general is a difficult optimization problem while for the latter even the feasibility problem is $\mathcal{NP}$–complete.

This paper presents a nonlinear integer programming model which covers both the balancing requirements of level scheduling and the constraints of car sequencing. For the solution of the problem we present a set partitioning/column generation approach. Solving the LP–relaxation of this model by column generation provides tight lower bounds for the optimal objective function value.

KEYWORDS: MIXED–MODEL ASSEMBLY LINES, JUST–IN-TIME PRODUCTION, STATION LOAD–/PART USAGE–CONSTRAINTS, SET PARTITIONING/COLUMN GENERATION

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. Anyway, it is desirable to distribute products with high work content evenly in order to reduce the risk of conveyor stoppage or the cost for utility work (cp., for instance, Tsai 1995).

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. The use of JIT methods makes it possible to satisfy customers' demands for several products without holding large inventories and without incurring large shortages.

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. An analysis of these topics can be found in, for instance, Thomopoulos 1967, Scholl 1995, and van Zante–de Fokkert and de Kok 1997.

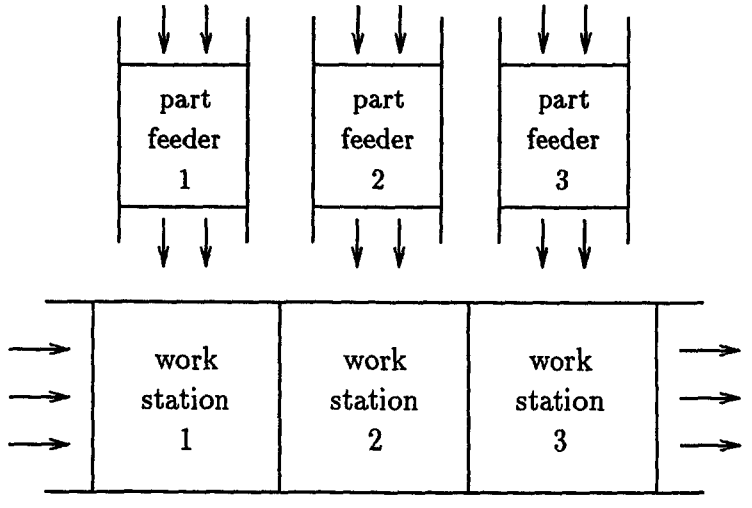
In the model sequencing problem, the shop floor is the focus of attention where we have to decide about the specific order in which the different models have to be launched onto the line. Usually, the demand rate of the models varies and the problem must be solved periodically.

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 preceeding 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 (cp. the shop floor layout of a final assembly line in Figure 1) or one feeder provides parts for several stations (cp., e.g., Sumichrast and Clayton 1996). 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, oftenly problems arise because of neglecting interdependencies between consecutive subsequences. The methods developed in this paper allow to evaluate subsequences consisting of about 20 copies in an analytical way. 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.

Figure 1: Final Assembly Line — Shop Floor Layout



Many assembly lines, such as those in the automobile industry, have dozens or hundreds of stations the performance of which is affected by customer–selected options on the products assembled. However, most assembly line sequencing algorithms developed for situations where the various options require significantly different amounts of processing time cannot consider so many stations or options effectively. Then the analytical methods

of Rachamadugu and Yano 1994 which compute a "criticality index" for selecting stations shall be used.

In addition to balancing and sequencing, there are several relevant issues such as lead times, work in process/kanbans, etc. which have to be considered also in order to design and operate JIT systems efficiently; we refer to the work by Bitran and Chang 1987, Krajewski, King, Ritzman, and Wong 1987, and Berkley 1992.

The problem dealt with in this paper may be characterized also as a one machine sequencing (or scheduling) problem with difficult sequencing constraints and with irregular criteria (which are not monotone in the job completion times). A survey of recent results for this area can be found in Hoogeven, Lenstra, and van de Velde 1997.

The exposition of our work is as follows: In Section 2 we review previous work. Section 3 introduces a set partitioning/column generation approach for the mixed–model assembly line sequencing problem under station load– and part usage constraints. The computational evaluation is presented in Section 4. Section 5 outlines some of the special cases covered by the set partitioning/column generation model. In addition, some extensions are discussed also. Finally, Section 6 provides a summary and an outlook for future work.

2 Review of Previous Work

In this section first we describe one of the fundamental models developed for level scheduling and discuss related work. Second, we present a formal model of the car sequencing problem.

Throughout the paper we use the following basic notation:

V	:	set of variants, index v
D_v	:	set of copies (demand) to be produced of variant v; for the sake of simplicity the copies are represented by numbers, i.e. $D_v = \{1, \ldots, \lvert D_v \rvert\}$, index i
T	:	total production volume (periods, cycles), i.e. $T = \{1, \ldots, \sum_{v \in V} \lvert D_v \rvert\}$, index t

2.1 Level Scheduling

The focus of most of the work done over the past years in JIT sequencing was on scheduling problems with penalties for both earliness and tardiness. Baker and Scudder 1989 give a comprehensive review of earlier research in this area. The main objective of another important class of scheduling problems considered in JIT production is the minimization of rates with which processes within a system supply their outputs. The main idea to achieve this is keeping the quantity of each product manufactured per unit time as close as possible to the demand for that product per unit time.

As outlined by Kubiak 1993 a JIT system, because being a pull system, initiates a supply process only if there is is another process that requires the supplying process' output (raw material, part, subassembly). Consequently, the final assembly line is the focus of scheduling.

Miltenburg 1989 has formulated the problem as a nonlinear integer programming problem where the objective is to minimize the total deviation of actual production from the

desired production rates. He developed an exact algorithm with exponential (in the number of products) worst case complexity, and, in additon, two heuristic procedures. Miltenburg, Steiner, and Yeomans 1990 propose a dynamic programming algorithm whose run time is also exponential in the number of products. Kubiak and Sethi 1991, 1994 subsequently developed an optimization algorithm that solves the problem and its extensions in polynomial time in the total demand for all products produced on the line over a given planning horizon. Similarly, Inman and Bulfin 1991 proposed a formulation which can be solved to optimality by ordering the copies according to the earliest due date (EDD) rule.

In order to describe the approach by Inman and Bulfin 1991 formally we use the following parameters and variables:

$f_{v,i}$: ideal position of copy $i \in D_v$ of variant $v \in V$

$x_{v,i}$: positive integer denoting the period in which copy $i \in D_v$ of variant $v \in V$ is scheduled

A formal definition of the ideal positions $f_{v,i}, v \in V, i \in D_v$, is given by equation (1).

$$f_{v,i} = (i - 1/2)\,|T| \,/\, |D_v| \tag{1}$$

Based on these definitions the level scheduling model can be stated by the eypressions (2)–(5).

$$\min \sum_{v \in V} \sum_{i \in D_v} (x_{v,i} - f_{v,i})^2 \tag{2}$$

$$\text{s.t.} \quad x_{v,i+1} \geq x_{v,i} + 1 \qquad v \in V, i \in D_v \setminus \{|D_v|\} \tag{3}$$

$$x_{v,i} \neq x_{v',i'} \qquad v \in V, i \in D_v, v' \in V, i' \in D_{v'}, (v,i) \neq (v',i') \tag{4}$$

$$x_{v,i} \in T \qquad v \in V, i \in D_v \tag{5}$$

The objective (2) minimizes the sum of deviations of the scheduled periods from the ideal ones. Inequalities (3) restrict the decision variables to be monotically increasing from copy to copy for each variant. Constraints (4) require the variables to be pairwise disjoint while (5) defines the domain of the variables. Clearly, constraints (4) are not in the standard format of integer programs. They simply require that exactly one unit can be produced in each period. Inman and Bulfin 1991 observe that the model (2)–(5) describes a single machine scheduling problem with penalties for both earliness and tardiness, where each copy of a product is a separate job and $f_{v,i}$ is the due date of job (v,i). They prove that the level scheduling model (2)–(5) can be solved to optimality by ordering the copies according to the EDD rule.

For illustrative purposes consider the instance given in Table 1 having $|V| = 6$ variants and a planning horizon of $|T| = 14$. Note that the first two columns and rows two to five can be skipped for the moment. For this instance the ideal positions are reproduced in Table 2. Finally, Table 3 provides the optimal level schedule with an objective function value of 10.8 (disregard rows three to six until later). Note, in this instance option $o = 3$ could be deleted because $|D_2| + |D_5| = 3 \leq H_3 = 3$.

A practical example of balancing the production of Coronas at Toyota is given by Monden 1983. 250 sedans, 125 hardtops, and 125 wagons must be produced within an

eight–hour production shift which essentially means that one car leaves the shop floor every minute. A balanced production schedule would be sedan, wagon, sedan, hardtop, sedan, wagon, sedan, hardtop, etc.

Apparently, the level scheduling model (2)–(5) does not take care about the work contents of the products which might be different. Hence, it neither allows to control the risk of conveyor stoppage nor enables to reduce the cost for utility work.

Recently, some other level scheduling topics have been addressed in the literature. First, the min–max criterion has been the subject of research instead of the min–sum criterion in the paper of Steiner and Yeomans 1993. Second, Steiner and Yeomans 1994 study the Pareto optimality version of the problem which has both, the min–sum and the min–max objective. Finally, multi–level approaches can be found in Miltenburg and Goldstein 1991, Miltenburg and Sinnamon 1992, Steiner and Yeomans 1996, and Kubiak, Steiner, and Yeomans 1997.

2.2 Car Sequencing

The subject of the so–called 'car sequencing' approach (cp. Parello, Kabat, and Wos 1986 and Parello 1988) also is to derive schedules for the final assembly line. In contrast to level scheduling it 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 affect station loads and that the implied part demand for the output of the feeder process has to be taken into account also. More precisely, the goal is to produce a sequence for the final assembly level while taking maximal station loads (capacities) and implied part usages explicitely 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.

The central part of the car sequencing approach shall be explained by means of the following example (which is of relevance in the final assembly of cars): 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. Constraints with respect to the used parts can be dervied similarly.

In order to describe the car sequencing model formally we use the following parameters and variables:

O : set of options, index o

$a_{v,o}$: 1, if variant $v \in V$ requires option $o \in O$ (0, otherwise)

$H_o : N_o$: at most H_o out of N_o successively sequenced copies may require option $o \in O$

T_o : set of constrained periods for option $o \in O$, i.e. $T_o = T \setminus \{|T| - N_o + 2, \ldots, |T|\}$

$Q^+_{t,o}$: set of forward–constrained periods for option $o \in O$ w.r.t. period t, i.e. $Q^+_{t,o} = \{t, \ldots, t + N_o - 1\}$

$x_{v,t}$: 1, variant $v \in V$ is assigned to period $t \in T$ (0, otherwise)

Table 1: Instance 1 — Data

o	$H_o : N_o$	$v=1$	$v=2$	$v=3$	$v=4$	$v=5$	$v=6$
1	2 : 3		×	×	×		×
2	2 : 4			×	×		
3	3 : 5		×			×	
4	2 : 6	×			×		
	$\lvert D_v \rvert$	4	1	2	2	2	3

Table 2: Instance 1 — Ideal Positions $f_{v,i}$

v	$\lvert D_v \rvert$	i			
1	4	1.75	5.25	8.75	12.25
2	1	7.00			
3	2	3.50	10.50		
4	2	3.50	10.50		
5	2	3.50	10.50		
6	3	2.30	7.00	11.60	

Table 3: Level Schedule of Instance 1 — Optimal Solution with Objective 10.8

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
v	1	6	3	4	5	1	2	6	1	3	4	5	6	1
$o=1$		×	×	×			×	×		×	×		×	
$o=2$			×	×						×	×			
$o=3$					×		×					×		
$o=4$	×			×		×			×		×			×

Table 4: Car Sequence of Instance 1 — Feasible Solution

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
v	1	1	2	3	5	3	1	4	6	5	6	6	1	4
$o=1$			×	×		×		×	×		×	×		×
$o=2$				×		×		×						×
$o=3$			×		×					×				
$o=4$	×	×					×	×					×	×

Based on these definitions the car sequencing model can be stated as a constraint satisfaction problem by expressions (6)–(9).

$$\sum_{v \in V} x_{v,t} = 1 \qquad t \in T \tag{6}$$

$$\sum_{t \in T} x_{v,t} = |D_v| \qquad v \in V \tag{7}$$

$$\sum_{v \in V} \sum_{\tau \in Q^+_{t,o}} a_{v,o}\, x_{v,\tau} \leq H_o \qquad o \in O, t \in T_o \tag{8}$$

$$x_{v,t} \in \{0,1\} \qquad v \in V, t \in T \tag{9}$$

Equations (6) are assignment constraints, i.e. exactly one copy has to be produced in each period. Inequalities (7) require to manufacture the prespecified number of copies of each product. Inequalities (8) restrict sequences to be feasible only if the "$H_o : N_o$" constraints are fulfilled. Finally, constraints (9) define the decision variables to be binary.

We 'extracted' the car sequencing model (6)–(9) out of the verbal description and the codes given in Parello, Kabat, and Wos 1986, Dincbas, Simonis, and van Hentenryck 1988, and Parello 1988. Note, the car sequencing model (6)–(9) is $\mathcal{NP}$–complete (cp. problem MP1 in Garey and Johnson 1979).

Recall instance 1 (cp. Table 1) where in addition to what was needed in the level scheduling context column two and rows two to five are of relevance now. (Note, an '×' indicates which variants require which options.) Without surprise, a closer look at the optimal level scheduling solution for this instance (cp. Table 3) reveals that it is infeasible within the periods $2 \leq t \leq 4$ with respect to the sequencing constraints imposed by option 1 and in the periods $1 \leq t \leq 6, 4 \leq t \leq 9, 6 \leq t \leq 11$, and $9 \leq t \leq 14$ with respect to the sequencing constraints imposed by option 4. Table 4 provides a feasible car sequencing solution for this instance.

The car sequencing model (6)–(9) pays attention to the work contents of the products. Hence, it allows to control the risk of conveyor stoppage or — depending on the preferences of management — enables to control the cost for utility work. Note, 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 open or closed station. This is advantageous 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 explicitely in order to control the risk of conveyor stoppage or the cost for utility work; cp., e.g., Yano and Rachamadugu 1991, Bard, Shtub, and Josh 1994, Tsai 1995, Bolat 1997, and Kim, Hyun, and Kim 1996.

In the case where each of the stations in series has a finite capacity buffer blocking of the line may also occur if one of the buffers is full. If the amount of storage needed is a function of the sequence imposed then sequencing has to take care of this type of blocking also. A related assembly line sequencing problem with blocking due to finite capacity buffers has been studied in McCormick, Pinedo, Shenker, and Wolf 1989.

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 1986, Dincbas, Simonis, and van Hentenryck 1988, Parello 1988, and Drexl and Jordan 1995. Unfortunateley, the performance of these approaches in general is totally disappointing. Drexl and Jordan 1995 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 fast workstation. As shown in this reference also, to use standard MIP–solvers is impractical, too.

At the end of this section we have the following intermediate result: While the level scheduling model covers 'smoothing' capabilities which are very relevant in a JIT environment the car sequencing model provides equations which suitably address constraints imposed on station loads and on part usages. Consequently, to combine both features within one single model — which is the subject of the remainder of the paper — gives a very interesting approach of practical importance which has not been studied before. Unfortunately, this combination comes up with an $\mathcal{NP}$–hard optimization problem. Hence, to compute tight lower bounds, the focus of the next section, is of primary concern.

3 Integrated Level Scheduling/Car Sequencing

This section is dedicated to the computation of lower bounds for the combined level scheduling/car sequencing problem. The lower bounds are calculated by solving the LP–relaxation of a restricted master problem via column generation. The use of column generation techniques is neat because the master problem is a set partitioning model with an exponential number of columns. For an introduction to column generation cp. e.g. Bradley, Hax, and Magnanti 1977.

3.1 Set Partitioning/Column Generation

The basic idea is to iteratively compute sequences for each of the variants by means of a shortest path model. From the set of sequences on hand those are chosen by the set partitioning model which respect the $H_o : N_o$ sequencing constraints (in the LP–relaxation).

In order to describe the set partitioning model formally we use the following parameters and variables:

$v(k)$	:	variant $v \in V$ column k is associated with
$\mathcal{S}^v$	:	set of columns representing sequences for variant $v \in V$, i.e. $\mathcal{S}^v = \{k \mid v(k) = v\}$, index k
$b_{v,i,k,t}$	:	1, if copy $i \in D_v$ of variant $v \in V$ is assigned to period $t \in T$ within sequence k where $v(k) = v$ (0, otherwise)
$F_{v(k),i,t}$	:	squared deviation of copy $i \in D_{v(k)}$ of variant $v(k)$ within sequence k from its ideal position $f_{v(k),i}$
c_k	:	objective function coefficient of sequence k related to variant $v(k)$, i.e. $c_k = \sum_{i \in D_{v(k)}} \sum_{t \in T} F_{v(k),i,t}$
y_k	:	1, if sequence k is part of the optimal solution (0, otherwise)

A formal definition of the functions $F_{v(k),i,t}$ is given by equation (10).

$$F_{v(k),i,t} = (t - f_{v(k),i})^2 \cdot b_{v(k),i,k,t} \tag{10}$$

Based on these definitions the (restricted) master problem can be stated by equations (11)–(15) as a set partitioning model.

$$\min \sum_{v \in V} \sum_{k \in \mathcal{S}^v} c_k \, y_k \tag{11}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{S}^v} y_k = 1 \qquad v \in V \tag{12}$$

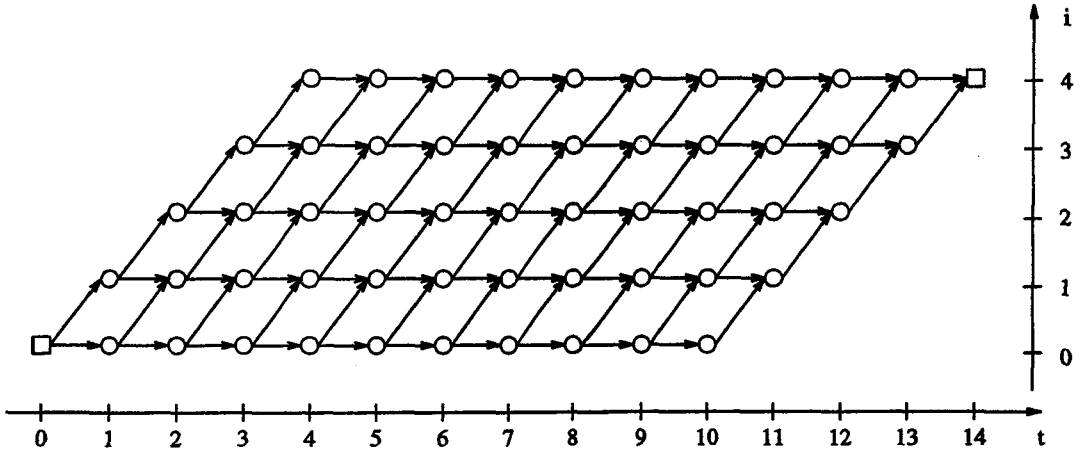
$$\sum_{v \in V} \sum_{i \in D_v} \sum_{k \in \mathcal{S}^v} b_{v,i,k,t} \, y_k = 1 \qquad t \in T \tag{13}$$

$$\sum_{v \in V} \sum_{i \in D_v} \sum_{k \in \mathcal{S}^v} \sum_{\tau \in Q^+_{t,o}} a_{v,o} \, b_{v,i,k,\tau} \, y_k \leq H_o \qquad o \in O, t \in T_o \tag{14}$$

$$y_k \in \{0,1\} \qquad k \in \cup_{v \in V} \mathcal{S}^v \tag{15}$$

The objective (11) is to select a subset of columns at minimum costs. Equations (12) make exactly one sequence per variant to be part of the solution, while equations (13) require to assign exactly one copy of any variant to each period. Finally, restrictions (14) are the sequencing constraints.

Figure 2: Instance 1 — Basic Structure of the Shortest Path Graph for Variant 1



3.2 Shortest Path Model

For each variant $v \in V$ sequences are computed by solving shortest path problems. Figure 2 illustrates the (shortest path) graph for variant 1 of instance 1 with $|D_1| = 4$ and

$|T| = 14$. Diagonal (horizontal) arcs denote the decision (not) to produce a copy of variant 1. This (shortest path) graph with source node '□' (left bottom) and sink node '□' (top right) covers all the possibilities to assign four copies to the 14 available periods.

In order to describe the shortest path model formally we use the following parameters and variables:

$Q^-_{t,o}$: set of backward–constrained periods for option $o \in O$ w.r.t. period t, i.e. $Q^-_{t,o} = \{t - N_o + 1, \ldots, t\} \cap T_o$

N^v : set of nodes of the graph associated with variant $v \in V$, i.e. $N^v = \{(i,t) \,|\, i \in D_v \cup \{0\}, t \in \{i, \ldots, |T| - |D_v| + i\}\}$

E^v : set of arcs of the graph associated with variant $v \in V$, i.e. $E^v = \{(h,j) \in N^v \times N^v \,|\, (h = (i-1, t-1) \text{ and } j = (i,t)) \text{ or } (h = (i,t-1) \text{ and } j = (i,t))$

$d^v_{h,j}$: original weight of arc $(h,j) \in E^v$

λ_v : dual variable associated with the one sequence per variant constraint (12), $\lambda_v \in \mathbb{R}$

μ_t : dual variable associated with the one variant per period constraint (13), $\mu_t \in \mathbb{R}$

$\pi_{o,t}$: dual variable associated with the $H_o : N_o$ constraint (14), $\pi_{o,t} \geq 0$

$g^v_{h,j}$: updated weight of arc $(h,j) \in E^v$

$x^v_{h,j}$: 1, if arc $(h,j) \in E^v$ is element of the shortest path (0, otherwise)

Equation (16) formally defines the original weights $d^v_{h,j}$ of arcs $(h,j) \in E^v$. Equation (17) explains how to calculate the arc weights taking the dual variables μ_t and $\pi_{o,t}$ into account.

$$d^v_{h,j} = \begin{cases} 0 & , \quad h = (i, t-1), j = (i,t) \\ (t - f_{v,i})^2 & , \quad h = (i-1, t-1), j = (i,t) \end{cases} \tag{16}$$

$$g^v_{h,j} = \begin{cases} d^v_{h,j} = 0 & , \quad h = (i, t-1), j = (i,t) \\ d^v_{h,j} - \mu_t - \sum_{o \in O} a_{v,o} \sum_{\tau \in Q^-_{t,o}} \pi_{o,\tau} & , \quad h = (i-1, t-1), j = (i,t) \end{cases} \tag{17}$$

Based on these definitions the objective function of the shortest path model for variant v can be stated by equation (18).

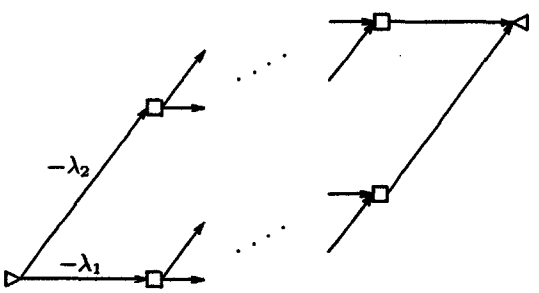
$$Z^v = \min \sum_{(h,j) \in E^v} g^v_{h,j} \, x^v_{h,j} - \lambda_v \tag{18}$$

Note that the shortest path graph is acyclic with node weights $g^v_{h,j} \in \mathbb{R}$. Because of the topological structure, the shortest path problems are solvable in linear time.

Finally, pricing out occurs if $\min\{Z^v \,|\, v \in V\} \geq 0$. This is accomplished by computing the shortest path in the overall shortest path graph comprising all the variants. Figure 3

shows what the overall shortest path graph is all about for the case of two variants. We just add a source node '$\triangleright$' and a sink node '$\triangleleft$', connect the variant–specific shortest path graphs to the source and sink node and count for the (constant) dual variables λ_v as shown in Figure 3. In our implementation, we compute at most $|V|$ columns per iteration, one for each variant $v \in V$ with $Z^v < 0$.

Figure 3: Overall Shortest Path Graph for Two Variants



If we apply the column generation technique to instance 1 given in Table 1 then the optimal solution of the LP–relaxation of the set partitiong model is integral — and hence we are done. Table 5 provides the optimal solution with an objective function of about 24 — which is more than twice as large as in the unconstrained case.

Table 5: SPP/CG Sequence of Instance 1 — Optimal Solution with Objective $24.30\bar{5}$

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
v	1	4	6	5	3	6	1	1	2	3	5	6	4	1
$o = 1$		×	×		×	×			×	×		×	×	
$o = 2$		×			×					×			×	
$o = 3$				×					×		×			
$o = 4$	×	×					×	×					×	×

Unfortunately, the size of the shortest path graph is growing exponentially in terms of $D_v, v \in V$, and T. Fortunately, simple techniques allow to reduce the size of the shortest path graph in a preprocessing stage.

3.3 Shortest Path Graph Reduction

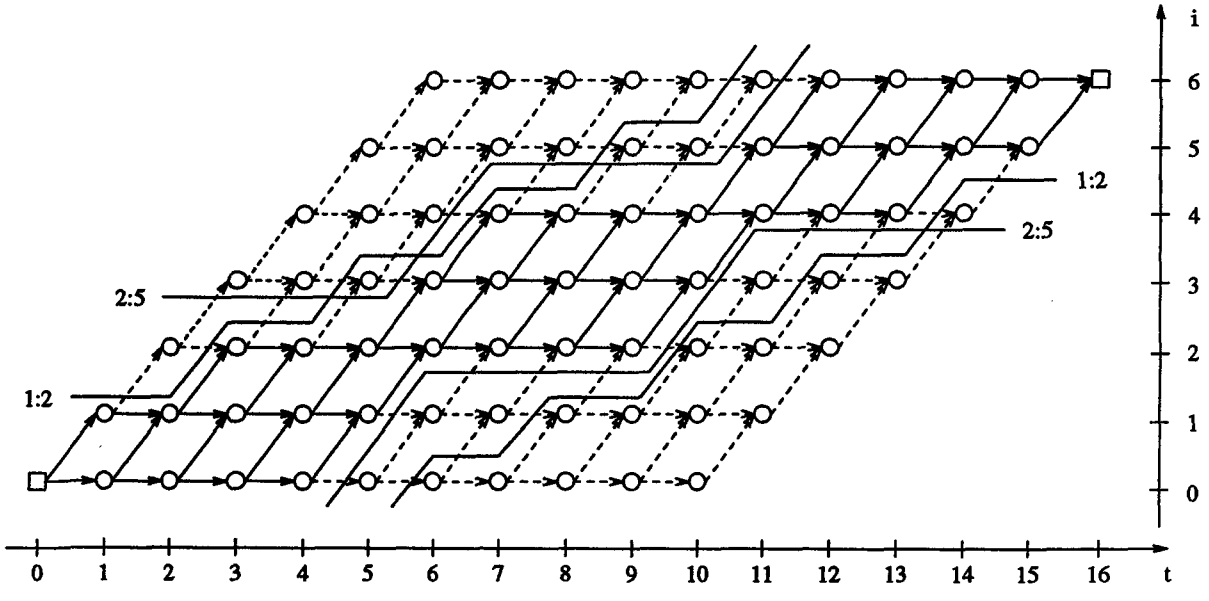
The number of potential columns $|\mathcal{S}^v|$ of variant v is defined by

$$|\mathcal{S}^v| = \binom{|T|}{|D_v|} \qquad \text{so} \qquad \sum_{v \in V} \binom{|T|}{|D_v|}$$

is the number of potential columns of the master problem.

Figure 4 explains the basic idea of the shortest path graph reduction techniques by means of an instance with $|T| = 16$ and $|D_v| = 6$. Paths which correspond to an in-sequence production of more than H_o out of N_o copies can be eliminated without affecting the set of feasible solutions. Figure 4 shows via dotted arcs that different $H_o : N_o$ constraints drop different subsets of arcs and nodes. Note, the shortest path graph reduction technique can be applied during a preprocessing stage and needs not be done over and over again.

Figure 4: Instance 2 — Shortest Path Graph Reduction — $|T| = 16, |D_v| = 6$



More formally, let us define a 0–1–valued function $\delta_v : N^v \rightarrow \{0,1\}$ which assigns the value zero to a node if this node (and arcs pointing to/from that node) can be deleted from the shortest path graph associated with variant $v \in V$. Since a variant $v \in V$ may require several options $o \in O, \delta_v$ can be defined as

$$\delta_v(i,t) = \prod_{o \in O} \delta_{v,o}(i,t) \qquad v \in V, (i,t) \in N^v \tag{19}$$

where $\delta_{v,o} : N^v \rightarrow \{0,1\}$ are 0–1–valued functions each of which, in analogy to δ_v, yields zero, if a node $(i,t) \in N^v$ can be deleted due to a restriction on option $o \in O$. We define

$$\delta_{v,o}(i,t) = \begin{cases} 1, & a_{v,o} = 0 \\ \delta^U_{v,o}(i,t) \cdot \delta^L_{v,o}(i,t), & \text{otherwise} \end{cases} \qquad v \in V, (i,t) \in N^v, o \in O \tag{20}$$

In turn, $\delta^U_{v,o} : N^v \to \{0,1\}$ is a 0–1–valued function which reveals those nodes in the upper left part of the graph that can be deleted. Likewise, $\delta^L_{v,o} : N^v \to \{0,1\}$ is a 0–1–valued function that indicates which nodes in the lower right part can be deleted. These functions are defined by (21) and (22) for all $v \in V, (i,t) \in N^v$ and $o \in O$. $\lfloor \alpha \rfloor$ denotes the greatest integer smaller than or equal to α.

$$\delta^U_{v,o}(i,t) = \begin{cases} 0, & i \geq 1 + H_o \left(1 + \left\lfloor \frac{t-i}{N_o - H_o} \right\rfloor\right) \\ 1, & \text{otherwise} \end{cases} \tag{21}$$

$$\delta^L_{v,o}(i,t) = \begin{cases} 0, & i \leq |D_v| - \left(1 + H_o \left(1 + \left\lfloor \frac{(|T|-t)-(|D_v|-i)}{N_o - H_o} \right\rfloor\right)\right) \\ 1, & \text{otherwise} \end{cases} \tag{22}$$

The reduction of the size of the shortest path graph is important because of three reasons: First, although requiring only a linear number of steps, solving smaller shortest path problems is faster than solving larger ones. Second, updating superfluous arcs of the shortest path graph would just waste CPU–time. Third, generating columns of the set partitioning model which subsequently are suppressed by the $H_o : N_o$ constraints (14) increases the size of the set partitioning model and, hence, the time to solve its LP–relaxation.

4 Computational Evaluation

The approach presented in this paper explores a new area, hence, no established test-bed is available. Therefore, first, we elaborate on the instances which are used in our computational study. Second, numerical results shall be presented.

4.1 Instance Generation

Even in current literature, the systematic generation of test instances does not receive much attention. Generally, two possible approaches can be found adopted in literature when having to come up with test instances. First, practical cases. Their strength is their high practical relevance while the obvious drawback is the absence of any systematic structure allowing to infer any general properties. Thus, even if an algorithm performs good on some practice cases, it is not guaranteed that it will continue to do so on other instances as well. Second, artificial instances. Since they are generated randomly according to predefined specifications, their plus lies in the fact that fitting them to certain requirements such as given probability distributions poses no problems. However, they may reflect situations with little or no resemblance to any problem setting of practical interest. Hence, an algorithm performing well on several such artificial instances may or may not perform

satisfactorily in practice. Because of the unavailability of practical cases we must choose the second approach here.

The generator we used in our computational study was designed such that the instances have two properties:[1] First, each instance has at least one feasible solution which is the result of the instance generation. This is an important aspect, because to compute lower bounds for an instance which does not have any feasible solution would make no sense. Clearly, having an initial feasible solution provides an initial upper bound also. Second, each instance comprises a non–trivial example of the integrated level scheduling/car sequencing variety. Non–triviality is assured by requesting each option 'as often as possible'. We decided to use $T, O, H_o : N_o$ as input and to produce V, D_v, and the matrix $(a_{v,o})$ as output.

The basic working principle of the generator is best illustrated by the use of the example given in Table 6. The first column specifies the input while columns 2 to 12 provide the output. More precisely, the $H_o : N_o$ constraints of each option are considered separately. Taking option $o = 1$ as an example we get row "$H_1 : N_1$" of Table 6. The procedure fills in as many options '×' as possible which — ceteris paribus — makes the $H_o : N_o$ constraints more difficult to maintain. We drop the first $2 = \min\{H_o \mid o \in O\}$ columns (because here every row has an entry '×') and generate $|T| + \min\{H_o \mid o \in O\}$ columns in total. Finally, we identify identical columns which gives the number of variants V and thus the number of copies $|D_v|$ for each variant $v \in V$. For the example of Table 6 we get $|V| = 5, |D_v| = 2, v \in V$, and

$$(a_{v,o}) = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

For experimental purposes we used the following testbed:

- $|O| \in \{3, 5, 7\}$
- $|T| \in \{10, 15, 20, 30, 40, 50\}$
- $\{H_o : N_o\} \in \left\{ \begin{array}{l} \{1:2 \ , \ 4:5 \ , \ 7:8 \ ; \ 3:4 \ , \ 6:7 \ ; \ 2:3 \ , \ 5:6\} \\ \{1:8 \ , \ 1:7 \ , \ 2:8 \ ; \ 1:6 \ , \ 2:7 \ ; \ 3:8 \ , \ 1:5\} \end{array} \right\}$

Note, the 'grouping' of the $H_o : N_o$ constraints by semicolons is done with respect to the numbers of options $|O| \in \{3, 5, 7\}$. In total we have $3 \times 6 \times 2 = 36$ instances.

Note that the choice of the set $\{H_o : N_o\}$ specifies two subsets of sequencing constraints with a different difficulty of finding feasible solutions. More precisely, as shown in Table 7 the $H_o : N_o$ constraints along the main diagonal comprise those constraints which are easy to maintain while those in the top right corner are very difficult to ascertain. In what follows we will refer to both as '$H_o : N_o$ — easy' and '$H_o : N_o$ — hard', respectively.

[1] All the instances used in this study are available in the internet via anonymous ftp (ftp://ftp.wiso.uni-kiel.de/pub/operations-research/cars-jit).

Table 6: Instance Generation — Input and Output

$\|T\| = 10$	t		1	2	3	4	5	6	7	8	9	10
$\|O\| = 3$	v		1	2	3	4	5	5	1	3	2	4
$H_1 : N_1 = 2 : 3$	×	×		×	×		×	×		×	×	
$H_2 : N_2 = 4 : 5$	×	×	×	×		×	×	×	×		×	×
$H_3 : N_3 = 3 : 6$	×	×	×				×	×	×			

Table 7: Degree of Difficulty — $H_o : N_o$

		N_o							
		1	2	3	4	5	6	7	8
H_o	1		×			×	×	×	×
	2			×				×	×
	3				×				×
	4					×			
	5						×		
	6							×	
	7								×
	8								

Table 8: Instance 1 — Initial Columns

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$v = 1$		1			1				1			1		
$v = 2$							1							
$v = 3$				1							1			
$v = 4$				1							1			
$v = 5$				1							1			
$v = 6$		1					1					1		

4.2 Computational Results

The methods described earlier have been imlemented in GNU C using the CPLEX callable library (LP–solver; cp. Bixby and Boyd 1996) on a 200 MHz Pentium Pro machine with 128 MB main memory.

Each instance has been solved starting with $|V|$ initial columns, one for each variant $v \in V$. Furthermore, the incidence vector $b_{v,i,v,t}, v \in V$, has been initialized according to equation (23). Table 8 shows what the outcome of equation (23) is for the instance in Table 1. Finally, an initial feasible solution is computed with the Big M method.

$$b_{v,i,v,t} = \begin{cases} 1, & t = \lfloor f_{v,i} + 0.5 \rfloor \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

Table 9 provides the lower and upper bounds obtained in our experiments. The entries within each cell have the following meaning:

- The symbol "•" indicates that the method yields a binary feasible solution, that is, we have an optimal solution.
- The symbol "o" indicates that the method does not yield a binary feasible solution. However, solving the master problem with all generated columns and with binary constraints gives a feasible solution with identical objective function value. Thus, again the methods provides the optimal objective function value.
- A value "$\underline{X\%}$" means that the method does not give a binary feasible solution. However, solving the master problem with all generated columns and with binary constraints leads to a feasible solution. The gap between upper and lower bound (UB and LB) is $X = 100 \cdot \frac{UB-LB}{LB}$. We conjecture that large gaps are due to poor upper bounds.
- A value "X%" means again that the method does not find a binary feasible solution. Unfortunately, the master problem with all generated columns and with binary constraints is either not feasible or cannot be solved within reasonable time. Thus the only available upper bound is the one given by the instance generator. Again, X is the gap between upper and lower bound. We are highly convinced that large gaps are due to poor upper bounds.

Table 10 provides information about the size of the last master problem. A value in this table is the ratio of the number of generated columns and the number of all possible columns, i.e.

$$100 \cdot \frac{\sum_{v \in V} |S^v|}{\sum_{v \in V} \left(\frac{|T|}{|D_v|} \right)}.$$

Table 9: Lower and Upper Bounds

	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$
$\|T\| = 10$	•	•	•	18%	•	•
$\|T\| = 15$	o	•	•	o	•	6%
$\|T\| = 20$	151%	•	o	14%	49%	12%
$\|T\| = 30$	71%	246%	100%	•	167%	98%
$\|T\| = 40$	•	281%	76%	255%	302%	122%
$\|T\| = 50$	314%	148%	124%	497%	168%	91%
	$H_o : N_o$ — easy			$H_o : N_o$ — hard		

Table 10: Size of the Last Master Problem — Percentages

	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$
$\|T\| = 10$	11.11	29.30	68.00	14.55	17.88	66.40
$\|T\| = 15$	1.48	6.56	37.56	2.11	1.72	8.04
$\|T\| = 20$	0.12	0.51	4.88	0.15	0.10	1.47
$\|T\| = 30$	0.00	0.00	0.03	0.00	0.00	0.02
$\|T\| = 40$	0.00	0.00	0.00	0.00	0.00	0.00
$\|T\| = 50$	0.00	0.00	0.00	0.00	0.00	0.00
	$H_o : N_o$ — easy			$H_o : N_o$ — hard		

Table 11: Size of the Last Master Problem — Absolute Numbers

	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$
$\|T\| = 10$	40	63	85	40	54	83
$\|T\| = 15$	92	120	169	68	115	135
$\|T\| = 20$	198	206	274	119	165	241
$\|T\| = 30$	444	611	569	283	384	498
$\|T\| = 40$	1,209	971	1,129	478	637	735
$\|T\| = 50$	2,240	2,293	1,409	1,616	1,180	1,179
	$H_o : N_o$ — easy			$H_o : N_o$ — hard		

Table 12: Run–Time Performance

	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$
$\|T\| = 10$	0.03	0.06	0.10	0.03	0.03	0.03
$\|T\| = 15$	0.19	0.27	0.62	0.08	0.18	0.18
$\|T\| = 20$	0.88	1.11	2.36	0.31	0.53	1.05
$\|T\| = 30$	9.31	22.17	24.14	2.41	5.97	9.28
$\|T\| = 40$	119.78	124.15	256.86	19.56	28.45	39.19
$\|T\| = 50$	1,073.94	2,447.47	415.50	355.15	161.90	147.25
	$H_o : N_o$ — easy			$H_o : N_o$ — hard		

Table 13: Run–Time Performance — One Column Per Iteration

	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$
$\|T\| = 10$	0.06	0.15	0.26	0.05	0.06	0.08
$\|T\| = 15$	0.39	0.69	1.10	0.12	0.26	0.49
$\|T\| = 20$	2.77	4.08	6.35	0.56	1.14	2.65
$\|T\| = 30$	24.92	63.36	67.71	5.17	14.52	18.38
$\|T\| = 40$	321.76	430.19	845.16	19.74	68.85	92.11
$\|T\| = 50$	1,605.81	>3,600.00	1,826.72	389.31	270.07	335.89
	$H_o : N_o$ — easy			$H_o : N_o$ — hard		

Table 14: Size of the Last Master Problem — Percentages — One Column Per Iteration

	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$	$\|O\| = 3$	$\|O\| = 5$	$\|O\| = 7$
$\|T\| = 10$	10.00	24.19	47.20	13.82	13.91	43.20
$\|T\| = 15$	1.32	5.51	22.00	1.51	1.15	6.19
$\|T\| = 20$	0.13	0.50	3.97	0.15	0.08	1.19
$\|T\| = 30$	0.00	0.00	0.03	0.00	0.00	0.03
$\|T\| = 40$	0.00	0.00	0.00	0.00	0.00	0.00
$\|T\| = 50$	0.00	0.00	0.00	0.00	0.00	0.00
	$H_o : N_o$ — easy			$H_o : N_o$ — hard		

The value 0.00 in the table means that the size actually is below 0.005%. We see that especially for large instances only a very small percentage of all potential columns is generated. This proves that column generation indeed pays off for our problem. Table 11 shows the size of the last master problem also, now in absolute figures. The run–times measured in CPU–seconds are provided in Table 12.

For getting the results presented so far, we generated at most $|V|$ columns per iteration, one for each variant v with $Z^v < 0$. To reveal that this is indeed a good idea, we also show the results when we generated at most one column per iteration which is determined by the overall shortest path. Table 13 provides the run–times while Table 14 shows the figures for the size of the last master problem. Comparing this with Table 12 and Table 10 the run–time upon termination is much shorter when more than just one column is generated per iteration. The price we have to pay for this is that slightly more columns are generated in total.

5 Special Cases and Extensions

In what follows we will point to some special cases covered by the model formulation. In addition, an important extension will be outlined also.

(i) We conjecture that the approach described in this paper is valid for many functions $F_{v,i,t}$ which can be 'locally' computed which means that $F_{v,i,t}$ only depends on copy i but not on other copies. Important special cases are where $F_{v,i,t}$ is the l_p–norm for $-\infty < p < \infty$. Note that this situation covers absolute deviations and weighted earliness/tardiness for $p = 1$ also. Note also, that some values p don't make sense, e.g. $p \in [0, 1)$.

Clearly, in practice the decision maker has to chose an appropriate value for p. Then the question arises how robust optimal solutions are with respect to changing values of p. Table 15 shows for instance 1 optimal objectives $Z(p)$ for different values of p. Apparently, even more interesting than what is presented in Table 15 would be to know how sensitive optimal sequences are with respect to changes of p.

(ii) If we replace the min–sum criterion through a min–max objective (l_p–norm for $p = \infty$) then only minor changes of the set partitiong/column generation approach and of the shortest path model are required. Note, however, that in the presence of $H_o : N_o$ constraints a min–max objective seems to be of minor relevance.

(iii) The approach covers also the situation where the work load of a station stems from more than one option. Consider the situation presented in the first six rows of Table 16. Assume that the load of a station depends on both the options $o = 2$ and $o = 3$. Similarly, the load of another station may jointly depend on the options $o = 2$ and $o = 4$. Then we generate two additional options $o = 6$ and $o = 7$. An added option gets a '×' if and only if variant v requies both options. Then each sequencing constraint $H_o : N_o$ of the newly generated options must express the maximal workload as a function of the number of successively sequenced copies requiring both options.

Table 15: Stability of Optimal Solutions With Respect to p

p	1	2	3	4	5
$Z(p)$	16.00	24.31	43.65	85.98	179.37

Table 16: Instance 3 — Data

o	$H_o : N_o$	$v=1$	$v=2$	$v=3$	$v=4$	$v=5$
1	2 : 3	×	×			×
2	3 : 4		×	×	×	×
3	3 : 7	×		×		×
4	2 : 6		×		×	
5	4 : 5				×	×
6	2 : 8			×		×
7	2 : 9		×		×	

6 Summary and Future Work

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. The use of JIT methods makes it possible to satisfy customers' demands for several products without holding large inventories and without incurring large shortages. In assembly systems, products usually are mounted on a conveyor belt. Operators move along with the belt while working on a product. 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. Obviously, the management style behind such distinct approaches is quite different. Anyway, it is desirable to distribute products with high work content evenly in order to reduce the risk of conveyor stoppage or the cost for utility work.

This paper presents a nonlinear integer programming model which covers both the balancing requirements of level scheduling and the constraints of car sequencing. Hence, it allows to control the risk of conveyor stoppage or — depending on the preferences of management — enables to control the cost for utility work while producing 'smooth' JIT schedules. For the solution of the problem we specify a set partitioning/column generation approach. Solving the LP–relaxation of this model by column generation provides tight lower bounds for the optimal objective function value.

As already mentioned in the introduction, in practice usually subsequences consisting

of only a few copies are used in a cyclic manner. The methods developed in this paper allow to compute lower bounds for instances having up to 50 copies, while for up to 20 copies in general we get feasible, oftenly optimal sequences. Hence, considerable improvements have been achieved.

Future research should be directed towards the development of local search methods which allow to compute feasible solutions for large–size problem instances. Furthermore, branch–and–price methods (cp. Barnhart, Johnson, Nemhauser, Savelsbergh, and Vance 1997) and exact branch–and–cut/row and column generation methods (cp. for instance van den Akker 1995) shall be the subject of research.

References

[1] BAKER, K.R., SCUDDER, G.D. "Sequencing with earliness and tardiness penalties: a review". *Operations Research*, 38:22–36, 1990.

[2] 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.

[3] BARNHART, C., JOHNSON, E.L., NEMHAUSER, G.L., SAVELSBERGH, M.W.P., VANCE, P.H. "Branch–and–price: column generation for huge integer programs". *Operations Research*, 1997 (to appear).

[4] BERKLEY, B.J. "A review of the kanban production control research literature". *Production and Operations Management*, 1:393–411, 1992.

[5] BITRAN, G.R., CHANG, L. "A mathematical programming approach to a deterministic Kanban system". *Management Science*, 33:427–441, 1987.

[6] BIXBY, N., BOYD, E. *Using the CPLEX Callable Library.* CPLEX Optimization Inc., 7710–T Cherry Park, Houston, TX, 1996.

[7] BOLAT, A. "Efficient methods for sequencing minimum job sets on mixed model assembly lines". *Naval Research Logistics*, 44:419–437, 1997.

[8] BRADLEY, S.P., HAX, A.C., MAGNANTI, T.L. *Applied Mathematical Programming.* Addison–Wesley, Reading, 1977.

[9] DINCBAS, 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)*, 290–295. München, 1988.

[10] DREXL, A., JORDAN, C. "Materialflußorientierte Produktionssteuerung bei Variantenfließfertigung". *Zeitschrift für betriebswirtschaftliche Forschung*, 47:1073–1087, 1995.

[11] GAREY, M.R., JOHNSON, D.S. *Computers and Intractability — A Guide to the Theory of NP–Completeness.* Freeman and Company, San Francisco, 1979.

[12] HOOGEVEN, J.A., LENSTRA, J.K., VAN DE VELDE, S.L. "Sequencing and scheduling: an annotated bibliography". In DELL'AMICO, M., MAFFIOLI, F., MARTELLO, S., editors, *Annotated Bibliographies in Combinatorial Optimization.* John Wiley & Sons, New–York, 1997 (to appear).

[13] INMAN, R.R., BULFIN, R.L. "Sequencing JIT mixed–model assembly lines". *Management Science*, 37:901–904, 1991.

[14] 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.

[15] KRAJEWSKI, L.J., KING, B.E., RITZMAN, L.P., WONG, D.S. "Kanban, MRP, and shaping the manufacturing environment". *Management Science*, 33:39–57, 1987.

[16] KUBIAK, W. "Minimizing variation of production rates in just–in–time systems: a survey". *European Journal of Operational Research*, 66:259–271, 1993.

[17] 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.

[18] KUBIAK, W., SETHI, S. "Optimal just–in–time schedules for flexible transfer lines". *The International Journal of Flexible Manufacturing Systems*, 6:137–154, 1994.

[19] KUBIAK, W., STEINER, G., YEOMANS, J.S. "Optimal level schedules for mixed–model, multi–level just–in–time assembly systems". *Annals of Operations Research*, 69:241–259, 1997.

[20] MCCORMICK, S.T., PINEDO, M.L., SHENKER, S., WOLF, B. "Sequencing in an assembly line with blocking to minimize cycle time". *Operations Research*, 37:925–935, 1989.

[21] MILTENBURG, G.J. "Level schedules for mixed–model assembly lines in just–in–time production systems". *Management Science*, 35:192–207, 1989.

[22] MILTENBURG, G.J., GOLDSTEIN, T. "Developing production schedules which balance part usage and smooth production loads for just–in–time production systems". *Naval Research Logistics*, 38:893–910, 1991.

[23] MILTENBURG, G.J., SINNAMON, G. "Algorithms for scheduling mixed–model just–in–time production systems". *IIE Transactions*, 24:121–130, 1992.

[24] MILTENBURG, G.J., STEINER, G., YEOMANS, J.S. "A dynamic programming algorithm for scheduling mixed–model, just–in–time production systems". *Mathematical and Computer Modelling*, 13(3):57–66, 1990.

[25] MONDEN, Y. *Toyota Production Systems.* Industrial Engineering and Management Press, Norcross, GA, 1983.

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

[27] 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.

[28] RACHAMADUGU, R.M.V., YANO, C.A. "Analytical tools for assembly line design and sequencing". *IIE Transactions*, 26(2):2–10, 1994.

[29] SCHOLL, A. *Balancing and Sequencing of Assembly Lines.* Physica, Heidelberg, 1995.

[30] STEINER, G., YEOMANS, J.S. "Level schedules for mixed–model, just–in–time processes". *Management Science*, 39:728–735, 1993.

[31] STEINER, G., YEOMANS, J.S. "A bicriterion objective for levelling the schedule of a mixed–model, JIT assembly process". *Mathematical and Computer Modelling*, 20(2):123–134, 1994.

[32] STEINER, G., YEOMANS, J.S. "Optimal level schedules in mixed–model, multi–level JIT assembly systems with pegging". *European Journal of Operational Research*, 95:38–52, 1996.

[33] 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.

[34] THOMOPOULOS, N.T. "Line balancing–sequencing for mixed–model assembly". *Management Science*, 14:B59–B75, 1967.

[35] TSAI, L.–H. "Mixed–model sequencing to minimize utility work and the risk of conveyor stopping". *Management Science*, 41:485–495, 1995.

[36] VAN DEN AKKER, J.M. *LP–Based Solution Methods for Single–Machine Scheduling Problems.* PhD–Thesis, Eindhoven University of Technology, 1995.

[37] VAN ZANTE–DE FOKKERT, J.I., DE KOK, T.G. "The mixed and multi model line balanacing problem: a comparison". *European Journal of Operational Research*, 100:399–412, 1997.

[38] YANO, C.A., RACHAMADUGU, R.M.V. "Sequencing to minimize work overload in assembly lines with product options". *Management Science*, 37:572–586, 1991.