Haase, Knut; Kimms, Alf

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

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 393

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 393

Lot Sizing and Scheduling with Sequence Dependent Setup Costs and Times and Efficient Rescheduling Opportunities

Knut Haase and Alf Kimms[1]

No. 393

Lot Sizing and Scheduling with Sequence Dependent Setup Costs and Times and Efficient Rescheduling Opportunities

Knut Haase and Alf Kimms[1]

February, 1996

Knut Haase, Alf Kimms, Lehrstuhl für Produktion und Logistik, Institut für Betriebswirtschaftslehre, Christian–Albrechts–Universität zu Kiel, Olshausenstraße 40, 24118 Kiel, Germany

[1] The work of this author was done with partial support from the DFG-project Dr 170/4-1.

Summary: This paper deals with lot sizing and scheduling for a single-stage 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. Furthermore, it will become clear that rescheduling can neatly be done.

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 hold 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 in a period, i.e. we have to consider sequencing and lot sizing simultaneously.

Despite of its relevance there has been done only little research 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 (cf. [18]), 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 produced per period. In [22] a DLSP-like model with sequence dependent setup costs was considered first. For the DLSP an exact branch-and-bound approach based on Lagrangean relaxation of the capacity constraints has been presented in [9] which is extended to the DLSP with sequence dependent setup costs (DLSPSD) in [10]. 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 [21]. The optimal enumeration method proposed by [13] is based on the so-called batch sequencing problem (BSP). It can be shown that the BSP is equivalent to the DSLPSDCT 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. Another new type of model has been published in [6], [7], and [12] which is called the proportional lot sizing and scheduling problem (PLSP). 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 [4] 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 [5]. In [11] 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 (cf. [8]). 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 give a mathematical formulation of the problem under concern. Afterwards, rescheduling is discussed in Section 3. In Section 4 an optimal enumeration method is outlined. The efficiency of the algorithm is tested by a computational study in Section 5.

2 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.

Especially for manufacturing companies this assumption is very realistic. For example, the preparation of a drilling machine for the production of a specific item may require the setup of a specific drill. If nobody has unmounted the drill a new lot of the same item can be produced without performing a new setup since the machine is still suitably prepared. We point out this simple fact because in the DLSP (cf. [2], [9], [21]) 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 the assumption 1 is also made in a wide variety of different lot sizing and scheduling models (cf. [1], [11], [12], [14], [15], [16], [17], [19], [20], [25]).

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

Assumption 2. For each item at most one lot will be produced in a period.

Note, assumption 2 is underlying the classical uncapacitated and capacitated lot sizing problems (cf. e.g. [4], [24], and [26], respectively), too.

Now, let $sc_{i,j}$ ($st_{i,j}$) denote the setup cost (setup time) for a setup from item i to item j. Since setup costs are in large parts opportunity costs for the time that is needed to perform a setup, we compute setup costs as follows:

Assumption 3. Setup cost has the form $sc_{i,j} = f(st_{i,j})$ where $f(\cdot)$ is a non-decreasing function.

Thus the larger (smaller) the setup time the larger (smaller) is the associated setup cost. Moreover, if we minimize setup costs we also minimize setup times.

In [10] 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 4. Setup times satisfy the triangle inequality, i.e. $st_{i,j} \leq st_{i,k} + st_{k,j}$ for all $i, j, k = 1, \ldots, J$.

where J is the number of different items to be considered. Due to assumption 3 the triangle inequality is also valid for the setup costs.

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.

Due to this assumption the lot size of an item in a period is equal to the quantity demanded in that period or it includes one or more future demands. Furthermore, if a lot size of an item in period t includes the demand of a period $s > t$ then all demands of the periods τ, $t \leq \tau \leq s$, are also included. Necessary to say, that the production of a lot can start in a period t and be finished in a period s, $s \geq t$. This, of course, eases the computation of lot sizes. But, when having a closer look at former work we observe that the DLSP, for instance, also implies easy-to-compute lot sizes (all-or-nothing). The BSP, for instance, assumes lot sizes being the sum of future demands.

As in [3] we state now the following assumption.

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

Thus a setup never starts in a period t and will be finished in period $s > t$.

In the following we have to define some notation.

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

$seq^{(n)}$ 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 $seq^{(n)}$ is given by $SC_n = \sum_{j=1}^{M_n-1} sc_{i_j,i_{j+1}}$ ($ST_n = \sum_{j=1}^{M_n-1} st_{i_j,i_{j+1}}$).

Efficiency can now be defined as follows:

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

Note, the computation of an efficient sequence is a traveling salesman problem in which the salesman starts at 'custom' i_1 and stops at 'custom' i_{M_n}. 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

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

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 defines the schedule in period t ($S_{n,t} = 1$) or not ($S_{n,t} = 0$) (an initial setup state is also taken into account, i.e. $\exists_{n \in \{1,...N\}} : S_{n,0} = 1 \wedge S_{n',0} = 0 \; \forall n' \neq n$).

The LSPSD can now be stated as follows:

Problem LSPSD

$$Minimize \sum_{n=1}^{N} \sum_{t=1}^{T} SC_n S_{n,t} + \sum_{j=1}^{J} \sum_{t=1}^{T} h_j I_{j,t} \tag{1}$$

subject to

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

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

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

$$q_{j,t} - B \sum_{n \in A_j} S_{n,t} \leq 0 \qquad j = 1,...,J; t = 1,...,T \tag{5}$$

$$I_{j,t-1} - B(1 - \sum_{n \in A_j \setminus F_j} S_{n,t}) \leq 0 \qquad j = 1,...,J; t = 2,...,T \tag{6}$$

$$\sum_{j=1}^{J} p_j q_{j,t} + \sum_{n=1}^{N} ST_n S_{n,t} \leq C_t \qquad t = 1,...,T \tag{7}$$

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

$$I_{j,t}, q_{j,t} \geq 0 \qquad j = 1,...,J; t = 1,...,T \tag{9}$$

The objective function (1) determines the total setup and holding costs. (2) are the inventory balances. (3) states that for each period we have to choose exactly one sequence in which items are scheduled. By (4) we satisfy that the setup state is preserved between two adjacent periods. (5) ensures that an item can only be produced in a period if the machine is setup for it. (6) guarantees that only a new lot is scheduled for an item if the inventory is empty (zero-switch-property). (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 do occur. Noteworthy to say that in contrast to other optimal approaches dealing with similar problems (e.g. [13]) we have made here rather general assumptions. For instance, an initial setup state is taken into account, there are no restrictive assumptions for the cost parameters such as $h_j = 1$ for all $j = 1,\ldots,J$, and the capacity may vary over time.

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

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

and $sc_{i,j} = 10st_{i,j}$ for $i, j = 1, ..., 3$. 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 the Table 1 contains all possible sequences. Suppose we would have $J = 4$, 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 by arbitration.

Table 1. Sequences and associated setup costs and times

n	$seq^{(n)}$	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

The optimal solution, computed with the standard solver LINDO [23], is $Z^* = 585$ costly. The associated production quantities and non-zero binary decision variables are

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

Table 2. Item specific sets of sequence indices

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

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 ($ST_7 + \sum_{j=1}^{3} p_j q_{j,2} = 5 + 2 \cdot 10 + 1 \cdot 75 = 100 = C_2$).

3 Rescheduling

If we have a look at a shop floor we see that machines are working, materials are moved from one place to the other, products are packed for shipment, and so on. Apparently, production schedules do affect all these operations. The impact on the other proceedings when changing the schedule usually is substantial and thus expensive due to transaction costs. It is not an easy task to handle all the interaction effects which would have to be considered. However, lost and/or additional orders may enforce modifications of the existing schedules, and trigger the process of rescheduling. To keep the production wheels humming and to avoid costs, these modifications should be kept in certain limits.

Now, we will show how rescheduling can easily be integrated into the LSPSD. Let us assume that new informations - due to customer requests - result in changes of the demand matrix only. That is, some entrances 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.

Therefore we define

SEQ_t the set of sequences which are allowed to be scheduled in period t, and

q_{jt}^f the minimum production quantity of item j which has to be scheduled in period t.

Now, if we replace (3) by

$$\sum_{n \in SEQ_t} S_{n,t} = 1 \qquad t = 1, ..., T \tag{10}$$

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

To give more insight we come back to the above example and assume now that $q^f_{jt} = q_{jt}$ for $j = 1, \ldots, 3$ and $t = 1, \ldots, 3$, i.e. we will not allow reduction of the previously optimal production quantities in the periods $t = 1, \ldots, 3$. Let, $q^f_{j4} = 0$ for $j = 1, \ldots, J$.

Furthermore, let T=5 now. This leads to the updated capacities $(C_t) =$ (25 5 50 100 100). Let new customer requests be given by the demand matrix

$$(d'_{j,t}) = \begin{pmatrix} 0 & 0 & 0 & 5 & 15 \\ 0 & 0 & 20 & 40 & 50 \\ 0 & 0 & 0 & 20 & 10 \end{pmatrix}$$

in which the already scheduled demands are excluded. Furthermore, suppose the planner wants that - concerning the optimal sequences derived in the above example - in period $t = 1$ the sequence is fixed, the sequences in period $t = 2$ and period $t = 3$ must include the same items as before whereas for period $t = 4$ and period $t = 5$ no restrictions are given. These requirements are taken into account by defining:

$$\begin{aligned} SEQ_1 &= \{seq^{(5)}\} \\ SEQ_2 &= \{seq^{(n)} | n = 7, 9, 10\} \\ SEQ_3 &= \{seq^{(n)} | n \in A_3\} \\ SEQ_4 &= \{seq^{(n)} | n = 1, \ldots, 15\} \\ SEQ_5 &= SEQ_4 \end{aligned}$$

Note, the sequences 2, 4, 5, 14, and 15 are not contained in SEQ_2, because the first item in these sequences does not equal the last item in sequence 5 which is the only one that can be scheduled in period 1.

Based on the new data we compute the following optimal production schedule

$$(q'_{j,t}) = \begin{pmatrix} 0 & 0 & 0 & 5 & 15 \\ 0 & 0 & 20 & 40 & 50 \\ 0 & 0 & 0 & 30 & 0 \end{pmatrix}$$

and $S'_{5,1} = S'_{7,2} = S'_{4,3} = S'_{14,4} = S'_{7,5} = 1$. We see that in period $t = 3$ the sequence is extended by item $j = 2$ whereas the sequences in period $t = 1$ and $t = 2$ are not changed. Thus, the schedule is only slightly modified as it would be a desired property in practice. This can also be seen by the following final production schedule

$$(q''_{j,t}) = (q^f_{j,t}) + (q'_{j,t}) = \begin{pmatrix} 10 & 10 & 0 & 5 & 15 \\ 35 & 75 & 20 & 40 & 50 \\ 20 & 0 & 50 & 30 & 0 \end{pmatrix}.$$

Thus, we have shown how easily the rescheduling aspect can be integrated in our MIP-formulation. 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).

4 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 $(seq_1, \ldots, seq_T)$ 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 $(i_1, \ldots, i_{k_t})$ where $k_t \geq 1$. Then, $first(seq_t) = i_1$ and $last(seq_t) = i_{k_t}$ are the first and the last item, respectively, in seq_t. $ST(seq_t)$ equals ST_n if seq_t is sequence n and thus $k_t = M_n$. Analogously, we define $SC(seq_t)$. Let $CD_{j,t} = \sum_{\tau=t}^{T}(d_{j,\tau} - q_{j,\tau})$ be the cumulative demand for item j in period 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.

Table 3. Outline of the enumeration in period t

Step 0: Compute SEQ_t.
while $(SEQ_t \neq \emptyset)$
{
Step 1: Choose $seq_t = (i_1, \ldots, i_{k_t}) \in SEQ_t$.
Step 2: $SEQ_t := SEQ_t - \{seq_t\}$.
Step 3: Set $cost_t = SC(seq_t) + \sum_{j=1}^{J}(CD_{j,t} - d_{j,t})h_j + cost_{t+1}$.
Step 4: Compute $q_{j,t}$ $(j = 1, \ldots, J)$.
Step 5: If "not bound" go to period $t-1$ ...
}
backtracking to period $t+1$.

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. $last(seq_t) = first(seq_{t+1})$ for $t = 1, \ldots, T-1$. In addition, SEQ_t must contain only those sequences which do not violate capacity constraints. That is to say, that all sequences seq_t contained in SEQ_t must fulfill

$$C_t - ST(seq_t) - \sum_{j=2}^{k_t} (p_{i_j} CD_{i_j,t}) \geq 0.$$

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 1* 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 4 directly corresponds to solving a linear program. Due to our assumption that lots (i.e. $CD_{j,t}$) must not be splitted, this turns out to be very easy: For all items $j \in \{i_2, \ldots, i_{k_t}\}$ we set $q_{j,t} = CD_{j,t}$. For all items $j \notin \{i_1, \ldots, i_{k_t}\}$ we set $q_{j,t} = 0$. Finally, for $j = i_1$ we set

$$q_{i_1,t} = min\{CD_{i_1,t}, \frac{C_t - ST(seq_t) - \sum_{j=2}^{k_t} (p_{i_j} CD_{i_j,t})}{p_{i_1}}\}.$$

If period 1 is under concern, we face a feasible solution if and only if $CD_{j,1}$ evaluates to zero now for all items $j = 1, \ldots, J$. And, $cost_1$ is the objective function value for the feasible solution at hand.

Finally in *Step 5* we test if the current state is bound to prune the search tree. Two tests are done here. First, we check if the remaining capacity suffices the production, i.e.

$$\sum_{\tau=1}^{t-1} C_\tau \geq \sum_{j=1}^{J} (CD_{j,t-1} + \sum_{\tau=1}^{t-2} d_{j,t}) p_j$$

must hold. Note, we disregard setup times for this capacity check. 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 $lowerbound_{1,t-1}$, that will additionally occur if we schedule sequences in the periods $t-1, \ldots, 1$. Then, we simply check $cost_t + lowerbound_{1,t-1} \geq upperbound$ 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 $lowerbound_{1,1}$. Then we solve an instance with 2 periods which gives $lowerbound_{1,2}$ 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.

5 Computational Study

To test the proposed method we run 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 $st_{i,j}$ $(i \neq j)$ is randomly chosen out of the interval $[1, 5]$ (and $st_{j,j} = 0$). 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 $[1, 5]$ 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 $sc_{i,j}$ for changing the setup state from item i to item j are computed by

$$sc_{i,j} = f_{sc} st_{i,j} \qquad i, j = 1, \ldots, J$$

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

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

where the capacity utilization U is systematically varied using $U = 0.4$, $U = 0.6$, and $U = 0.8$. Note, the utilization of capacity is an estimate only, because setup times do 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. In summary, we have

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

instances.

Tables 4 to 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.

Table 4. Run-Time Performance for $f_{sc} = 50$ 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									

As expected, it turns out that the parameters J and T do have a significant impact on the run-time performance. 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 $T = 4$ 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 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.

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									

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 [23] 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 six seconds to give the optimum result.

6 Conclusion and Future Work

In this paper we proposed a model for lot sizing and scheduling with sequence dependent setups. It is a large time bucket model and thus fits to real-world situations where we find periods such as shifts, days or weeks naturally be given. Within these periods scheduling can be done on the basis of a continuous time axis. In addition, an optimal solution procedure is presented which follows the idea of branch-and-bound.

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.

In summary, this method competes with approaches for similar but more restrictive cases [13] where instances with up to $J = 10$ are solved. In our terminology, these instances have up to 38 entries in the external demand matrix. By the way, the procedure in [13] outperforms the code given in [21].

Future work should deal with solving model extensions. Multiple scarce capacities would for instance be an important subject for which the method should be modified. Taking multi-level product structures into account would also be an extension worth to be considered.

Beside this, the method should be refined. Additional branching and bounding rules should improve the results.

Acknowledgement

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

References

1. G. R. Bitran and H. Matsuo. The multi-item capacitated lot size problem: Error bounds on Manne's formulation. *Management Science*, 32:350–359, 1986.
2. D. Cattrysse, M. Salomon, R. Kuik and L. N. Van Wassenhove. A dual ascent and column generation heuristic for the discrete lotsizing and scheduling problem with setup-times. *Management Science*, 39:477–486, 1993.
3. C. Dillenberger, L. F. Escudero, A. Wollensak, and W. Zhang. On practical resource allocation for production planning and scheduling with period overlapping setups. *European Journal of Operational Research*, 75:275–286, 1994.
4. D. M. Dilts and K. D. Ramsing. Joint lot sizing and scheduling of multiple items with sequence dependent setup costs. *Decision Sciences*, 20:120–133, 1989.

5. G. Dobson. The cyclic lot scheduling problem with sequence-dependent setups. *Operations Research*, 40:736–749, 1992.
6. A. Drexl and K. Haase. Sequential-analysis-based randomized-regret-methods for lotsizing and scheduling. 1993. Forthcoming in *Journal of the Operational Research Society*.
7. A. Drexl and K. Haase. Proportional lotsizing and scheduling. *International Journal of Production Economics*, 40:73–87, 1995.
8. G. D. Eppen and R. K. Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35:832–848, 1987.
9. B. Fleischmann. The discrete lot-sizing and scheduling problem. *European Journal of Operational Research*, 44:337–348, 1990.
10. B. Fleischmann. The discrete lot-sizing and scheduling problem with sequence-dependent setup-costs. *European Journal of Operational Research*, 75:395–404, 1994.
11. K. Haase. Capacitated lot-sizing with sequence dependent setup costs. *OR Spektrum*, 18:51–59, 1996.
12. K. Haase. *Lotsizing and scheduling for production planning, Lecture Notes in Economics and Mathematical Systems*, Vol. 408. Springer-Verlag, Berlin, 1994.
13. C. Jordan and A. Drexl. Lotsizing and scheduling by batch sequencing. 1995. Under review for *Management Science*.
14. U. S. Karmarkar, S. Kekre, and S. Kekre. The dynamic lot-sizing problem with startup and reservation costs. *Operations Research*, 35:389–398, 1987.
15. U. S. Karmarkar and L. Schrage. The deterministic dynamic product cycling problem. *Operations Research*, 33:326–345, 1985.
16. A. Kimms. Multi-level, single-machine lot sizing and scheduling (with initial inventory). 1993. Forthcoming in *European Journal of Operational Research*.
17. A. Kimms. Competitive methods for multi-level lot sizing and scheduling: tabu search and randomized regrets. 1994. Forthcoming in *International Journal of Production Research*.
18. L. S. Lasdon and R. C. Terjung. An efficient algorithm for multi-item scheduling. *Operations Research*, 19:946–969, 1971.
19. T. L. Magnanti and R. Vachani. A strong cutting plane algorithm for production scheduling with changeover costs. *Operations Research*, 38:456–473, 1990.
20. Y. Pochet and L. A. Wolsey. Solving multi-item lot-sizing problems using strong cutting planes. *Management Science*, 37:53–67, 1991.
21. M. Salomon, M. M. Solomon, L. N. Van Wassenhove, Y. D. Dumas, and S. Dauzere-Peres. Discrete lotsizing and scheduling with sequence dependent setup times and costs. 1995. Forthcoming in *European Journal of Operational Research*.
22. L. Schrage. The multiproduct lot scheduling problem. In M. A. H. Dempster and et al., editors, *Deterministic and stocastic scheduling*, 233–244. Dordrecht/Holland, 1982.
23. L. Schrage. Linear, integer and quadratic programming with LINDO, 3rd edition Redwood City, 1986.
24. J.-M. Thizy and L. N. Van Wassenhove. Lagrangean relaxation for the multi-item capacitated lot-sizing problem: A heuristic implementation. *IIE Transactions*, 17:308–313, 1985.
25. T. J. Van Roy and L. A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45–57, 1987.

26. H. M. Wagner and T. H. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5:89–96, 1958.