Jordan, Carsten

A two-phase genetic algorithm to solve variants of the batch sequencing problem

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 363

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

Suggested Citation: Jordan, Carsten (1995) : A two-phase genetic algorithm to solve variants of the batch sequencing problem, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 363, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
https://hdl.handle.net/10419/155431

Nr. 363

A Two-Phase Genetic Algorithm to Solve Variants of the Batch Sequencing Problem

Carsten Jordan

e-mail: Jordan@bwl.uni-kiel.de
Lehrstuhl für Produktion und Logistik
Christian-Albrechts-Universitaet Kiel
D - 24118 Kiel

Abstract

We introduce the batch sequencing problem with item and batch availability for the single machine and two machine flow-shop case. We propose a genetic algorithm which solves all variants through a decomposition of the problem into a Phase I-Batching and a Phase II-Scheduling decision. The batch sequencing problem is closely related to the discrete lotsizing and scheduling problem (DLSP). Computational experience shows that our algorithm favourably compares with procedures for the DLSP.

Zusammenfassung

Wir betrachten das Batch Sequencing Problem mit geschlossener und offener Produktweitergabe fuer den Ein-Maschinen- und Zwei-Maschinen Flow-Shop Fall. Ein genetischer Algorithmus loest alle Varianten durch Dekomposition der Loesung in eine Phase I-Batching und eine Phase II-Scheduling Entscheidung. Das Batch Sequencing Problem haengt eng zusammen mit dem Discrete Lotsizing and Scheduling Problem (DLSP). Rechenergebnisse zeigen, dass der genetische Algorithmus leistungsfaehiger ist als die DLSP Loesungsverfahren.

Sommaire

Nous considerons le batch sequencing problem avec item et batch availability au cas d'une seule machine et pour le deux machine flow-shop environment. Un algorithme du type "genetic algorithm" resoud tous les cas differents. Le batch sequencing problem est très proche au discrete lotsizing and scheduling problem (DLSP). Resultats numeriques montrent la superiorité de l'algorithme proposé sur les algorithmes proposés pour le DLSP.

1 Introduction

In certain manufacturing systems significant setups are required to change production from one item to another, e.g. plastic molding and chemical engineering. Thus productivity can be increased by batching jobs to avoid setups. At the same time demands of the different products occur at different points in time within the planning horizon. To satisfy dynamic demand large inventories must be hold if production is run with large batches, or frequent setups are required if inventory levels are low. Significant setup times, which consume scarce production capacity, further complicate the scheduling problem. This situation is modeled in the batch sequencing problem (BSP).

The set of jobs is partitioned into families and a setup is required if a job follows a job from another family or if a new batch of the same family is started. Only jobs from the same family can form a batch. Jobs in one batch are produced consecutively without a setup. We consider problems where the number of jobs is large compared to the number of families. Each job must be scheduled between time zero and its deadline. All jobs in one family have different deadlines. So a family must be further partitioned into batches to assure feasibility with respect to the deadlines.

In the BSP we distinguish two cases (cf. Santos and Magazine [21]):

- batch availability, i.e. jobs of one batch are only available (to satisfy demand) after completion of the entire batch, and
- item availability, i.e. jobs complete individually and are available at their completion time.

Complexity results are given in Bruno and Downey [7] who refer to the BSP as job class scheduling problem and show that even the feasibility problem is NP-hard for nonzero setup times. Monma and Potts [18] refer to the results of [7] to derive the complexity of other scheduling problems and give a pseudopolynomial algorithm. Unal and Kiran [22] propose a heuristic to solve the feasibility problem of the BSP with item availability.

Scheduling models which involve batching are e.g. presented in Santos and Magazine [21]. Properties of optimal schedules for problems without deadlines are derived in Ahn and Huyn [1] and Mason and Anderson [17]. A survey of recent results is found in Webster and Baker [23].

The discrete lotszing and scheduling problem (DLSP) is similar to the job class scheduling problem. Salomon et al. [20] refer to the results in [7] to examine the complexity of the DLSP. Potts and van Wassenhove [19] stress the relationship of batching and lotsizing problems. A reformulation of the DLSP as a BSP and an exact solution procedure is given in Jordan and Drexl [12].

A near optimal dual ascent and column generation algorithm for the DLSP with setup times is presented in Cattrysse et al. [8]. Brüggeman and Jahnke [5] consider the DLSP with the additional assumption of batch availability. An extension of the model to a 2-stage production process is given in Brüggeman and Jahnke [6]. In both cases they solve the extended DLSP with a simulated annealing approach.

The outline of the paper is as follows: We first introduce the model in Section 2. In Section 3 a genetic algorithm is presented which turns out to accommodate easily the different variants of the basic model. We illustrate the transformation of a DLSP into a BSP in Section 4. In Section 5 we compare the performance of the genetic algorithm with procedures to solve the DLSP. We use the results of Brüggeman and Jahnke [5], Brüggeman and Jahnke [6] and Cattrysse et al. [8] as benchmarks for our algorithm. Conclusions follow in Section 6.

2 The batch sequencing problem

First we introduce the notation for the BSP. Parameters concerning the families are given in Table 1. We assume sequence-independent setup times st_i and costs sc_i. Setup times and costs depend only on the family i a setup is performed to.

Table 1: Parameters of the Families

i	index of the family, $i = 1, ..., I$
K_i	number of jobs in family i
st_i	setup time to family i
sc_i	setup costs to family i

Attributes for jobs are given in Table 2. The j-th job of family i is indexed with a tuple (i, j). A job (i, j) has a processing time $p_{(i,j)}$, a deadline $d_{(i,j)}$ and a weight $w_{(i,j)}$.

Table 2: Jobattributes

(i, j)	denotes the j-th job of family i, $i = 1, ..., I$, $j = 1, ..., K_i$
$p_{(i,j)}$	processing time of the j-th job of family i
$d_{(i,j)}$	deadline of the j-th job of family i
$w_{(i,j)}$	earliness weight of the j-th job of family i

All jobs in one family are labelled in order of increasing deadlines $d_{(i,j)}$. We assume that the

time between successive deadlines of one family is large enough to perform at least one setup to another family before processing the next job, i.e.

$$\forall i,j \quad \exists g : d_{(i,j-1)} + st_g + st_i \leq d_{(i,j)} - p_{(i,j)}.$$

Otherwise we could batch jobs a priori and consider them as a single job (cf. Unal and Kiran [22]). Cf. Figure 1 with a BSP instance with 2 families and 5 jobs. The 3 jobs in family 1 all have different deadlines. Setup time to family 1 (2) is 1 (2) units of time.

The BSP is solved in two phases, i.e.

- Phase I: how are families partitioned into batches, referred to as Phase I-Batching, cf. Table 3, and
- Phase II: how do we schedule batches, referred to as Phase II-Scheduling, cf. Table 4.

Table 3: Decision Variables Phase I-Batching

j_b	the b-th batch of family i starts with job (i, j_b)
NB_i	family i is partitioned into NB_i batches
$B_{(i,b)}$	denotes the b-th batch of family $i, b = 1, ..., NB_i$, $B_{(i,b)} = \{(i,l) \mid l = j_b, ..., j_{b+1} - 1\}$
Resulting batch attributes	
$P_{(i,b)} = st_i + \sum_{l=j_b}^{j_{b+1}-1} p_{(i,l)}$	processing time of the b-th batch of family i
$D^{ba}_{(i,b)}$ $(D^{ia}_{(i,b)})$	deadline of the b-th batch of family i for batch (item) availability

In Phase I-Batching we decide whether a job starts a batch or not. The start job (i, j_b) is the first, job $(i, j_{b+1} - 1)$ the last job in batch $B_{(i,b)}$. In Figure 1 family 1 is partitioned into two batches, $B_{(1,1)} = \{(1,1),(1,2)\}$ and $B_{(1,2)} = \{(1,3)\}$. The attributes batch processing time and batch deadline result from the partitioning of families into batches.

We distinguish between batch and item availability with different batch deadlines. For batch availability a job is available after completion of the whole batch, $D^{ba}_{(i,b)}$ equals the deadline of the start job (i, j_b), cf. (1). In Figure 1 the deadline of the first batch of family 1 is $D^{ba}_{(1,1)} = d_{(1,1)} = 7$.

$$D^{ba}_{(i,b)} = d_{(i,j_b)} \tag{1}$$

For item availability the deadline $D^{ia}_{(i,b)}$ is different (cf. (2)). Processing batch $B_{(i,b)}$ as late as possible the start job (i, j_b) completes at its deadline. Cf. Figure (1), if $B_{(1,1)}$ completes at $D^{ia}_{(1,1)} = 9$ we have $C_{(1,j_1)} = C_{(1,1)} = d_{(1,1)} = 7$.

$$D^{ia}_{(i,b)} = d_{(i,j_b)} + \sum_{l=j_b+1}^{j_{b+1}-1} p_{(i,l)} \tag{2}$$

The batch processing time $P_{(i,b)}$ is the sum of all job processing times in the batch plus the setup time st_i, we have e.g. $P_{(1,1)} = 5$.

For so called burn-in models the batch processing time may be the maximum instead of the sum of the processing times of the jobs in a batch. Lee et al. [14] consider these problems, which arise in the production of wafers in semiconductor manufacturing.

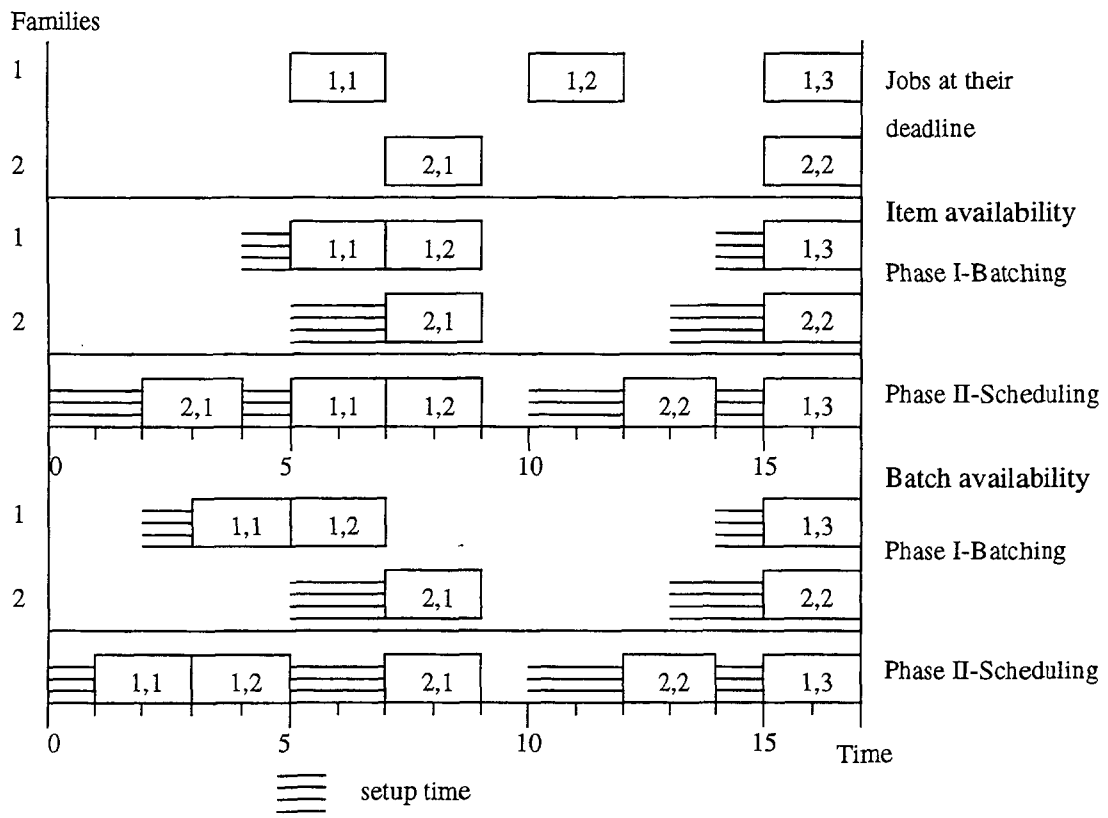

Figure 1: BSP-instance, Phase I-Batching, Phase II-Scheduling

Scheduling batches in Phase II we can treat batches as independent jobs. The Phase-I Batching determines already the setups. The decision variables for Phase II-Scheduling are given in Table 4.

Table 4: Decision Variables Phase II-Scheduling

SB	Sequence of all batches
$B_{(i_{[k]},b_{[k]})}$	denotes the batch at position k
$CB_{(i,b)}$	completion time of the b-th batch of family i
Resulting job completion times	
$C_{(i,j)}$	completion time of job (i,j)
	$C_{(i,j_{b+1}-1)} = CB_{(i,b)}$,
	$C_{(i,l)} = C_{(i,l+1)} - p_{(i,l+1)}, \quad l = j_b, ..., j_{b+1} - 2$

We must sequence batches satisfying constraints (3). We initialize (3) with $CB_{(i_{[0]},b_{[0]})} = 0$. A batch must complete before the next batch starts and not later than its deadline. In (3) we have $D_{(i,b)} = D^{ba}_{(i,b)}$ if we consider batch availability, or $D_{(i,b)} = D^{ia}_{(i,b)}$ for item availability. For given batch completion times $CB_{(i,b)}$ we recursively compute the job completion times $C_{(i,j)}$ of the jobs in batch $B_{(i,b)}$, cf. Table 4.

$$CB_{(i_{[k-1]},b_{[k-1]})} \leq min\left\{CB_{(i_{[k]},b_{[k]})} - P_{(i_{[k]},b_{[k]})}; D_{(i_{[k-1]},b_{[k-1]})}\right\} \qquad k = 1, ..., \sum_{i=1}^{I} NB_i \qquad (3)$$

We can now express the feasibility problem BSPF-BA of the BSP for batch availability.

BSPF-BA : Find batches $B_{(i,b)}$, a sequence SB, and completion times $CB_{(i,b)}$ subject to (3) and (1).

The feasibility problem BSPF-IA for item availability differs in the deadline only.

BSPF-IA : Find batches $B_{(i,b)}$, a sequence SB, and completion times $CB_{(i,b)}$ subject to (3) and (2).

Feasible solutions of **BSPF-BA** and **BSPF-IA** can be evaluated with respect to earliness and setup costs. Phase I-Batching determines the setup costs. Phase II-Scheduling determines batch completion times $CB_{(i,b)}$ and thus the job completion times $C_{(i,j)}$. The earliness of a job is $d_{(i,j)} - C_{(i,j)}$. Total costs F of a schedule are computed in (4).

$$F = \sum_{i=1}^{I} \sum_{j=1}^{K_i} w_{(i,j)} \cdot (d_{(i,j)} - C_{(i,j)}) + \sum_{i=1}^{I} NB_i \cdot sc_i \qquad (4)$$

The optimization problem of the BSP is expressed as follows:

BSP : Find a solution of **BSPF-BA** (**BSPF-IA**, respectively) which minimizes F.

So far we implicitely assumed that jobs in one family can be ordered w.r.t. increasing deadlines. This is stated in Theorem 1, which allows us to restrict our search for feasible solutions.

Theorem 1 *There exists a feasible solution of* **BSPF-IA** *and* **BSPF-BA** *where jobs* (i,j) *of one family* i *are scheduled in nondecreasing order of their deadlines (earliest deadline within families EDDWF).*

Proof: Refer to Monma and Potts [18] or Unal and Kiran [22]. □

Unfortunately, a solution for **BSP** with arbitrary weights $w_{(i,j)}$ may not have the EDDWF property. But for the special case where weights and processing times are identical, i.e. $w_{(i,j)} = p_{(i,j)}$, we can again restrict our search to EDDWF solutions.

Theorem 2 *For the special case* $w_{(i,j)} = p_{(i,j)}$ *there exists an optimal solution of* **BSP** *where jobs* (i,j) *of one family* i *are scheduled in EDDWF order.*

Theorem 2 is easily proven with an interchange argument. As **BSPF-IA** and **BSPF-BA** are already NP-hard, we will restrict the (heuristic) search of the genetic algorithm to EDDWF solutions. A similar approach is choosen by Woodruff and Spearman [24].

BSP is extended to the 2-machine flow-shop case in Section 3.2. We assume identical processing and setup times on the first Machine 1 and the second Machine 2. In the 2-machine flow-shop case we successively solve **BSP** for Machine 2 first and then for Machine 1.

3 A genetic algorithm

In the following section we present a genetic algorithm (GA) to solve **BSP**. This algorithm combines the meta-strategy of genetic search with a heuristic for the subproblem Phase II-Scheduling. For an introduction to genetic algorithms cf. e.g. Goldberg [11] or Liepins and Hilliard [15]. Genetic search is performed only for Phase I-Batching, the search space is thus much smaller than for the solution of the whole problem. The heuristic for Phase II-Scheduling is derived from standard scheduling algorithms for known polynomially solvable cases.

The algorithm solves all the different variants of **BSP** because we solve the problem in two phases. It is applicable to item and batch availability as well and covers also the 2-machine flow-shop case. An extension to more than 2 machines is straightforward.

3.1 Phase I-Batching: The single machine case

Applying a genetic algorithm (GA) for a specific problem we have to encode the solution of the problem in a genetic string. A suitable coding of the solution (the *individual* in the GA) should need a short string and use a small alphabet for the genes. Moreover, the standard GA operations crossing over and mutation should maintain feasibility of the offsprings, i.e. the coding should be *consistent* with the operators of the GA. How to maintain feasibility is a typical problem one encounters applying GA's. Liepins and Hilliard [15] give an example for the TSP where the crossover operator does not maintain tours. "Repair" algorithms may now convert infeasible to feasible offsprings, or the crossover operator may be modified. But this way we "disturb" the evolution process based on building blocks and schemes (cf. Goldberg [11]). A poor performance is the consequence in most cases.

For our algorithm we choose a binary representation of the Phase I-Batching as the genetic string, cf. Figure 2. Each gene represents the batching of one job: If a job is the start job (i, j_b) of a batch $B_{(i,b)}$, the gene is set to one and zero otherwise. The jobs $(i, 1), i = 1, ..., I$, are always start jobs of a batch and therefore omitted in the genetic string. The string length is $L = \sum_{i=1}^{I} K_i - I$. For the instance of Figure 1 the string length is 3.

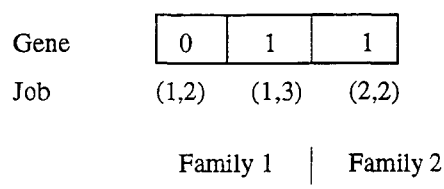

Figure 2: Genetic String of Phase I-Batching

The string in Figure 2 represents the Phase I-Batching shown in the Gantt Chart in Figure 1. Job $(1, 1)$ (not represented in the string) is the start job of the first batch $B_{(1,1)}$. Job $(1, 2)$ is batched with the preceeding job $(1, 1)$ (the gene is set to zero) and job $(1, 3)$ starts again a new batch (the gene is set to one). The same way family 2 is partitioned into two batches, job $(2, 2)$ starts a new batch (the gene is set to one). This string is used in a genetic algorithm to search for the best batching decision. Each gene may assume value 0 or 1, independently of the other genes. Note, that in a string representing a permutation of jobs we may not have two times the same value of a gene, values of one gene are not independent. We never obtain infeasible offsprings applying the genetic operators crossover or mutation. Moreover, we use a small alphabet for the genes and a string shorter than the total number of jobs. Coding the solution this way is thus well suited to be used in a genetic algorithm.

3.2 Phase I-Batching: The two machine flow-shop case

In the following we extend **BSP** to the two-machine flow-shop case with batch availability. All jobs of one family pass two machines, we assume equal processing times $p_{(i,j)}$ and setup times on both machines. A batch on Machine 1 (M1) must be completed before processing on Machine 2 (M2) can start. In the two-machine case batches on M2 can again be interpreted as jobs (= demands) which can be batched on M1. The deadline of the batch on M1 is the start of its first job on M2. However, the setup can be anticipated on M2, cf. Figure 3. In the genetic string we have to include additional information how jobs (=batches) on M2 are batched on M1. We enlarge the string alphabet to the values [0;1;2] and define its meaning as follows:

$$\text{value of the gene} = \begin{cases} 1 & , \quad \text{the job starts a batch on M1 } \textit{and} \text{ M2} \\ 2 & , \quad \text{the job starts a batch only on M2} \\ 0 & , \quad \text{the job } (i,j) \text{ is batched with job } (i,j-1) \end{cases}$$

In Figure 3 we illustrate Phase I-Batching (for only one item) on M1 and M2. The string which determines the batching decisions is given below the Gantt chart. The jobs at their deadline are first batched on M2 in 4 batches. Batches on M2 can again be interpreted as jobs at their deadline for M1. On M1 we have to schedule 4 jobs (=batches on M2) with deadlines 4, 9, 13 and 16 and processing times of 2, 1, 1 and 2. These 4 jobs are batched in 2 batches on M1. Batches on M1 must end before processing on M2 starts: on M2 Jobs $(1,4)$,$(1,5)$ and $(1,6)$ form two batches, but one batch on M1. Job $(1,5)$ starts a batch only on M2 (the gene is set to 2), job $(1,4)$ starts a batch on both machines (the gene is set to 1) and batches all three Jobs $(1,4)$, $(1,5)$ and $(1,6)$ on M1. The starttime 13 on M2 of the batch with job $(1,4)$ is the deadline for the batch on M1. The setup on M2 for job $(1,4)$ can be anticipated before the batch on M1 is completed. Job $(1,1)$ always starts a batch on both machines and is thus ommitted in the string.

In fact, the string [02120] can be transformed into M1 batching with string [00100] and M2 batching with string [01110] according to the former definition for the single machine case. The heuristic solution of the two-machine problem is straightforward: First we solve **BSP** on M2, compute the deadlines and solve **BSP** on M1.

3.3 Phase II-Scheduling

Minimizing F (cf. (4)) for a given Phase I-Batching we no longer have to worry about setups, which are already determined. The resulting subproblem Phase II-Scheduling consists of scheduling batches between time zero and their deadline.

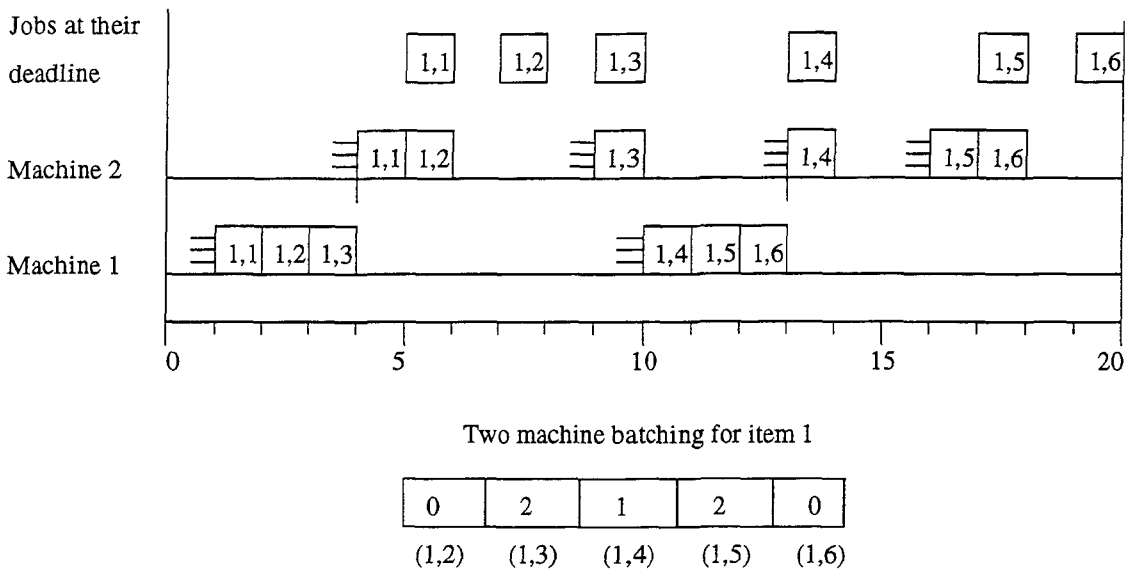

Figure 3: Phase I-Batching for the Two-Machine Flow-Shop case with Batch Availability

We solve this subproblem by considering the corresponding "mirror problem": In the mirror problem the time axis is reverted. The maximal deadline (=planning horizon) becomes time zero, deadlines convert into release dates and time zero converts into the planning horizon, which is a common deadline for all jobs. For a given batching the feasiblility problem is exactly the decision version of the single machine problem with release dates, i.e. *is there a schedule with a makespan less than a certain* C, where C is the common deadline. We denote this problem as $(1/\ r_j/\ C^{max})$ in the 3-field notation of Lageweg et al. [13]. $(1/\ r_j/\ C^{max})$ is solved scheduling jobs in nondecreasing order of release dates (ERD). After Phase I-Batching we thus have to schedule batches in nondecreasing order of their deadlines (EDD) to solve **BSPF-IA** (or **BSPF-BA**, respectively). Consequently, the feasibility problem for a given batching can be solved polynomially.

Minimizing the earliness costs for a given batching the mirror problem is $(1/\ r_j/\ C^{\omega})$ with a common deadline for all batches (=jobs), the objective changes to minimize weighted completion time. Unfortunately, $(1/\ r_j/\ C^{\omega})$ is NP-hard, cf. Lenstra et al. [16]. Note, that Bianco and Ricciardelli [4] propose an exact algorithm for $(1/\ r_j/\ C^{\omega})$.

But for the purpose of a good heuristic we can improve the EDD scheduling of the batches with an idea similar to the Smith-rule for the $(1/\ /\ C^{\omega})$ problem: For each batch we define a specific

weight $W_{(i,b)}$ in (5) (for a similar idea cf. Ahn and Huyn [1]).

$$W_{(i,b)} = \frac{\sum_{l=j_b}^{j_{b+1}-1} w_{(i,l)}}{P_{(i,b)}} \qquad b = 1, ..., NB_i \tag{5}$$

Note, that for the special case $p_{(i,j)} = w_{(i,j)}$ we always have $\sum_{k=j_b}^{j_{b+1}-1} w_{(i,k)} \leq P_{(i,b)}$ as $P_{(i,b)}$ includes setup time. $W_{(i,b)} \in [0;1]$ may be interpreted as a (specific) weight giving the fraction of time in a batch which is spent for production rather than setup. Whenever batches are scheduled earlier than their deadline in the EDD schedule we order them in increasing $W_{(i,b)}$.

The algorithm **Phase II-Scheduling** moves backwards in time and minimizes earliness costs heuristically.

Phase-II Scheduling

1. Sort all batches in order of decreasing deadlines. Set the time instant t at the maximal deadline.
2. Choose among all unscheduled batches $B_{(i,b)}$ with $D_{(i,b)} \geq t$ the one with maximal $W_{(i,b)}$.
3. Schedule the batch w.r.t. constraint (3) and set t at the starting time of the batch. If there is no unscheduled batch with $D_{(i,b)} \geq t$ set t at the maximal deadline of all unscheduled batches.
4. As long as there are unscheduled batches, `GOTO` 2, otherwise `STOP`.

Performing the algorithm for the example in Figure 1 t is set to 17 in Step 1. Batch $B_{(1,2)}$ (which contains job $(1,3)$) and batch $B_{(2,2)}$ have a deadline ≥ 17. We have $W_{(1,2)} = \frac{2}{3} > W_{(2,2)} = \frac{2}{4}$. Thus batch $B_{(1,2)}$ is scheduled closer to its deadline than $B_{(2,2)}$.

3.4 The algorithm GABSP

In the genetic algorithm to solve **BSP** (GABSP) the strings in a population are recombined through mutation and crossover until a prespecified number of generations is reached. We did not perform numerical experiments to find the best values for the different parameters but choose them according to the general recommendations of Goldberg [11].

Genetic search is performed only for Phase I-Batching. A solution for each string is found after Phase II-Scheduling. The following algorithm **EvaluateString** computes the fitness of each string in a new population.

EvaluateString

1. Compute the batches and batch attributes deadline, processing time and weight on M2.
2. Perform Phase II-Scheduling for M2.
3. Compute deadlines, processing times and weights for the batches on M1.
4. Perform Phase II-Scheduling for M1.
5. Evaluate F on M1 and M2 and assign the sum to the fitness of the string.

Clearly, the single machine case does not include steps 3 and 4.

4 Transforming lotsizing into batch sequencing instances

The objective function (4) of **BSP** takes into account setup as well as earliness costs. Interpreting earliness as holding costs, (4) expresses the classical tradeoff in lotsizing problems. We can model **BSP** as a discrete lotsizing and scheduling problem (DLSP). The demand matrice of a DLSP and the jobs at their deadline express the same: The demand (=a job) must be produced (=scheduled) between period (=time) zero and its occurrence (=deadline).

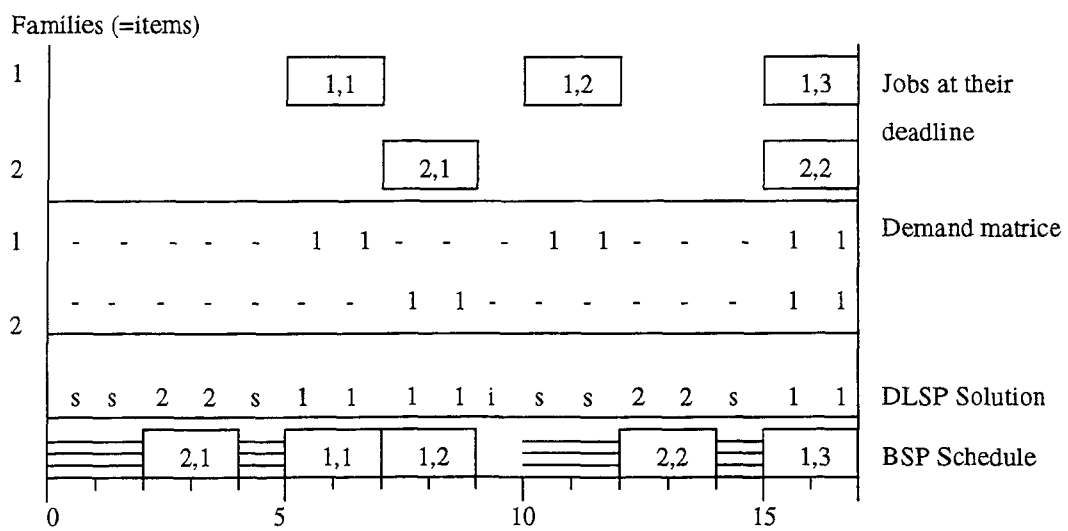

Figure 4: Tranformation between the DLSP and **BSP**

Figure 4 provides the demand matrice for the BSP instance in Figure 1. The [2 x 17] DLSP demand matrice for 2 items and 17 periods reflects the jobs of two families at their deadline.

For item 1(2) setup time is 1(2) periods. The DLSP solution tells us what to do in each period: we have [i,s,1,2] as idle or setup time or production of item 1 or 2, respectively. Interpreting consecutive "ones" in the DLSP demand matrice as a job at its deadline and families as items we transform DLSP into BSP instances, cf. Figure 5. Holding costs h_i of an item i are expressed via earliness weights of the jobs, i.e. $w_{(i,j)} = h_i \cdot p_{(i,j)}$. Via this transformation DLSP instances in the papers of Cattrysse et al. [8] and Brüggemann, Jahnke [5], [6] are available for **BSP**. They all report computational experience with instances where holding costs h_i are equal to one for all items. Therefore Theorem 2 holds and we can restrict the search to EDDWF solutions.

A detailed overview about the DLSP is beyond the scope of this paper, cf. e.g. [9], [12] and [20].

5 Computational results

In this section we compare the performance of GABSP with DLSP solution procedures. We transform the DLSP instances into instances of **BSP** (cf. Figure 5). GABSP is then compared with the procedure proposed by Cattrysse et al. [8], the single-machine procedure of Brüggemann and Jahnke [5] and their two machine procedure [6]. Note, that GABSP solves all the different variants, with a modification only in the assignment of batch attributes in Phase I-Batching.

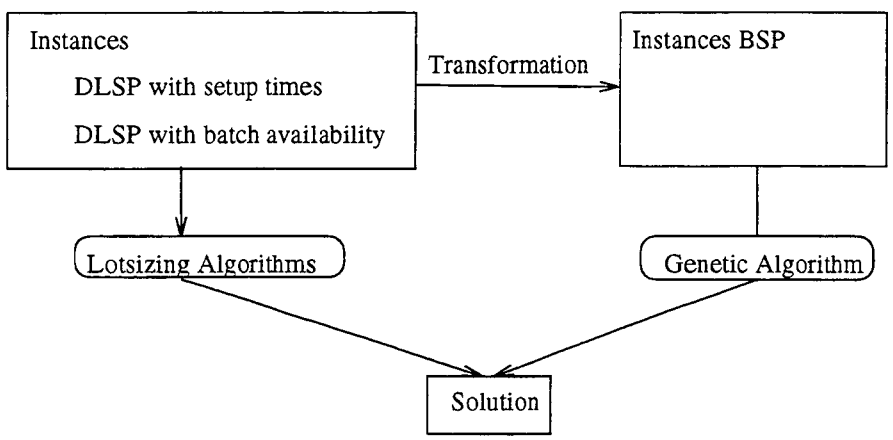

Figure 5: Comparison of DLSP and **BSP** solution procedures

5.1 The single-machine DLSP with setup times

A dual ascent and column generation procedure (DACGP) for the single-machine DLSP is proposed in Cattrysse et al. [8]. It is a near optimal procedure and derives a lower bound as

well. For GABSP batch deadlines must be computed for item availability via constraints (2). From [8] we take the (largest) instances generated for item N and period T combination with $(N,T) = \{(2,60);(4,60);(6,60)\}$. The DLSP instances are transformed into instances for **BSP** with 2, 4 and 6 families, respectively. For all instances optimal solutions are calculated with an exact Branch&Bound procedure described in Jordan and Drexl [12] so that for each instance the deviation A from the optimal solution is known. GABSP has been coded in C, for DACGP we took a FORTRAN code provided by Cattrysse. Experiments were run on a 486/33 Mhz PC. Table 5 gives the results for the different item - period combinations (N,T) which transform into instances for **BSP** with an average number $\#J$ of jobs. Capacity utilization ρ is high (H), medium (M) or low (L), respectively. In each (N,T,ρ) class we aggregated over 30 instances and give the average deviation A_{avg} and maximum deviation A_{max} in % from the optimum. Not all of the 30 instances in each problem class have a feasible solution. This is denoted with $\#IF$ in the last column. The number of problems where the heuristics do not find a solution is denoted by $\#I$.

Table 5: Comparison of the solution quality between DACGP and GABSP

			GABSP			DACGP			
(N,T)	$\#J$	ρ	A_{avg}	A_{max}	$\#I$	A_{avg}	A_{max}	$\#I$	$\#IF$
(2,60)	19	L	0,07	1,4	2	0,02	0,5	2	2
	25	M	0,23	2,6	7	0	0	7	7
	29	H	0,56	4,3	9	0,32	4,3	10	9
(4,60)	21	L	0,14	2,6	3	0,17	3,2	3	3
	31	M	0,63	5,8	5	0,20	3,1	6	5
	35	H	0,82	6,3	10	0,52	5,6	11	10
(6,60)	22	L	0,11	1,0	1	0,21	4,0	1	1
	33	M	0,38	4,5	7	0,79	7,7	10	7
	35	H	0,41	6,6	10	0,60	5,1	10	10

From Table 5 we note that average A_{avg} and maximum deviation A_{max} for DACGP and GABSP do not differ significantly. Note e.g. that GABSP finds all solutions in $(6,60,M)$ while DACGP fails to find a solution to some of the feasible instances. Solution quality of GABSP is slightly better for $N = 6$ than DACGP.

A comparison of the computation times is left out because DACGP generates a lower bound while GABSP does not. Moreover, computation times for a genetic algorithm directly depend on the choice of parameters. For the results in Table 5 we choose a population size of 100, a maximal

number of generations (=iterations) of 500. GABSP stops if 100 consecutive generations do not improve the solution. Good results for GABSP can be obtained also for smaller populations and less iterations but then GABSP sometimes fails to solve instances where a feasible solution is difficult to find. The maximal CPU-seconds in the problem classes for GABSP (DACGP) vary between 4.8 (5.9) and 24.9 (177.0).

5.2 The single-machine DLSP with batch availability

Brüggemann and Jahnke propose in [5] a simulated annealing procedure to solve the single-machine DLSP with batch availability. Now batch deadlines in the GABSP are computed via constraints (1) for batch availability. In [5] computational experience is reported for a hard instance (hard to find a feasible solution) with $N = 6$ items and $T = 60$ periods, which takes 1204 sec to solve on a 486/20 PC. We solve the same instance on a 486DX2/66 PC in 10 sec to find approximately the same solution. The sequence of batches SB and also the objective function value are slightly different in our solution.

5.3 The two-stage DLSP with batch availability

In [6] Brüggemann and Jahnke extend their simulated annealing approach to a two-stage DLSP with batch availability, referred to as BRJSA. This is the two-machine flow-shop case described in Section 3.2. Brüggemann and Jahnke give computational experience for instances with $N = 6$ items and $T = 60$ periods in 6 problem classes with 15 randomly generated instances each. Approximate capacity utilization ρ is given in % for each class, e.g. ρ is between 60 and 70 % in the first class. For high capacity utilization, i.e. $\rho > 0.88$, it is differentiated between a hard and an easy problem class. We choose a similar experimental design as Brüggemann and Jahnke and repeat GABSP 10 times (=10 trials) for each instance with a rather small population size $|PS|$. So GABSP may come up with infeasible solutions in some of the 10 trials, which also happened with BRJSA. We denote the number of unsuccessful trials of BRJSA (GABSP) in a problem class by $\#UT$. Cf. Table 6, we apply both algorithms in problem class 88-92hard. The number of unsuccessful trials $\#UT$ is 29 (6) out of 10·15 trials for BRJSA (GABSP). For BRJSA experiments are conducted on a 486/20 PC, for GABSP on a faster 468DX2/66 PC, so we multiplied times on the latter with factor 5. Table 6 gives average running times R_{avg} ($5 \cdot R_{avg}$) in seconds for BRJSA (GABSP) and the population size $|PS|$ of GABSP.

Table 6 shows a smaller solution time for GABSP versus BRJSA in all problem classes. Similar to the results in [6] objective function values of GABSP differ only slightly over the 10 trials, so that we conclude that the algorithm has converged to a "good" solution. But more important than the

Table 6: Comparison of the solution times between BRJSA and GABSP

			BRJSA		GABSP		
(N,T)	$\#J$	ρ	R_{avg}	$\#UT$	$5 \cdot R_{avg}$	$\#UT$	$\|PS\|$
(6,60)	22	60-70	356	-	20	-	50
	22	70-80	351	-	20	-	50
	22	80-85	400	-	20	-	50
	22	85-88	492	2	20	-	50
	22	88-92easy	670	-	40	2	100
	22	88-92hard	1119	29	40	6	100

comparison of objective function values is the ability of GABSP to find a feasible solution faster than BRJSA.

Objective function values can only be compared for the example in [6], (Table 1 and Figures 3 to 5). There a solution is given for Stage 2, which consists of a sequence of 10 batches. This solution can be improved interchanging the positions of the Batch 2 with 3 and Batch 6 with 7, which is the solution of GABSP on M2 for this instance (comparing the specific (batch) weights $W_{(i,b)}$ this improvement is obvious).

6 Conclusions

We presented a genetic algorithm for some variants of the batch sequencing problem. One algorithm solves all the different variants. To evaluate our approach we solve instances of the discrete lotsizing and scheduling problem for the different variants and demonstrate the performance of our algorithm. It should be noted that setup-times $st_i \in \{0;1;2\}$ in all DLSP instances and thus are not too big. We expect the advantages of our approach to be more pronounced if setups have a greater significance.

The main feature of the approach is the decomposition of the solution procedure in a Phase I-Batching and a Phase II-Scheduling decision. Search is performed only for Phase I-Batching where we use the meta-strategy of a genetic algorithm. The algorithm in Phase II-Scheduling takes advantage from the knowledge about polynomially solvable scheduling problems. The decomposition allows to solve variants of one problem with the same approach. Extensions to multiple (more than two) machines and other batching types (e.g. burn-in-models) are straightforward.

Acknowledgements

We are indebted to D. Cattrysse and W. Brüggeman to make available their instances and their code. Furthermore we wish to thank J. Wachsmuth and J. Kurth for fruitful discussions and the programming.

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] Baker, K.R., 1974. *Introduction to sequencing and scheduling*, Wiley, New York.

[3] Baker, K., 1994. Scheduling groups of jobs. Paper presentend at the ORSA/TIMS Conference, Boston, April 1994.

[4] Bianco, L. and S. Ricciardelli, 1982. Scheduling of a single machine to minimize total weighted completion time subject to release dates. *Naval Research Logistics Quarterly*, Vol. 29, pp. 151-167.

[5] Brüggeman, W. and H. Jahnke, 1992. DLSP with multi-item batch production. Proceedings of a Joint German/US Conference, Operations Resarch in Production planning and control, Springer, Hagen, Germany, ISBN 3-540-56444-6.

[6] Brüggeman, W. and H. Jahnke, 1994. DLSP for 2-stage multi item batch production. *International Journal of Production Research*, Vol. 32, pp. 755-768.

[7] 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.

[8] Cattrysse, 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.

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

[10] 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.

[11] Goldberg, D., 1989. *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, USA.

[12] Jordan, C. and A. Drexl., 1994. Lotsizing and scheduling by batch sequencing. Working Paper No. 343, University of Kiel, Germany.

[13] Lageweg, B.J., J.K. Lenstra and A.H.G. Rinnoy Khan, 1982. Computer-aided complexity classification of combinatorial problems. *Communications of the ACM* , Vol. 25, pp. 817-822.

[14] Lee, C.Y., R. Uzsoy and L.A. Martin-Vega, 1992. Efficient algorithms for scheduling semiconductor burn-in operations, *Operations Research*, Vol. 40, pp. 764-774.

[15] Liepins, G.E. and M.R. Hilliard, 1989. Genetic algorithms: foundations and applications. *Annals of Operations Research*, Vol. 21, pp. 31-58.

[16] Lenstra, J.K., A.H.G. Rinnoy Kan and P. Brucker, 1977. *Complexity of machine scheduling problems*, Studies in integer programming, North-Holland, Amsterdam.

[17] 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.

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

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

[20] 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.

[21] Santos, C. and M. Magazine, 1985. Batching in single operation manufacturing systems, *Operations Research Letters*, Vol. 4, pp. 99-103

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

[23] Webster, S. and K.R. Baker, 1994. Scheduling groups of jobs on a single machine, Working Paper No. 307, The Amos Tuck School of Business Administration, Hanover, NH, USA.

[24] 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.