Jordan, Carsten; Drexl, Andreas

Discrete lotsizing and scheduling by batch sequencing

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 438

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

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 438

Discrete Lotsizing and Scheduling by Batch Sequencing

Jordan/Drexl

Manuskripte
aus den
Instituten für Betriebswirtschaftslehre
der Universität Kiel

No. 438

Discrete Lotsizing and Scheduling by Batch Sequencing

Jordan/Drexl

April 1997

to appear in *Management Science*

Carsten Jordan, Andreas Drexl, Lehrstuhl für Produktion und Logistik, Institut für Betriebswirtschaftslehre, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24118 Kiel, Germany
email: Jordan@bwl.uni-kiel.de
Drexl@bwl.uni-kiel.de
URL: http://www.wiso.uni-kiel.de/bwlinstitute/Prod
ftp://ftp.wiso.uni-kiel.de/pub/operations-research

Abstract: The discrete lotsizing and scheduling problem for one machine with sequence dependent setup times and setup costs is solved as a single machine scheduling problem, which we term the batch sequencing problem. The relationship between the lotsizing problem and the batch sequencing problem is analyzed. The batch sequencing problem is solved with a branch & bound algorithm which is accelerated by bounding and dominance rules. The algorithm is compared with recently published procedures for solving variants of the DLSP and is found to be more efficient if the number of items is not large.

KEYWORDS: DISCRETE LOTSIZING AND SCHEDULING, SEQUENCE DEPENDENT SETUP TIMES AND SETUP COSTS, BATCH SEQUENCING, BRANCH & BOUND ALGORITHM, BOUNDING/DOMINANCE RULES.

1 Introduction

In certain manufacturing systems a significant amount of setup is required to change production from one type of products to another, such as in the scheduling of production lines or in chemical engineering. Productivity can then be increased by batching in order to avoid setups. However, demand for different products arises at different points in time within the planning horizon. To satisfy dynamic demand, either large inventories must be kept if production is run with large batches or frequent setups are required if inventory levels are kept low. Significant setup times, which consume scarce production capacity, tend to further complicate the scheduling problem. The discrete lotsizing and scheduling problem (DLSP) is a well-known model for this situation.

In the DLSP, demand for each item is dynamic and back-logging is not allowed. Prior to each production run a setup is required. Setup costs and setup times depend on either the next item only (sequence independent), or on the sequence of items (sequence dependent). Production has to meet the present or future demand, and the latter case also incurs holding costs. The planning horizon is divided into a finite number of (short) periods. In each period at most one item can be produced, or a setup is made ("all or nothing production"). An optimal production schedule for the DLSP minimizes the sum of setup and holding costs.

The relationship between the DLSP and scheduling models in general motivated us to solve the DLSP as a batch sequencing problem (BSP). We derive BSP instances from DLSP instances and solve the DLSP as a BSP. Demand for an item is interpreted as a job with a deadline and a processing time. Jobs corresponding to demand for the same item are grouped into one family. Items in the DLSP are families in the BSP. All jobs must be processed on a single machine between time zero and their respective deadlines, while switching from a job in one family to a job in another family incurs a (sequence dependent) setup time and setup cost. Early completion of jobs is penalized by earliness costs which correspond to holding costs. As for the DLSP, an optimal schedule for the BSP minimizes the sum of setup and earliness costs.

The DLSP was first introduced by Lasdon and Terjung [13] with an application to production scheduling in a tire company. Complexity results for the DLSP and its extensions are examined in Salomon et al. [17], where the close relationship of the DLSP to job (class) scheduling problems is emphasized. A broader view on lotsizing and scheduling problems is given in Potts and Van Wassenhove [16]. An approach based on lagrangean relaxation is proposed by Fleischmann [7] for the DLSP without setup times. Fleischmann [8] utilizes ideas from solution procedures for vehicle routing problems to solve the DLSP with sequence dependent setup costs. The DLSP with sequence independent setup times and setup costs is examined by Cattrysse et al. [4]. In a recent work, Salomon et al. [18] propose a dynamic programming

based approach for solving the DLSP with sequence dependent setup times and setup costs to optimality. The results of [4], [8] and [18] will serve as a benchmark for our approach for solving the BSP.

Haase and Kimms [12] present a new mathematical model formulation for lotsizing and scheduling with sequence dependent setup times and costs which considers only efficient sequences. In addition, they provide a branch & bound algorithm which solves instances optimally and efficiently. A local search algorithm for lotsizing and scheduling with sequence dependent setup costs is presented in Haase [11]. A review of recent lotsizing and scheduling research can be found in Drexl and Kimms [5]. Schutten et al. [20] present an exact branch-and-bound algorithm for single-machine scheduling with release dates, due dates, family setup times, and maximum lateness as objective, respectively.

The complexity of scheduling problems with batch setup times is investigated by Bruno and Downey [2] and Monma and Potts [15]. Bruno and Downey show the feasibility problem to be NP-hard if setup times are nonzero. Solution procedures for scheduling problems with batch setup times are studied in Unal and Kiran [21], Ahn and Hyun [1] and Mason and Anderson [14]. In [21] the feasibility problem of the BSP is addressed and an effective heuristic is proposed. In [1] and [14], algorithms to minimize mean flow time are proposed. Webster and Baker [24] survey recent results and derive properties of optimal schedules for various batching problems.

The contribution of the paper is twofold. First, we solve the DLSP as a BSP and state the equivalence between both models such that we can solve either the DLSP or the BSP. Second, we present an algorithm that solves the BSP faster than known procedures solving the DLSP.

The paper is organized as follows: we present the DLSP and the BSP in Section 2 and provide a numerical example in Section 3. The relationship between the two models is analyzed in Section 4. Section 5 presents a timetabling procedure to convert a sequence into a minimum cost schedule, and in Section 6 we describe a branch & bound algorithm for solving the BSP. A comparison of our algorithm with solution procedures solving variants of the DLSP is found in Section 7. Summary and conclusions follow in Section 8.

2 Model Formulations

The DLSP is presented with sequence dependent setup times and setup costs, we refer to this problem as SDSTSC. SDSTSC includes the DLSP with sequence independent setups (SISTSC), sequence dependent setup costs but zero setup times (SDSC), and the generic DLSP with sequence independent setup costs and zero setup times (cf. Fleischmann [7]) as special cases.

The DLSP parameters are given in Table 1. Items (families) in the DLSP (BSP) are indexed by i, and h_i denotes holding costs per unit of item i and period. Production has to fulfill the demand $q_{i,t}$ for item i in period t. Setup costs $sc_{g,i}$ are "distributed" over $\max\{1, st_{g,i}\}$ setup periods by defining per-period setup costs $sc^p_{g,i}$. The decision variables are given in Table 2: we set $Y_{i,t} = 1$ if production takes place for item i in period t. $V_{g,i,t} = 1$ indicates a setup from item g to item i in period t, and $I_{i,t}$ denotes the inventory of item i at the end of period t.

In the mixed binary formulation of Table 3, the objective (1) minimizes the sum of setup costs $sc^p_{g,i}$ (per setup period $st_{g,i}$) and inventory holding costs. Constraints (2) express the inventory balance. The "all or nothing production" is enforced by constraints (3): in each period, the machine either produces at full unit capacity, undergoes setup for an item, or is idle, i.e. $Y_{0,t} = 1$ for an idle period. For $st_{g,i} = 0$

constraints (4) instantiate $V_{g,i,t}$ appropriately. Constraints (5) couple setup and production whenever $st_{g,i} > 0$: if item i is produced in period t and item g in period $t-\tau-1$ then the decision variable $V_{g,i,t-\tau} = 1$ for $\tau = t - st_{g,i}, \ldots, t-1$. Constraints (6) ensure, that the $Y_{i,t}$ are in correct positions relative to one another, therefore, we have to set $Y_{0,\tau} = 1$ for $\tau \leq 0$ in (10). Constraints (7) prevent any back-logging. Finally, the variables $I_{i,\tau}$, $V_{g,i,\tau}$, and $Y_{i,\tau}$ are initialized for $\tau \leq 0$, by constraints (10). Due to the "all or nothing production", we can write down a DLSP schedule in terms of a period-item assignment in a string $\nu = (\nu_1, \nu_2, \ldots, \nu_T)$. ν specifies the action in each period, i.e. $\nu_t = i$, if $Y_{i,t} = 1$, $i = 0, \ldots, N$, or $\nu_t = \mathrm{a}$, if $V_{g,i,t} = 1$ (for $st_{g,i} > 0$).

Table 1: *Parameters of the DLSP*

i	index of item (=family), $i = 0, \ldots, N$, 0 denotes the idle machine
t	index of periods, $t = 1, \ldots, T$
$q_{i,t}$	demand of item i in period t
h_i	holding costs per unit and period of item i
$st_{g,i}$	setup time from item g to item i, $g, i = 0, \ldots, N$
$sc^p_{g,i}$	setup costs per setup period from item g to item i, $g, i = 0, \ldots, N$
$sc_{g,i}$	setup costs from item g to item i, $g, i = 0, \ldots, N$
	$sc_{g,i} = sc^p_{g,i} \max\{1, st_{g,i}\}$

Table 2: *Decision Variables of the DLSP*

$Y_{i,t}$	1, if item i is produced in period t, and 0 otherwise.
	$Y_{0,t} = 1$ denotes idle time in period t
$V_{g,i,t}$	1, if the machine is setup for item i in period t, while the previous item
	was item g, and 0 otherwise
$I_{i,t}$	inventory of item i at the end of period t

The BSP is a family scheduling problem (cf. e.g. Webster and Baker [24]). Parameters (cf. Table 4) related to the N families are the index i, the number of jobs n_i in each family, and the total number of jobs J. Index $i = 0$ denotes the idle machine. As for the DLSP, holding costs h_i represent the costs for holding one unit of family i in inventory for one period of time. Setup times $st_{g,i}$ and setup costs $sc_{g,i}$ are given for each pair of families g and i. The set of jobs is partitioned into families i, the j-th job of family i is indexed by the tuple (i, j). Associated with each job (i, j) are a processing time $p_{(i,j)}$, a deadline $d_{(i,j)}$, and a weight $w_{(i,j)}$. Job weights $w_{(i,j)}$ are proportional to the quantity (=processing time) of the job (*proportional weights*), they are derived from h_i and $p_{(i,j)}$. We put the tuple in brackets to index the job attributes because the tuple denotes a job as *one* entity.

Table 3: *Model of the DLSP*

$$\text{Min} \quad Z_{DLSP} = \sum_{i=1}^{N}\sum_{t=1}^{T}\left(\sum_{g=0}^{N} sc^{p}_{g,i} V_{g,i,t} + h_i I_{i,t}\right) \tag{1}$$

subject to

$$I_{i,t-1} + Y_{i,t} - q_{i,t} = I_{i,t} \qquad i = 1,\ldots,N; \quad t = 1,\ldots,T \tag{2}$$

$$\sum_{i=0}^{N} Y_{i,t} + \sum_{\{g,i|st_{g,i}>0, g\neq i\}} V_{g,i,t} = 1 \qquad t = 1,\ldots,T \tag{3}$$

$$V_{g,i,t} \geq Y_{g,t-1} + Y_{i,t} - 1 \qquad \begin{cases} g = 0,\ldots,N; \quad i = 1,\ldots,N; \quad g \neq i; \\ st_{g,i} = 0; \quad t = 1,\ldots,T; \end{cases} \tag{4}$$

$$V_{g,i,\tau} \geq Y_{i,t} + Y_{g,t-st_{g,i}-1} - 1 \qquad \begin{cases} g = 0,\ldots,N; \quad i = 1,\ldots,N; \quad g \neq i; \\ st_{g,i} > 0 \quad \tau = t - st_{g,i},\ldots,t-1; \\ t = 1,\ldots,T \end{cases} \tag{5}$$

$$\sum_{g=0}^{N} Y_{g,t-st_{g,i}-1} \geq Y_{i,t} \qquad \begin{cases} g = 0,\ldots,N; \quad i = 1,\ldots,N; \\ st_{g,i} > 0; \quad t = 1,\ldots,T \end{cases} \tag{6}$$

$$I_{i,t} \geq 0 \qquad i = 1,\ldots,N; \quad t = 1,\ldots,T \tag{7}$$

$$V_{g,i,t} \in \{0,1\} \qquad \begin{cases} g = 0,\ldots,N; \quad i = 1,\ldots,N; \quad g \neq i; \\ t = 1,\ldots,T \end{cases} \tag{8}$$

$$Y_{i,t} \in \{0,1\} \qquad i = 0,\ldots,N; \quad t = 1,\ldots,T \tag{9}$$

$$I_{i,\tau} = V_{g,i,\tau} = 0; Y_{0,\tau} = 1 \qquad g,i = 0,\ldots,N; \quad \tau \leq 0 \tag{10}$$

The decision variables are given in Table 5. The *sequence* π denotes the processing order of the jobs, $(i_{[k]}, j_{[k]})$ denotes the job at position k; together with completion times $C_{(i,j)}$ of each job we obtain the *schedule* σ. A conceptual model formulation for the BSP is presented in Table 6. $Z_{BSP}(\sigma)$ denotes the sum of earliness and setup costs for a schedule σ, which is minimized by the objective (11). The earliness $d_{(i,j)} - C_{(i,j)}$ is weighted by $w_{(i,j)}$, and setup costs $sc_{i_{[k-1]},i_{[k]}}$ are incurred between jobs of different families. Each job is to be scheduled between time zero and its deadline, while respecting the sequence on the machine as well as the setup times. This is done by constraints (12). Constraints (13) set P_k equal to one if there is idle time between two consecutive jobs. We then have a setup time $st_{0,i_{[k]}}$ from the idle machine rather than a sequence dependent setup time $st_{i_{[k-1]},i_{[k]}}$. Initializations of beginning and end of the schedule are given in (15) and (16), respectively.

Remark 1 For the BSP and DLSP parameters we assume that:

1. setup times and setup costs satisfy the triangle inequality, i.e. $st_{g,i} \leq st_{g,l} + st_{l,i}$ and $sc_{g,i} \leq sc_{g,l} + sc_{l,i}$, $g,i,l = 0,\ldots,N$.

Table 4: *Parameters of the BSP*

n_i	number of jobs of family i, $J = \sum_{i=1}^{N} n_i$: total number of jobs
(i,j)	denotes the j-th job of family i, $i = 1,\ldots,N$, $j = 1,\ldots,n_i$
$p_{(i,j)}$	processing time for the j-th job of family i
$d_{(i,j)}$	deadline for the j-th job of family i
$w_{(i,j)}$	earliness weight per unit time for the j-th job of family i
B	big number

Table 5: *Decision variables of the BSP*

π	sequence of all jobs, $\pi = ((i_{[1]},j_{[1]}),(i_{[2]},j_{[2]}),\ldots,(i_{[k]},j_{[k]}),\ldots,(i_{[J]},j_{[J]}))$
$(i_{[k]},j_{[k]})$	denotes the job at position k
$C_{(i,j)}$	completion time of job (i,j)
P_k	1, if there is idle time between the jobs $(i_{[k-1]},j_{[k-1]})$ and $(i_{[k]},j_{[k]})$, and 0, otherwise

Table 6: *Model of the BSP*

$$\text{Min} \quad Z_{BSP}(\sigma) = \sum_{k=1}^{J} \left[w_{(i_{[k]},j_{[k]})} (d_{(i_{[k]},j_{[k]})} - C_{(i_{[k]},j_{[k]})}) + sc_{i_{[k-1]},i_{[k]}} + P_k (sc_{0,i_{[k]}} - sc_{i_{[k-1]},i_{[k]}}) \right] \tag{11}$$

subject to

$$C_{(i_{[k-1]},j_{[k-1]})} \leq \min \left\{ C_{(i_{[k]},j_{[k]})} - p_{(i_{[k]},j_{[k]})} - st_{i_{[k-1]},i_{[k]}} - P_k (st_{0,i_{[k]}} - st_{i_{[k-1]},i_{[k]}}), d_{(i_{[k-1]},j_{[k-1]})} \right\}$$
$$k = 1,\ldots,J \tag{12}$$

$$B \cdot P_k - (C_{(i_{[k]},j_{[k]})} - p_{(i_{[k]},j_{[k]})} - st_{i_{[k-1]},i_{[k]}} - C_{(i_{[k-1]},j_{[k-1]})}) \geq 0 \qquad k = 1,\ldots,J \tag{13}$$

$$P_k \in \{0,1\} \qquad k = 1,\ldots,J \tag{14}$$

$$(i_{[0]},j_{[0]}) = (0,0);\ d_{(0,0)} = C_{(0,0)} = 0 \tag{15}$$

$$C_{(i_{[J]},j_{[J]})} = d_{(i_{[J]},j_{[J]})} \tag{16}$$

2. there are no setups within a family, i.e. $st_{i,i} = sc_{i,i} = 0$, and no tear-down times and costs, i.e. $st_{i,0} = sc_{i,0} = 0$, $i = 0, \ldots, N$.

3. "Longer" setup times lead to "higher" setup costs, i.e. $sc_{g,i} = f(st_{g,i})$ with a nondecreasing function $f(\cdot)$.

4. there is binary demand in the DLSP, i.e. $q_{i,t} \in \{0, 1\}$.

5. jobs of one family are labeled in order of increasing deadlines, and deadlines do not interfere, i.e. $d_{(i,j)} + p_{(i,j+1)} \leq d_{(i,j+1)}$.

Remark 1 states in (1.) that it is not beneficial to perform two setups in order to accomplish one. Mason and Anderson [14] show that problems with nonzero tear-downs can easily be converted into problems with sequence dependent setups and zero tear-downs, which motivates (2.). With (1.) and (2.) we have $st_{0,i} \geq st_{g,i}$ (for all $g, i = 1, \ldots, N$), which holds analogously for setup costs. Thus, the third term in the objective (11) is always nonnegative. In (3.) we exclude the case that shorter setup times may have higher setup costs. anticipates the "all or nothing production" for each item i (cf. also Salomon et al. [17]) and is basically the same assumption as (5.).

The main observation that motivated us to consider the DLSP as a special case of the BSP is that the $(q_{i,t})$-matrix is sparse, especially if setup times are significant. The basic idea is to interpret *items* in the DLSP as *families* in the BSP and to regard *nonzero demand* in the DLSP as *jobs* with a deadline and a processing time in the BSP. In order to solve the DLSP as a special case of the BSP we derive BSP instances from DLSP instances in the following way: setup times and setup costs in the BSP and DLSP are identical, and the job attributes of the BSP instances are derived from the $(q_{i,t})$-matrix by Definitions 1 and 2.

Definition 1 *BSPUT(DLSP) is defined as a BSP instance with unit time jobs derived from a DLSP instance. For each family i there are $n_i = \sum_{t=1}^{T} q_{i,t}$ jobs. An entry $q_{i,t} = 1$ $(i = 1, \ldots, N,\ t = 1, \ldots, T)$ denotes a job (i, j) where $p_{(i,j)} = 1$, $w_{(i,j)} = h_i$, and $d_{(i,j)} = t$ $(j = 1, \ldots, n_i)$.*

Definition 2 *BSP(DLSP) is defined as a BSP instance derived from a DLSP instance. A sequence of consecutive "ones" in the $(q_{i,t})$-matrix, i.e. $q_{i,t} = 1$ $(t = t_1, \ldots, t_2)$ denotes a job (i, j) where $p_{(i,j)} = t_2 - t_1 + 1$, $w_{(i,j)} = h_i p_{(i,j)}$, and $d_{(i,j)} = t_2$, $j = 1, \ldots, n_i$. The number of times that a sequence of consecutive ones appears for an item i defines n_i.*

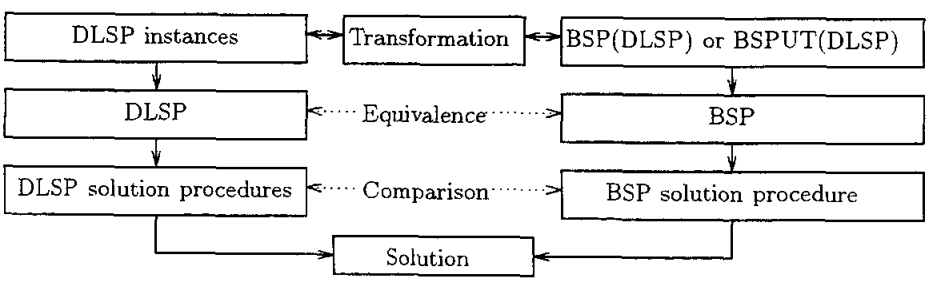

Figure 1: *Comparison of DLSP and BSP*

Figure 1 provides the framework for the BSP-DLSP comparison: after transforming DLSP instances into BSP instances, we compare the performance of solution procedures and the quality of the solutions. The difference between the approaches is as follows: in the DLSP, decisions are made anew in each individual period t, represented by decision variables $Y_{i,t}$ and $V_{g,i,t}$ (cf. Table 2). In the BSP, we decide how to schedule jobs, i.e. we decide about the completion times of the jobs. BSP and DLSP address the same underlying planning problem, but use different decision variables. Brüggemann and Jahnke [3] make another observation which concerns the transformation of instances: a DLSP instance may be *not* polynomially bounded in size while the size of the BSP(DLSP) instance is polynomially bounded. On that account, in [3] it is argued, that the $(q_{i,t})$-matrix is not a "reasonable" encoding for a DLSP instance in the sense of Garey and Johnson [10] because BSP(DLSP) describes a problem instance in a more concise way.

3 Numerical Example

In this section, we provide an example illustrating the generation of BSPUT(DLSP) and BSP(DLSP). This example will also be used to demonstrate certain properties of the BSP.

Table 7: *Numerical Example: Setup and Holding Costs*

$st_{g,i}$	1	2	3	$sc_{g,i}$	1	2	3	h_i
0	1	2	1	0	5	10	5	
1	0	1	1	1	0	5	5	1
2	0	0	1	2	0	0	5	3(1)
3	1	2	0	3	5	10	0	1

In Figure 2 we illustrate the equivalence between both models. The corresponding parameters setup times, setup costs and holding costs are given in Table 7. Figure 2 shows the demand matrix $(q_{i,t})$ of DLSP and the jobs at their respective deadlines of BSPUT(DLSP) and BSP(DLSP).

For BSPUT(DLSP), we interpret each entry of "one" as a job (i,j) with a deadline $d_{(i,j)}$. Processing times $p_{(i,j)}$ are equal to one for all jobs. We summarize the BSPUT(DLSP) parameters in Table 8. An optimal DLSP schedule with $h_2 = 3$ is the string ν^a in Figure 2 (with entries $\{0, \mathrm{a}, 1, 2, 3\}$ for idle or setup time or for production of the different items, respectively). This schedule is represented by σ_a^{ut} for BSPUT(DLSP), and is displayed in Table 8. Both schedules have an optimal objective function value of $Z_{BSP}(\sigma_a^{ut}) = 44$.

In BSP(DLSP), consecutive "ones" in the demand matrix $(q_{i,t})$ are linked to one job. The number of jobs is thus smaller in BSP(DLSP) than in BSPUT(DLSP). For instance, jobs $(1,2)$ and $(1,3)$ in BSPUT(DLSP) are linked to one job $(1,2)$ in BSP(DLSP), compare σ_b^{ut} and σ_b. However, a BSP(DLSP) schedule cannot represent σ_a^{ut} in Figure 2 since there we need unit time jobs. For BSP(DLSP) we now let the cost parameters $h_i = 1$, $(i = 1, 2, 3)$ (i.e. equal holding costs for all families) and $sc_{0,3} = 10$: then, ν^b is the optimal DLSP schedule and σ_b the optimal BSP(DLSP) schedule. Again, the optimal objective

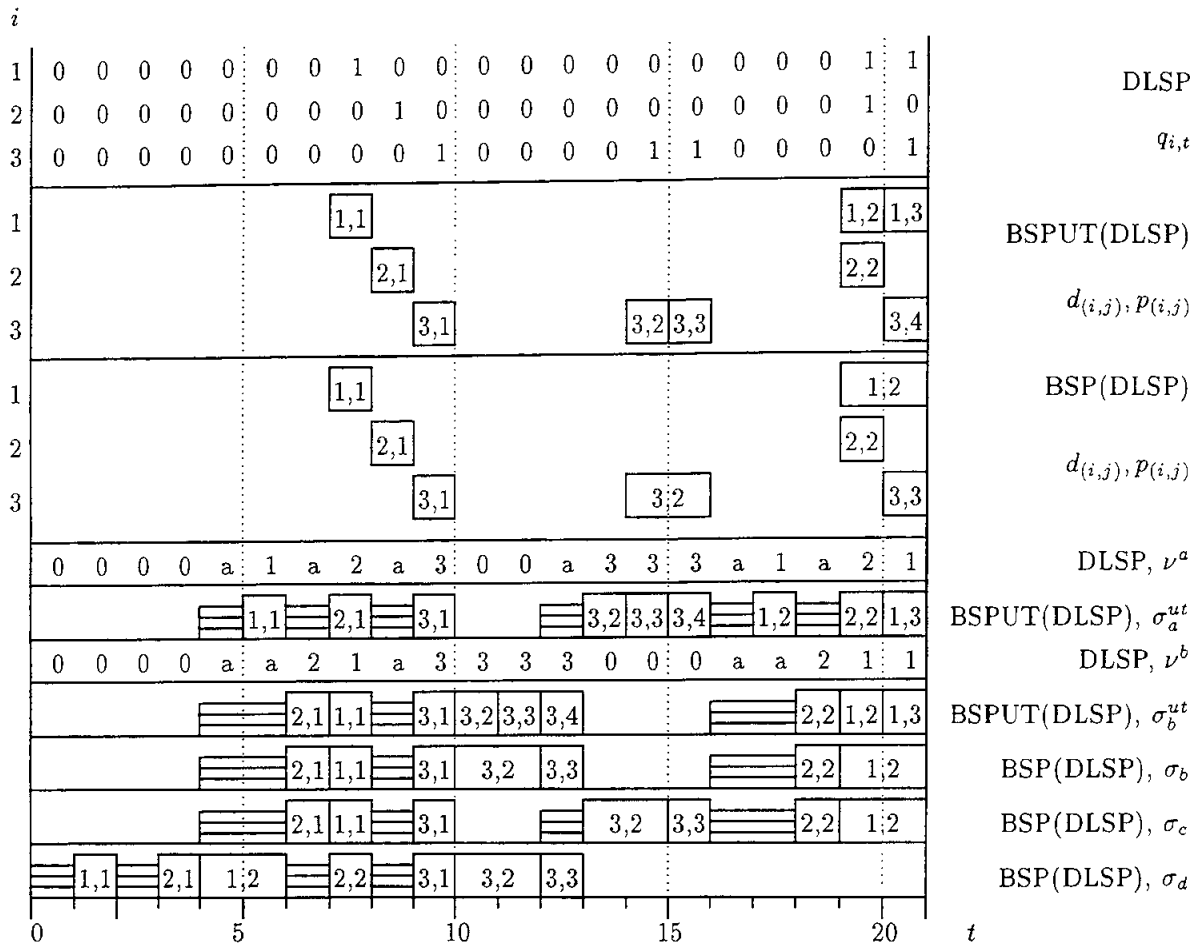

Figure 2: *DLSP, BSPUT(DLSP) and BSP(DLSP)*

function value is $Z_{BSP}(\sigma_b) = 44$ for ν^b and σ_b.

The example shows that the same schedules can be obtained from different models. In the next section we formally analyze the equivalence between the DLSP and the BSP.

4 Equivalence Between DLSP and BSP

In the BSP we distinguish between sequence and schedule. A BSP schedule may have inserted idle time so that the processing order does not (fully) describe a schedule. In the following we will say that job (i^1, j^1) is *consecutively sequenced* before job (i^2, j^2) if job (i^2, j^2) is sequenced at the *next* position. If we *consecutively schedule* job (i^1, j^1) before (i^2, j^2), there is no idle time between both jobs, i.e. the term in brackets in constraints (13) equals zero. A sequence π for the BSP consists of groups, where a *group* is an (ordered) set of consecutively sequenced jobs which belong to the same family. On the other hand, a schedule consists of (one or several) *blocks*. Jobs in one block are consecutively scheduled, different blocks are separated by idle time (to distinguish from setup time). Jobs in one block may belong to different families, and both block and group may consist of a single job. As an example refer to Figure 2 where

Table 8: *BSPUT(DLSP) Instance and Solution*

(i,j)	$(1,1)$	$(1,2)$	$(1,3)$	$(2,1)$	$(2,2)$	$(3,1)$	$(3,2)$	$(3,3)$	$(3,4)$	
$d_{(i,j)}$	8	20	21	9	20	10	15	16	21	
$p_{(i,j)}$	1	1	1	1	1	1	1	1	1	
$w_{(i,j)}$	1	1	1	3	3	1	1	1	1	
k	1	2	3	4	5	6	7	8	9	
$(i_{[k]}, j_{[k]})$	$(1,1)$	$(2,1)$	$(3,1)$	$(3,2)$	$(3,3)$	$(3,4)$	$(1,2)$	$(2,2)$	$(1,3)$	
$C_{(i_{[k]},j_{[k]})}$	6	8	10	14	15	16	18	20	21	σ_a^{ut}

both σ_c and σ_d consist of five groups, σ_c forms two blocks, and σ_d is only one block.

For a given sequence π, a BSP schedule $\hat{\sigma}$ is called *semiactive* if $C_{(i,j)}$ is constrained by either $d_{(i,j)}$ or the start of the next job; no job can be scheduled later or *rightshifted* in a semiactive schedule $\hat{\sigma}$. We can derive $\hat{\sigma}$ from a sequence π if constraints (12) are equalities and P_k is set to zero. The costs $Z_{BSP}(\hat{\sigma})$ are a lower bound for costs $Z_{BSP}(\sigma)$ of a BSP schedule σ because $\hat{\sigma}$ is the optimal schedule of the relaxed BSP in which constraints (13) are omitted. However, in the semiactive schedule there may be idle time and it may be beneficial to schedule some jobs earlier, i.e. to *leftshift* some jobs to save setups (which will be our concern in the timetabling procedure in Section 5).

In both models we save setups by batching jobs. In the DLSP, a *batch* is a non-interrupted sequence of periods where production takes place for the same item $i \neq 0$, i.e. $Y_{i,t} = 1, t = t_1, \ldots, t_2$. In the BSP, jobs of one group which are consecutively scheduled without a setup are in the same *batch*. A batch must not be preempted by idle time. In Figure 2, the group of family 3 forms two batches in schedule σ_c whereas this group is one batch in σ_b.

We will call a sequence π (schedule σ) an *EDDWF* sequence (schedule) if jobs of one family are sequenced (scheduled) in nondecreasing order of their deadlines (where EDDWF abbreviates earliest deadline within families). Ordering the jobs in EDDWF is called *ordered batch scheduling problem* in Monma and Potts [15]. By considering only EDDWF sequences, we reduce the search space for the branch & bound algorithm described in Section 6.

We first consider BSPUT(DLSP) instances. The following theorem states, that for BSPUT(DLSP) we can restrict ourselves to EDDWF sequences.

Theorem 1 *Any BSPUT(DLSP) schedule σ can be converted into an EDDWF schedule $\tilde{\sigma}$ with the same cost.*

Proof: Recall that jobs of one family all have the same weights and processing times. In a schedule σ, let A, B, C represent parts of σ (consisting of several jobs), and C_A, C_B, C_C (p_A, p_B, p_C) the completion (processing) times of the parts. Consider a schedule where jobs are not ordered in EDDWF, i.e. $\sigma = (C_A, C_{(i,j_2)}, C_B, C_{(i,j_1)}, C_C)$. Thus $C_{(i,j_2)} < C_{(i,j_1)} \leq d_{(i,j_1)} < d_{(i,j_2)}$. The schedule $\tilde{\sigma} = (C_A, \tilde{C}_{(i,j_1)}, C_B, \tilde{C}_{(i,j_2)}, C_C)$ with $\tilde{C}_{(i,j_1)} = C_{(i,j_2)}$, $\tilde{C}_{(i,j_2)} = C_{(i,j_1)}$ has the same objective function value because $w_{(i,j_1)} = w_{(i,j_2)} = h_i$. The completion times of the parts A, B, C do not change because

$p_{(i,j_1)} = p_{(i,j_2)} = 1$. Interchanging jobs can be repeated until σ is an EDDWF schedule, completing the proof. □

A DLSP schedule ν and a BSPUT(DLSP) schedule σ are called *corresponding solutions* if they define the same decision. A schedule $\nu = (\nu_1, \nu_2, \ldots, \nu_T)$ and a schedule σ are corresponding solutions if for each point in time $t = 1, \ldots, T$ the following holds: *(i)* $\nu_t = i$ and in σ the job being processed at t belongs to family i, *(ii)* $\nu_t =$ a and a setup is performed in σ, and *(iii)* $\nu_t = 0$ and the machine is idle in σ.

Figure 2 gives an example for corresponding solutions: ν^a corresponds to σ_a^{ut}, and ν^b corresponds to σ_b^{ut}. We can always derive entries in ν from σ, and completion times in σ can always be derived from ν if σ is an EDDWF schedule.

Theorem 2 *A schedule σ is feasible for BSPUT(DLSP) if and only if the corresponding solution ν is feasible for DLSP, and ν and σ have the same objective function value.*

Proof: Obvious. □

As a consequence of Theorem 2, a schedule σ is optimal for BSPUT(DLSP) if and only if the corresponding solution ν is optimal for DLSP, which constitutes the equivalence between DLSP and BSP for BSPUT(DLSP) instances. We can thus solve DLSP by solving BSPUT(DLSP).

In general, however, the more attractive option will be to solve BSP(DLSP) because the number of jobs is smaller.

Definition 3 *In a schedule σ, let a production start of family i be the start time of the first job in a batch. Let inventory for family i build between $C_{(i,j)}$ and $d_{(i,j)}$. The schedule σ is called regenerative if there is no production start for a family i as long as there is still inventory for family i.*

The term "regenerative" stems from the regeneration property found by Wagner and Whitin [23] (for similar ideas cf. e.g. Vickson et al. [22]). Each regenerative schedule is also an EDDWF schedule, but the reverse is not true. If a schedule σ is regenerative, jobs (i,j) and $(i,j+1)$ are in the same batch if $C_{(i,j+1)} - p_{(i,j+1)} \leq d_{(i,j)}$ holds. Furthermore, in a regenerative BSPUT(DLSP) schedule σ, jobs from consecutive "ones" in $(q_{i,t})$ are scheduled consecutively (recall for instance σ_b^{ut} and σ_b in Figure 2); hence a regenerative BSPUT(DLSP) schedule represents a BSP(DLSP) schedule as well. In Figure 2, schedule σ_d is not regenerative: a batch for family $i = 1$ is started at $t = 4$ though there is still inventory for $i = 1$.

We first show that we do not lose feasibility when restricting ourselves to regenerative schedules only.

Theorem 3 *If σ is a feasible BSPUT(DLSP) or BSP(DLSP) schedule then there is also a feasible regenerative schedule $\tilde{\sigma}$.*

Proof: In a schedule σ, let i^B (i^A) be the family to which the first (last) job in part B (A) belongs. Consider a non-regenerative schedule σ, i.e. $\sigma = (C_A, C_{(i,j)}, C_B, C_{(i,j+1)}, C_C)$. Jobs (i,j) and $(i,j+1)$ are not in one batch though $C_{(i,j+1)} - p_{(i,j+1)} \leq d_{(i,j)}$.

Consider schedule $\tilde{\sigma} = (C_A, \tilde{C}_B, \tilde{C}_{(i,j)}, C_{(i,j+1)}, C_C)$ with $\tilde{C}_{(i,j)} = C_{(i,j+1)} - p_{(i,j+1)}$ and $\tilde{C}_B = C_B - p_{(i,j)}$, where (i,j) and B are interchanged and (i,j) and $(i,j+1)$ are in one batch. $\tilde{\sigma}$ is feasible because $\tilde{C}_{(i,j)} \leq d_{(i,j)}$. By leftshifting B we do not violate feasibility. Furthermore, due to the triangle inequality we have $st_{i^A,i^B} \leq st_{i^A,i} + st_{i,i^B}$. Thus, B can be leftshifted by $p_{(i,j)}$ time units without affecting C_A. Interchanging jobs can be repeated until σ is regenerative which proves the theorem. □

An illustration for the construction of regenerative schedules is depicted in Figure 3. Interchanging (i,j) and B, we obtain from σ the regenerative schedule $\tilde{\sigma}$.

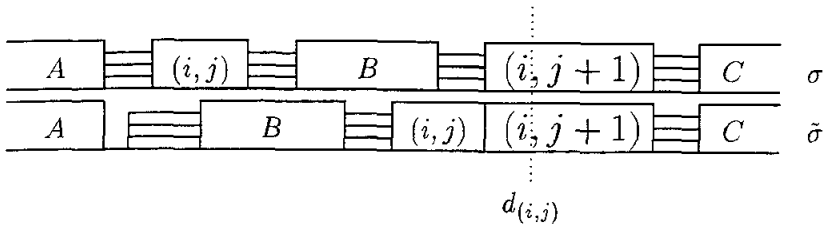

Figure 3: *Regenerative Schedule*

Unit processing times are not needed for the proof of Theorem 3, so we have in fact two results: first, to find a feasible schedule we may consider BSP(DLSP) instead of BSPUT(DLSP). Second, for BSP(DLSP) we only need to search over regenerative schedules to find a feasible schedule. Theorem 3 is a stronger result than the one found by Salomon et al. [17] and Unal and Kiran [21] who only state the first result. Moreover, if holding costs are equal, the next theorem extends this result to optimal schedules.

Theorem 4 *If σ is an optimal BSPUT(DLSP) or BSP(DLSP) schedule and h_i is constant for all i, then there is also an optimal regenerative schedule $\tilde{\sigma}$.*

Proof: Analogous to the proof of Theorem 3 we now must consider the change of the objective function value if (i,j) and B are interchanged. Without loss of generality, let $h_i = 1$ for all i, then $p_{(i,j)} = w_{(i,j)}$. Let $Z_{BSP}(\sigma)$ ($Z_{BSP}(\tilde{\sigma})$) denote the objective function value of σ ($\tilde{\sigma}$).

For part B, which is leftshifted, we have $w_B \leq p_B$ because processing time in part B is at most p_B, but B may contain setups as well. Interchanging B and (i,j), the objective changes as follows:

$$Z_{BSP}(\tilde{\sigma}) = Z_{BSP}(\sigma) - w_B(\tilde{C}_B - C_B) - w_{(i,j)}(\tilde{C}_{(i,j)} - C_{(i,j)}) - sc_{i^A,i} - sc_{i,i^B} + sc_{i^A,i^B}$$

$$\overset{(i)}{\leq} Z_{BSP}(\sigma) + w_B p_{(i,j)} - w_{(i,j)} p_B \overset{(ii)}{\leq} Z_{BSP}(\sigma) + p_B p_{(i,j)} - p_{(i,j)} p_B = Z_{BSP}(\sigma)$$

Due to the triangle inequality, setup costs and setup times in σ are not larger than in $\tilde{\sigma}$, i.e. $-sc_{i^A,i} - sc_{i,i^B} + sc_{i^A,i^B} \leq 0$. Furthermore, if setup time is saved (as in Figure 3), we will not increase setup costs due to Remark 1, which explains (i). We leftshift B by $p_{(i,j)}$ and rightshift (i,j) by p_B with $w_B \leq p_B$, which explains (ii). Thus $Z_{BSP}(\tilde{\sigma}) \leq Z_{BSP}(\sigma)$, which proves the theorem. □

Considering regenerative schedules, we again achieve a considerable reduction of the search space. To summarize we have so far obtained the following results: *(i)* DLSP and BSP are equivalent for BSPUT(DLSP). *(ii)* BSP(DLSP) is feasible if and only if DLSP is feasible. *(iii)* For equal holding costs an optimal BSP(DLSP) schedule is optimal for DLSP. When instances with unequal holding costs are solved, the theoretical difference between BSP(DLSP) and DLSP in *(iii)* has only a small effect: computational results in Section 7.3 will show that there is almost always an optimal regenerative BSPUT(DLSP) schedule to be found by solving BSP(DLSP).

5 A Timetabling Procedure for a Given Sequence

For a given sequence π the following timetabling procedure decides how to partition π into blocks, or equivalently, which consecutively *sequenced* jobs should be consecutively *scheduled*. In the BSP model

formulation of Table 6, we have $P_k = 1$ if the job at position k starts a new block, or $P_k = 0$ if it is blocked with the preceding job. By starting a new block at position k, we save earliness costs at the expense of additional setup costs. In Figure 2, earliness costs of σ_b are higher than for σ_c but we save one setup in σ_b.

In the timetabling procedure, we start with the semiactive schedule and leftshift some of the jobs to find a minimum cost schedule. Consider the example in Figure 2: for the sequence $\pi = ((2,1),(1,1),(3,1),(3,2),(3,3),(2,2),(1,2))$ the semiactive schedule $\hat{\sigma}$ is given in Figure 4. We first consider two special cases. If we omit constraints (13) of the BSP (so that each group is a batch and idle time may preempt the batch) timetabling is trivial: the semiactive schedule is optimal for a given sequence because no job can be rightshifted to decrease earliness costs (and because setup costs are determined by π and not by σ). Timetabling is also trivial if earliness weights are zero (i.e. $h_i = 0$ for all i and therefore $w_{(i,j)} = 0$ for all (i,j)): in this case, we can leftshift each job (without increasing earliness costs) until the resulting schedule is one block (e.g. schedule σ_d in Figure 2 is one block and no job can be leftshifted). We have $sc_{g,i} \leq sc_{i,0} + sc_{0,i} = sc_{0,i}$, setup costs are minimized if jobs are scheduled in a block, and there is an optimal schedule which consists of one block.

In the general case, we need some definitions: block costs bc_{k_1,k_2} are the cost contribution of a block from position k_1 to k_2, i.e.

$$bc_{k_1,k_2} = \sum_{k=k_1}^{k_2} w_{(i_{[k]},j_{[k]})}(d_{(i_{[k]},j_{[k]})} - C_{(i_{[k]},j_{[k]})}) + \sum_{k=k_1+1}^{k_2} sc_{i_{[k-1]},i_{[k]}}.$$

The block size bs_k is the number of jobs which are consecutively scheduled after job $(i_{[k]}, j_{[k]})$ (with $(i_{[k]}, j_{[k]})$ included). For instance, in Figure 4 we have $bs_1 = 3$ (for $(i_{[1]}, j_{[1]}) = (2,1)$).

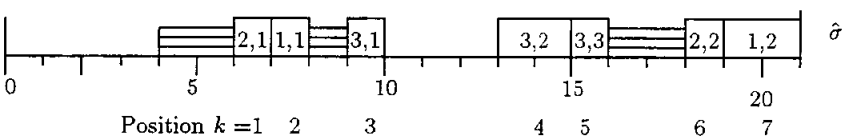

Figure 4: *Semiactive Schedule*

Let $f_k(b)$ denote the costs of a schedule from position k to J if $bs_k = b$. Let denote f_k^* the costs of the minimum cost schedule and bs_k^* the corresponding block size at position k. The recurrence equation for determining f_k^* and bs_k^* for $k = J, \ldots, 1$ is:

$$f_k^* = \min_{b=1,\ldots,bs_{k+1}^*+1} \{f_k(b)\} \tag{17}$$

$$f_k(bs_k^*) = f_k^*$$

$$f_k(b) = bc_{k,k+b-1} + sc_{0,i_{[k+b]}} + f_{k+b}^*$$

$$f_{J+1}^* = 0;\ i_{[J+1]} = 0;\ bs_{J+1}^* = 0 \tag{18}$$

In equation (17) we take the minimum cost for $bs_k = 1, \ldots, bs_{k+1}^* + 1$ where $bs_{k+1}^* + 1$ is the *maximum* block size at position k (and a new block starts at $k + bs_{k+1}^* + 1$). For a given block size b, $f_k(b)$ is the sum of block costs from position k to position $k + b - 1$, the setup $sc_{0,i_{[k+b]}}$ to the next block and the minimum cost f_{k+b}^*. Minimum cost of a sequence are computed via f_k^* for $k = 1$, and the complexity of

the algorithm is $\mathcal{O}(\mathcal{J}^6)$. Basically, equation (17) must be computed for every sequence. However, some simplifications are possible: if two jobs can be consecutively scheduled in the semiactive schedule, it is optimal to increment $bs_k^* = bs_{k+1}^* + 1$, because $sc_{g,i} \leq sc_{0,i}$, so that equation (17) needs not be evaluated. Consequently, if the semiactive schedule is one block, timetabling is again trivial: each group in $\hat{\sigma}$ equals one batch, the whole schedule forms a block, and $\hat{\sigma} = \sigma$.

If setups are sequence independent, a minimum cost schedule can be derived with less effort as follows: let the group size gs_k at position k denote the number of consecutively sequenced jobs at positions $r > k$ that belong to the same family as job $(i_{[k]}, j_{[k]})$. Then, for sequence independent setups, equation (17) must be evaluated only for $b = 1, \ldots, gs_k$. The reasoning is as follows: jobs of *different* groups are leftshifted to be blocked only if we can save setup cost. Then, for consecutive groups of families g and i we would need $sc_{g,i} < sc_{g,0} + sc_{0,i} = sc_{0,i}$, which does not hold for sequence independent setups; therefore, we only need to decide about the leftshift *within* a group.

An example for the computations of equation (17) is given in Table 9 for the semiactive schedule in Figure 4 (see the cost parameters in Table 7). The schedule $\hat{\sigma}$ contains idle time, and we determine f_k^* and bs_k^* for each position k, $k \leq J$.

Consider jobs (2,2), (3,3) and (3,2) at positions 6,5 and 4 in Figure 4. Up to position 4 the semiactive schedule is one block, and we increment bs_k^*, which is denoted by entries (-) in Table 9. After job (3,1), $\hat{\sigma}$ has inserted idle time between positions 3 and 4 ($d_{(3,1)} = 10$) and all different block sizes must be considered to find the minimum cost schedule. In Table 9, we find $f_3^* = f_3(1)$, i.e. we start a new block after position $k = 3$ and split the group into two batches, as done for σ_c in Figure 2. For the objective function value, we add $sc_{0,2}$ to f_1^* and obtain a cost of $Z_{BSP}(\sigma_c) = 40$.

6 Sequencing Algorithm

In this section we present a branch & bound algorithm for solving the BSP to optimality, denoted as SABSP. Jobs are sequenced backwards, i.e. at stage 1 a job is assigned to position J, at stage 2 to position $J - 1$, at stage s to position $J - s + 1$. An s-partial sequence π^s assigns s jobs to the last s positions of sequence π, in addition to that an s-partial schedule σ^s also assigns completion times to each job in π^s. A partial schedule ω^s is called completion of σ^s if ω^s extends σ^s to a schedule σ which schedules all jobs, and we write $\sigma = (\omega^s, \sigma^s)$.

We only examine EDDWF sequences, as if there were precedence constraints between the jobs. The precedence graph for the example in Figure 2 is shown in Figure 5. Using the EDDWF ordering, we decide in fact at each stage s which *family* to schedule. A job is eligible at stage s if all its (precedence related) predecessors are scheduled. An s-partial schedule (corresponding to a node in the search tree) is extended by scheduling an eligible job at stage $s+1$. We apply depth-first search in our enumeration and use the bounding, branching, and dominance rules described in Sections 6.1 and 6.2 to prune the search tree.

Each (s-partial) sequence π^s uniquely defines a minimum cost (s-partial) schedule σ^s by the timetabling procedure. Enumeration is done over all sequences and stops after all sequences have been (implicitly) examined; the best solution found is optimal. The implementation of SABSP takes advantage of the fact that equation (17) needs not be recalculated for every σ^s and, in the case of backtracking, the computation of equation (17) has already been accomplished for the partial schedule to which we backtrack.

Table 9: *Computations of Equation (17) for the Example in Figure 4*

$(i_{[k]}, j_{[k]})$	k	f_k^*	bs_k^*	b	$f_k(b) =$	$bc_{k,k+b-1}$	$+ sc_{0,i_{[k+b]}} + f_{k+b}^*$
$(0,0)$	8	0	0				
$(1,2)$	7	0	1	1	$0 =$	$0 + 0 + 0$	
$(2,2)$	6	-	-	1		-	
		1	2	2	$1 =$	$1 + 0 + 0$	
$(3,3)$	5	-	-	1		-	
		-	-	2		-	
		16	3	3	$16 =$	$(5+10+1) + 0 + 0$	
$(3,2)$	4	-	-	1		-	
		-	-	2		-	
		-	-	3		-	
		18	4	4	$18 =$	$(2+5+10+1) + 0 + 0$	
$(3,1)$	3	23	1	1	$23 =$	$0 + 5 + 18$	
				2	$29 =$	$8 + 5 + 16$	
				3	$27 =$	$(8+8) + 10 + 1$	
				4	$35 =$	$(8+8+10+4) + 5 + 0$	
				5	$36 =$	$(8+8+10+4+0+6) + 0 + 0$	
$(1,1)$	2	-	-	1		-	
		28	2	2	$28 =$	$5 + 5 + 18$	
$(2,1)$	1	-	-	1		-	
		-	-	2		-	
		30	3	3	$30 =$	$(2+5) + 5 + 18$	

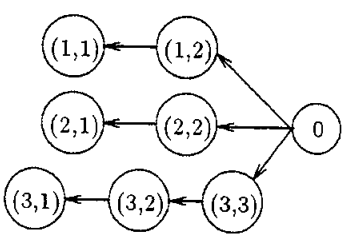

Figure 5: *EDDWF Precedence Graph for Backward Sequencing*

Table 10: *Attributes of Partial Schedules*

(i^s, j^s)	job under consideration at stage s
UB	upper bound, objective function value of the current best schedule
$t(\sigma^s)$	start time of σ^s, i.e. $C_{(i^s,j^s)} - p_{(i^s,j^s)}$
$c(\sigma^s)$	cost of σ^s without the setup for (i^s, j^s)
$\mathcal{AS}^s(\mathcal{US}^s)$	set of jobs already scheduled (unscheduled) in the s-partial schedule σ^s
$\mathcal{UI}^s$	set of families to which jobs in $\mathcal{US}^s$ belong to
$\mathcal{G}_1(\sigma^s)$	set of jobs which form the first block of σ^s
$w_1(\sigma^s)$	sum of earliness weights of jobs in $\mathcal{G}_1(\sigma^s)$, i.e.
	$w_1(\sigma^s) = \sum_{(i,j)\in\mathcal{G}_1(\sigma^s)} w_{(i,j)}$

Table 10 lists attributes of s-partial schedules. For each scheduling stage s we identify the job (i^s, j^s) under consideration, and the start time $t(\sigma^s)$ and costs $c(\sigma^s)$ of the s-partial schedule. The set of currently scheduled (unscheduled) jobs is denoted by $\mathcal{AS}^s$ ($\mathcal{US}^s$). $\mathcal{UI}^s$ denotes to which families the jobs in $\mathcal{US}^s$ belong; UB is the current upper bound.

6.1 Bounding and Branching Rules

The **feasibility bound** states that for a given σ^s, all currently unscheduled jobs in $\mathcal{US}^s$ must be scheduled between time zero and $t(\sigma^s)$ and, furthermore, we need a setup time for each family in $\mathcal{UI}^s$. More formally, define

$$\mathsf{T}^s = \sum_{i\in\mathcal{UI}^s} \min_{\substack{g=0,\ldots,N\\ g\neq i}} \{st_{g,i}\} + \sum_{(i,j)\in\mathcal{US}^s} p_{(i,j)}.$$

Then, σ^s has no feasible completion ω^s if $t(\sigma^s) - \mathsf{T}^s < 0$.

The **cost bound** states that cost $c(\sigma^s)$ of an s-partial schedule is a lower bound for all extensions of σ^s, and for any completion ω^s at least one setup for each family in $\mathcal{UI}^s$ must be performed. We define

$$\mathsf{C}^s = \sum_{i\in\mathcal{UI}^s} \min_{\substack{g=0,\ldots,N\\ g\neq i}} \{sc_{g,i}\}.$$

Then, σ^s cannot be extended to a schedule $\sigma = (\omega^s, \sigma^s)$ which improves UB if $\mathsf{C}^s + c(\sigma^s) \geq UB$.

Both bounds are checked for each s-partial schedule σ^s. Clearly, T^s and C^s can be easily updated during the search. We also tested a more sophisticated lower bound where all unscheduled jobs were scheduled in EDD order without setups. In this way we were able to derive a lower bound on the earliness costs as well and check feasibility more carefully, but computation times did not decrease.

If only regenerative schedules need to be considered to find the optimal schedule (cf. Theorem 4), we employ a **branching rule** as follows: scheduling (i^s, j^s) at stage s (with $j^s > 1$), job $(i^s, j^s - 1)$ becomes eligible in the EDDWF precedence graph. If $t(\sigma^s) \leq d_{(i^s,j^s-1)}$, we schedule $(i^s, j^s - 1)$ at stage $s+1$ (i.e. we batch (i^s, j^s) and $(i^s, j^s - 1)$) and do not consider any other job as an extension of σ^s. We need not

enumerate partial schedules where σ^s is extended by a job (g,j) where $g \neq i^s$ because then the resulting schedule is non-regenerative.

6.2 Dominance Rules

The most remarkable reduction of computation times comes as a result of the **dominance rules**. The dominance rules of SABSP compare two s-partial schedules σ^s and $\overline{\sigma}^s$, which schedule the same set of jobs, so that $\mathcal{AS}^s = \overline{\mathcal{AS}}^s$. In this notation, schedule $\overline{\sigma}^s$ denotes the s-partial schedule currently under consideration, while σ^s denotes a previously enumerated schedule which may dominate $\overline{\sigma}^s$. A partial schedule σ^s dominates $\overline{\sigma}^s$ if it is more *efficient* in terms of time and cost: σ^s starts later to schedule the job-set, i.e. $t(\sigma^s) \geq t(\overline{\sigma}^s)$, and σ^s incurs less cost, i.e. $c(\sigma^s) \leq c(\overline{\sigma}^s)$. If the family i^s of the job scheduled at stage s differs in σ^s and $\overline{\sigma}^s$ we make both partial schedules "comparable" with a setup from $\overline{i^s}$ to i^s: we compare time and cost but subtract setup times and setup costs appropriately.

If a schedule σ^s is not dominated, we store for the job set $\mathcal{AS}^s$ and family i^s the pair $t(\sigma^s)$ and $c(\sigma^s)$ which is "most likely" to dominate other s-partial schedules. Note that the number of partial schedules is exponential in the number of items N so that storage requirements for the dominance rules grow rapidly if N increases.

For a **formal description** of the dominance rules we need several definitions (cf. Table 10): all jobs which form a block with (i^s, j^s) belong to the set $\mathcal{G}_1(\sigma^s)$, and the sum of earliness weights in $\mathcal{G}_1(\sigma^s)$ is denoted as $w_1(\sigma^s)$. The dominance rules take into account the block costs for all extensions of σ^s and $\overline{\sigma}^s$: we consider for σ^s the maximum, for $\overline{\sigma}^s$ the minimum costs incurred by blocking; σ^s then dominates $\overline{\sigma}^s$ if $c(\sigma^s)$ plus an *upper* bound on block costs is less or equal $c(\overline{\sigma}^s)$ plus a *lower* bound on block costs.

An upper bound on the block costs for σ^s is given by sc_{0,i^s} (recall that $sc_{0,i} \geq sc_{g,i}$). Then, σ^s starts a new block. But a tighter upper bound can be found for start times close to $t(\sigma^s)$: in order to save costs we can leftshift all the jobs in $\mathcal{G}_1(\sigma^s)$ (but only these), because after $\mathcal{G}_1(\sigma^s)$ we perform a new setup from the idle machine. $\mathcal{G}_1(\sigma^s)$ is the *largest* block which may be leftshifted. Let $pbt(\sigma^s)$ denote the time where the cost increase due to a leftshift of $\mathcal{G}_1(\sigma^s)$ exceeds sc_{0,i^s}. We then have $w_1(\sigma^s)(t(\sigma^s) - pbt(\sigma^s)) = sc_{0,i^s}$ and define the pull-back-time $pbt(\sigma^s)$ of an s-partial schedule σ^s as follows:

$$pbt(\sigma^s) = t(\sigma^s) - \frac{sc_{0,i^s}}{w_1(\sigma^s)}.$$

Consequently, for time t, $pbt(\sigma^s) \leq t \leq t(\sigma^s)$, an upper bound on block costs is given by leftshifting $\mathcal{G}_1(\sigma^s)$; for $t < pbt(\sigma^s)$, block costs are bounded by sc_{0,i^s}. In this argumentation we implicitely assume, that holding inventory during setup time must not exceed setup costs.

A lower bound on the block costs for $\overline{\sigma}^s$ is given in the same way as for σ^s, but now we consider the *smallest* block that can be leftshifted, which is simply job $(\overline{i^s}, \overline{j^s})$.

We can now state the dominance rule: we differentiate between $i^s = \overline{i^s}$ (Theorem 5) and $i^s \neq \overline{i^s}$ (Theorem 6).

Theorem 5 *Consider two s-partial schedules σ^s and $\overline{\sigma}^s$ with $\mathcal{AS}^s = \overline{\mathcal{AS}}^s$ and $i^s = \overline{i^s}$. Let $\triangle^i = t(\sigma^s) - t(\overline{\sigma}^s)$ and $\triangle^{ii} = t(\overline{\sigma}^s) - pbt(\sigma^s)$. σ^s dominates $\overline{\sigma}^s$ if*

$$\begin{aligned} &(i) && t(\overline{\sigma}^s) \leq t(\sigma^s), \\ &(ii)-a && c(\overline{\sigma}^s) \geq c(\sigma^s) + \min\{\triangle^i w_1(\sigma^s), sc_{0,i^s}\} \quad \textit{for} \quad st_{0,i^s} \leq \triangle^i \end{aligned}$$

$$(ii)-b \qquad c(\overline{\sigma}^s) \geq c(\sigma^s) + \Delta^i w_1(\sigma^s) \qquad \textit{for} \quad st_{0,i^s} > \Delta^i \quad \textit{and}$$

$$(iii) \qquad c(\overline{\sigma}^s) + \min\{\Delta^{ii} w_{(i^s,j^s)}, sc_{0,i^s}\} \geq c(\sigma^s) + sc_{0,i^s}.$$

Proof: Any completion ω^s of $\overline{\sigma}^s$ is also a feasible completion of σ^s because of (i); if $(\omega^s, \overline{\sigma}^s)$ is feasible, (ω^s, σ^s) is feasible, too. Due to (ii), for any ω^s, the schedule (ω^s, σ^s) has lower costs than $(\omega^s, \overline{\sigma}^s)$.

In the following we consider the cost contributions of σ^s and $\overline{\sigma}^s$ due to leftshifting, when we extend σ^s and $\overline{\sigma}^s$. Consider Figure 6 for an illustration of the situation in a time-cost diagram. We have $i^s = \overline{i^s}$ and due to EDDWF also $(i^s, j^s) = (\overline{i^s}, \overline{j^s})$. The solid line represents the upper bound on block costs for σ^s. For $pbt(\sigma^s) \leq t \leq t(\sigma^s)$, it is less expensive to leftshift $\mathcal{G}_1(\sigma^s)$ (and extra costs are $\Delta^i w_1(\sigma^s)$), while for $t < pbt(\sigma^s)$ a setup from the idle machine to i^s is performed (with cost sc_{0,i^s}). However, in the case that $\Delta^i < st_{0,i}$ (see $(ii)-b$), we must leftshift $\mathcal{G}_1(\sigma^s)$ and blocking cost are not bounded by sc_{0,i^s}. The broken line represents the lower bound on block costs for $\overline{\sigma}^s$. The smallest block that can be leftshifted is the job (i^s, j^s).

In order to prove that $\overline{\sigma}^s$ will never have less costs than σ^s due to blocking, we check the costs at points (ii) and (iii): at (ii) (which is either $(ii)-a$ or $(ii)-b$), we compare the costs at $t(\overline{\sigma}^s)$ while at (iii) we compare them at $pbt(\sigma^s)$. Between (ii) and (iii) costs increase linearly, and for $t < pbt(\sigma^s)$ we know that there is a monotonous cost increase for $\overline{\sigma}^s$, while costs of σ^s no longer increase. Thus, if (ii) and (iii) are fulfilled, cost contributions of σ^s are less than those of $\overline{\sigma}^s$, i.e. there is no completion ω^s such that $Z_{BSP}(\omega^s, \overline{\sigma}^s) < Z_{BSP}(\omega^s, \sigma^s)$, completing the proof. □

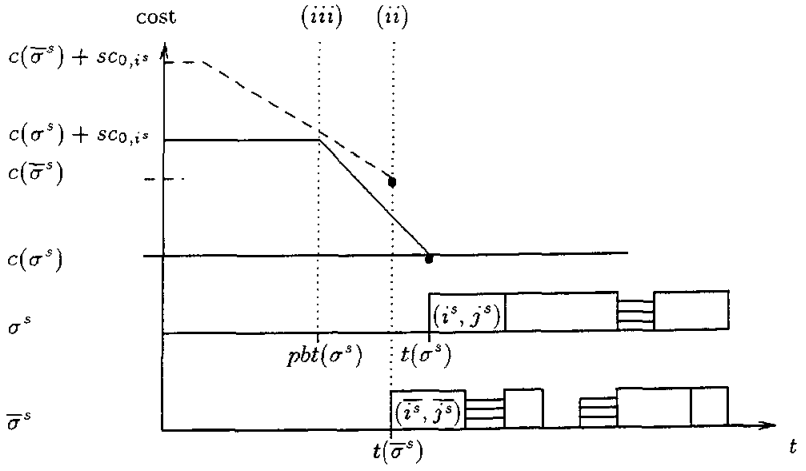

Figure 6: *Illustration of Theorem 5*

For the example in Figure 2, Figures 7 and 8 illustrate Theorem 5 with 3-partial schedules σ^3 and $\overline{\sigma}^3$. In Figure 7, we have $\mathcal{G}_1(\sigma^3) = \mathcal{G}_1(\overline{\sigma}^3) = \{(3,3),(2,2),(1,2)\}$, $t(\sigma^3) \geq t(\overline{\sigma}^3)$ and $pbt(\sigma^3) = 15 - 5/4 = 13.75$. Checking (ii), we have $22 \geq 16 + \min\{1 \cdot 4, 5\}$, while for (iii) we have $22 + \min\{0.25 \cdot 1, 5\} \geq 16 + 5$, so that $\overline{\sigma}^3$ is dominated. Figure 8 illustrates the effect of block costs, but with modified data as follows: $sc_{0,3} = 10$ and $d_{3,2} = 15$. Thus $pbt(\sigma^3) = 13 - 10/3$, and $\Delta^{ii} = 10/3$. Checking Theorem 5, we have (ii) $16 \geq 15 + \min\{0 \cdot 3, 5\}$, but (iii) $16 + \min\{10/3 \cdot 2, 10\} \geq 15 + 10$ is not fulfilled. Thus, σ^3 does *not* dominate $\overline{\sigma}^3$, though $c(\sigma^3) < c(\overline{\sigma}^3)$ and $t(\sigma^3) = t(\overline{\sigma}^3)$. Figure 8 shows that $c(\sigma^4) > c(\overline{\sigma}^4)$ if $\mathcal{G}_1(\sigma^4)$ is leftshifted.

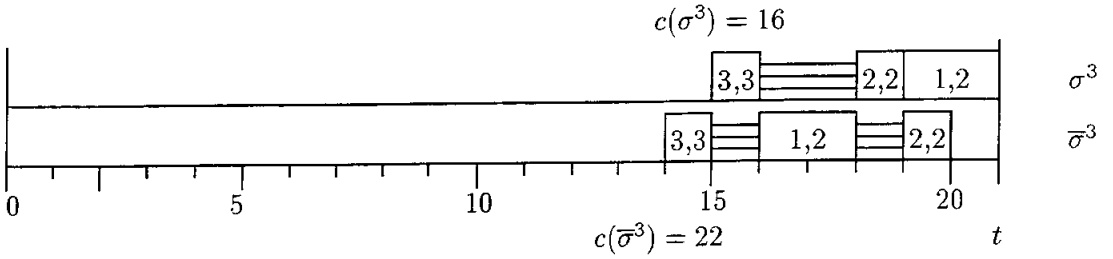

Figure 7: *Theorem 5: σ^3 dominates $\overline{\sigma}^3$*

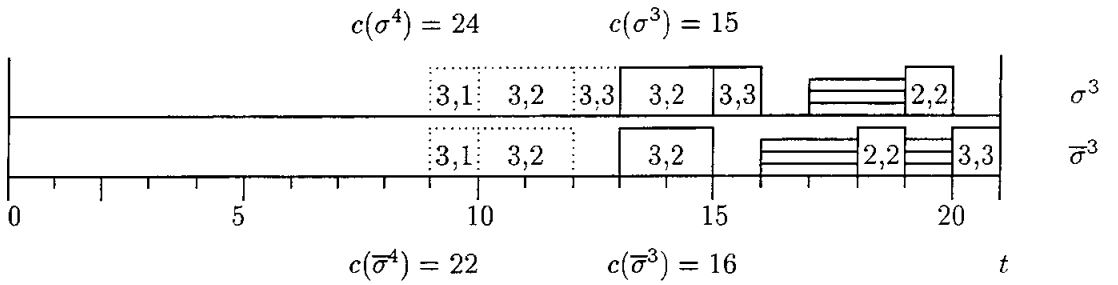

Figure 8: *Theorem 5: σ^3 does not dominate $\overline{\sigma}^3$*

In the second dominance rule for the case $i^s \neq \overline{i^s}$, we must consider $sc_{0,i}$ instead of $sc_{g,i}$ to take block costs into account.

Theorem 6 *Given two s-partial schedules σ^s and $\overline{\sigma}^s$ with $\mathcal{AS}^s = \overline{\mathcal{AS}}^s$ and $i^s \neq \overline{i^s}$, σ^s dominates $\overline{\sigma}^s$ if*

(i) $\quad t(\overline{\sigma}^s) + st_{\overline{i^s},i^s} \leq t(\sigma^s)$ *and*

(ii) $\quad c(\overline{\sigma}^s) \geq c(\sigma^s) + sc_{0,i^s}$.

Proof: Let i^ω denote the family of the (last) job in a completion ω^s of $\overline{\sigma^s}$. Then $st_{i^\omega,i^s} \leq st_{i^\omega,\overline{i^s}} + st_{\overline{i^s},i^s}$, analogously for setup costs due to the triangle inequality. Thus any completion ω^s of $\overline{\sigma}^s$ is also a feasible completion of σ^s because of (i); if $(\omega^s, \overline{\sigma}^s)$ is feasible, (ω^s, σ^s) is feasible, too. Due to (ii), for any ω^s, the schedule (ω^s, σ^s) has lower costs than $(\omega^s, \overline{\sigma}^s)$.

The difference is that now also block costs are taken into account in (ii): when leftshifting $\mathcal{G}_1(\sigma^s)$ in an extension of σ^s, we have $c(\sigma^s) + sc_{0,i^s}$ as an upper bound for the cost contribution. A trivial lower bound for the cost contribution of $\overline{\sigma}^s$ is $c(\overline{\sigma}^s)$. Thus σ^s dominates $\overline{\sigma}^s$ as any ω^s completes σ^s at lower costs, completing the proof. □

Finally, an alternative way to solve the BSP is a dynamic programming approach. We define the job-sets as states and apply the dominance rules in the same way. An implementation of this approach was less efficient and is described in Jordan [9].

7 Computational Results

From the analysis in Section 4 we know that we address the same planning problem in BSP and DLSP, and that we find corresponding solutions. Consequently, in this section we compare the performance of algorithms solving the BSP with procedures for solving variants of the DLSP. The comparison is made on the DLSP instances used to test the DLSP procedures; we take the instances provided by the cited authors and solve them as BSP(DLSP) or BSPUT(DLSP) instances (cf. Figure 1). An exception is made for reference [8] where we use randomly generated instances.

The different DLSP variants are summarized in Table 11. For the DLSP, in the first column the reference, in the second the DLSP variant is displayed. The fourth column denotes the proposed algorithm, the third column shows whether computational results for the proposed algorithm are reported for equal or unequal holding costs. Depending on the holding costs, the different DLSP variants are solved as BSP(DLSP) or BSPUT(DLSP) instances. With the exception of reference [18], the DLSP procedures are tested with equal holding costs, so that regenerative schedules are optimal in [4] and [8].

Table 11: *Solving Different DLSP Variants as a BSP*

DLSP				BSP	
Author	Variant	Holding Costs	Algorithm	Instances	Properties of Schedules
Cattrysse et al. [4]	SISTSC	$h_i = 1$	DACGP	BSP(DLSP)	EDDWF and regenerative
Fleischmann [8]	SDSC	$h_i = 1$	TSPOROPT	BSP(DLSP)	EDDWF and regenerative
Salomon et al. [18]	SDSTSC	$h_i > 0$	TSPTWA	BSPUT(DLSP)	EDDWF
		$h_i = 0$	TSPTWA	BSP(DLSP)	EDDWF and one block

7.1 Sequence Independent Setup Times and Setup Costs (SISTSC)

In Cattrysse et al. [4], a mathematical programming based procedure to solve SISTSC is proposed. Cattrysse et al. [4] refer to their procedure as dual ascent and column generation procedure (DACGP). The DLSP is first formulated as a set partitioning problem (SPP) where the columns represent the production schedule for one item i; the costs of each column can be calculated separately because setups are sequence independent. DACGP then computes a *lower bound* for the SPP by column generation, new columns can be generated solving a single item subproblem by a (polynomial) DP recursion. In DACGP a feasible schedule, i.e. an *upper bound*, may be found in the column generation step, or is calculated by an enumerative algorithm with the columns generated so far. If in neither case a feasible schedule is found, an attempt is made with a simplex based procedure.

The (heuristic) DACGP generates an upper and a lower bound, SABSP solves BSP(DLSP) to optimality. DACGP is coded in FORTRAN, SABSP is coded in C. DACGP was run on an IBM-PS2 Model 80 PC (80386 processor) with a 80387 mathematical coprocessor, we implemented SABSP on the same machine to make computation times comparable.

Computational results for the DACGP are reported only for identical holding costs ($h_i = 1$) for all items. Consequently, we solve DLSP as BSP(DLSP) and only need to consider regenerative schedules, cf. Theorem 4. Furthermore, the timetabling procedure requires fewer computations in equation (17) as setups are sequence independent.

The DLSP instances with nonzero setup times are provided by the authors of [4]. They generated instances for item-period combinations $\{(N,T)\} = \{(2,20),(2,40),(4,40),(2,60),$
$(4,60),(6,60)\}$. We refer only to instances with $T = 60$ because smaller instances are solved much faster by SABSP than by DACGP. The DLSP instances have setup times $st_{g,i}$ of either 0, 1 or 2 periods. The average setup-time per item (over all instances) is (approximately) 0.5, making setup times not very significant. For each item-period combination instances with different (approximate) capacity utilizations ρ were generated: low (L) capacitated ($\rho < 0.55$), medium (M) ($0.55 \leq \rho < 0.75$) and high (H) capacitated instances ($\rho \geq 0.75$). Approximate capacity utilization is defined as $\rho = (1/T)\sum_{i,t} q_{i,t}$. 30 instances were generated for each (N,T,ρ) combination, amounting to $3 \cdot 3 \cdot 30 = 270$ instances in total.

In Table 12, we use $\#J$ to denote the average number of jobs in BSP(DLSP) for the instance size (N,T) of the DLSP. For DACGP we use $\triangle_{avg}$ to denote the average gap (in percent) between upper and lower bound. $\#_{inf}$ is the number of instances found infeasible by the different procedures and R_{avg} denotes the average time (in seconds) needed for the 30 instances in each class. For DACGP, all values in Table 12 are taken from [4].

In the comparison between DACGP and SABSP, the B&B algorithm solves problems with $N = 2$ and $N = 4$ much faster; the number of sequences to examine is relatively small. For $N = 6$ computation times of SABSP are in the same order of magnitude than for DACGP. In (6,60,M) the simplex based procedure in DACGP finds a feasible integer solution for one of the 10 instances claimed infeasible by DACGP. Thus, in (6,60,M), 9 instances remain unsolved by DACGP, whereas SABSP finds only 7 infeasible instances. DACGP also fails to find existing feasible schedules for (N,T,ρ) =(2,60,H), (4,60,M). Recall that SABSP takes advantage of a small solution space, keeping the enumeration tree small and thus detecting infeasibility or a feasible schedule quite quickly. DACGP tries to improve the lower and upper bound, which is difficult without an initial feasible schedule. Therefore the (heuristic solution procedure) DACGP may fail to detect feasible schedules if the solution space is small.

For the same problem size (N,T) in DLSP, the number of jobs J in BSP(DLSP) may be very different. Therefore, solution times differ considerably for SABSP. Table 13 presents the frequency distribution of solution times. In every problem class the majority of instances is solved in less than the average time for DACGP.

7.2 Sequence Dependent Setup Costs (SDSC)

An algorithm for solving SDSC is proposed by Fleischmann [8]. Fleischmann transforms the DLSP into a traveling salesman problem with time windows (TSPTW), where a *tour* corresponds to a production schedule in SDSC. Fleischmann calculates a lower bound by lagrangean relaxation; the condition that

Table 12: *Comparison of DLSP and BSP Algorithms for SISTSC*

			DLSP DACGP			BSP SABSP	
(N,T)	$\#J$	ρ	$\triangle_{avg}$	R_{avg}	$\#_{inf}$	R_{avg}	$\#_{inf}$
(2, 60)	19	L	0.17	25.8	2	0.1	2
	25	M	0.20	76.3	7	0.2	7
	29	H	1.22	274.9	10	0.1	9
(4, 60)	21	L	0.15	38.9	3	4.2	3
	31	M	0.47	120.8	6	11.8	5
	35	H	1.43	268.7	11	7.1	10
(6, 60)	22	L	0.13	56.2	1	36.9	1
	33	M	0.70	264.9	10	149.0	7
	35	H	0.99	274.1	10	98.7	10

(386 PC with coprocessor)

Table 13: *Frequency Distribution of Solution Times of SABSP*

		Number of instances solved faster than ... [sec]								
(N,T)		< 0.1	< 1	< 10	< 30	< 100	< 300	< 1000	≥ 1000	$< R_{avg}$ (DACGP)
(2, 60)	L	10	20							30
	M	4	26							30
	H	13	17							30
(4, 60)	L		4	23	3					30
	M		2	16	10	2				30
	H	3	4	13	10					30
(6, 60)	L	1		4	5	10	8	2		25
	M	1	1	3	5	12	5	2	1	26
	H	1		4	5	10	8	2		28

(386 PC with coprocessor)

each node is to be visited exactly once, is relaxed. An upper bound is calculated by a heuristic, that first constructs a tour for the TSPTW and then tries to improve the schedule using an Or-opt operation. In Or-opt, pieces of the initial tour are exchanged to obtain an improved schedule. Or-opt is repeated until no more improvements are found. We refer to Fleischmann's algorithm as TSPOROPT. TSPOROPT was coded in Fortran, experiments were performed on a 486DX2/66 PC with the original code provided by Fleischmann.

Fleischmann divides the time axis into micro and macro periods. Holding costs arise only between macro periods, and demand occurs only at the end of macro periods. Thus a direct comparison of TSPOROPT and SABSP using Fleischmann's instances is not viable; instead, we use randomly generated BSP instances which are then transformed into DLSP instances. We generated 30 instances for $N = 5$ families and low (L) ($\rho \approx 0.75$) or high (H)($\rho \approx 0.97$) capacity utilization. Note that for zero setup times, ρ does not depend on the schedule; the feasibility problem is polynomially solvable. In BSP, we have an average number $\#J = 33$ of jobs with a processing time out of the interval $[1, 4]$. In DLSP, we have an average $T = 73$ for high (H) and $T = 100$ for low (L) capacitated instances. Holding costs are identical, and we solve BSP(DLSP). From [8] we select the 2 setup cost matrices S4 and S6 which satisfy the triangle inequality: in S4 costs equal 100 for $g < i$ and 500 for $g > i$. For S6 we have only two kinds of setups: items $\{1, 2, 3\}$ and $\{4, 5\}$ form two setup-groups, with minor setup costs of 100 within the setup-groups and major setup costs of 500 from one setup-group to the other.

In Table 14 results are aggregated over the 30 instances in each class. We use $\triangle_{avg}$ to denote the average gap between lower and upper bound in % for TSPOROPT and R_{avg} ($\tilde{R}_{avg}$) to denote the average time for TSPOROPT (SABSP) in seconds. We denote by $\triangle Z_{best}$ the average deviation in % of the objective function value of the heuristic TSPOROPT from the optimal one found by SABSP. Table 14

Table 14: *Comparison of DLSP and BSP Algorithms for SDSC*

setup cost				TSPOROPT			SABSP
matrix	(N,T)	$\#J$	ρ	$\triangle_{avg}$	$\triangle Z_{best}$	R_{avg}	$\tilde{R}_{avg}$
S4	(5,75)	33	H	5.2	4.1	3.8	0.1
	(5,100)	33	L	15.9	15.3	38.2	24.6
S6	(5,75)	33	H	3.7	2.0	3.4	0.2
	(5,100)	33	L	39.2	15.8	33.9	107.8

(486DX2/66 PC)

shows that $\triangle_{avg}$ can be quite large for TSPOROPT. Solution times of SABSP are short for high capacitated instances and long for low ones. For S4, TSPOROPT generates a very good lower bound, we have $\triangle_{avg} \approx \triangle Z_{best}$ and the deviation from the optimal objective is due to the poor heuristic upper bound. On the other hand, for S6 both the lower and the upper bound are not very close to the optimum. Note that SABSP does not solve large instances of SDSC with 8 or 10 items whereas Fleischmann reports computational experience for instances of this size as well. The feasibility bound is much weaker for zero

setup times, or, equivalently, the solution space is much larger, making SABSP less effective. For the instances in Table 14, however, SABSP yields a better performance.

7.3 Sequence Dependent Setup Times and Setup Costs (SDSTSC)

In Salomon et al. [18], Fleischmann's transformation of the DLSP into a TSP with time windows (TSPTW) is extended for nonzero setup times in order to solve SDSTSC. Nodes in the TSP network represent positive demands, and all nodes must be visited within a certain time window. The transformed DLSP is solved by a dynamic programming approach designed for TSPTW problems (cf. Dumas et al. [6]), we refer to the procedure in [18] as TSPTWA. Paths in the TSP network correspond to partial schedules. Similar to the dominance rule for SABSP, in TSPTWA paths may dominate other paths via a cost dominance, or they may be eliminated because they cannot be extended, which corresponds to the feasibility bound.

TSPTWA is coded in C and run on a HP9000/730 workstation (76 mips, 22 M flops). SABSP runs on a 486DX2/66 PC. In order to test TSPTWA Salomon et al. [18] use randomly generated instances, in which, similar to [4], setup times $st_{g,i} \in \{0,1,2\}$. Unfortunately, the setup times do not satisfy the triangle inequality. A "triangularization" (e.g. with the Floyd/Warshall algorithm) often results in setup times equal to zero. So we adjusted the setup times "upwards" (which is possible in this case because $st_{g,i} \in \{0,1,2\}$) and as a result, setup times are rarely zero. We added 4 (8) units to the planning horizon for $N = 3$ and $N = 5$ ($N = 10$) in order to obtain the same (medium) capacity utilization as in [18]. In this way, instances are supposed to have the same degree of difficulty for TSPTWA and SABSP: the smaller solution space due to correcting $st_{g,i}$ upwards is compensated by a longer planning horizon.

In [18] instances are generated for $T = 20, 40, 60$, and we take the (largest) instances for the item-period combination $\{(N,T)\} = \{(10,40),(3,60),(5,60),(10,60)\}$. The instances have a medium (M) capacity utilization $0.5 \leq \rho \leq 0.75$ because setup times are nonzero. For each (N,T) combination, 30 instances with and without holding costs are generated. Holding costs differ among the items. Consequently, we solve BSPUT(DLSP) if $h_i > 0$ and BSP(DLSP) if $h_i = 0$. Furthermore, we need not apply the timetabling procedure in the latter case because the optimal schedule is one block. In Table 15, $\#F$ ($\#\tilde{F}$) denotes the number of problems solved by TSPTWA (SABSP) within a time limit of 1200 sec (1200 sec) and a memory limit of 20 MB (10 MB). $\#J$ denotes the average number of jobs for the BSP. $\tilde{R}_{avg}$ ($\tilde{R}^B_{avg}$) denotes the average time SABSP requires to solve the instances (considering only regenerative schedules). The average time is calculated over all instances which are solved within the time limit, $\tilde{R}_{avg}$ is put in brackets if not all instances are solved. The last column shows the results if we consider only regenerative schedules during enumeration for $h_i > 0$: $\triangle Z^B_{max}$ provides the maximal deviation in % from the optimal schedule (which may be non-regenerative).

Table 15 demonstrates that SABSP succeeds in solving some of the problems which remained unsolved by TSPTWA. Solution times of SABSP are relatively short compared with TSPTWA for $N = 3$ and $N = 5$. Solution times increase for $N = 10$, and instances can only be solved if the number of jobs is relatively small. Instances become difficult for nonzero, especially for unequal holding costs. If we only enumerate over regenerative schedules, solution times for SABSP decrease. Moreover, only one instance is not solved to optimality for $(N,T) = (3,64)$. Thus, even for unequal holding costs optimal schedules are regenerative in most cases. Furthermore, for $(N,T) = (10,48)$ $((10,68))$, 29 (3) instances would have been solved within the time limit of 1200 sec if only regenerative schedules would have been considered.

Table 15: *Comparison of DLSP and BSP Algorithms for SDSTSC*

			TSPTWA		SABSP			
(N,T)	$\#J$	h_i	$\#F$	R_{avg}	$\#\tilde{F}$	$\tilde{R}_{avg}$	$\tilde{R}^B_{avg}$	$\triangle Z^B_{max}$
$(10,48)$	25	$=0$	21	<1200	30	3.5	-	-
	25	>0	1	<1200	21	(489.8)	(373.1)	0.0
$(3,64)$	38	$=0$	30	<1200	30	0.8	-	-
	38	>0	30	<1200	30	5.1	1.4	0.4
$(5,64)$	38	$=0$	25	<1200	30	28.0	-	-
	38	>0	0		30	140.3	36.3	0.0
$(10,68)$	38	$=0$	4	<1200	14	(41.6)	-	-
	38	>0	0		2	(438.6)	(488.4)	0.0

(486DX2/66 PC)

8 Summary and Conclusions

In this paper, we examined both the discrete lotsizing and scheduling problem (DLSP) and the batch sequencing problem (BSP).

We presented model formulations for the DLSP and for the BSP. In the DLSP, decisions regarding what is to be done are made in each individual period, while in the BSP, we decide how to schedule jobs. The DLSP can be solved as a BSP if the DLSP instances are transformed. For each schedule of one model there is a corresponding solution for the other model. We proved the equivalence of both models, meaning that for an optimal schedule of the BSP the corresponding solution of the DLSP is also an optimal schedule, and vice versa.

In order to solve the BSP effectively, we tried to restrict the search to only a subset of all possible schedules. We found out that jobs of one family can be preordered according to their deadlines. Furthermore, for equal holding costs, it is optimal to start production for a family only if there is no inventory of this family.

When solving the BSP with a branch & bound algorithm to optimality, we face the difficulty that already the feasibility problem is difficult. We must maintain feasibility and minimize costs at the same time. Compared with other scheduling models, the objective function is rather difficult for the BSP. A tight lower bound could thus not be developed. We therefore used dominance rules to prune the search tree. Again, the difficult objective function complicates the dominance rules and forces us to distinguish different cases.

In order to evaluate our approach, we tested it against (specialized) procedures for solving variants of the DLSP. Despite the fact that we have no effective lower bound, our approach proved to be more efficient if (*i*) the number of items is small, and (*ii*) instances are hard to solve, i.e. capacity utilization is

high and setup times are significant. It is then "more appropriate" to schedule jobs than to decide what to do in each individual period.

In the DLSP, the time horizon is divided into small periods and all parameters are based on the period length. In the BSP, all parameters can also be real numbers: setup times, in particular, are not restricted to being multiples of a period length. The different models also result in different problem sizes for DLSP and BSP: the problem size for DLSP is essentially the number of items N and periods T while the problem size for the BSP depends on the number of families and jobs.

We conjecture that our approach is advantageous for instances with few items and a small solution space (i.e. long setup times and high capacity utilization), where the job sequence is the main characteristic of a solution. In such cases we managed to solve instances with 10 (5) families and 30 (50) jobs on a PC. DLSP solution procedures are thought to be better suited for lower capacitated instances with many items, setup times that are not very significant, and parameters which differ among the periods. It is then appropriate to decide anew for each individual period.

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

Acknowledgments

We are indebted to Dirk Cattrysse and Marc Solomon who made available their instances, and to Bernhard Fleischmann who made available his code. Furthermore, we would like to thank three anonymous referees for their valuable comments on earlier versions of this paper.

References

[1] AHN, B.H. AND J.H. HYUN, 1990. Single facility multi-class job scheduling. *Computers and Operations Research*, Vol. 17, pp. 265-272.

[2] 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.

[3] BRÜGGEMANN, W. AND H. JAHNKE, 1994. Remarks on: "Some extensions of the discrete lotsizing and scheduling problem" by Salomon, M., L.G. Kroon, R. Kuik and L.N. Van Wassenhove. Working Paper, Institut für Logistik and Transport, University Hamburg, Germany.

[4] CATTRYSSE, D., M. SALOMON, R. KUIK AND L.N. 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.

[5] DREXL, A. AND A. KIMMS, 1996. Lot sizing and scheduling: survey and extensions. Working Paper, University Kiel, to appear in *European Journal of Operational Research.*

[6] DUMAS, Y., J. DESROSIERS, E. GELINAS AND M.M. SOLOMON, 1995. Technical Note: An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, Vol. 43, pp. 367-371.

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

[8] FLEISCHMANN, B., 1994. The discrete lot-sizing and scheduling problem with sequence-dependent setup-costs. *European Journal of Operational Research*, Vol. 75, pp. 395-404.

[9] JORDAN, C., 1995. *Batching and Scheduling - Models and Methods for Several Problem Classes.* Lecture Notes in Economics and Mathematical Systems No. 437, Springer, Berlin 1996.

[10] GAREY, M.R. AND D.S. JOHNSON, 1979. *Computers and intractability — a guide to the theory of NP-completeness.* Freeman, San Francisco.

[11] HAASE, K., 1996. Capacitated lot-sizing with sequence dependent setup costs. *OR Spektrum*, Vol. 18, pp. 51-59.

[12] HAASE, K. AND A. KIMMS, 1996. Lot sizing and scheduling with sequence dependent setup costs and times and efficient rescheduling opportunities. Working Paper, University Kiel.

[13] LASDON L.S. AND R.C. TERJUNG, 1971. An efficient algorithm for multi-item scheduling. *Operations Research*, Vol. 19, pp. 946-969.

[14] MASON, A.J. AND E.J. ANDERSON, 1991. Minimizing flow time on a single machine with job classes and setup times. *Naval Research Logistics*, Vol. 38, pp. 333-350.

[15] MONMA, C.L. AND C.N. POTTS, 1989. On the complexity of scheduling with batch setup-times. *Operations Research*, Vol. 37, pp. 798-804.

[16] 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.

[17] 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.

[18] SALOMON, M., M.M. SOLOMON, L.N. VAN WASSENHOVE, Y.D. DUMAS, S. DAUZERE-PERES, 1995. Discrete lotsizing and scheduling with sequence dependent setup times and costs. Working Paper, to appear in *European Journal of Operational Research.*

[19] SANTOS, C. AND M. MAGAZINE, 1985. Batching in single operation manufacturing systems. *Operations Research Letters*, Vol. 4, pp. 99-103

[20] SCHUTTEN, J.M.J., S.L. VAN DE VELDE AND W.H.M. ZIJM, 1996. Single-machine scheduling with release dates, due dates and family setup times. *Management Science*, Vol. 42, pp. 1165-1174.

[21] UNAL, A. AND A.S. KIRAN, 1992. Batch sequencing. *IIE Transactions*, Vol. 24, pp. 73-83.

[22] VICKSON, R.G., M. MAGAZINE AND C. SANTOS, 1993. Batching and sequencing of components at a single facility. *IIE Transactions*, Vol. 25, pp. 65-70.

[23] WAGNER, H.M. AND T.M. WHITIN, 1958. Dynamic version of the economic lot size model. *Management Science*, Vol. 5, pp. 89-96.

[24] WEBSTER, S. AND K.R. BAKER, 1995. Scheduling groups of jobs on a single machine. *Operations Research*, Vol. 43, pp. 692-704.