Jordan, Carsten; Drexl, Andreas

Lotsizing and scheduling by batch sequencing

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 343

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

Nr. 343

Lotsizing and Scheduling by Batch Sequencing

Jordan, C. and A. Drexl

April 1994

Carsten Jordan and Andreas Drexl, Institut für Betriebswirtschaftslehre,
Christian-Albrechts-Universität zu Kiel, Olshausenstr.40, D - 24118 Kiel, F.R.G.

Abstract: The discrete lotsizing and scheduling problem with setup-times is transformed into a single-machine scheduling problem, denoted as batch sequencing problem, which integrates sequence-dependent setups as well. The relationship between the lotsizing and the batch sequencing problem is analyzed. The batch sequencing problem is solved with an enumerative algorithm which is accelerated through bounding and dominance rules. Computational results show that this algorithm solves even the special case of sequence-independent setups more efficiently than a procedure for the discrete lotsizing and scheduling problem which has been recently published.
Keywords: DISCRETE LOTSIZING AND SCHEDULING, SETUP-TIMES, BATCH SEQUENCING, SEQUENCING ALGORITHM, BOUNDING/ DOMINANCE RULES.

1. Introduction

In lotsizing and scheduling models the size and sequence of production lots or batches of different items is determined. In this paper we consider the single-stage single-machine case with setup-times. The problem can be stated as a discrete lotsizing and scheduling problem (DLSP) with setup-times, denoted as DLSPST. In the DLSPST demand for each item is dynamic, backlogging is not allowed. Before each production run a setup is made. Setup-costs and -times depend either on the future item only (sequence-*independent*) or on the sequence of items (sequence-*dependent*). Production takes place to meet present or future demand, in the latter case holding costs are incurred. The planning horizon is segmented into a finite number of (small) periods. In each period at most one item can be produced or a setup is made. An optimal production schedule for the DLSPST minimizes the sum of setup- and holding costs.
The close relationship of the DLSPST to scheduling models in general has motivated us to state the DLSPST as a batch sequencing problem, denoted as BSP. Demand for an item is interpreted as a job with a deadline and a processing time. Jobs of the same item are grouped into families. All jobs must be processed on a single machine between time zero and their deadline, switching from a job in one family to a job in another family incurs (sequence-dependent) setup-costs and -times. Early completion of a job is penalized via earliness costs which correspond to holding costs. An optimal schedule for the BSP minimizes the sum of setup- and earliness costs.

The DLSP has first been introduced by Lasdon/Terjung (1971) with an application for production scheduling in a tire company. A problem similar to the DLSP is considered by Gascon/Leachman (1988) and solved via dynamic programming. An approach based on Lagrangean relaxation is proposed by Fleischmann (1990) for the DLSP without setup-times. Fleischmann (1992) incorporates ideas from solution procedures for vehicle routing problems to solve the DLSP with sequence-dependent setup-costs. The DLSPST is examined by Catrysse et al. (1993). They compute lower and upper bounds with a dual-ascent and column generation heuristic. Their results will serve as a benchmark for our approach to solve the batch sequencing problem (BSP). Complexity results for the DLSP and its extensions are examined in Salomon et al. (1991), where the close relationship of the DLSP to job (class) scheduling problems is emphasized. A broader view of lotsizing and scheduling problems is given in Potts/Van Wassenhove (1992).

Solution procedures for scheduling problems with batch setup-times are considered in Unal/Kiran (1992) and Woodruff/Spearman (1992). Unal/Kiran (1992) address the feasibility of the batch sequencing problem and propose an effective heuristic, Woodruff/Spearman (1992) solve an extension of the BSP with a tabu search heuristic. Both papers stress the relationship to lot-scheduling problems. The complexity of scheduling problems with batch setup-times is investigated by Bruno/Downey (1978) and Monma/Potts (1989).

In Section 2 we state both models and illustrate the transformation with an example. In Section 3 their relationship is investigated, Section 4 presents a sequencing algorithm to solve the BSP. Computational results are presented in Section 5, conclusions follow in Section 6.

2. The DLSPST and the BSP

In this section we state the DLSPST as it is presented in Catrysse et al. (1993), but without considering production costs. We then introduce the BSP. The BSP is stated and solved for the case of sequence-*dependent* setups, therefore including sequence-*independent* setups of the DLSPST as a special case. In Section 3 we show, that for equal holding costs for all items an optimal solution for the BSP is also optimal for the corresponding DLSPST. Thus solution procedures for the BSP and the DLSPST are compared for instances with sequence-independent setups and equal holding costs. The transformation of the DLSPST into the BSP is illustrated with an example.
The parameters of the DLSPST are introduced in Table 1.

Table 1. Parameters of the DLSPST

t	periods $t = 1,...,T$
i	items $i = 1,...,N$
$q_{i,t}$	demand of item i in period t
h_i	holding costs per unit and period of item i
s_i	setup-costs to item i
a_i	setup-time (in periods) to item i
A	set of items with nonzero setup-times = $\{i \mid a_i > 0,\ i=1,...,N\}$

In the DLSPST in each period either one item is produced at full (unit) capacity or a setup is made or the machine is idle (cf. Salomon et al. (1991)). The setup state is not kept during idle time, i.e. if item i ($i \in A$) is not produced in period $t-1$ a setup of a_i periods is needed before t to produce item i in period t (for item $i \notin A$ we have a setup at the beginning of period t). With the decision variables of Table 2 we state the DLSPST in the equations (1) to (7).

Table 2. Decision variables of the DLSPST

$y_{i,t}$	*1*, if item *i* is produced in period *t*; *0*, otherwise;
$v_{i,t}$	*1*, if a setup for item *i* is performed in period *t*; *0*, otherwise;
$I_{i,t}$	inventory of item *i* at the end of period *t*

$$Z_{DLSPST} = min \sum_{i=1}^{N} \sum_{t=1}^{T} s_i v_{i,t} + h_i I_{i,t} \quad (1)$$

subject to

$$\sum_{i=1}^{N} y_{i,t} + \sum_{i \in A} v_{i,t} \leq 1 \qquad t=1,...,T \quad (2)$$

$$I_{i,t-1} + y_{i,t} - q_{i,t} = I_{i,t} \qquad i=1,...,N;\ t=1,...,T \quad (3)$$

$$v_{i,t-a_i+k} \geq y_{i,t} - y_{i,t-1} \qquad i \in A;\ t=a_i+1,...,T;\ k=0,...,a_i-1 \quad (4)$$

$$v_{i,t} \geq y_{i,t} - y_{i,t-1} \qquad i \notin A;\ t=1,...,T \quad (5)$$

$$I_{i,t} \geq 0 \qquad i=1,...,N;\ t=1,...,T \quad (6)$$

$$y_{i,t}, v_{i,t} \in \{0,1\} \qquad i=1,...,N;\ t=1,...,T \quad (7)$$

The objective function (1) minimizes the sum of setup- and holding costs. In (2) setup or production for at most one item is allowed. Equations (3) are inventory balance constraints and (4) and (5) together with (1) appropiately set the variables $y_{i,t}$, $v_{i,t}$ for items with nonzero and zero setup-times, respectively (cf. Catrysse et al. (1993)). To state the BSP we need the parameters of Table 3.

Table 3. Parameters of the BSP

j	job $j = 1,...,J$
d_j	deadline of job j
p_j	processing time of job j
K_j	item i, job j belongs to
$e_j = p_j h_{K_j}$	earliness weight per unit time of job j
$st_{g,i}$	setup-time from item g to item i, $g=0,...,N;\ i=1,...,N;$ $= a_i$ for $g \neq i;\ = 0$, for $g=i;$
$sc_{g,i}$	setup-cost from item g to item i, $g=0,...,N;\ i=1,...,N;$ $= s_i\ max\{1, a_i\}$ for $g \neq i;\ = 0$, for $g=i;$
B	big number

In the DLSPST we assume w.l.o.g. binary demand, i.e., $q_{i,t} \in \{0,1\}$ (cf. Magnanti/ Vachani (1990)). For each item i in the BSP we interpret a sequence of "consecutive ones" in $q_{i,t}$ as one job j. Along this way we derive the job attributes as follows: The last period of such a sequence is the deadline d_j, and the length of the sequence yields the processing time p_j. Jobs j of one item i belong to the same family (=item) K_j (cf. Figure 1). The earliness weight e_j of a job j is the processing time p_j multiplied with the holding costs of its item K_j. For the BSP setup-times $st_{g,i}$ and -costs $sc_{g,i}$ from item g to item i are given in a matrix, thus in general setups can be sequence-*dependent*. We obtain the entries of the matrices from the DLSPST parameters (cf. Tables 3 and 1), where item $g=0$ denotes the idle machine. Note that in this special case setups are sequence-*independent*.

With the decision variables in Table 4 we give the "conceptual" formulation (8) - (11) for the BSP.

Table 4. Decision variables for the BSP

π	job sequence
j_k	k-th job of the sequence π
X_j	completion time of job j
Y_k	*1*, if a setup (from the idle machine) to the k-th job is performed; *0*, otherwise;

$$Z_{BSP} = \min \sum_{k=1}^{J} e_{j_k} \cdot (d_{j_k} - X_{j_k}) + sc_{K_{j_{k-1}}, K_{j_k}} + Y_k \cdot sc_{0,K_{j_k}} \tag{8}$$

subject to

$$X_{j_{k-1}} \le \min\{X_{j_k} - p_{j_k} - st_{K_{j_{k-1}}, K_{j_k}}; d_{j_{k-1}}\} \qquad k = 1,\ldots,J \tag{9}$$

$$B \cdot Y_k - (X_{j_k} - p_{j_k} - X_{j_{k-1}}) \ge 0 \qquad k = 2,\ldots,J;\ K_{j_{k-1}} = K_{j_k} \tag{10}$$

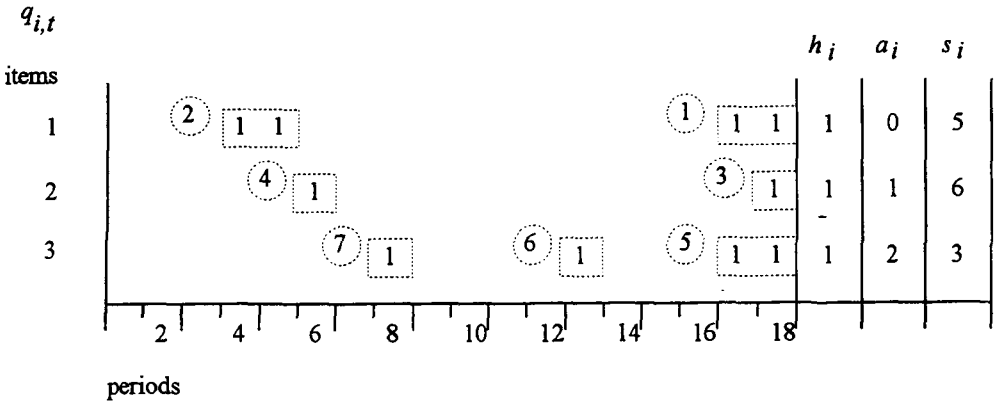
$$X_0 = j_0 = d_0 = K_0 = Y_1 = 0;\ Y_k \in \{0;1\},\ k = 2,\ldots,J; \tag{11}$$

In this formulation the decision variable "job sequence" π is an index, thus the formulation cannot be used for conventional mixed-integer-programming solvers.

The objective function (8) minimizes weighted earliness and setup-costs. Constraint (9) imposes the sequence on the machine and enforces the completion time X_j of each job j to be less or equal its deadline d_j. If consecutive jobs j_{k-1} and j_k of the same item are not processed immediately after each other, Y_k is set to one in (10), a new setup (from the idle machine) is performed and taken into account in the objective (8). Initialization of variables is provided in (11). In the BSP the unit capacity constraints (2) of the DLSPST transform into the condition, that the single machine can handle only one job at a time. Recall that the BSP integrates sequence-*dependent* setups.

A DLSPST instance with $N=3$ items and $T=18$ periods with the parameters $q_{i,t}$, h_i, a_i and s_i is given in Figure 1. Entries $q_{i,t}=0$ are omitted. Dotted rectangles and circles denote the jobs (=demands) and job-numbers of the BSP, respectively.

Figure 1. Interpretation of demands as jobs



For item 1 we have $q_{i,t}=1$ in periods 18, 17 and 5, 4, respectively, and derive the attributes of jobs 1 and 2 (cf. Table 5). The setup-time matrix $st_{g,i}$ is obtained directly from a_i, the setup-cost matrix through s_i $max\{1, a_i\}$, setups are sequence-independent. We yield 7 jobs labelled consecutively within each family in the order of decreasing deadlines.

Table 5. Job attributes and setup-time and -cost matrix

j	d_j	p_j	K_j	e_j
1	18	2	1	2
2	5	2	1	2
3	18	1	2	1
4	6	1	2	1
5	18	2	3	2
6	13	1	3	1
7	8	1	3	1

	$st_{g,i}$	to 1	to 2	to 3
	0	0	1	2
	1	0	1	2
from	2	0	0	2
	3	0	1	0
	$sc_{g,i}$			
	0	5	6	6
	1	0	6	6
from	2	5	0	6
	3	5	6	0

The DLSPST solution can be represented with a string of length 18 with entries [-, a, 1, 2, 3] for idle time, setup-time and production of the different items, respectively, cf. Figure 2. This DLSPST solution is the schedule of the job sequence π = (4, 2, 7, 6, 5, 3, 1). Both solutions yield the optimal objective function value $Z_{DLSPST} = Z_{BSP} = 48$.

For higher setup-costs to item 3 we obtain another solution, cf. Figure 3. Jobs 6 and 5 are now scheduled contiguously behind job 7 to avoid a second setup to item 3.

Figure 2. DLSPST and BSP solutions

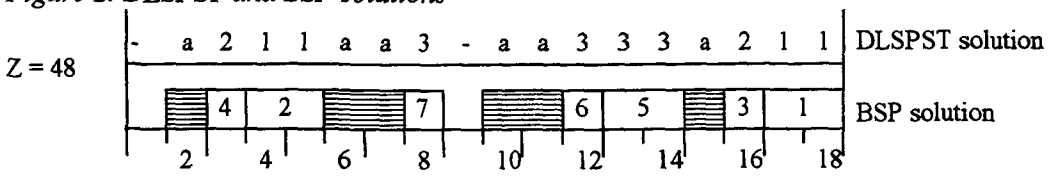
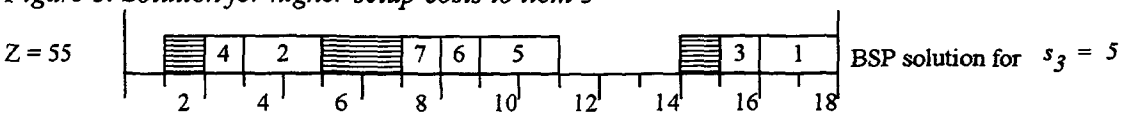


Figure 3. Solution for higher setup-costs to item 3



Note that the problem size $N \times T$ (items×periods) of the DLSPST can be quite different from the problem size J (number of jobs) of the BSP.

3. Relationship between the DLSPST and the BSP

We first give some definitions and then analyze the relationship between the DLSPST and the BSP.

In the DLSPST we define a *batch* as a noninterrupted sequence of periods where production takes place for one item. In the BSP jobs of one item which are processed contiguously without idle time between them form a *batch.* E.g., in Figure 2 jobs 7 and 6, 5 form two batches, in Figure 3 jobs 7, 6, 5 form one batch.

We refer to *demand splitting* as splitting one job j in two jobs j' and j'' on two different batches, the corresponding DLSPST schedule is called *preemptive* (cf. Salomon et al. (1991)). The DLSPST allows *demand splitting,* while the BSP does not, i.e. jobs must not be preempted, only *nonpreemptive* schedules are allowed.

In the BSP the earliness weight e_j of a job j is proportional to its processing time p_j, cf. Table 3. Thus for h_i identical for all items i the weighted processing time p_j/e_j is identical for all jobs. The following theorems state that in this case demand splitting does not need to be considered. Furthermore, we only need to consider schedules, where jobs within one family (=item) are sequenced in an earliest deadline order, referred to as EDDWF ordering (earliest deadline within families, cf. Woodruff/ Spearman (1992)). EDDWF ordering reduces the size of the enumeration tree, cf. Section 4.

All theorems are easily proven with exchange arguments (cf. e.g. Monma/ Potts (1989)), therefore the proofs are omitted.

Definition 1. For an EDDWF ordering we denote with $j \leftarrow i$ the case where $j = i + 1$ and $K_i = K_j$. So $j \leftarrow i$ denotes that i and j belong to the same item and due to the labelling within an item in order of decreasing deadlines, i is the job at the "next" deadline, cf. e.g. Figure 1 the jobs $4 \leftarrow 3$.

A schedule $\sigma = (A, i, B, j, C)$ denotes the completion times of jobs A, i, B, j and C. We refer to A, B, C as composite jobs, where two or more jobs are aggregated to one composite job. From a schedule σ we

derive the earliness costs of each job and from the sequence of items we derive the setup-costs, thus *costs* of a schedule σ can easily be determined (cf. (8)).

Theorem 1. Assume $j \leftarrow i$ and a schedule $\sigma = (A, i, j, B)$. Then a schedule $\sigma' = (A, j, i, B)$ can be constructed with equal costs. As a consequence, jobs j, i in one batch can be scheduled according to EDDWF.

For $j \leftarrow i$ and j, i in different batches interchanging j and i can be seen as a special case of demand splitting:

Theorem 2. Assume $j \leftarrow i$ and a schedule $\sigma = (A, i, B, j, C)$. Without changing the completion time of B we can construct a schedule $\sigma' = (A, j, i', B, i'', C)$ or $\sigma'' = (A, j', B, j'', i, C)$ with costs equal to costs of σ, so that jobs j, i are split but scheduled in EDDWF order.

To find *feasible* schedules we do not need to consider *demand splitting* or *preemptive* schedules (cf. Unal/Kiran (1992), Salomon et al. (1991)):

Theorem 3. Assume a feasible *preemptive* schedule σ for the DLSPST which corresponds to a schedule with job j split in j' and j'' on two batches, $\sigma = (A, j', B, j'', C)$. Then the schedule $\sigma' = (A, B, j, C)$ is a feasible *nonpreemptive* schedule for the DLSPST which corresponds to a feasible schedule for the BSP.

In Theorem 3 we can schedule B earlier without violating feasibility. Theorem 4 states, that it is sufficient to consider nonpreemptive schedules in the special case of equal holding cost:

Theorem 4. Let all holding costs be equal, i.e. w.l.o.g. h_i=1 for all items $i=1,...,N$. Assume a feasible *preemptive* schedule σ for the DLSPST. Then there is a feasible *nonpreemptive* schedule σ' for the DLSPST which corresponds to a feasible schedule for the BSP with less or equal costs.

Thus in the case of equal holding costs an optimal schedule for the BSP is also optimal for the DLSPST.

4. Solving the BSP by job sequencing

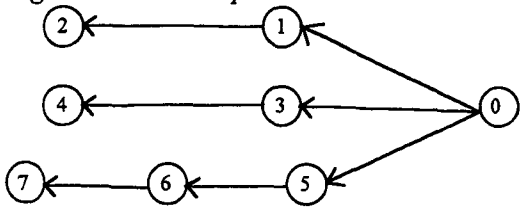
In the following we present an algorithm to solve the BSP to optimality. We derive a schedule from the job sequence and describe the basic enumeration scheme in Section 4.1. In Sections 4.2 to 4.4 we provide bounding and dominance rules in order to prune large parts of the search tree.

4.1 Sequencing algorithm

In the following we assume jobs within the families being (re-)labeled according to the EDDWF ordering as in Figure 1. The sequences permitted by EDDWF preordering can be represented with a

precedence graph, cf. Figure 4 for our instance. We schedule a project with a serial structure on a single facility. The EDDWF preordering substiantially reduces the number of sequences.

Figure 4. EDDWF precedence network



The sequencing algorithm constructs partial schedules starting at the planning horizon (the largest deadline) with the (fictitious) job $j = 0$, i.e. the root of the search tree. At any stage a job is eligible, if all its predecessors (of the precedence graph) are scheduled. Partial schedules are extended by scheduling an eligible job in each stage of the enumeration tree. This way the *last* job scheduled defines the starttime of a (partial) schedule σ. Iff a partial schedule σ' is bounded or dominated by another partial schedule σ, backtracking occurs. Extending partial schedules we apply a depth-first strategy. A schedule σ comprises a solution of the BSP when all J jobs have been scheduled. The algorithm stops after all sequences have been examined, the best solution found is optimal.

From the job sequence π we derive a semiactive schedule, which is then transformed into the schedule σ. Therefore we can use the terms sequence and schedule interchangeably. Schedule σ is the minimal cost schedule for sequence π, so that only the sequences have to be enumerated.

For a sequence π given, we compute a semiactive schedule starting with $X_{j_J} = d_{j_J}$ as follows:

$$X_{j_{k-1}} = \min \{ X_{j_k} - p_{j_k} - st_{K_{j_{k-1}}, K_{j_k}} ; d_{j_{k-1}} \} \qquad k = J,\ldots,2 \qquad (9)$$

In this semiactive schedule no local right-shift of a single job is possible. From the semiactive schedule we derive schedule σ solving a subproblem: W.r.t. a semiactive schedule we denote a *group* of jobs as all consecutive jobs belonging to the same item. Within the groups we (may) have to decide which jobs shall be batched, referred to as a *single-item subproblem*. This subproblem only needs to be solved if there is idle time behind the *first* job of a group in the semiactive schedule. Similar to the Wagner-Whitin algorithm (cf. Wagner/Whitin (1958)) we can solve this single item subproblem via dynamic programming.

Consider Figures 2 and 3 with the job sequence π = (4, 2, 7, 6, 5, 3, 1). The semiactive schedule for π is shown in Figure 2, another schedule is shown in Figure 3. In both schedules jobs 7, 6, 5 of item 3 form a group. In the semiactive schedule, cf. Figure 2, job 7 is the first job of the group and the machine is idle behind it. For $s_3=3$ it is optimal to split the group into two batches, the semiactive schedule is identical with schedule σ. In Figure 3 for $s_3=5$ a second setup to item 3 with costs of $a_3 \cdot s_3 = 10$ is more

expensive than leftshifting jobs 5, 6 which causes earliness costs of 9. Therefore jobs 6, 5 are scheduled contiguously behind job 7 and for higher setup-costs we yield another schedule σ for π. We refer to leftshifting jobs to avoid a second setup as *pulling back* jobs. Note, that in both cases the first job of the group, here job 7, is scheduled at its deadline.

In the sequencing algorithm first all jobs of one group are batched. Then, we only need to solve the single-item subproblem if the costs incurred due to pulling back the first group of a partial schedule exceed the setup-costs, e.g. extending the partial sequence (6,5,3,1) to (7,6,5,3,1) for $s_3=3$.

4.2 Bounding rules

In order to present bounding and dominance rules we need attributes of partial schedules, cf. Table 6.

Table 6. Attributes of partial schedules I

UB	upper bound (costs of the current best solution)
σ	(partial) schedule of the job sequence $(j_h, j_{h-1}, \ldots, j_1)$
$f(\sigma)$	item i, the first job j_h (=last job scheduled) of schedule σ belongs to
$t(\sigma)$	starttime of the first job j_h of schedule σ
$c(\sigma)$	costs of schedule σ without the setup to j_h
$J(\sigma)$	set of currently unscheduled jobs
$I(\sigma)$	set of items, the unscheduled jobs belong to $= \{i \mid i = K_j, j \in J(\sigma)\}$

The objective function value of the best solution computed during enumeration is denoted with UB. Scheduling the job sequence $(j_h, j_{h-1}, \ldots, j_1)$ semiactively we denote with $t(\sigma)$ the starttime of the last scheduled job j_h and with $f(\sigma)$ the item it belongs to. The costs $c(\sigma)$ of a partial schedule σ can easily be updated when a new job is scheduled, in some cases we may have to solve a single-item subproblem, cf. Section 4.1. For each partial schedule σ the sets of unscheduled jobs $J(\sigma)$ and unscheduled items $I(\sigma)$ are known.

Two bounds are easily derived, a feasibility bound and a cost bound:

A lower bound for the time $T(\sigma)$ necessary to schedule the jobs $j \in J(\sigma)$ is the minimal time needed to perform a setup to each family contained in $I(\sigma)$ and the sum of all processing times of jobs in $J(\sigma)$. By defining $T(\sigma) = \sum_{i \in I(\sigma)} st_{0,i} + \sum_{j \in J(\sigma)} p_j$, we derive the

Feasibility bound: Fathom a partial schedule σ if

$T(\sigma) > t(\sigma)$, *because there is no feasible extension of* σ.

A lower bound for the costs $C(\sigma)$ necessary to schedule the jobs $j \in J(\sigma)$ is the sum of costs needed to perform a setup to each item in $I(\sigma)$.

By defining $C(\sigma) = \sum_{i \in I(\sigma)} sc_{0,i}$, we derive the

Cost bound: Fathom a partial schedule σ if

$c(\sigma) + C(\sigma) \geq UB$, *because there is no extension of* σ *with lower overall costs.*

Remark 1. For sequence-dependent setups we consider in each column i the minimum of $st_{g,i}$ and $sc_{g,i}$, $g=0,...,N$, $g \neq i$.

4.3 Dominance rules

We need to define further attributes of partial schedules to describe the dominance rules, cf. Table 7.

Table 7. Attributes of partial schedules II

$G_1(\sigma)$	set of jobs which form the *first* group of a (partial) schedule σ $= \{j_h, j_{h-1}, ..., j_{h-k} \mid K_{j_{h-k}} = f(\sigma), k = 0,...,z(\sigma)\}$
$z(\sigma)$	number of jobs in the first group
$w_1(\sigma)$	sum of earliness weights of $G_1(\sigma)$ $= \sum_{j \in G_1(\sigma)} e_j$
$W(\sigma)$	sum of earliness weights of jobs in $J(\sigma)$ $= \sum_{j \in J(\sigma)} e_j$
$pbs(\sigma)$	pull back start time of the first job j_h of a schedule σ
$pc(\sigma)$	additional earliness costs for schedule σ if $G_1(\sigma)$ is pulled back $= w_1(\sigma)\ max\{0,\ t(\sigma) - pbs(\sigma)\}$

Two different partial schedules σ and σ' are compared if they schedule the same set of jobs, i.e. $J(\sigma) = J(\sigma')$. A schedule σ' is dominated if another schedule σ is more *efficient* in terms of time *and* costs, so that σ starts later (leaves more space for jobs in $J(\sigma)$ which may be scheduled later) and $c(\sigma) \leq c(\sigma')$. Unfortunately, the dominance rules are complicated through the fact that jobs must be scheduled in a "DLSP" like fashion (no idle time in a batch): Comparing two partial schedules we have to take into account that the first group $G_1(\sigma)$ of a partial schedule σ may be "pulled back" to avoid another setup.

Thus we need an upper bound on the costs transforming a partial schedule σ (where $G_1(\sigma)$ is scheduled semiactively) into a DLSP schedule, denoted as pulling cost $pc(\sigma)$.
The earliest time a job j can be "pulled back" to is its starttime scheduling it contiguously behind the job of its item $i = K_j$ with the smallest deadline. In Figure 1 job 6 may start at time 8 due to a "pull back", job 5 may be pulled back to $8 + p_6 = 9$. Along this way a "pull back start" of each job is derived, independently of a schedule σ. For a given partial schedule σ we define $pbs(\sigma)$ as the pull back start of the *first* job j_h of σ. Now $max\{0, t(\sigma) - pbs(\sigma)\}$ is the time intervall $G_1(\sigma)$ may maximally be pulled back. Weighting this time with $w_1(\sigma)$, we obtain the pulling costs $pc(\sigma)$ of a partial schedule σ, the required upper bound. Costs through pulling back $G_1(\sigma)$ occur if σ is extended with a job j, $K_j = f(\sigma)$ and $d_j < t(\sigma)$ (e.g. extending the partial schedule (6, 5, 3, 1) to (7, 6, 5, 3, 1) in Figure 3).

In addition, we need an upper bound on the cost decrease if jobs $j \in J(\sigma)$ are delayed. Consequently, $W(\sigma)$ denotes the sum of earliness costs of jobs in $J(\sigma)$. Weighting $W(\sigma)$ with a delay we get the required upper bound on the costs decrease.

Dominance rules for different first items

In the following we assume that two partial schedules σ and σ' schedule the same set of jobs, i.e. $J(\sigma) = J(\sigma')$, but the items of the first jobs of σ and σ' are different, i.e. $f(\sigma) \neq f(\sigma')$. We ask for conditions to hold so that schedule σ dominates schedule σ'.

Remark 2. All dominance rules are stated for sequence-*dependent* setups. The setup-time and -cost matrices have to satisfy the triangle inequality. We need furthermore the following monotonicity property between $st_{g,i}$ and $sc_{g,i}$: If a permutation P' of the set of items (including 0) leads to higher setup-times than a permutation P, then the setup-costs of P' must exceed setup-costs of P, too (for sequence-*independent* setups different permutations P and P' lead to the same setup-costs and -times).

Case I: $t(\sigma') + st_{f(\sigma'), f(\sigma)} \leq t(\sigma)$

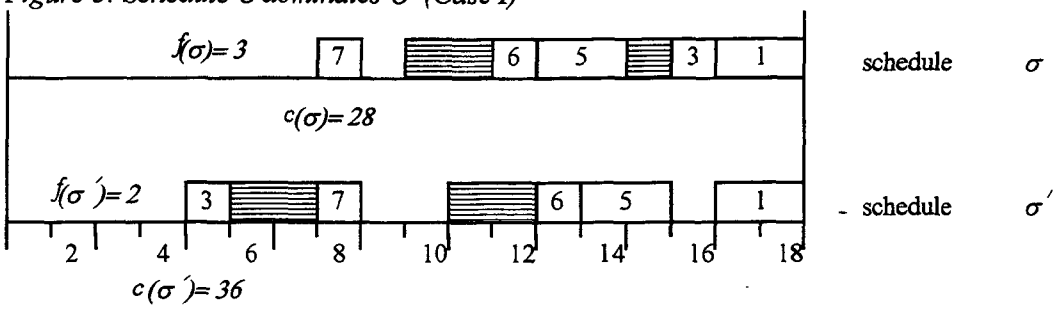
Schedule σ' is dominated by schedule σ if
$c(\sigma) + pc(\sigma) \leq c(\sigma) - sc_{f(\sigma'), f(\sigma)}$.

In Case I schedule σ' starts earlier than schedule σ. Schedule σ' is dominated by σ, if maximal costs of σ are lower than actual costs of σ' minus setup-costs from $f(\sigma')$ to $f(\sigma)$.

Proof: Let ω be a partial schedule of all jobs $j \in J(\sigma)$ which extends σ to a feasible solution. With the triangle inequality valid for the setups an extension ω of σ' cannot schedule jobs later than schedule σ. For an extension ω of σ' we can find a better solution scheduling ω before σ and performing a setup after ω from $f(\sigma')$ to $f(\sigma)$. □

Consider Figure 5 as an example for Case I: Schedule σ starts at time 7, schedule σ' at time 4. We have $pc(\sigma) = pc(\sigma') = 0$ because $t(\sigma') < d_4$, the pull back start of job 3, and job 7 cannot be pulled back because it is the job of item 3 with the smallest deadline. So we fathom schedule σ' as we have $28 + 0 \leq 36 - 6$.

Figure 5. Schedule σ dominates σ' (Case I)



Case II: $t(\sigma') + st_{f(\sigma'), f(\sigma)} > t(\sigma)$

Let $\Delta = t(\sigma') + st_{f(\sigma'), f(\sigma)} - t(\sigma)$.

Schedule σ' is dominated by schedule σ if
$c(\sigma) + pc(\sigma) + \Delta \cdot W(\sigma) \leq c(\sigma') - sc_{f(\sigma'), f(\sigma)}$ *and*

schedule σ has a feasible extension.

In Case II an extension of σ' may schedule all jobs in $J(\sigma)$ Δ periods later than any extension of σ. In this case σ must have a feasible extension to dominate a schedule σ' which leaves more space for jobs in $J(\sigma)$.

Proof: An extension ω of σ' can have lower earliness costs than extending σ with ω, but not less than $\Delta \cdot W(\sigma)$ (cf. Figure 8). Due to the monotonicity property between setup-times and -costs we will not find an extension ω of σ', which is infeasible for σ and has lower setup-costs than the existing feasible extension of σ. □

A geometrical illustration of Cases I and II is given in Figure 8.

Dominance rules for equal first items

Again we have $J(\sigma) = J(\sigma')$ and also the items of the first jobs of σ and σ' are equal, i.e. $f(\sigma) = f(\sigma')$. Furthermore the first job j_h of both partial schedules σ and σ' is identical as we schedule jobs in EDDWF order. So also the pull back start time is equal for both partial schedules, i.e. $pbs(\sigma) = pbs(\sigma')$. Clearly, for $G_1(\sigma) = G_1(\sigma')$ and $t(\sigma') = t(\sigma)$, σ' is dominated by σ if $c(\sigma) \leq c(\sigma')$. Cf. Figure 6, where $G_1(\sigma) = G_1(\sigma') = \{5\}$ and $21 \leq 23$.

If the schedules σ and σ' differ in their first group, the pulling costs need to be considered: In Figure 7 we have $t(\sigma) = 12 \geq t(\sigma') = 10$ and $c(\sigma) = 12 \leq c(\sigma') = 17$. But if we now extend $\sigma = (6, 5, 3)$ to $\sigma =$

(7, 6, 5, 3) we have $c(\sigma) = 24$, extending $\sigma' = (6, 3, 5)$ to $\sigma' = (7, 6, 3, 5)$ produces $c(\sigma') = 19$. In this example schedule σ does *not* dominate σ' due to $pc(\sigma) = 12 > pc(\sigma') = 2$.

Figure 6. Schedule σ dominates σ' (Case III)

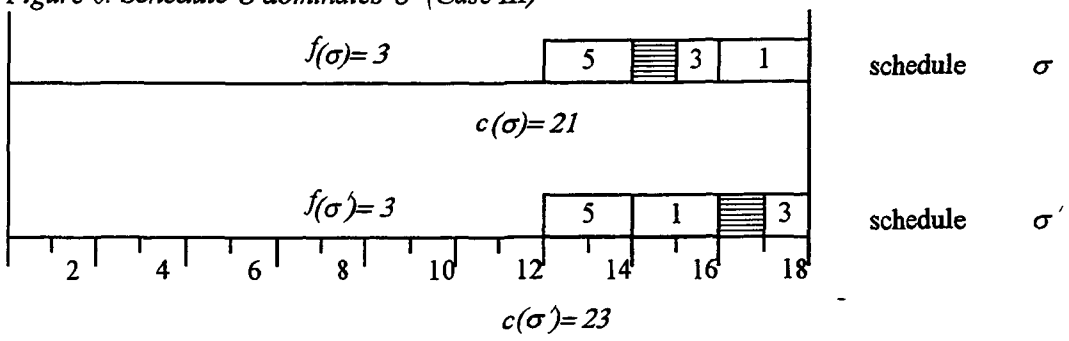
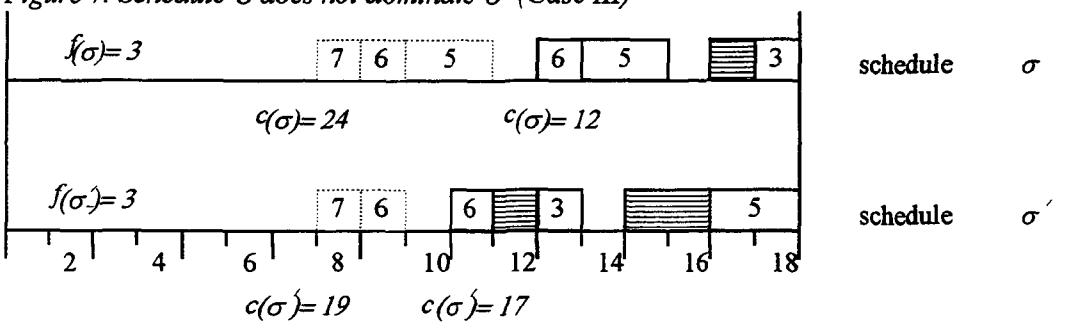


Figure 7. Schedule σ does not dominate σ' (Case III)



For equal first items we have the following rules:

Case III: $t(\sigma') \leq t(\sigma)$

Let $\Delta = t(\sigma') - t(\sigma)$.

Schedule σ' is dominated by schedule σ if:

$c(\sigma) + \Delta \cdot pc(\sigma) \leq c(\sigma')$ *and* $c(\sigma) + pc(\sigma) \leq c(\sigma') + pc(\sigma')$.

In Case III schedule σ starts later than σ'.

Proof. Again any extension ω of σ' can be scheduled before σ with less or equal cost. Scheduling $G_1(\sigma)$ or $G_1(\sigma')$ at any time between $pbs(\sigma') = pbs(\sigma)$ and $t(\sigma)$, schedule σ has lower costs than σ. □

Case IV: $t(\sigma') > t(\sigma)$

Schedule σ' is dominated by schedule σ if

$c(\sigma) + \Delta \cdot W(\sigma) \leq c(\sigma')$, $c(\sigma) + pc(\sigma) \leq c(\sigma') + pc(\sigma')$ *and*

schedule σ has a feasible extension.

In Case IV an extension ω of σ' may schedule all jobs in $J(\sigma)$ Δ periods later and we need the correction term $\Delta \cdot W(\sigma)$ (cf. Figure 9). Schedule σ must have a feasible extension to dominate schedule σ'.

Figure 8. Illustration for Cases I and II

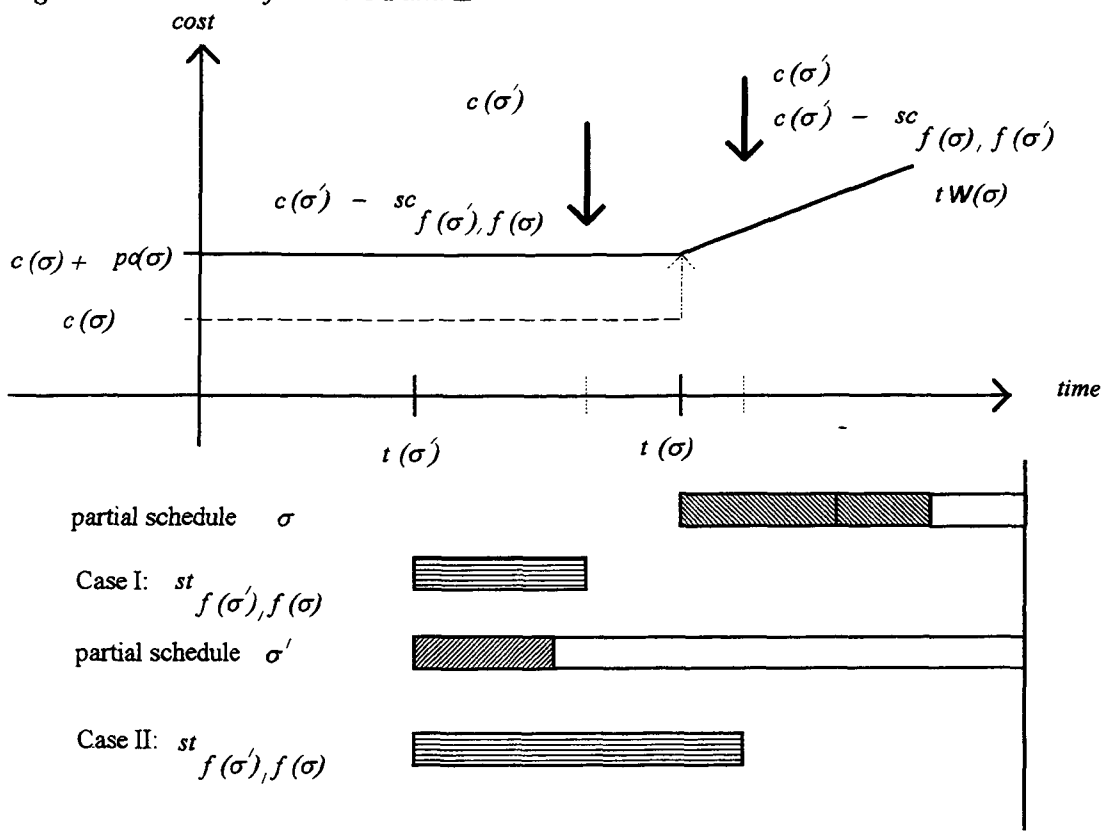
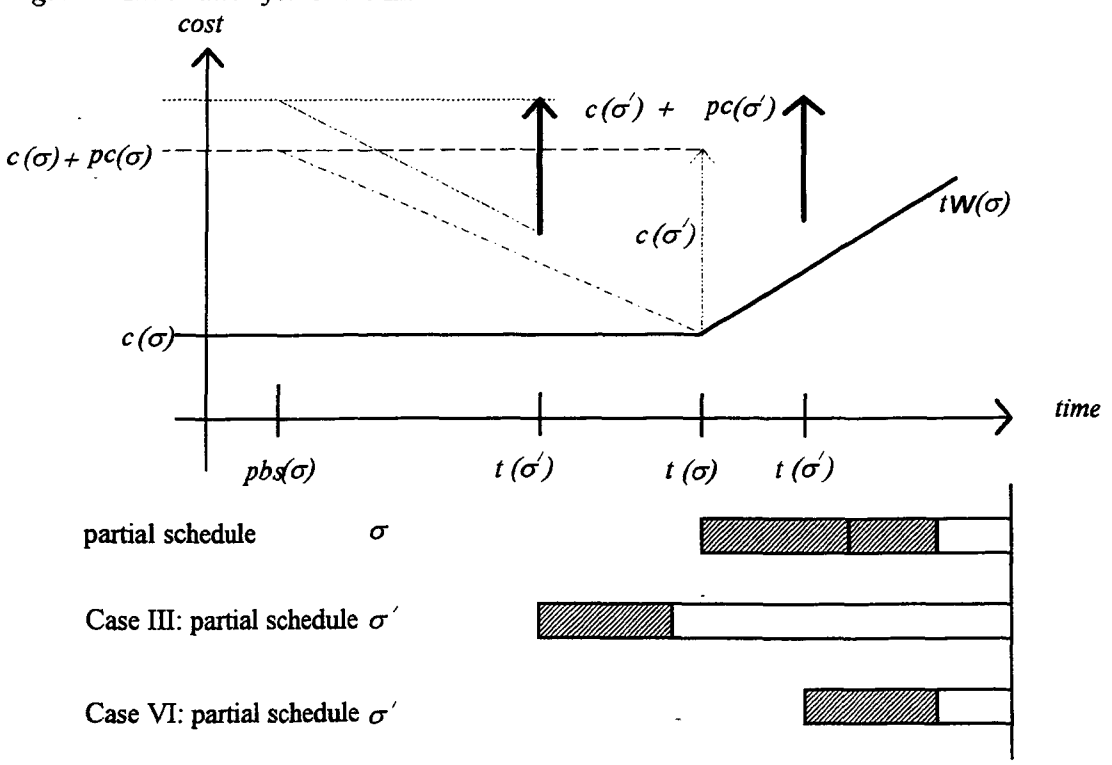


Figure 9. Illustration for Cases III and VI



Proof. Refer to the proofs for Case II and Case III.

In Figure 9 a geometrical illustration of Cases III and IV is given. Costs through pulling back increase linearily between $pbs(\sigma) = pbs(\sigma')$ and $t(\sigma)$ or $t(\sigma')$, and in this intervall costs of schedule σ' are higher than costs of σ.

Item extension dominance rule

A local pruning of the search tree is proposed by Park et al. (1993) concerning the extensions of partial schedules σ: As long as we can extend a schedule σ with jobs from item $f(\sigma)$, which are *early* in the semiactive schedule, we do not need to consider any *other* extension of σ. All jobs in $G_1(\sigma)$ are early, they can be considered as one single job. More formally:

Extend a partial schedule σ only with job j (all other extensions are dominated)
for:

$$j \in J(\sigma),\ K_j = f(\sigma),\ t(\sigma) < d_j$$

Cf. Figure 2, the partial schedule σ = (5, 3, 1) can be extended with the (eligible) jobs 2, 4 and 6 (cf. Figure 4). Job 6 is early extending schedule σ = (5, 3, 1) to σ = (6, 5, 3, 1). So we do not need to examine the extensions of σ = (5, 3, 1) with jobs 2 and 4.

The item extension rule is easily proven with an exchange argument. We apply this rule together with a branching rule where the next job scheduled to extend σ belongs to the same item as $f(\sigma)$, so that setups are avoided.

5. Computational results for the nonpreemptive case

In nonpreemptive schedules jobs in the BSP, which correspond to demands in the DLSPST, must not be split on two batches. In the DLSPST schedules may be preemptive, which is not allowed for the BSP. But the BSP includes the case of sequence-*dependent* setups, while setups in the DLSPST must be sequence-*independent*. Figure 10 shows the different types of schedules and setups covered by the corresponding models. Sequence-independent setups are a special case of sequence-dependent setups, as well as nonpreemptive schedules form a subset of the preemptive ones.

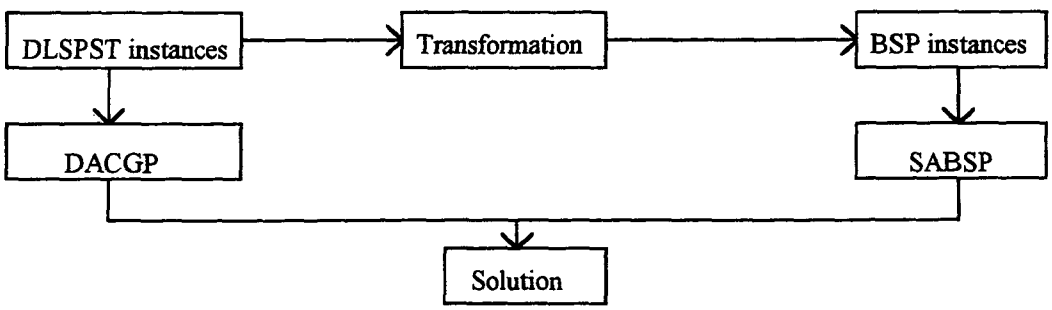
Figure 10. Types of schedules and setups for the different models

schedules \ setups	sequence independent	sequence dependent
nonpreemptive	DLSPST BSP	BSP
preemptive	DLSPST	—

In Catrysse et al. (1993) computational results for the DLSPST are reported only for equal holding costs for all items. In Section 3 we stated that in this case a *nonpreemptive* schedule is optimal for the DLSPST. Thus the algorithms to solve the BSP and the DLSPST are directly comparable. Catrysse et al. (1993) refer to their procedure as the dual ascent column generation procedure, denoted as DACGP. They calculate a lower bound and an upper bound by means of column generation. The gap ΔZ provides the difference between upper and lower bound, which is the maximal deviation of DACGP from the optimal objective function value. We refer to our sequencing algorithm to solve the BSP as SABSP. SABSP stops after the enumeration is finished, the best solution found is optimal.

To compare DACGP and SABSP we solve the DLSPST instances with nonzero setup-times provided by Catrysse et al. (1993). The DLSPST instances are transformed into BSP instances and solved with SABSP, cf. the framework provided in Figure 11. All instances have holding costs $h_i = 1$ for all items i so that Theorem 4 of Section 3 holds. Thus the BSP solution is optimal for the DLSPST.

Figure 11. Transforming and solving DLSPST instances



The DACGP is coded in FORTRAN, the SABSP in C. Both algorithms are implemented on an IBM-PS2 Model 80 with mathematical coprocessor 80387.

Catrysse et al. (1993) generated problems for item-period combinations $\{(N,T)\} = \{(2,20), (2,40), (4,40), (2,60), (4,60), (6,60)\}$. In the following we refer only to problems with $T=60$, because smaller problems are solved much faster by SABSP than by DACGP. The DLSPST instances have setup-times a_i of either 0, 1 or 2 periods, the average setup-time per item is (approximately) 0.5, so that items often have zero setup-time. For each item-period combination there are instances with different (approximate) capacity utilization ρ: low (L) capacitated $(\rho \leq 0.55)$, medium (M) $(0.55 \leq \rho \leq 0.75)$ and high (H) capacitated problems $(\rho > 0.75)$. There are 30 instances for each (N,ρ) combination, so that in total we

yield $3x3x30 = 270$ instances. In Table 8 we denote with $\#J$ the average number of jobs in the transformed BSP instances. For the DACGP we denote with ΔZ the maximal gap (in percent) between upper and lower bound, with #I the number of problems found infeasible and with R the average time (in seconds) needed for the 30 instances. The SABSP is exact, we denote with R′ the average time (in seconds) needed for solving the instances to optimality and #I′ denotes the number of problems found infeasible by SABSP.

Table 8. Results within each problem class (30 instances)

			DACGP			SABSP	
(N,T)	*#J*	*ρ*	ΔZ	#I	R	R′	#I′
(2,60)	19	L	0.17	2	25.8	0.1	2
	25	M	0.20	7	76.3	0.2	7
	29	H	1.22	10	274.9	0.1	9
(4, 60)	21	L	0.15	3	38.9	5.5	3
	31	M	0.47	6	120.8	18.1	5
	35	H	1.43	(1 11	268.7	11.4	10
(6, 60)	22	L	0.13	1	56.2	48.4	1
	33	M	0.70	(2 10	264.9	401.8	7
	35	H	0.99	10	274.1	195.4	10

(1: Feasible integer solution found by simplex based procedure for one problem instance (cf. Catrysse et al. (1993))

(2 LP-relaxation is feasible but no integer solution found for one problem instance (cf. Catrysse et al. (1993))

Problems with $N = 2$ and 4 are solved much faster by SABSP than by DACGP, the number of sequences to examine is relatively small. For $N = 6$ computation times of SABSP are comparable to DACGP. In the problem class $(N,T,\rho) = (6,60,M)$ average time of SABSP is larger than the one of DACGP, but at least two existing solutions are not found by DACGP. Furthermore, the average time in this class would be 133.6 sec without considering one instance, which takes 8181 sec with SABSP. Note, that there are other problem classes $(N,T,\rho) = (2,60,H), (4,60,M)$ where DACGP does not find existing feasible solutions.

Solution times differ considerably for SABSP. Table 9 gives the frequency distribution of solution times and shows that in every problem class the majority of instances is solved in less than R, the average time for DACGP. The bounding and dominance rules stated in Section 4 are applied at each node of the search tree. Dominance rules for equal items are more often succesfully applied than those for different first items, and Case I and III more often than Case II and IV. But none of the rules could be omitted without worsening the results.

Table 9. Frequency distribution of solution times

		Number of instances solved in less than .. [sec]								
(N,T)	ρ	< 0.1	< 1	< 10	< 30	< 100	< 300	< 1000	< 10000	< R
(2,60)	L	10	20							30
	M	4	26							30
	H	13	17							30
(4,60)	L		3	23	4					30
	M		12	13	5					30
	H	3	4	9	12	2				30
(6,60)	L		2	11	8	4	3	2		25
	M	1	1	1	5	13	4	4	1	23
	H	1		2	6	9	4	7	1	21

Computational results for sequence-dependent setups

For sequence-dependent setups only a small sample of instances with 6 items (manually generated) is examined: integer processing and setup-times are out of the intervall [1..10], setup-costs are an integer multiple of setup-times. Deadlines for the 6-item problems are uniformely distributed over the planning horizon. We examine different kinds of setup structures: With 3G (2G) we denote setups where the 6 items are grouped into 3 (2) families with large intra- and small interfamily setups, SQ denotes that setups from item i to item $j > i$ are small and large for $j < i$ (e.g. from a light to a dark colour). With RD matrices with random entries (satisfying the triangle inequality) are denoted. Lower (L) and medium (M) capacitated problems are generated multiplying all deadlines of the high (H) capacitated instances with 1.3 and 1.6, respectively. In Table 10 *#J* denotes the number of jobs of the BSP and *(N,T)* the size of the corresponding DLSPST. Furthermore the solution times in seconds on the IBM-PS2 Model 80 for the different setup structures are given.

Table 10. CPU times for sequence-dependent setups

			SABSP			
#J	*(N,T)*	ρ	3G	2G	SQ	RD
30	*(6,300)*	L	1248	1480	607	1503
30	*(6,240)*	M	929	1280	557	1080
30	*(6,190)*	H	221	242	161	187

The results show that computation times decrease with a smaller solution space, i.e. a higher capacity utilization, especially from medium (M) to high (H) utilization.

6. Summary and conclusions

In this paper we consider the discrete lotsizing and scheduling problem with setup-times, denoted as DLSPST and the batch sequencing problem, denoted as BSP. Interpreting demands in the DLSPST as jobs we transform the DLSPST into a BSP. The dual ascent and column generation procedure (DACGP) to solve the DLSPST introduced in Catrysse et al. (1993) is compared with a sequencing algorithm (SABSP) to solve the BSP. DACGP solves the DLSPST heuristically and provides an upper and lower bound, SABSP solves the BSP to optimality. In the DLSPST setups must be sequence-*independent*, but demands can be split on two batches (preemptive schedules). The BSP integrates sequence-dependent setups but jobs must not be preempted, we only consider nonpreemptive schedules. Catrysse et al. (1993) report computational experience with DACGP for a set of instances with equal holding costs for all items. In this case the sequencing algorithm for the BSP (SABSP) solves also the corresponding DLSPST exactly. Our results show, that SABSP is more efficient than DACGP, exact solutions are found in a shorter time than heuristic solutions.

SABSP is conjectured to be advantageous for instances with few items and a small solution space (i.e. long setup-times, high capacity utilization) whereas DACGP is supposed to be better suited for lower capacitated instances with many items and smaller setup-times.

In the future we will extend the BSP to multilevel structures and multiple machines.

Acknowledgement: The authors are indepted to Dirk Catrysse, Katholieke Universiteit Leuven, for providing the test instances.

References

Bruno, J. and P. Downey, 1978. Complexity of task sequencing with deadlines, setup-times and changeover costs, *SIAM Journal on Computing,* Vol. 7, pp. 393-404.

Catrysse, D., M. Salomon, R. Kuik and L. van Wassenhove, 1993. A dual ascent and column generation heuristic for the discrete lotsizing and scheduling problem with setup-times, *Management Science,* Vol. 39, pp. 477 - 486.

Fleischmann, B., 1990. The discrete lot-sizing and scheduling problem, *European Journal of Operational Research*, Vol. 44, pp. 337 - 348.

Fleischmann, B., 1992. The discrete lot-sizing and scheduling problem with sequence-dependent setup-costs, Working paper, University of Augsburg, to appear in *European Journal of Operational Research.*

Gascon, A. and R.C. Leachman, 1988. A dynamic programming solution to the dynamic, multi-item, single-machine scheduling problem, *Operations Research*, Vol. 36, pp. 50-56.

Lasdon L.S. and R.C. Terjung, 1971. An efficient algorithm for multi-item scheduling, *Operations Research*, Vol. 19, pp. 946-969.

Magnanti, T.L. and R. Vacchani, 1990. A strong cutting plane algorithm for production scheduling with changeover costs. *Operations Research*, Vol. 38, pp. 456-473.

Monma, C.L. and C.N. Potts, 1989. On the complexity of scheduling with batch setup-times, *Operations Research*, Vol. 37, pp. 798-804.

Park, M., R. Dattero and J.J.Kanet, 1993. Single machine batch scheduling with setup-times, Working paper, Florida Atlantic University, USA.

Potts, C.N. and L.N. van Wassenhove, 1992. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity, *Journal of the Operational Research Society*, Vol. 43, pp. 395-406.

Salomon, M., L.G. Kroon, R. Kuik and L.N. van Wassenhove, 1991. Some extensions of the discrete lotsizing and scheduling problem, *Management Science*, Vol. 37, pp. 801-812.

Unal, A. and A.S. Kiran, 1992. Batch sequencing, *IIE Transactions*, Vol. 24, pp. 73-83.

Wagner, H.M. and T.M. Whitin, 1958. Dynamic version of the economic lot size model, *Management Science*, Vol. 5, pp. 89 - 96.

Woodruff, D.L. and M.L. Spearman, 1992. Sequencing and batching for two classes of jobs with deadlines and setup-times, *Production and Operations Management*, Vol.1, pp. 87-102.