ELSEVIER

Int. J. Production Economics 66 (2000) 159–169

international journal of production economics

www.elsevier.com/locate/dsw

Lot sizing and scheduling with sequence-dependent setup costs and times and efficient rescheduling opportunities

Knut Haase, Alf Kimms*

Institut für Betriebswirtschaftslehre, Christian Albrechts University of Kiel, Olshausenstr. 40, 24118 Kiel, Germany

Received 15 October 1998; accepted 17 August 1999

Abstract

This paper deals with lot sizing and scheduling for a single-stage, single-machine production system where setup costs and times are sequence dependent. A large-bucket mixed integer programming (MIP) model is formulated which considers only efficient sequences. A tailor-made enumeration method of the branch-and-bound type solves problem instances optimally and efficiently. The size of solvable cases ranges from 3 items and 15 periods to 10 items and 3 periods. Furthermore, it will become clear that rescheduling can neatly be done. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Lot sizing; Scheduling; Production planning and control; Rescheduling; Sequence-dependent setup times

1. Introduction

For many production facilities the expenditures for the setups of a machine depend on the sequence in which different items are scheduled on the machine. Especially when a machine produces items of different family types setups between items of different families are substantially more costly and time consuming than setups between items of the same family. In such a case a just-in-time philosophy will cause frequent setups, i.e. large total setup costs and long total setup times. To reduce the expenditures for the setups items may be produced in lots which satisfy the demand of several periods. The amount of a production quantity in a period which will be used to satisfy demand in later periods must then be held in inventory. This incurs holding costs. Therefore, we have to compute a schedule in which the sum of setup and holding costs is minimized. In the case of sequence-dependent setup costs the calculation of the setup costs requires the computation of the sequence in which items are scheduled, i.e. we have to consider sequencing and lot sizing simultaneously.

Despite its relevance only little research has been done in the area of lot sizing and scheduling with sequence-dependent setups. Some papers have been published which are related to the so-called discrete lot sizing and scheduling problem [4], denoted as DLSP. In the DLSP the planning horizon is divided into a large number of small periods (e.g. hours, shifts, or days). Furthermore, it is assumed that the production process always runs full periods without changeover and the setup state is not preserved over idle time. Such an "all-or-nothing" policy implies that at most one item will be

* Corresponding author.
E-mail address: Kimms@bwl.uni-kiel.de (A. Kimms).

produced per period. In [1] a DLSP-like model with sequence-dependent setup costs was considered first. For the DLSP with sequence-dependent setup costs (DLSPSD) an exact algorithm is presented in [2]. There the DLSPSD is transformed into a traveling salesman problem with time windows which is then used to derive lower bounds as well as heuristic solutions. An exact solution method for the DLSP with sequence-dependent setup costs and times (DLSPSDCT) is proposed in [3]. The optimal enumeration method proposed by [4] is based on the so-called batch sequencing problem (BSP). It can be shown that the BSP is equivalent to the DLSPSDCT for a restricted class of instances. The solution methods for the DLSPSDCT and the BSP require large working spaces, e.g. for instances with six items and five demands per item a working space of 20 megabytes is required. Recently, another new type of model has been published which is called the proportional lot sizing and scheduling problem (PLSP) [5]. The PLSP is based on the assumption that at most one setup may occur within a period. Hence, at most two items are producible per period. It differs from the DLSP regarding the possibility to compute continuous lot sizes and to preserve the setup state over idle time. A regret-based sampling method is proposed to consider sequence-dependent setup costs and times. In [6] an uncapacitated lot-sizing problem with sequence-dependent setup costs is considered. A heuristic for a static, i.e. constant demand per period, lot-scheduling problem with sequence-dependent setup costs and times is introduced in [7]. In [8] the so-called capacitated lot-sizing problem with sequence-dependent setup costs (CLSD) is presented. As in the PLSP, the setup state can be preserved over idle time. But in contrast to the DLSP and PLSP many items are producible per period. Hence, the DLSP and PLSP are called small-bucket problems and the CLSD is a large-bucket problem. For a review of lot-sizing and scheduling models we refer to [5]. A large-bucket problem with sequence-dependent setup costs and times is not considered in the literature so far. In this paper we will close this gap.

The text is organized as follows: in the next section we briefly describe the practical background that inspired our work on this subject. In Section 3, we give a mathematical formulation of the problem under concern. Afterwards, rescheduling is discussed in Section 4. In Section 5, an optimal enumeration method is outlined. The efficiency of the algorithm is tested by a computational study in Section 6.

2. A real-world case

Linotype-Hell AG, Kiel (Germany), manufactures high technology machines for the setting and printing business. A case study coauthored by one of the authors and provided in [9] informally describes the situation up to 1995 as follows. Although a commercial Siemens software package already is in use for production planning and control, demands are usually not met right in time. A milling machine, a so-called BZV07-2000, is identified as being the bottleneck. Hence, one searches for alternatives out of this situation. Buying an additional milling machine is considered to be too expensive. And using outside capacities is not wanted, because the know-how should be kept inside the firm. Hence, one of the authors suggests using the available capacity of the milling machine more efficiently by improved production planning.

The production planning problem for the milling machine indeed is a lot-sizing and scheduling problem with sequence-dependent setups. Setting the milling machine up actually means to load a specific program into memory that runs the numerically controlled milling machine, and to mount specific tools and holders. The sequence-dependent setup time consists of taking off tools and holders, loading another program, and mounting other tools and holders. The shortcoming of the planning software already in use is that average data for setup times are used as input for planning, and sequence dependencies are not considered. But, in practice, total setup times result from planning. As a consequence, the idea is near to formulate and to solve a model for this particular problem.

This case actually inspired the work on this paper and may thus serve for validating the work. Unfortunately, from our point of view, in 1996 Linotype-Hell AG started using SAP software and our contact persons were busy with this project since then.

3. A mixed-integer programming formulation

In this section we introduce the lot-sizing and scheduling-problem with sequence-dependent setup costs and times, denoted as LSPSD. Before we present a mathematical formulation of the LSPSD we have to give the underlying assumptions and have to introduce some definitions.

Assumption 1. The setup state is kept up over idle time.

We point out this simple fact because in the DLSP (cf. [3,10]) it is assumed that the setup state is lost after idle time. Only a few practical applications seem to require the loss of the setup state. This is emphasized by the fact that Assumption 1 is also made in a wide variety of different lot-sizing and scheduling models (cf. [5]).

We consider a large-bucket problem. This is to say that more than one item can be produced per period (e.g. per week).

Now, let $\mathrm{sc}_{i,j}$ ($\mathrm{st}_{i,j}$) denote the setup cost (setup time) for a setup from item i to item j. We compute setup costs as follows:

Assumption 2. Setup cost has the form $\mathrm{sc}_{i,j} = f_j + f_{\mathrm{sc}}\mathrm{st}_{i,j}$ where f_j are fixed cost and f_{sc} are opportunity cost per unit of setup time.

In [2] instances are considered where the triangle inequality for the setup costs is not fulfilled. For practical purposes this seems to be not a very important case. Hence, we exclude such solutions by the following assumption:

Assumption 3. Setup times satisfy the triangle inequality, i.e. $\mathrm{st}_{i,j} \leqslant \mathrm{st}_{i,k} + \mathrm{st}_{k,j}$ for all $i, j, k = 1, \dots, J$, where J is the number of different items to be considered.

Due to assumption 2 the triangle inequality is also valid for the setup costs.

Corollary 4. *For each item at most one lot will be produced in a period.*

Note, Corollary 4 is underlying the classical uncapacitated and capacitated lot-sizing problems (cf. e.g. [6]), too.

Furthermore, we assume the following:

Assumption 5. If the production of an item starts in a period then the inventory of the item must be empty at the end of the previous period.

Assumption 6. Each setup will be performed within a period.

Definition 7. The ordered set $\mathrm{seq}^{(n)} := (i_1, \dots, i_k, \dots, i_{M_n})$ denotes a sequence n of M_n different items within a period where i_1 (i_k, i_{M_n}) is said to be the first (kth, last) item of $\mathrm{seq}^{(n)}$.

$\mathrm{seq}^{(n)}$ is a sequence in which some items can be (efficiently) scheduled where it needs to be defined what efficient really means.

Definition 8. The setup cost (setup time) of the sequence $\mathrm{seq}^{(n)}$ is given by $\mathrm{SC}_n = \sum_{j=1}^{M_n-1} \mathrm{sc}_{i_j,i_{j+1}}$ ($\mathrm{ST}_n = \sum_{j=1}^{M_n-1} \mathrm{st}_{i_j,i_{j+1}}$).

Efficiency can now be defined as follows:

Definition 9. Consider two feasible sequences $\mathrm{seq}^{(n)}$ and $\mathrm{seq}^{(n')}$ with $(i_1, \dots, i_{M_n})$, and $(i'_1, \dots, i'_{M_{n'}})$, respectively. $\mathrm{seq}^{(n)}$ dominates $\mathrm{seq}^{(n')}$ if $\mathrm{SC}_n < \mathrm{SC}_{n'}$, $i_1 = i'_1, i_{M_n} = i_{M_{n'}}$, and $\mathrm{seq}^{(n')}$ consists exactly of the same items as $\mathrm{seq}^{(n)}$. $\mathrm{seq}^{(n)}$ is called efficient if there exists no other sequence $\mathrm{seq}^{(n'')}$ that dominates $\mathrm{seq}^{(n)}$.

Note, the computation of an efficient sequence is a traveling salesman problem (TSP) in which the salesman starts at ‘custom’ i_1 and stops at ‘custom’ i_{M_n}. Nowadays, optimal solution procedures for the TSP which work in reasonable time with hundreds of customers exist. In the following we need to consider only efficient sequences.

Remark. The set of all sequences may be constrained by additional restrictions to give the set of feasible sequences. This will turn out to be important for rescheduling.

Before we give a MIP-model formulation to define the problem at hand precisely, let us introduce some notation.

Parameters

J	the number of different items
N	the total number of efficient sequences
T	the number of periods
A_j	the set of indices which are associated to efficient sequences which contain item j, i.e. $A_j = \{n \in \{1,\dots,N\} \mid j \in \mathrm{seq}^{(n)}\}$
B	big number; e.g. $B \geqslant \max\{\sum_{t=1}^{T} d_{j,t} \mid j = 1,\dots,J\}$
C_t	the capacity available in period t
$d_{j,t}$	the demand for item j at the end of period t
F_j	the set of indices which are associated to efficient sequences in which item j is scheduled as the first item, i.e. $F_j = \{n \in \{1,\dots,N\} \mid \mathrm{seq}^{(n)} = (j,\dots,i_{M_n})\}$
h_j	holding cost which is incurred to hold one unit of item j at the end of a period in inventory
L_j	the set of indices which are associated to efficient sequences in which item j is scheduled as the last item, i.e. $L_j = \{n \in \{1,\dots,N\} \mid \mathrm{seq}^{(n)} = (i_1,\dots,j)\}$
p_j	capacity needs for producing one unit of item j
SC_n	setup cost which is incurred for scheduling sequence n
ST_n	setup time which is required to schedule sequence n

Decision variables

$I_{j,t}$	the inventory of item j at the end of period t ($I_{j,0} = 0$ without loss of generality)
$q_{j,t}$	the quantity of item j to be produced in period t
$S_{n,t}$	a binary variable indicating whether sequence n is used in period t ($S_{n,t} = 1$) or not ($S_{n,t} = 0$) (an initial setup state is also taken into account, i.e. $\exists_{n\in\{1,\dots,N\}}: S_{n,0} = 1 \wedge S_{n',0} = 0 \ \forall n' \neq n$).

The LSPSD can now be stated as follows:

Problem LSPSD

$$\text{Minimize} \quad \sum_{n=1}^{N}\sum_{t=1}^{T} \mathrm{SC}_n S_{n,t} + \sum_{j=1}^{J}\sum_{t=1}^{T} h_j I_{j,t} \tag{1}$$

subject to

$$I_{j,t-1} + q_{j,t} - I_{j,t} = d_{j,t}, \quad j = 1,\dots,J;\ t = 1,\dots,T, \tag{2}$$

$$\sum_{n=1}^{N} S_{n,t} = 1, \quad t = 1,\dots,T, \tag{3}$$

$$\sum_{n\in L_j} S_{n,t-1} - \sum_{n\in F_j} S_{n,t} = 0, \quad j = 1,\dots,J;\ t = 1,\dots,T, \tag{4}$$

$$q_{j,t} - B\sum_{n\in A_j} S_{n,t} \leqslant 0, \quad j = 1,\dots,J;\quad t = 1,\dots,T, \tag{5}$$

$$I_{j,t-1} - B\left(1 - \sum_{n\in A_j\setminus F_j} S_{n,t}\right) \leqslant 0, \quad j = 1,\dots,J;\quad t = 2,\dots,T, \tag{6}$$

$$\sum_{j=1}^{J} p_j q_{j,t} + \sum_{n=1}^{N} \mathrm{ST}_n S_{n,t} \leqslant C_t, \quad t = 1,\dots,T, \tag{7}$$

$$S_{n,t} \in \{0, 1\}, \quad j = 1,\dots,J,\ t = 1,\dots,T, \tag{8}$$

$$I_{j,t}, q_{j,t} \geqslant 0, \quad j = 1,\dots,J,\ t = 1,\dots,T. \tag{9}$$

The objective function (1) determines the total setup and holding costs. Constraints (2) are the inventory balances. Constraint (3) states that for each period we have to choose exactly one sequence in which items are scheduled. By constraint (4) we satisfy that the setup state is preserved between two adjacent periods. Constraint (5) ensures that an item can only be produced in a period if the machine is setup for it. Constraint (6) guarantees that a new lot is scheduled for an item only if the inventory is empty (zero-switch-property). Constraints (7) are the capacity constraints. These constraints also include that all setups are done within a period completely, i.e. no setup is performed over a period border. The last two constraints, (8) and (9), properly define the domains of the binary and continuous variables, respectively. The non-negative conditions of the inventory variables ensure that no shortages occur.

For example, let $S_{1,0} = 1$, $S_{n,0} = 0$ for $n > 1$, $T = 4$, $J = 3$, $(h_j) = (1, 1, 2)$, $(p_j) = (2, 1, 1)$, $(C_t) = (100, 100, 100, 100)$, and

$$(d_{j,t}) = \begin{pmatrix} 10 & 10 & 0 & 10 \\ 30 & 50 & 30 & 40 \\ 20 & 0 & 50 & 20 \end{pmatrix}.$$

Furthermore, let

$$(\mathrm{st}_{i,j}) = \begin{pmatrix} 0 & 5 & 10 \\ 5 & 0 & 15 \\ 10 & 15 & 0 \end{pmatrix}$$

and $\mathrm{sc}_{i,j} = 10\mathrm{st}_{i,j}$ for $i,j = 1,\dots, 3$. We derive the (efficient) sequences and the associated setup times and costs as given in Table 1. The item specific sets A_j, F_j, and L_j are provided in Table 2. Note, to keep the example small and clear we have chosen $J = 3$ and thus Table 1 contains all possible sequences. If we had $J = 4$, then for example either the sequence (1, 2, 3, 4) or (1, 3, 2, 4), but not both, would have to be considered as an efficient sequence. Note, if both sequences are efficient, we can choose one of them arbitrarily.

The optimal solution, computed with a standard solver, is $Z^* = 585$. The associated production quantities and non-zero binary decision variables are

$$(q_{j,t}) = \begin{pmatrix} 10 & 10 & 0 & 10 \\ 35 & 75 & 0 & 40 \\ 20 & 0 & 50 & 20 \end{pmatrix}$$

and $S_{5,1} = S_{7,2} = S_{3,3} = S_{14,4} = 1$, respectively. In the solution item $j = 2$ is scheduled as the last item in period $t = 1$ and as first item in period $t = 2$ which allows five units of the demand $d_{2,2}$ to be produced in period $t = 1$. This is necessary, since the capacity in period $t = 2$ is completely used up for setups and production ($\mathrm{ST}_7 + \sum_{j=1}^{3} p_j q_{j,2} = 5 + 2\times 10 + 1\times 75 = 100 = C_2$).

Table 1
Sequences and associated setup costs and times

n	$\mathrm{seq}^{(n)}$	ST_n	SC_n
1	(1)	0	0
2	(1, 2)	5	50
3	(1, 3)	10	100
4	(1, 2, 3)	20	200
5	(1, 3, 2)	25	250
6	(2)	0	0
7	(2, 1)	5	50
8	(2, 3)	15	150
9	(2, 1, 3)	15	150
10	(2, 3, 1)	25	250
11	(3)	0	0
12	(3, 1)	10	100
13	(3, 2)	15	150
14	(3, 1, 2)	15	150
15	(3, 2, 1)	20	200

4. Rescheduling

Now, we will show how rescheduling (cf. [11]) can easily be integrated into the LSPSD. Let us assume that new information – due to customer requests – result in changes of the demand matrix only. That is, some entries in the demand matrix are increased and others are decreased. Furthermore, we will allow an extension of the planning horizon.

Requirements for a new schedule can be expressed by restricting the set of valid sequences in a period and reducing the available capacity in a period. The capacity reduction is due to lot sizes which are already scheduled in a period and that should not decrease.

Table 2
Item specific sets of sequence indices

j	A_j	F_j	L_j
1	$\{1,\dots, 5, 7, 9, 10, 12, 14, 15\}$	$\{1,\dots, 5\}$	$\{1, 7, 10, 12, 15\}$
2	$\{2, 4,\dots, 10, 13, 14, 15\}$	$\{6,\dots, 10\}$	$\{2, 5, 6, 13, 14\}$
3	$\{3, 4, 5, 8,\dots, 15\}$	$\{11,\dots, 15\}$	$\{3, 4, 8, 9, 11\}$

Therefore, we define

SEQ_t	the set of sequences which are allowed to be scheduled in period t, and
$q^{\mathrm{f}}_{j,t}$	the minimum production quantity of item j which has to be scheduled in period t. Note, $q^{\mathrm{f}}_{jt} > 0$ implies $\mathrm{SEQ}_t \subseteq A_j$.

Now, if we replace (3) by

$$\sum_{n \in \mathrm{SEQ}_t} S_{n,t} = 1, \quad t = 1, \dots, T, \tag{10}$$

and update the capacities using $C'_t = C_t - \sum_{j=1}^{J} p_j q^{\mathrm{f}}_{j,t}$ for $t = 1, \dots, T$ the LSPSD is extended for rescheduling.

In summary we find out that rescheduling equals the scheduling process but reduces the solution space when compared with the original instance. Rescheduling therefore needs less computational effort than finding a first schedule (unless we extend the planning horizon).

5. A fast enumeration scheme

To find an optimal solution for a particular LSPSD instance we must first note that once we have fixed all the binary variables $S_{n,t}$ in the MIP-formulation above, the remaining subproblem is an LP-problem. In other words, enumerating all the T-tuples $(\mathrm{seq}_1, \dots, \mathrm{seq}_T)$ where seq_t denotes the sequence chosen in period t, and solving the corresponding LP-problem then, will reveal the optimal solution. Unfortunately, the number of T-tuples is quite large, $((J-1)2^{(J-2)} + 1)^T$ to be precise.

Hence, we need a more sophisticated approach to tackle this problem. In its essence, the procedure that we propose is a branch-and-bound (B&B) method. Roughly speaking, we start in period T, perform a branching step by choosing seq_T, and then move on to period one step by step doing backtracking in-between if necessary.

To provide more details, we use the following notation: If seq_t is a sequence of the form $(i_1, \dots, i_{k_t})$ where $k_t \geq 1$, then, first $(\mathrm{seq}_t) = i_1$ and $\mathrm{last}(\mathrm{seq}_t) = i_{k_t}$ are the first and the last item, respectively, in seq_t. $\mathrm{ST}(\mathrm{seq}_t)$ equals ST_n if seq_t is sequence n and thus $k_t = M_n$. Analogously, we define $\mathrm{SC}(\mathrm{seq}_t)$. Let $\mathrm{CD}_{j,t} = \sum_{\tau=t}^{T} d_{j,\tau} - \sum_{\tau=t+1}^{T} q_{j,\tau}$ be the cumulative remaining demand for item j in period t. Note, initially we have $q_{j,t} = 0$ for all items j in all periods t. Furthermore, let SEQ_t be the set of all efficient sequences to be considered in period t.

Now, we are ready to describe the solution scheme (see Table 3) in more detail.

Some things need to be discussed. First, in *Step 0*, SEQ_t can of course be chosen as the set of all sequences. But, we can eliminate all those sequences which contain items with a zero cumulative demand. In other words, items for which no demand occurs, need not be produced. The only exception from that is that the last item in a sequence must equal the first item in seq_{t+1} sequence, i.e. $\mathrm{last}(\mathrm{seq}_t) = \mathrm{first}(\mathrm{seq}_{t+1})$ for $t = 1, \dots, T-1$. In addition, SEQ_t must contain only those sequences which do not violate capacity constraints. That is

Table 3
Outline of the enumeration in period t

```
Step 0: Compute $\mathrm{SEQ}_t$.
Step 1: Set $q_{j,t} = 0$ $(j = 1, \dots, J)$.
while ($\mathrm{SEQ}_t$ $\emptyset$)
{
    Step 2: Choose $\mathrm{seq}_t = (i_1, \dots, i_{k_t}) \in \mathrm{SEQ}_t$.
    Step 3: $\mathrm{SEQ}_t := \mathrm{SEQ}_t - \{\mathrm{seq}_t\}$.
    Step 4: Set $\mathrm{cost}_t = \mathrm{SC}(\mathrm{seq}_t) + \sum_{j=1}^{J} (\mathrm{CD}_{j,t} - d_{j,t}) h_j + \mathrm{cost}_{t+1}$.
    Step 5: Compute $q_{j,t}$ $(j = 1, \dots, J)$.
    Step 6: If "not bound" go to period $t - 1$ ...
}
backtracking to period $t + 1$.
```

to say, that all sequences seq_t contained in SEQ_t must fulfill

$$C_t - \text{ST}(\text{seq}_t) - \sum_{j=2}^{k_t} (p_{i_j} \text{CD}_{i_j,t}) \geqslant 0.$$

Remember, that lots (except the one for the first item in a sequence) must not range over period borders. For rescheduling, the choice of SEQ_t must be a subset of the valid sequences in period t.

In *Step 2* seq_t can be chosen arbitrarily. In our implementation we choose long sequences before we choose short ones. Ties are broken in lexicographical order.

Step 5 directly corresponds to solving a linear program. Due to our assumption that lots (i.e. $\text{CD}_{j,t}$) must not be split, this turns out to be very easy: For all items $j \in \{i_2, \ldots, i_{k_t}\}$ we set $q_{j,t} = \text{CD}_{j,t}$. For all items $j \notin \{i_1, \ldots, i_{k_t}\}$ we set $q_{j,t} = 0$. Finally, for $j = i_1$ we set

$$q_{i_1,t} = \min\left\{\text{CD}_{i_1,t}, \frac{C_t - \text{ST}(\text{seq}_t) - \sum_{j=2}^{k_t}(p_{i_j}\text{CD}_{i_j,t})}{p_{i_1}}\right\}.$$

If period 1 is under concern, we face a feasible solution if and only if the total demand for periods 1 to T is equal to the total production of periods 1 to T for all items $j = 1, \ldots, J$. And, cost_1 is the objective function value for the feasible solution at hand.

Finally, in *Step 6* we test if we can prune the search tree. Two tests are done here. First, we check if the remaining capacity exceeds the capacity demand, i.e.

$$\sum_{\tau=1}^{t-1} C_\tau \geqslant \sum_{j=1}^{J} \left(\text{CD}_{j,t-1} + \sum_{\tau=1}^{t-2} d_{j,\tau}\right) p_j + \sum_{j \in P_t} \min_{i \in (P_t \cup \{\text{last}(\text{seq}_0)\}) \setminus \{j\}} \{\text{st}_{i,j}\}$$

with

$$P_t = \left(\left\{\{1, \ldots, J\} \mid \text{CD}_{j,t-1} + \sum_{\tau=1}^{t-2} d_{j,\tau} > 0\right\} \cup \{\text{first}(\text{seq}_t)\}\right) \Big\backslash \{\text{last}(\text{seq}_0)\}$$

must hold. P_t is the set of items which must be produced in periods $1, \ldots, t-1$ and for which a setup must occur. Note, only a lower bound of the setup time is considered here. Second, we test if the current situation is bounded by costs. Assume, that we have an upper bound of the overall problem, say *upperbound*. Furthermore assume, that we have a lower bound of costs, say $\mathit{lowerbound}_{1,t-1}$, that will additionally occur if we schedule sequences in the periods $t-1, \ldots, 1$. Then, we simply check $\text{cost}_t + \text{lowerbound}_{1,t-1} \geqslant \text{upperbound}$ to prune the tree.

The efficiency of our procedure highly depends on these lower bounds.

Let us discuss the lower bounds only, since the upper bounds are computed using standard techniques (i.e. starting with infinity we update the upper bound whenever a feasible solution is found that improves the current best bound). Before we start trying to solve an instance with J items and T periods, we cut the horizon at the end and solve the resulting instance with J items and the $T-1$ periods $t = 1, \ldots, T-1$. Following the same lines, before we solve the $T-1$ periods instance we solve the $T-2$ periods instance and so on. Note, that if a smaller instance is not feasible, the larger cannot be either. In summary we start with an instance of 1 period only which provides the lower bound $\text{lowerbound}_{1,1}$. Then we solve an instance with 2 periods which gives $\text{lowerbound}_{1,2}$ and so on until we are done. The trick here is that we can use the lower bounds computed by solving small instances when we solve the large instances. As computational studies have shown, the speed-up is dramatic. E.g. running instances with three items and 10 periods of time took more than an hour without these bounds and now terminates after a few seconds.

This bounding scheme is very efficient in terms of both, run-time and memory space. While the former one will be verified in the next section, the latter one should be evident.

6. Computational study

To test the proposed method we ran a C-implementation on a Power PC computer with 80 MHz measuring run-time performances. A total of 540 instances were systematically generated as follows: For all items we choose $p_j = 1$. The machine is assumed to be set up for item 1 initially. The number of items J ranges from 2 to 10 items and the number of periods T ranges from 3 to 10, 15, and 20 periods. We then randomly generated an external demand matrix with 10 items (rows) and 20 periods (columns) where each entry $d_{j,t}$ is chosen out of the interval [40,60] with uniform distribution. Hence, this matrix contains no zero values which possibly would reduce the number of sequences to be considered per period. Analogously, a setup time matrix with 10 items is generated where each entry $\mathrm{st}_{i,j}$ $(i \neq j)$ is randomly chosen out of the interval [2,10] (and $\mathrm{st}_{j,j} = 0$). The choice of setup times is done so that all triangle inequalities are fulfilled. Holding costs for 10 items are randomly chosen, too, where each value h_j is drawn out of the interval [2,10] with uniform distribution. For an instance with J items and T periods we then use the data given in the first J rows and the first T columns of the external demand matrix, the first J rows and columns of the setup time matrix, and the first J entries of the holding cost vector. This implements the concept of common random numbers in our tests. The setup cost $\mathrm{sc}_{i,j}$ for changing the setup state from item i to item j are computed by

$$\mathrm{sc}_{i,j} = f_{\mathrm{sc}} \mathrm{st}_{i,j}, \quad i,j = 1, \ldots, J,$$

where the parameter f_{sc} is systematically varied using $f_{\mathrm{sc}} = 50$ and 500. The capacity per period C_t is determined according to

$$C_t = \frac{\sum_{j=1}^{J} d_{j,t}}{U}, \quad t = 1, \ldots, T,$$

where the capacity utilization U is systematically varied using $U = 0.4$, 0.6, and 0.8. Note, the utilization of capacity is an estimate only, because setup times do not affect the computation of C_t. Hence, a value $U = 0.8$ actually means that the utilization of capacity by production and setup actions is greater than 80% on average. Note, the more items there are, the more capacity is consumed by setup time. In summary, we have

$$|\{2, \ldots, 10\}| \times |\{3, \ldots, 10, 15, 20\}| \times |\{50, 500\}|$$
$$|\{0.4, 0.6, 0.8\}| = 540 \text{ instances.}$$

Tables 4–9 provide the run-time results of our study. All results are given in CPU–seconds. A time limit of 3600 CPU-seconds is used. Missing entries thus indicate that the corresponding instance cannot be solved optimally within one hour on our platform. Zeroes indicate that the method needs less than 0.5 CPU-seconds to compute the optimum solution. The run-times given here do not include the time needed to compute the efficient sequences. This is because in a real-world situation the number of items J does not change in the short-term and thus solving the set of traveling salesman problems needs to be done once and for all. The effort for doing so can thus be neglected.

As expected, it turns out that the parameters J and T do have a significant impact on the run-time performance. In almost all cases, the run-time grows faster with J than with T. For instance, see Table 4 where the instance with $J = 7$ and $T = 4$ terminated after 61 CPU-seconds. For $J = 8$ and

Table 4
Run-time performance for $f_{\mathrm{sc}} = 50$ and $U = 0.4$

	$T = 3$	4	5	6	7	8	9	10	15	20
$J = 2$	0	0	0	0	0	0	0	0	0	1
$J = 3$	0	0	0	0	0	0	0	0	5	162
$J = 4$	0	0	0	1	2	5	10	39	1966	
$J = 5$	0	1	3	12	20	96	173	2052		
$J = 6$	2	9	39	83	209	1375	2658			
$J = 7$	13	61	181	511	1295					
$J = 8$	144	1729								
$J = 9$	537									
$J = 10$	2428									

Table 5
Run-time performance for $f_{sc} = 50$ and $U = 0.6$

	$T = 3$	4	5	6	7	8	9	10	15	20
$J = 2$	0	0	0	0	0	0	0	0	0	1
$J = 3$	0	0	0	0	0	0	0	0	5	136
$J = 4$	0	0	0	1	2	5	9	37		
$J = 5$	0	1	2	11	19	93	167	2001		
$J = 6$	2	9	38	82	204	1352	2633			
$J = 7$	13	60	179	505	1282					
$J = 8$	194	2570								
$J = 9$	613									
$J = 10$										

Table 6
Run-time performance for $f_{sc} = 50$ and $U = 0.8$

	$T = 3$	4	5	6	7	8	9	10	15	20
$J = 2$	0	0	0	0	0	0	0	0	0	1
$J = 3$	0	0	0	0	0	0	0	1	15	530
$J = 4$	0	0	1	2	3	9	17	46		
$J = 5$	0	2	9	33	73	214	325	2965		
$J = 6$	2	13	56	208	768	2856				
$J = 7$	7	24	228	552	1203					
$J = 8$	100	1303								
$J = 9$	471									
$J = 10$										

Table 7
Run-time performance for $f_{sc} = 500$ and $U = 0.4$

	$T = 3$	4	5	6	7	8	9	10	15	20
$J = 2$	0	0	0	0	0	0	0	0	1	8
$J = 3$	0	0	0	0	1	2	4	7	164	
$J = 4$	0	0	1	6	20	78	331	732		
$J = 5$	0	3	20	125	1560					
$J = 6$	1	11	176	1350						
$J = 7$	4	153	1581							
$J = 8$	39	3501								
$J = 9$	201									
$J = 10$	2489									

$T = 4$ we measure 1729 CPU-seconds, and for $J = 7$ and $T = 5$ we need 181 CPU-seconds.

Varying the setup costs (measured by the parameter f_{sc}) and the capacity utilization U does not drastically affect the order of magnitude of problem sizes that can be solved within reasonable time. It cannot be stated that higher capacity usage gives shorter computation times. Comparing Table 4 with Table 6 indicates that larger instances can be solved when capacity usage is low. But, this result cannot be validated when comparing Table 7 with Table 9. Also, it is not true that higher setup costs

Table 8
Run-time performance for $f_{sc} = 500$ and $U = 0.6$

	$T = 3$	4	5	6	7	8	9	10	15	20
$J = 2$	0	0	0	0	0	0	0	0	0	5
$J = 3$	0	0	0	0	0	1	2	3	175	
$J = 4$	0	0	2	6	18	50	118	314		
$J = 5$	0	2	15	81	554	2138				
$J = 6$	1	18	216	862						
$J = 7$	5	134	1580							
$J = 8$	49									
$J = 9$	247									
$J = 10$	2300									

Table 9
Run-time performance for $f_{sc} = 500$ and $U = 0.8$

	$T = 3$	4	5	6	7	8	9	10	15	20
$J = 2$	0	0	0	0	0	0	0	0	0	2
$J = 3$	0	0	0	0	0	0	0	1	14	378
$J = 4$	0	0	0	1	2	10	24	64		
$J = 5$	0	1	3	9	27	127	316	2551		
$J = 6$	0	4	13	56	143	1155	3069			
$J = 7$	2	8	50	189	721					
$J = 8$	47	489	2071							
$J = 9$	57	765								
$J = 10$	826									

make instances easier to solve. Compare for instance Table 6 with Table 9 where this seems to be the case, whereas a comparison of Table 4 with Table 7 does not give such a proof.

Since we used instances with fully filled demand matrices the results can be seen as worst case estimates on the run-time performance. Facing instances with sparse demand matrices would give shorter run-times, because the number of sequences to be considered within a period decreases. This is due to the fact that items with no cumulative demand need not be scheduled and thus sequences containing such items need not be enumerated. A similar argument applies to the effort for rescheduling. Since rescheduling means to impose some restrictions on the sequences that are allowed to be scheduled, its run-time will be less than what can be read in the tables.

A benchmark test with the standard solver LINDO gives convincing results. Within 3600 CPU-seconds, LINDO is able to solve the instances with four items and six periods. In contrast to that, our procedure needs less than 6 seconds to give the optimum result.

7. Conclusions

In this paper we proposed a model for lot sizing and scheduling with sequence-dependent setups which was inspired from a practical case at Linotype-Hell AG. The key element for the efficiency of the method is based on an idea derived from problem specific insights. Roughly speaking, this idea is that if we know what items to produce in a period but we do not know the lot sizes yet, we can nevertheless determine the sequence in which these items are to be scheduled.

In contrast to other approaches which suffer from large memory requests, the presented procedure requires modest capacities. This is mainly due to a novel idea for computing lower bounds to prune the search tree. Memorizing partial schedules seems to be avoidable now. Beside the low memory space usage, the lower bounding technique amazes with high speed-ups.

The size of the instances that are solved is of practical relevance as it is proven by case studies in [2] (food industry) and [9] (discrete part manufacturing) where instances with less than 10 items occur.

Acknowledgements

We are grateful to Andreas Drexl for his steady support. Also, we thank Steffen Wernert.

References

[1] L. Schrage, The multiproduct lot scheduling problem, in: M.A.H. Dempster et al. (Eds.), Deterministic and Stocastic Scheduling, Dordrecht/Holland, 1982, pp. 233–244.

[2] B. Fleischmann, The discrete lot-sizing and scheduling problem with sequence-dependent setup-costs, European Journal of Operational Research 75 (1994) 395–404.

[3] M. Salomon, M.M. Solomon, L.N. Van Wassenhove, Y.D. Dumas, S. Dauzere-Peres, Solving the discrete lotsizing and scheduling with sequence dependent set-up costs and set-up times using the Travelling Salesman Problem with time windows, European Journal of Operational Research 100 (1997) 494–513.

[4] C. Jordan, A. Drexl, Lotsizing and scheduling by batch sequencing, Management Science 44 (1998), 698–713.

[5] A. Drexl, A. Kimms, Lot sizing and scheduling — survey and extensions, European Journal of Operational Research 99 (1997) 221–235.

[6] D.M. Dilts, K.D. Ramsing, Joint lot sizing and scheduling of multiple items with sequence dependent setup costs, Decision Sciences 20 (1989) 120–133.

[7] G. Dobson, The cyclic lot scheduling problem with sequence-dependent setups, Operations Research 40 (1992) 736–749.

[8] K. Haase, Capacitated lot-sizing with sequence dependent setup costs, OR Spektrum 18 (1996) 51–59.

[9] K. Haase, L. Göpfert, Engpassorientierte Fertigungssteuerung bei reihenfolgeabhängigen Rüstvorgängen in einem Unternehmen der Satz- und Drucktechnik, Zeitschrift für Betriebswirtschaft 66 (1996) 1511–1526.

[10] D. Cattrysse, M. Salomon, R. Kuik, L.N. Van Wassenhove, A dual ascent and column generation heuristic for the discrete lotsizing and scheduling problem with setup-times, Management Science 39 (1993) 477–486.

[11] A. Kimms, Stability measures for rolling schedules with applications to capacity expansion planning, master production scheduling, and lot sizing, Omega 26 (1998) 355–366.