Annals of Operations Research 96 (2000) 17–38

Solving a chemical batch scheduling problem by local search

Peter Brucker [a,*] and Johann Hurink [b]

[a] *Department of Mathematics, University of Osnabrück, D-49069, Osnabrück, Germany*
[b] *University of Twente, Enschede, The Netherlands*

In this paper the following chemical batch scheduling problem is considered: a set of orders has to be processed on a set of facilities. For each order a given amount of a product must be produced by means of chemical reactions before a given deadline. The production consists of a sequence of processes whereby each process has to be performed by one facility out of a given subset of facilities allowed for this process. The processing times depend on the choice of the facility and the processing is done in batch mode with given minimum and maximum sizes. The problem is to assign the processes to the facilities, splitting them into batches, and scheduling these batches in order to produce the demands within the given deadlines.

For the scheduling part of the problem we present an approach based on the following steps. First, a procedure to calculate the minimum number of batches needed to satisfy the demands is presented. Based on this, the given problem is modeled in two different ways: as a general shop scheduling problem with set-up times or as scheduling problem with positive time-lags. Finally, a two-phase tabu search method is presented which is based on the two different formulations of the problem. The method is tested on some real world data.

Keywords: case study, batch production, tabu search, general shop problem, time-lags, mixed graph scheduling

1. Introduction

In this case study we consider a multi-product batch processing problem which models a special situation in the chemical processing industry. The problem may be formulated as follows.

q orders $A_1, \ldots, A_q$ must be executed. For order A_i a chemical process must provide a_i units of a product before a given deadline d_i. The process starts with a set of raw materials from which the product is derived by a sequence of chemical processing tasks $P_{i1}, \ldots, P_{in_i}$. The production process can be represented by an acyclic network connecting the processing tasks according to the material flows. The predecessors of a processing task P_{ij} provide different inputs which are transformed by P_{ij} into outputs serving as inputs for successors of P_{ij}. Part of the output of P_{ij} may also be used

* Supported by the Deutsche Forschungsgemeinschaft, Project "Komplexe Maschinen-Schedulingprobleme".

again as the input of P_{ij}. In this case P_{ij} is called a loop processing task.

Processing tasks are assigned to processors. We have m facilities (processors) $M_1, \ldots, M_m$. Not all processors can perform all processing tasks, i.e., associated with each P_{ij} there is a subset $M_{ij} \subseteq \{M_1, \ldots, M_m\}$ of the set of processors which are suitable for performing the corresponding chemical process. The processing tasks P_{ij} are performed in batch mode on processor M_k, i.e., we have a batch production with a minimum and maximum batch size $\underline{b}_{ijk}$ and $\overline{b}_{ijk}$, which depend on the chemical reactions involved and the capacity of the reactors used. Thus, it is usually necessary to split a task into several batches which are processed one after the other on the assigned facility. A batch is processed without interruption in time p_{ijk}, which only depends on the reactions performed and on the facility used.

The costs $C_{ijk}(x)$ of producing a P_{ij}-batch of $\underline{b}_{ijk} \leqslant x \leqslant \overline{b}_{ijk}$ units on facility M_k are given by

$$C_{ijk}(x) = \begin{cases} C^1_{ijk}x + C^2_{ijk} & \text{if } x > 0, \\ 0 & \text{if } x = 0. \end{cases} \tag{1.1}$$

(C^1_{ijk} denotes the variable costs of producing one unit and C^2_{ijk} the fixed costs of producing one batch of P_{ij} on M_k.)

Besides the production costs, there are constant set-up costs C_{set} if a facility moves from the production of one order to the production of another order. In addition to the set-up costs, the movement from the production of one order to the production of another order on a facility leads to a constant set-up time P_{set} during which the facility cannot be used for processing.

The objective is to

- assign the processing tasks to the facilities,
- split the tasks into batches, and
- schedule the batches

in such a way that

- the demands are satisfied in time and
- the total costs are minimized.

The above defined batching problem can be solved in principle by a two-phase approach. In the first phase the processing tasks of all orders are assigned to processors. Based on this assignment, the minimal number and sizes of the batches necessary to satisfy the demands are calculated and time constraints between batches are fixed. Since the production costs only depend on the machine assignment and the sizes of batches, the decisions in this first phase completely determine the production costs. Furthermore, after phase 1 it is ensured that the demands are satisfied. In the second phase the problem of scheduling the given batches on the facilities while respecting the time constraints is solved. In this phase the set-up costs and the given deadlines are crucial.

In this work we will not consider the assignment of tasks to machines, but assume that this assignment is already given (our industrial partner provided us with its preferred machine assignment). Thus, we will mainly concentrate on the construction of batches and on the scheduling problem given in the second phase. First, in section 2 we describe how for a given machine assignment the processing tasks of an order can be split into batches and give the precedence relations between these batches. Afterwards, in section 3 we present two different models for the batch scheduling problem derived in the previous section. Based on these two models, in section 4 we give tabu search methods to solve the batch scheduling problem. Finally, some computational results on a real world instance and concluding remarks are presented.

A number of articles deal with scheduling problems for chemical batch processes. For comprehensive literature reviews we refer to Rippui [10] and Reklaitis [9]. In Blömer and Günther [2], Burkard et al. [5] and Kondili et al. [7] mixed integer linear programming formulations for batching problems related to our problems are given. In these formulations deadlines for the demands and set-up times and set-up costs for the facilities are not taken into account. On the other hand, we do not consider the batch sizes as variables and do not introduce capacity constraints on storage. Storage was never a bottleneck in our type of application.

One previous work on an application of local search to a batch problem is due to Löhl et al. [8]. However, in this paper a specific problem arising in polymerization processes has been solved using a genetic algorithm. Finally, an approach proposed by Burkard et al. [4] should be mentioned which shows some similarities with ours.

2. Batch production of a single order

In this section we will derive the batch scheduling problem which results from the stated chemical batch problem if a fixed machine assignment is given. To achieve a solution of the scheduling problem, we may model the batch production of each order separately. We start with a continuous system consisting of a network of processing tasks. This model is presented in section 2.1. It is used in section 2.2 to calculate the inputs and outputs which are necessary to satisfy a given demand, i.e., to produce a given amount of the ordered product. This is accomplished by a backward calculation. The next step is to split each processing task into batches. A corresponding algorithm is presented in section 2.3. Another complication is the existence of loop processes, which is discussed in section 2.4. Due to loop processes and to the fact that the batch size is bounded from below the inputs and outputs must be modified appropriately during the backward calculation. Finally, in section 2.5 precedence relations between the batches are introduced.

2.1. A continuous model

A chemical production process without batching restrictions can be represented by an acyclic graph $G = (V, A)$ in which the nodes $V = \{1, \ldots, n\}$ represent the

different processing tasks and the arcs represent the output/input relations between the tasks. To describe these relations more precisely, assume that a processing task k transforms given inputs of altogether I_k units into outputs of altogether

$$L_k := t_k I_k \tag{2.1}$$

units, where the t_k are given constants. The input I_k consists of several products which have to be mixed in some given ratio. These products are provided by preceding processing tasks and/or raw material. Thus, for each predecessor j of processing task k, we have a value M_{jk} which denotes the fraction of input I_k which is provided by processing task j. Similarly, the output of a processing task splits into several products which are used as input for succeeding processing tasks or which may be used to satisfy demands. Again, the ratio between these products is fixed. By T_{jk} we denote the fraction of the output L_j which is provided to processing task k. Ideally, we have $L_j T_{jk} = M_{jk} I_k = M_{jk} L_k / t_k$ or

$$L_k = \alpha_{jk} L_j \quad \text{with } \alpha_{jk} := \frac{T_{jk} t_k}{M_{jk}}. \tag{2.2}$$

(2.2) describes how the output of processing task j is transformed into the output of processing task k. In graph G this is represented by an arc (j, k) with weight α_{jk}. There is a special terminal vertex (vertex without successors), say vertex n, which provides the ordered product as the output. The inputs of sources (i.e., of vertices without predecessors) correspond to the raw materials needed to run the production process.

2.2. Backward calculation

To satisfy a demand of a units we have to solve the following system of linear equations:

$$\begin{aligned} L_n &= a, \\ L_k &= \alpha_{jk} L_j \quad \text{for all } (j,k) \in A. \end{aligned} \tag{2.3}$$

(2.3) has a solution if and only if

$$\alpha_{jj_1}\alpha_{j_1j_2}\cdots\alpha_{j_{s-1},j_s}\alpha_{j_sk} = \alpha_{ji_1}\alpha_{i_1i_2}\cdots\alpha_{i_{r-1},i_r}\alpha_{i_rk}$$

for any two paths $j \to j_1 \to j_2 \to \cdots \to j_s \to k$ and $j \to i_1 \to i_2 \to \cdots \to i_r \to k$ connecting j and k in $G = (V, A)$. If (2.3) does not have a solution we relax the constraints $L_k = \alpha_{jk} L_j$ to $L_k \leqslant \alpha_{jk} L_j$, i.e., we now have to find minimal values $L_1, \ldots, L_n$ satisfying

$$\begin{aligned} L_n &= a, \\ \frac{1}{\alpha_{jk}} L_k &\leqslant L_j \quad \text{for all } (j,k) \in A, \end{aligned}$$

or, equivalently,

$$L_n = a,$$
$$L_j = \max_{(j,k)\in A} \frac{1}{\alpha_{jk}} L_k \quad \text{for all } j = 1, \ldots, n-1. \tag{2.4}$$

Values $L_1, \ldots, L_n$ satisfying (2.4) can be achieved by considering the nodes of G in a reverse topological order and calculating for each of them the value L_j due to (2.4).

Since graph G is acyclic and has only one terminal vertex, the above algorithm will always terminate with output amounts L_i for all tasks i. For any source s of $G = (V, A)$ the necessary input (which will be raw material) is given by $I_s = L_s/t_s$.

Since the amount of output for a task is calculated as the maximum amount which is used for the successors (see (2.4)), not all the output resulting from this task will be used by the succeeding tasks. We assume that these excesses are stored in corresponding buffers and have no further influence.

2.3. Batching

In the previous subsection, for each processing task we calculated an amount L of output which has to be produced. Due to physical constraints a facility can not produce arbitrary amounts of output within the production of one batch. More precisely, the output L_v of each batch v must satisfy the conditions

$$\underline{b} \leqslant L_v \leqslant \overline{b}. \tag{2.5}$$

Thus, to produce L units of output, several batches may be necessary, or it may even be impossible to produce exactly amount L. In the latter case we have to increase L. In the following we will determine in dependence of L, $\underline{b}$, and $\overline{b}$ a set of feasible batches (in the sense of (2.5)) which produces a minimal amount $\tilde{L} \geqslant L$ of output. To accomplish this we create $q_1 \geqslant 0$ batches of size $\overline{b}$, $q_2 \geqslant 0$ batches of size $\underline{b}$, and at most one extra batch of size b with $\underline{b} < b < \overline{b}$. During this process, L is increased by a minimal amount of ΔL units if necessary.

The numbers q_1, q_2, size b of a possible extra batch, and the incremental value ΔL are calculated by the following algorithm. We assume that L is not a multiple of $\overline{b}$. Otherwise we have $q_1 = L/b$, $q_2 = 0$, and $\Delta L = 0$.

Algorithm batching

1. $q_1 := \lfloor L/\overline{b} \rfloor$; *current number of large batches*
2. $q_2 := 0$; *current number of small batches*
3. $d := L - q_1\overline{b}$ *remainder*
4. IF $d \geqslant \underline{b}$ THEN $b := d$ *extra batch*
   ELSE
   BEGIN
5. Calculate the greatest integer l with $l(\overline{b} - \underline{b}) + d < \underline{b}$;

6. If $l \geqslant q_1$ THEN *no extra batch*
   BEGIN
7. $q_2 := q_1 + 1$;
8. $\Delta L := \underline{b} - (q_1(\overline{b} - \underline{b}) + d)$;
9. $L := L + \Delta L$;
10. $q_1 := 0$
   END;
   ELSE
   BEGIN
11. $q_1 := q_1 - (l + 1)$;
12. IF $(l + 1)(\overline{b} - \underline{b}) + d = \underline{b}$ THEN *no extra batch*
13. $q_2 := l + 1$
   ELSE *extra batch*
14. BEGIN $q_2 := l; b := (l + 1)(\overline{b} - \underline{b}) + d$ END
   END
   END

Note that the number of batches created by this algorithm is minimal. For given $\underline{b}$ and $\overline{b}$ the expression $L + \Delta L$ as a function $f(L)$ of L has a form shown in figure 1 (for the case $\underline{b} = 3$ and $\overline{b} = 4$). An explicit description of this function can be derived. Due to the batching restrictions of a processing task k it may be necessary to increase L_k. This is done during the backward calculation in the following way (see figure 1):

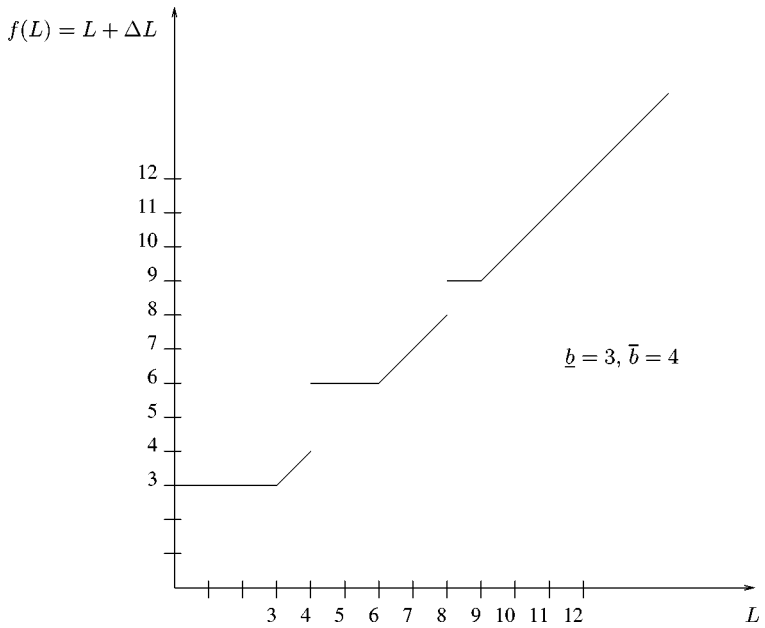


Figure 1. The function $f(L)$.

if L_k is the current value then we calculate the smallest function value $f(L) \geqslant L_k$ and replace L_k by $f(L)$.

2.4. Loop processing tasks

A loop processing task k is a processing task which uses part of its output L_k as the input of k. More specifically, this part of the output is a product which consists of the same mixture of products as the input of this processing task, i.e., this part of the output of a batch may be used to reduce the amount of input for the next batch of the same processing task. By T_{kk} we denote the fraction of the output of a batch returned as input for the next batch. A corresponding fraction of the last batch cannot be used by task k. Thus if L_k is the total output of processing task k then the input is decreased by $T_{kk}(L_k - v)$ where v is the output of the last batch (we always choose a batch with the smallest output as the last batch).

Example 1. We consider a processing task k with $L_k = 30$, $\underline{b}_k = 5$, $\overline{b}_k = 7$ and $t_k = 0.5$. Thus we have five batches with sizes $7, 7, 6, 5, 5$. Without loop this would imply $I_k = 60$.

If we now change k into a loop processing task with $T_{kk} = 1/2$ then 50% of the output of each batch is returned and we have a situation shown in figure 2. The modified input is $I_k = 60 - (30 - 5)/2 = 47,5$. Furthermore, the total output of 30 units is split into 15 units which are passed to the successors, 12.5 units which are used as additional input for process k, and 2.5 units stored in a buffer.

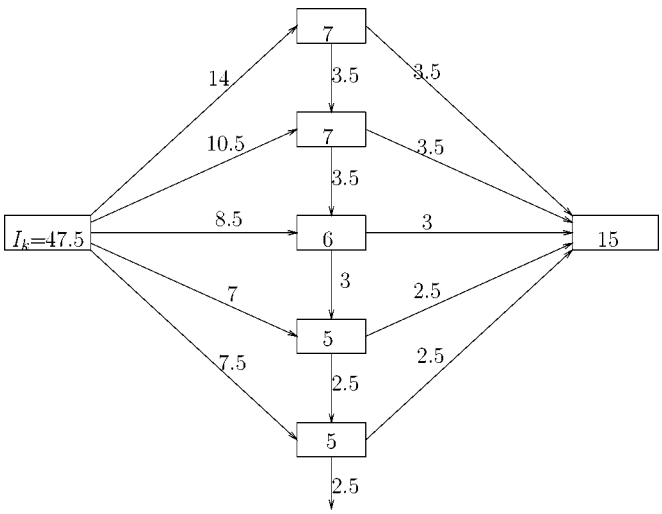


Figure 2. A loop processing task.

2.5. Precedence relations between batches

As indicated in sections 2.3 and 2.4 each task will be split into several batches which all have to be processed by the same machine. In principle, the processing sequence of the batches of one task is arbitrary. However, we will fix this sequence due to the following two reasons:

- Fixing the sequence of the batches of a task will reduce the number of decisions to be made and, therefore, will make the problem much easier. Since the processing times of all the batches belonging to one task are equal, and since there are at most three different sizes of batches for a task, differences between two sequences of batches will only occur due to precedence constraints (to start the kth batch of a task different numbers of batches of predecessors of this task may have to be finished) and, thus, we expect that fixing the sequences of the batches for a task will not have much influence on the quality of the solutions achieved by heuristics.
- Without fixing the sequence of the batches of a task it is not possible to a priori translate the precedences between tasks into precedences between batches, contrary to the case of fixed sequences (see next paragraph). Thus, again, the problem will become easier to handle.

We have chosen to sequence the batches of a task in order of nonincreasing size. To model the sequence we introduce corresponding precedences between the batches, i.e., we represent a processing task by a chain of batches in nonincreasing order.

Furthermore, for a precedence relation between a processing task k and a processing task j we can introduce precedences from certain batches of k to certain batches of j. At every point where enough output has been produced by batches of k for a new batch of j a corresponding arc is introduced. The example shown in figure 3 demonstrates this procedure. The output of the batches of k are denoted by $L_{k\nu}$ and the inputs of j by $I_{j\nu}$.

3. Two models for the production problem

In this section we consider the production problem of scheduling a given set of orders. We assume that the tasks have already been assigned to machines and that by the procedure described in section 2 for each order a model which is defined by an acyclic graph in which the nodes represent batches has been calculated. In section 3.1 we combine these models in a general shop model with set-up times and introduce the mixed graph model as a useful tool for the general shop problem. A restricted version of this model is discussed in section 3.2. The basis of this approach is that batches belonging to the same processing task are aggregated to a single task and that the time dependences between the tasks are modeled by generalized precedences (with non-negative time-lags) between the aggregated tasks.

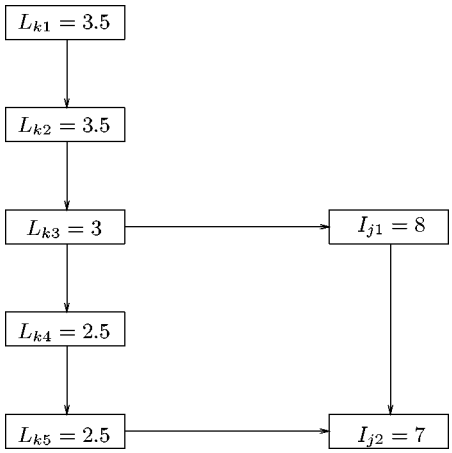


Figure 3. Precedence relations between batches.

3.1. A general shop model

A general shop problem can be described as follows. We have n jobs $i = 1, \ldots, n$ and m machines (processors, facilities) $M_1, \ldots, M_m$. Each job i consists of a set of operations O_{ij} ($j = 1, \ldots, n_i$) with processing times p_{ij}. Each operation O_{ij} must be processed on a dedicated machine $\mu_{ij} \in \{M_1, \ldots, M_m\}$. There may be precedence relation $O_{ij} \rightarrow O_{kl}$ between operations O_{ij}, O_{kl} of the jobs. Each job can only be processed by one machine at a time and each machine can only process one job at a time. Usually the objective is to find a feasible schedule that minimizes some objective function of the finishing times C_i of the jobs $i = 1, \ldots, n$.

In a general shop model with set-ups the set of all operations is divided into disjoint groups $G_1, \ldots, G_q$. If on a machine two operations belonging to different groups are processed one after the other then a set-up time and also set-up costs are to be considered. We assume that all set-up times and all the set-up costs are the same, i.e., they are given by constants P_{set} and C_{set}. The multi-product batch processing problem can be formulated as special general shop problems with set-ups in which the jobs correspond to the processing tasks of all orders and the operations correspond to the batches of the processing tasks. Precedences are given by the precedences between batches of the same orders as defined in section 2.5. Set-up times and set-up costs are incurred if processing on a machine turns from one order to another, i.e., all operations belonging to an order l form a group G_l.

In each group G_l there is a terminal job which corresponds to the terminal processing task of the corresponding order. Let C_l be the completion time of this terminal task and let $T_l = \max\{0, C_l - d_l\}$ be the tardiness for order l. Our objective is to find a schedule S which minimizes

$$f(S) = C_{\text{set}} \left(\sum_{k=1}^{m} z_k \right) + W \sum_{l=1}^{q} T_l. \tag{3.1}$$

In (3.1) z_k denotes the number of set-ups on facility M_k. Furthermore, W is a large penalty which forces all orders to be produced on time if this is possible. We denote the problem of finding a schedule which minimizes (3.1) by GS.

It is convenient to represent feasible solutions of the general shop problem as orientations of edges in a mixed graph $G = (V, C, D)$ where V is the set of all operations, $C = C_1 \cup C_2 \cup R$ is a set of (directed) arcs called conjunctions and $D = D_1 \cup D_2$ is a set of (undirected) edges called disjunctions. The sets C_1, C_2, R, D_1, D_2 are defined as follows:

C_1 represents the set of all chain precedences between the batches of each processing task,

C_2 represents all other precedences which are defined between batches belonging to the same order,

R is the set of all arcs of the form $(0, i)$ connecting a dummy start operation $0 \in V$ with all operations without predecessors. We associate the weights P_{set} and the costs C_{set} with these arcs,

D_1 is the set of edges connecting all pairs of operations of the same order which are to be processed on the same machine and which are not connected by a (directed) path of conjunctions,

D_2 is the set of edges connecting all pairs of operations belonging to different orders which are to be processed on the same machine. We associate the weights P_{set} and the costs C_{set} with these edges.

We further label the vertices by the processing times of the corresponding operation.

A set S of arcs constructed by orienting all edges in D is called a selection. A selection S is feasible if the directed graph $(V, C \cup S)$ does not contain a cycle. A feasible selection provides a semiactive feasible schedule defined by completion times C_{ij} of operations O_{ij} which are calculated as follows. Set C_{ij} equal to the length of a longest path from 0 to O_{ij}, where the length of a path is the sum of labels of operations and of set-up weights of conjunctions and (oriented) disjunctions along this path. This resulting schedule has minimal completion times for all jobs within the set of feasible schedules respecting the given selection. On the other hand, it is easy to see that each feasible schedule leads to a unique feasible selection. Thus, the search space may be restricted to the set of all feasible selections if the objective function is monotone increasing in the completion times of the jobs.

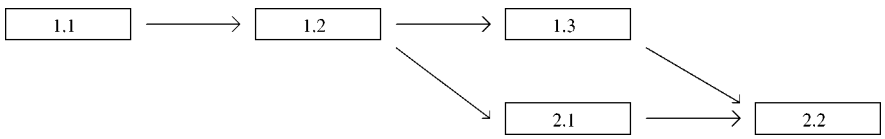
3.2. A time-lag model

The general shop model presented in the previous subsection has the disadvantage that for practical instances the number of operations will become very large since for each batch we have to introduce a corresponding operation. Thus, it will be difficult to develop efficient solution methods based on this model only. Therefore we consider

a second model in which we will restrict the possible solutions for instances of the general shop model to such solutions where all operations (batches) belonging to a job (processing task) are sequenced consecutively without interruption on their machine. Remember that all operations of a job have to be scheduled on the same machine! This restriction is motivated from practice (the hand-made solutions are of this type) and has the effect that the number of possible solutions decreases drastically. Based on these restrictions on the set of feasible solutions, we may simplify the general shop problem by considering all operations belonging to a processing task as one job. The processing time of this job will be equal to the sum of the processing times of the underlying operations. Since operations which are joined together to a job belong to the same group we can associate with each job a unique group and the set-ups which occurred between operations in the former model now occur between jobs. Thus, we are still looking for a schedule which minimizes the changeover costs.

It remains to adapt the given precedences between the operations in the general shop model to the new model. Obviously, the chain precedences between the operations belonging to one processing task become superfluous since these operations now form one job. The additional precedences between the batches of different processing task k and j introduced in section 2.5 may be translated into minimal time distances (non-negative time-lags) l_{kj} between the starting times S_k and S_j of the corresponding new jobs: $S_k + l_{kj} \leqslant S_j$. The value l_{kj} may be achieved as follows. Calculate from the first batch of the processing task k a longest path (length of a path $=$ sum of weights of the vertices on the path) to the last batch of the processing task j. The length of this longest path minus the processing time of job j gives the minimal time-lag l_{kj} between the starting time S_k of job k and the starting time S_j of job j (see also example 2).

Example 2. Consider 2 processing tasks 1 and 2. Task 1 splits into 3 batches each with processing time 5, and task 2 splits into 2 batches each with processing time 7. The precedence relations between the batches are assumed to be as follows:



The length of a longest path from batch 1.1 to batch 2.2 is given by 24 and, since the processing time of task 2 is 14, the time-lag l_{12} between the starting time S_1 of job 1 and the starting time S_2 of job 2 is 10.

Summarizing, we consider a time-lag model in which the jobs are to be scheduled on dedicated machines with time-lags between the jobs and group-dependent set-up times. Furthermore, an objective function depending on the lateness of some of the jobs and on the number of set-ups (see (3.1)) has to be minimized. In analogy to the previous subsection, we will again use a mixed graph $G = (V, C, D)$ to represent

feasible solutions in the time-lag model. V now is the set of all jobs. The set C of conjunctions is given by the union of two sets $C_1 \cup R$, where

C_1 represents the minimal time-lags which exist between jobs. An arc $(i, j) \in C_1$ will get a weight corresponding to the minimal time-lag between the finishing time of job i and the start of job j; i.e., the weight of an arc (i, j) is equal to $l_{ij} - p_i$, since l_{ij} denotes the minimal time distance between the starting times of the two jobs i and j (observe, that these time-lags may become negative, since they represent finish–start relations; e.g., in example 2 job 1 and 2 get a time-lag of length $10 - 15 = -5$). If between two jobs which have to be processed on the same machine a conjunction (or a path of conjunctions) with negative length exists, we will replace this conjunction by a conjunction with weight 0 (or introduce a new conjunction with weight 0).

R is the set of all arcs from a dummy starting operation to vertices without predecessors. Again, we associate the weights P_{set} and the costs C_{set} with these arcs.

The set of disjunctions is given by $D = D_1 \cup D_2$ where now

D_1 is the set of edges connecting all pairs of jobs of the same order which are to be processed on the same machine and which are not connected by a (directed) path of conjunctions,

D_2 is the set of edges connecting all pairs of jobs belonging to different orders which are to be processed on the same machine. Again, we associate the weights P_{set} and the costs C_{set} with these edges.

As in the previous case, each feasible selection (orientation of all edges in D without producing a cycle) provides a feasible schedule and vice versa.

4. A two-phase local search approach

In this section we will present a heuristic solution procedure for the considered batch scheduling problem. The basic goal of designing this method was to develop a solution method which could solve instances for our industrial partner ($\approx$ 50 orders, 30 machines, 200 processing tasks, and 1700 batches) in reasonable time ($\leqslant$ 10 minutes for order acceptance and $\leqslant$ one hour for process planning). Based on the good results achieved for many other optimization problems (see, e.g., Aarts and Lenstra [1]), we have chosen a tabu search-based method.

After our first experiences, it emerged that the general shop model presented in section 3.1 was too detailed to allow tabu search to build up a good solution from scratch in reasonable time. Therefore, we developed a two-phase approach, where in the first phase the "rough" time-lag model is used to change an initial solution to a good solution and in the second phase the general shop model is used to further improve the solution reached after the first phase.

In subsection 4.1 we will present the neighborhood structures which are used as the basis of the two tabu search approaches and in subsection 4.2 we will sketch the complete two-phase method.

4.1. Neighborhood structures

The basic idea of local search methods is to iteratively move through the set of feasible solutions. In each step a new solution is chosen from a subset of solutions near the current solution, the so-called neighborhood of the current solution. Obviously, the neighborhood structure used has a large impact on the way in which the search process will behave. Therefore, the definition of suitable neighborhoods is crucial for the success of local search. In the following we will introduce neighborhoods for the considered batch scheduling problem. Since we have two different models for this problem (the general shop model and the time-lag model) in principle we have to define two neighborhoods. However, since for both models solutions can be represented by selections in a mixed graph, and since for both models the objective function is the same, we can present the neighborhoods jointly. The basic idea is to get a neighbored solution by changing the order of operations (jobs) on one machine slightly, i.e., to orient a few disjunctive edges in D in a different way. Therefore, in the following if we speak about the selection of a mixed graph these results may be adapted to both of the models presented in the previous section. To obtain a unique terminology, we will always call the operation or job belonging to a vertex of the mixed graph an operation. Since the goal is to decrease the objective function (3.1), we will use this function to determine disjunctive edges which will be changed. As already mentioned, the second part $\sum_{l=1}^{q} T_l$ is the most important part of (3.1). In the following we will derive necessary conditions for changing disjunctive edges in order to improve the sum of tardiness $\sum_{l=1}^{q} T_l$ of a given solution. First we need some definitions.

For a given selection S of a mixed graph $G = (V, C, D)$, a *critical tree* T corresponding to S is a spanning subtree of longest paths connecting the dummy root 0 to all other vertices in the directed graph $(V, C \cup S)$. A sequence $u_1, \ldots, u_b$ of vertices, which form a path in a critical tree T, are called a *block* if the sequence contains at least two vertices and if it has one of the two following properties:

1. All operations represented by nodes in the sequence are processed on the same machine and enlarging the sequence would yield a sequence violating this condition.
2. The sequence is achieved from a block (according to 1 or 2) by deleting the first and last vertex of this block.

These definitions are a generalization of the definition of blocks and critical paths for the general shop scheduling problem with sequence-dependent set-up times used by Brucker and Thiele [3]. The main difference is that we replace the critical path by a critical tree since we are considering a sum objective and not a bottleneck objective.

The following theorem will be the basis for constructing neighborhoods.

Theorem 1. Let S and S' be two feasible selections for a mixed graph $G = (V, C, D)$ and let the sum of tardiness $\sum_{l=1}^{l} T_l$ of the solution S' be smaller than that of S. Then for each critical tree T corresponding to S at least one block B of T exists, such that one of the following conditions holds:

- in S' one operation which is not equal to the first operation of block B has to be processed before the first operation of B or
- in S' one operation which is not equal to the last operation of block B has to be processed after the last operation of B.

Proof. In principle, the theorem follows from the fact that a decrease of the sum of tardiness $\sum_{l=1}^{l} T_l$ is only possible if at least for one terminal operation the completion time will decrease. Furthermore, as mentioned in Brucker and Thiele [3] a decrease of the completion time of a terminal operation is only possible if at least one block on the path from 0 to this vertex in T is destroyed in the way mentioned in the theorem. The base of this proof is, that there are two different possibilities to decrease the completion time of a terminal operation. The first is to destroy a block on the longest path to this terminal operation resulting from property 1 of the definition of a block. This can be achieved by shifting an operation of this block before the first or after the last operation of the block. However, since also set-up times have an influence on the length of a path, a second possibility is to reorder operations within a block resulting from property 1. It is easy to see that each reordering will result in a change of a first or a last operations for at least one of the blocks resulting from property 2 of the definition of a block (details of the proof are given in Thiele [12]). □

Based on this theorem we will define two neighborhoods. Consider a feasible selection S for a mixed graph $G = (V, C, D)$ and a corresponding critical tree T. Then each feasible selection S' belongs

- to the neighborhood N_{exchange} of S if it is achieved from S via exchanging the order of the first two or last two operations of a block in T;
- to the neighborhood N_{shift} of S if it is achieved from S via shifting an operations of a block which is not equal to the first (last) operation of the block directly before (after) its block.

These neighborhoods are somehow related to the most commonly used neighborhoods for the job-shop problem (see Aarts and Lenstra [1]).

First numerical tests indicated that for these neighborhoods the number of neighbors was often very large, which led to long computational times for the tabu search approaches. The main reason is that many blocks defined by condition 1 contain several "inner" blocks achieved via condition 2 of the block definition. On the other hand, changes related to these "inner" blocks only may reduce the completion times of a terminal operation by saving a set-up time.

This led to the definition of two further neighborhoods with a smaller number of neighbors. They are achieved by applying the above operators only to blocks satisfying

condition 1 (machine blocks) in the above definition (resulting in subneighborhoods of N_{exchange} and N_{shift}) and by furthermore adding all selections which are achieved by shifting an operation of a block before or after an operation of the same block which belongs to the same group. The latter added neighbors are those neighbors where a set-up time may be removed. The resulting neighborhoods are denoted by $\overline{N}_{\text{exchange}}$ and $\overline{N}_{\text{shift}}$, respectively. Note that in general these neighborhoods are not subneighborhoods of the neighborhoods N_{exchange} and N_{shift} since some of the neighbors added for possibly removing a set-up time may not belong to N_{exchange} or N_{shift}.

4.2. The overall heuristic

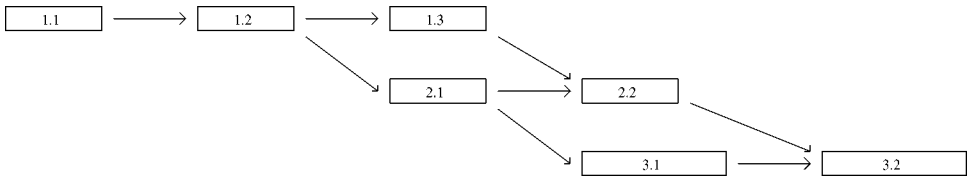
Based on the two models for the considered batching problem presented in the previous section, we have developed the following heuristic:

1. Calculate an initial solution for the time-lag model using a priority-based heuristic.
2. Improve this solution by applying a tabu search approach for the time-lag model using the neighborhoods presented in the previous subsection.
3. Convert the resulting solution of the time-lag model into a solution of the general shop model.
4. Improve this solution by applying a tabu search approach for the general shop model using the neighborhoods presented in the previous subsection.

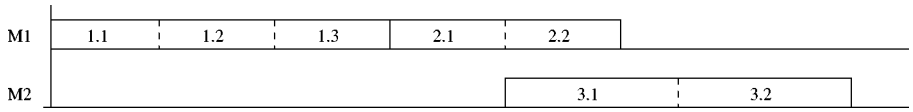
In the following we will describe the four steps of the heuristic in more detail. The initial solution for the time-lag model is iteratively calculated by a priority-based heuristic. In each step, first, from the set of admissible jobs (jobs for which all predecessors have already been scheduled) the job with minimal finishing time (when scheduled next) is calculated. Afterwards, the set of admissible jobs is restricted to all jobs which may start (with respect to the time-lags) before this time. From this reduced set a job with minimal difference between its finishing time plus tail and the deadline of the corresponding order is chosen to be scheduled next (tail = lower bound on the distance between the completion time of this job and the completion time of the corresponding terminal job).

The conversion of a solution of the time-lag model into a solution of the general shop model in step 3 of the above heuristic is done by splitting each job into its batches (operations). This process will not change the schedule and, thus, nor the objective function. However, we applied an additional quick procedure to improve the solution of the general shop model achieved by this direct transformation. When in the solution of the time-lag model on a machine more than one job of one order (group) are scheduled consecutively, we will mix the corresponding operations in the following way (see also example 3). If more than one operation of the corresponding jobs may be scheduled according to the precedence constraints, we chose for an operation of that job which is scheduled last in the time-lag model. This change of the solution may enable other operations of this order to start earlier on other machines.

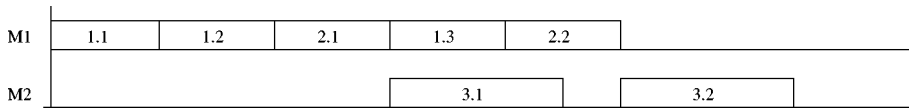
Example 3. Consider 3 processing tasks 1, 2, and 3 all belonging to the same order. Task 1 splits into 3 batches and tasks 2 and 3 both split into 2 batches. The precedence relations between the batches are as follows:



Assuming that both tasks 1 and 2 are processed on machine M_1 and that task 3 is processed on machine M_2, a schedule in the time-lag model may look as follows:



After applying the improvement procedure, the schedule for the general shop model is as follows:



Computational results show that the additional modifications of the converted solution may improve the objective value considerably (see next section).

It remains to describe the tabu search approaches used in steps 2 and 4. As already mentioned in the previous subsection, the representation of solutions and, thus, also the used neighborhoods, are equal for both models. Thus, for both models we basically used the same tabu search approach. In the remaining part of this section we will briefly describe the main elements of this tabu search approach:

Neighborhood reduction. To save computational time, we have restricted the neighborhoods to operators which result from blocks lying on paths in the critical tree from the root to a terminal operation which is late (all other operators will not reduce the completion time of a late terminal operation and, thus, will not reduce the lateness). If no due dates are violated (which hardly will occur in real world applications), we may choose terminal operations of jobs which are closest to their due dates. By these neighborhood reductions we save computational time.

Tabu conditions. For the neighborhoods which contain shifts (N_{shift}, $\overline{N}_{\text{exchange}}$ and $\overline{N}_{\text{shift}}$) we insert the triple containing the old machine predecessor of the shifted operation, the shifted operation, and the old machine successor of the shifted operation, into the tabu list. A neighbor is declared tabu if it results from a shift where the shifted operation together with its new machine predecessor, or the shifted operation together with its new machine successor, or the old machine predecessor and successor of the shifted operation occur as neighbors in one triple of the tabu

list. This setting ensures that a solution can not be reached again as long as the corresponding entry stays in the tabu list.

For the neighborhood N_{exchange} (which only consists of interchanging adjacent operations) the two interchanged operations are inserted into the tabu list and a reverse interchange is declared as tabu.

Tabu list management. For the tabu list we have implemented both a fixed length tabu list and a dynamical tabu list management (see Dell'Amico and Trubian [6]).

Neighbor selection. For selecting a non-tabu neighbor of a given solution, we have implemented two different strategies. The first is the most commonly used strategy. It selects the best non-tabu neighbor (*best-fit*). The second (*first-fit*) selects the first non-tabu neighbor which improves the current solution. If no improving solution exists, it selects the best non-improving.

Termination. The tabu search method stops if for a certain number ($k_{\max}$) of iterations the best found solution has not been improved.

5. Computational results

In this section we report on some results of a computational study achieved with the developed local search approach. We implemented the methods in C and tested them on a Sun Sparc Station 10/512.

The aim of the study was to give our industrial partner some idea of what quality of solutions can be achieved with our solution method and how long it takes to generate them. Furthermore, since until now in the plant all batches of a processing order are scheduled one after the other (i.e., they produce solution on the base of the time-lag model), they wanted to get an idea of how much the quality of solutions achieved for the general shop model differs from the quality of the solutions for the time-lag model (i.e., whether or not it makes sense to do a more detailed planning). As a test data set our industrial partner gave us a set containing 54 orders which had to be produced on 30 machines (orders of 3 months). For each order the precedence relations between the processing tasks, the amount to be produced, and the deadline was given. In addition to the set of allowable machines and corresponding process times (in days), the respective machine preferred by the plant management was also given for each processing task. The set-up time of a machine is always one day.

Using these data we constructed three different instances. For the first the preferable machine assignment from practice was used (we call this instance INDUSTRY). For the second we fixed machines from the sets of allowable machines such that all machines were as much as possible used equally (EQUAL) and for the third a machine assignment was generated randomly (RANDOM). The first two of these instances are more of practical interest, whereas the third leads to unequal loads on the machines. To specify the objective function we decided to weight the violation of a deadline by a day 10 times higher than the costs for one set-up; i.e., we defined $W = 10$ and $C_{\text{set}} = 1$.

Several versions of the proposed local search approach were applied to the three instances. The versions differ in the chosen neighborhood, the tabu list management, the tabu list length, the neighbor selection, and the value of $k_{\max}$.

In the following we present some results of these tests and draw conclusions from them. Note, that these conclusions are related to instances of the type which occur at our industrial partner. For different types of instances or for adaption of the presented local search method to variations of the problem further tests have to show whether the given conclusions remain valid.

Firstly, we report on the results using only the time-lag model. As a general result, we can state that the first-fit neighbor selection outperformed the best fit: the same or even better results were attained using less computation time. Therefore, we will only present results for first-fit versions. Furthermore, on average there was almost no difference between the fixed tabu list length or a dynamical tabu list management. We have chosen to use a dynamical tabu list management with minimal tabu list length 0 and different values TL_{tl} for the maximal length of the tabu list ($TL_{tl} \in \{5, 10, 15, 20, 30, 50\}$). These values were selected on the basis of preliminary tests. Table 1 gives the average objective values taken over all six different tabu list lengths (see columns VAL) and the corresponding average computation times in minutes: seconds (see columns CPU) using the four different neighborhoods and two different values of $k_{\max}$ ($k_{\max} \in \{100, 1000\}$). Besides these values also the objective values of the initial solution calculated by a priority rule-based heuristic (INIT) and a lower bound for the best objective value for solutions of the time-lag model (LB) are given. The lower bound is based on a lower bound for the minimal number of set-ups on a machine (number of different orders on this machine minus 1) and on lower bounds on the tardiness of the orders which were calculated using some one-machine relaxations. Due to the decomposition and relaxations applied to obtain the lower bounds, it may be expected that the lower bounds are not very close to the optimal value. For the random instance this gap has to be larger since no good decomposition of orders to machines could be used to calculate the lower bound on the tardiness (for details, see Schönemann [11]).

Looking at all three instances, one can state that the shift neighborhoods were most successful in achieving good solutions. For the first two instances (which are of most interest in practical applications) the gap between the initial solution and the lower bound was drastically reduced and for the third instance a large improvement was also achieved (note that we suppose that the LB for this instance is quite bad and thus the achieved quality also seems to be quite good). Both versions of the exchange neighborhoods are only able to achieve good results for one instance. Regarding the computation times for a fixed value of $k_{\max}$, one observes that the exchange neighborhoods use less time than the shift neighborhoods (as expected). However, comparing the long runs of the exchange neighborhoods with the short runs of the shift neighborhoods one can conclude that one should prefer the shift neighborhoods, especially neighborhood N_{shift}, if one wants to solve the time-lag model.

Now we will report on the improvements which may be achieved by transforming

Table 1
Local search for the time-lag model.

Neighborhood	$k_{\max}$	INDUSTRY		EQUAL		RANDOM	
		VAL	CPU	VAL	CPU	VAL	CPU
INIT		48148		42514		59018	
N_{exchange}	100	16150	0:29	11228	0:31	33145	0:51
N_{exchange}	1000	16042	2:15	11007	2:48	30509	5:11
$\overline{N}_{\text{exchange}}$	100	14841	0:33	12208	0:37	28713	0:43
$\overline{N}_{\text{exchange}}$	1000	14777	3:54	12079	3:28	27709	5:38
N_{shift}	100	13915	1:31	10989	1:33	28191	2:50
N_{shift}	1000	13649	8:08	10957	9:18	27888	20:42
$\overline{N}_{\text{shift}}$	100	13901	2:45	11596	2:11	29207	6:39
$\overline{N}_{\text{shift}}$	1000	13639	15:51	11426	10:59	28435	41:08
LB		12147		9133		10653	

Table 2
Improvement by the conversion procedure.

Neighborhood	INDUSTRY		EQUAL		RANDOM	
	tl	CONV	tl	CONV	tl	CONV
N_{exchange}	16096	15048	11118	10224	31827	31778
$\overline{N}_{\text{exchange}}$	14809	14155	12144	11143	28211	28130
N_{shift}	13782	12711	10973	10126	28039	27989
$\overline{N}_{\text{shift}}$	13777	12686	11511	10606	28821	28758
LB	12147		9133		10653	

the solutions achieved with the time-lag model to general shop solutions as described in subsection 4.2. In table 2 the average improvement of the final solutions of the different versions of the tabu search heuristics by this conversion procedure can be found. In columns tl the average objective values over both $k_{\max}$ values and in columns CONV the average objective values after conversion are given.

For the two practical instances the conversion procedure improves the time-lag model solution by approximately 7%. However, for the random instance the improvement is only very small. The corresponding computation times to achieve the improvement are only a fraction of a second.

As a final step of the overall heuristic, we apply the tabu search approach to the general shop model using the converted solutions as an initial solution. Based on the results for the time-lag model and some preliminary tests we only used the neighborhood N_{shift} and we fixed the $k_{\max}$ value to 100 for the general shop model. The tabu list was managed as a list with fixed length TL_{gs} ($TL_{gs} \in \{5, 10, 15, 20\}$). For the neighbor selection we will report on both best-fit and first-fit strategies. As initial solutions for the second phase we used the solutions of the time-lag model achieved with $k_{\max} = 1000$ and values $10, 15, 20$, and 30 for TL_{tl}. The corresponding results

Table 3
Local search for the general shop model.

Neighb.	Neigh. Sel.	INDUSTRY		EQUAL		RANDOM	
		VAL	CPU	VAL	CPU	VAL	CPU
N_{exchange}	best-fit	13231	40 : 25	9881	30 : 21	30077	41 : 27
N_{exchange}	first-fit	13185	26 : 34	9802	23 : 35	30217	32 : 40
$\overline{N}_{\text{exchange}}$	best-fit	12775	50 : 15	10040	33 : 40	26266	39 : 14
$\overline{N}_{\text{exchange}}$	first-fit	12720	37 : 11	9819	22 : 07	25899	31 : 13
N_{shift}	best-fit	12493	35 : 18	9817	35 : 02	27492	53 : 11
N_{shift}	first-fit	12602	29 : 31	9842	28 : 51	27533	40 : 22
$\overline{N}_{\text{shift}}$	best-fit	12332	45 : 24	10455	39 : 40	28212	66 : 50
$\overline{N}_{\text{shift}}$	first-fit	12362	40 : 35	10679	29 : 41	28331	57 : 35
LB		12147		9133		10653	
BEST		11947		9618		25257	

are presented in table 3. Again, the table contains the average values over all possible combinations of values for TL_{tl} and TL_{gs}. Besides these results, in the last row the table contains the best objective value which was found within the computational tests (see column BEST).

Again, we can state that the neighbor selection process has no great influence on the quality of the solutions, however, for first-fit the corresponding computation times are shorter. Comparing the results of table 3 with those of the columns CONV of table 2, we see that the tabu search approach for the general shop model again reduces the objective values significantly. For the two relevant practical instances the best found value is below or close to the (not good) lower bound of the time-lag model (which is used in practice). Comparing the neighborhoods (used to calculate the initial solutions), we can state that after using the tabu search approach for the general shop model the differences get smaller and no real winner can be pointed out.

Returning to the questions which were the starting point of the computational study we can summarize the results as follows:

- For order acceptance (computation times $\leqslant$ 10 minutes) one should use the tabu search approach on the basis of the time-lag model using the neighborhood N_{shift}, and afterwards convert the resulting solution to a solution of the general shop model.
- For process planning the overall heuristic should be applied using the neighborhoods $\overline{N}_{\text{exchange}}$ or N_{shift} for the time-lag model, and the neighborhood N_{shift} for the general shop model.
- The quality of the solutions achieved is good. Even within the time available for order acceptance we are able to achieve for the two relevant practical instances solutions close to the lower bound of the time-lag model. With longer computational times for one instance we can even beat the lower bound.
- By switching to a more detailed planning (i.e., switching from the time-lag model to the general shop model), the company can realize better schedules.

- The chosen machine assignment at the plant is not the best. It should be investigated whether or not even better assignments than our EQUAL distribution can be constructed.

6. Conclusion

We have presented a method to model a chemical batch scheduling problem as a discrete optimization problem and have presented a two-phase tabu search procedure to solve it. The method has been tested on data provided by our industrial partner. The computational results which are based on given assignments of processing tasks to processors are very promising. It seems that these results can still be improved by considering the assignment as part of the decision process. Research in this direction is an interesting topic for future work.

Acknowledgements

We acknowledge the fruitful cooperation with Bayer A.G. We also thank Carsten Schönemann for implementing the proposed procedures. Furthermore, the authors are grateful to the anonymous referees for their constructive comments.

References

[1] E.H.L. Aarts and J.K. Lenstra, eds., *Local Search in Combinatorial Optimization* (Wiley, Chichester, 1997).
[2] F. Blömer and H.-O. Günther, LP-based heuristics for scheduling chemical batch processes, Technical Report 04.98, Department of Industrial Management, Technical University Berlin (1998). To appear in Computers in Industry.
[3] P. Brucker and O. Thiele, A branch and bound method for the general-shop problem with sequence dependent setup-times, OR Spektrum 18 (1996) 145–161.
[4] R.E. Burkard, M. Hujter, B. Klinz, R. Rudolf and M. Wennink, A process scheduling problem arising from chemical production planning, Optimization Methods and Software 10 (1998) 175–196.
[5] R.E. Burkard, M. Kocher and R. Rudolf, Rounding strategies for mixed integer programs arising from chemical production planning, Yugoslav Journal of Operations Research 8 (1998) 9–23.
[6] M. Dell'Amico and M. Trubian, Applying tabu search to the job-shop scheduling problem, Annals of Operations Research 41 (1993) 231–252.
[7] E. Kondili, C.C. Pantelidis and R.W.H. Sargent, A general algorithm for short-term scheduling of batch operations – I. MILP formulation, Computers Chem. Enging. 17 (1993) 211–227.
[8] T. Löhl, C. Schulz and S. Engell, Sequencing of batch operations for a highly coupled production process: genetic algorithms versus mathematical programming, Computers Chem. Enging. 22 (1998) 579–585.
[9] G.V. Reklaitis, Overview of scheduling and planning of batch process operations, in: *Batch Processing Systems Engineering*, eds. G.V. Reklaitis et al. (Springer, Berlin, 1996) pp. 660–705.
[10] D.W.T. Rippui, Batch process systems engineering: a retrospective and prospective review, Computers in Industry (1993).
[11] C. Schönemann, Lokale Suche zur Lösung van Batch-Schedulingproblemen in der chemischen Industrie, Diplomarbeit, Universität Osnabrück (1998).

[12] O. Thiele, Ein Branch und Bound-Verfahren für das General-Shop Problem mit gruppenabhängigen Rüstzeiten, Diplomarbeit, Universität Osnabrück (1994).