A Cyclic Approach to Large-Scale Short-Term Planning of Multipurpose Batch Plants

Christoph Schwindt

Institute of Management and Economics
Clausthal University of Technology

Norbert Trautmann

Departement für Betriebswirtschaftslehre
University of Bern

1 Introduction

In the process industries, final products arise from chemical and physical transformations of materials on processing units. In batch production mode, the total requirements for intermediate and final products are divided into individual batches. To produce a batch, at first the input materials are loaded into a processing unit. Then a transformation process, called a *task*, is performed, and finally the output products are unloaded from the processing unit. Typically, a plant is operated in batch production mode when a large number of different products are processed on multi-purpose equipment. That is why we consider multi-purpose processing units, which can operate different tasks. Symmetrically, a task may be executed on different processing units, in which case the duration of the task may depend on the processing unit used. For a practical example of a multi-purpose batch production plant we refer to the case study presented by Kallrath (2002).

The minimum and maximum filling levels of a processing unit give rise to lower and upper bounds on the batch size. The input and the output proportions of the products consumed or produced, respectively, by a task are either fixed or variable within prescribed bounds. In general, storage facilities of limited capacity are available for stocking raw materials, intermediates, and final products. Some products are perishable and must be consumed immediately after production.

Between consecutive executions of different tasks on a processing unit, a changeover with sequence-dependent duration is necessary. Since the changeover times may be considerably large, the plant is generally configured according to a subset of the required final products. Before processing the next set of final products, the plant has to be reconfigured,

which requires the completion of all operations. In this context, the objective of makespan minimization is particularly important for ensuring a high resource utilization and short customer lead times. Given the primary requirements for final products, the short-term planning problem studied in this paper consists in computing a feasible production schedule with minimum makespan.

Various solution methods for this problem are known from literature. Most of them follow a monolithic approach, which addresses the problem as a whole, starting from a mixed-integer linear programming formulation of the problem. In those models, the time horizon is divided into a given number of time periods, the period length being either fixed (time-indexed formulations, cf. e.g., Kondili et al. 1993) or variable (continuous-time formulations, see e.g., Ierapetritou and Floudas 1998 or Castro et al. 2001). The main disadvantage of the monolithic approaches is that the CPU time requirements for solving real-world problems tend to be prohibitively high (cf. Maravelias and Grossmann 2004). To overcome this difficulty, different heuristics reducing the number of variables have been developed (cf. e.g., Blömer and Günther 1998).

A promising alternative approach is based on the decomposition of the short-term planning problem into interdependent subproblems. Such decomposition methods have, for example, been proposed by Brucker and Hurink (2000), Neumann et al. (2002), and Maravelias and Grossmann (2004). The solution approach developed in what follows is based on the hierarchical decomposition into a batching and a batch-scheduling problem presented in Neumann et al. (2002). Batching provides a set of batches for the intermediate and final products needed to satisfy the primary requirements. Batch scheduling allocates the processing units, intermediates, and storage facilities over time to the processing of the batches arising from the batching step. The batching problem can be formulated as a mixed-integer nonlinear program (MINLP) of moderate size, which can be solved using standard mathematical programming software. For solving the batch-scheduling problem, a truncated branch-and-bound method and a priority-rule-based method have been developed by Neumann et al. (2002) and Schwindt and Trautmann (2004), respectively. Within a reasonable amount of CPU time, good feasible solutions to problem instances with up to 100 batches can be computed with both methods. Recently, Gentner et al. (2004) have proposed a decomposition of the batch-scheduling problem which partitions the set of all batches into a sequence of subsets. The assignment of the batches to the individual subsets is determined stepwise by solving a binary linear program in each iteration. This decomposition method is able to approximatively solve batch-scheduling instances with

up to about 3000 batches in the space of several hours of CPU time (cf. Gentner et al. 2004 and Gentner 2005).

In this paper, we present a cyclic scheduling approach to the short-term planning problem. A preliminary version of this method can be found in Trautmann (2005). The basic idea consists in reducing the size of the batch-scheduling problem by computing a cyclic subschedule, which is executed several times. The set of batches belonging to one cycle is determined by solving an MINLP, which also provides the number of cycles needed to satisfy the primary requirements (*cyclic batching problem*). To guarantee that the resulting batch-scheduling problem remains tractable, we impose an upper bound on the number of batches per cycle. The subschedule is then obtained by scheduling the batches on the processing units subject to material-availability and storage-capacity constraints (*cyclic batch-scheduling problem*). The latter problem is solved using the priority-rule based method proposed in Schwindt and Trautmann (2004).

The remainder of this paper is organized as follows. In Section 2 we formulate the cyclic batching problem as an MINLP and briefly discuss structural issues. Section 3 is devoted to the cyclic batch-scheduling problem and the generation of a complete production schedule by efficiently concatenating copies of the subschedule. In Section 4 we report on results of an experimental performance analysis.

2 Cyclic Batching

Let T be the set of all tasks and let β_τ and ε_τ be the batch size and the number of batches for task $\tau \in \mathrm{T}$. By Π_τ^- and Π_τ^+ we denote the sets of input and output products, respectively, of task $\tau \in \mathrm{T}$. $\Pi_\tau := \Pi_\tau^- \cup \Pi_\tau^+$ is the set of all input and output products of task τ, and $\Pi := \cup_{\tau \in \mathrm{T}} \Pi_\tau$ is the set of all products considered. In addition to β_τ and ε_τ, the (negative) proportions $\alpha_{\tau\pi} < 0$ of all input products $\pi \in \Pi_\tau^-$ and the (positive) proportions $\alpha_{\tau\pi} > 0$ of all output products $\pi \in \Pi_\tau^+$ have to be determined for all tasks $\tau \in \mathrm{T}$ such that

$$\sum_{\pi \in \Pi_\tau^+} \alpha_{\tau\pi} = -\sum_{\pi \in \Pi_\tau^-} \alpha_{\tau\pi} = 1 \quad (\tau \in \mathrm{T}) \tag{1}$$

Proportions $\alpha_{\tau\pi}$ and batch sizes β_τ have to be chosen within prescribed intervals $[\underline{\alpha}_{\tau\pi}, \overline{\alpha}_{\tau\pi}]$ and $[\underline{\beta}_\tau, \overline{\beta}_\tau]$, i.e.,

$$\underline{\alpha}_{\tau\pi} \le \alpha \le \overline{\alpha}_{\tau\pi} \quad (\tau \in \mathrm{T},\ \pi \in \Pi_\tau) \tag{2}$$

$$\underline{\beta}_\tau \le \beta \le \overline{\beta}_\tau \quad (\tau \in \mathrm{T}) \tag{3}$$

Let T_π^- and T_π^+ be the sets of all tasks consuming and producing, respectively, product $\pi \in \Pi$ and let $\Pi^p \subset \Pi$ be the set of perishable products. Then equations

$$\alpha_{\tau\pi}\beta_\tau = -\alpha_{\tau'\pi}\beta_{\tau'} \quad (\pi \in \Pi^p, (\tau, \tau') \in \mathrm{T}_\pi^+ \times \mathrm{T}_\pi^-) \tag{4}$$

ensure that the amount of product $\pi \in \Pi^p$ produced by one batch of some task $\tau \in \mathrm{T}_\pi^+$ can immediately be consumed by any task $\tau' \in \mathrm{T}_\pi^-$ consuming π.

By $\Pi^i \subset \Pi$ we denote the set of intermediates. In order to obtain a cyclic solution, which allows us to execute the same subschedule an arbitrary number of times, the amount of an intermediate π produced within one cycle must be equal to the amount of π consumed, i.e.,

$$\sum_{\tau \in \mathrm{T}_\pi} \alpha_{\tau\pi}\beta_\tau\varepsilon_\tau = 0 \quad (\pi \in \Pi^i) \tag{5}$$

Proportions $\alpha_{\tau\pi}$, batch sizes β_τ, and the numbers of batches ε_τ define the set of batches belonging to one cycle. The number of cycles needed is a decision variable $\nu \in \mathbb{Z}_{\geq 0}$ whose value depends on the given primary requirements for final products. Let $\Pi^f \subset \Pi$ be the set of all final products and let ϱ_π be the primary requirement less the initial stock of product $\pi \in \Pi^f$. The final inventory of product π then equals $\nu \sum_{\tau \in \mathrm{T}} \alpha_{\tau\pi}\beta_\tau\varepsilon_\tau$. This amount must be sufficiently large to match the requirements ϱ_π for π, i.e.,

$$\nu \sum_{\tau \in \mathrm{T}_\pi} \alpha_{\tau\pi}\beta_\tau\varepsilon_\tau \geq \varrho_\pi \quad (\pi \in \Pi^f) \tag{6}$$

In addition, the number of batches within one cycle must not exceed the prescribed upper bound $\bar{\varepsilon}$, i.e.,

$$\sum_{\tau \in \mathrm{T}_\pi} \varepsilon_\tau \leq \bar{\varepsilon} \tag{7}$$

Finally, let p_τ be the mean processing time of task τ on the alternative processing units. To minimize the workload to be scheduled in the batch-scheduling step, the objective function is chosen to be the total mean processing time $\nu \sum_{\tau \in \mathrm{T}} p_\tau\varepsilon_\tau$. In sum, the cyclic batching problem reads

$$\text{(C-BP)} \quad \begin{cases} \text{Minimize} & \nu \sum_{\tau \in \mathrm{T}} p_\tau\varepsilon_\tau \\ \text{subject to} & (1) \text{ to } (7) \\ & \varepsilon_\tau \in \mathbb{Z}_{\geq 0} \quad (\tau \in \mathrm{T}) \\ & \nu \in \mathbb{Z}_{\geq 0} \end{cases}$$

For a given value of ν, problem (C-BP) can be transformed into a mixed-binary linear program with binary decision variables θ_τ^μ ($\tau \in \mathrm{T}$, $\mu = 1, \ldots, \overline{\varepsilon}$) being equal to one exactly if $\varepsilon_\tau \geq \mu$ and continuous decision variables $\xi_{\tau\pi}^\mu$ ($\tau \in \mathrm{T}$, $\pi \in \Pi_\tau$, $\mu = 1, \ldots, \overline{\varepsilon}$) with $\xi_{\tau\pi}^\mu = \alpha_{\tau\pi}\beta_\tau$ if $\theta_\tau^\mu = 1$ and $\xi_{\tau\pi}^\mu = 0$, otherwise (see Neumann et al. 2002 for details). Now suppose without loss of generality that $\max_{\pi \in \Pi^f} \rho_\pi > 0$. Due to $\nu > 0$ for any feasible solution to (C-BP), inequality (6) can be rewritten as

$$\frac{\rho_\pi}{\nu} - \sum_{\tau \in \mathrm{T}_\pi} \sum_{\mu=1}^{\overline{\varepsilon}} \xi_{\tau\pi}^\mu \leq 0 \quad (\pi \in \Pi^f) \tag{8}$$

Let ($\overline{\text{C-BP}}$) denote the continuous relaxation of the reformulated batching problem with decision variables θ_τ^μ and $\xi_{\tau\mu}^\mu$. Since for each $\pi \in \Pi^f$ the left-hand side of (8) is a convex function and because all remaining constraints of ($\overline{\text{C-BP}}$) are linear, the feasible region of ($\overline{\text{C-BP}}$) is a convex set. Moreover, the objective function $\nu \sum_{\tau \in \mathrm{T}} p_\tau \sum_{\mu=1}^{\overline{\varepsilon}} \theta_\tau^\mu$ of ($\overline{\text{C-BP}}$) is increasing on the feasible region.

3 Cyclic Batch-Scheduling and Concatenation

In this section we explain our method for solving the batch-scheduling problem. In Subsections 3.1 and 3.2, where we closely follow the presentation of Schwindt and Trautmann (2004), we are concerned with the scheduling of the batches belonging to one cycle. In Subsection 3.3 we show how a complete production schedule for the execution of the ν cycles can be efficiently constructed from the cyclic subschedule.

3.1 Statement of the Cyclic Batch-Scheduling Problem

Recall that solving the batching problem has provided us with the set of batches belonging to one cycle. For what follows, the processing of a batch on a processing unit is called an *operation*. Suppose that $n = \sum_{\tau \in \mathrm{T}} \varepsilon_\tau$ operations $1, \ldots, n$ have to be scheduled. For notational convenience we introduce two fictitious operations 0 and $n+1$ representing the production start and the production end, respectively. $\widetilde{V} := \{1, \ldots, n\}$ is the set of all real operations, and $V := \widetilde{V} \cup \{0, n+1\}$ is the set of all operations. Let $S_i \geq 0$ be the *start time* sought of operation i. Then S_{n+1} coincides with the production makespan, and vector $S = (S_i)_{i \in V}$ with $S_0 = 0$ is called a *schedule*.

Each **processing unit** can be viewed as a unit-capacity *renewable resource with changeovers* (cf. Neumann et al. 2003, Sect. 2.14). Let $\mathcal{R}^\rho$ be the set of all renewable resources and $\mathcal{R}_i^\rho$ be the set of those alternative renewable resources on which operation i can be carried out. For $i \in \widetilde{V}$ and $k \in \mathcal{R}_i^\rho$, the binary decision variable x_{ik} indicates whether or not resource k processes operation i ($x_{ik} = 1$ or $x_{ik} = 0$, respectively). Each real operation i must be executed on exactly one processing unit, i.e.,

$$\sum_{k \in \mathcal{R}_i^\rho} x_{ik} = 1 \quad (i \in \widetilde{V}) \tag{9}$$

Vector $x = (x_{ik})_{i \in \widetilde{V}, k \in \mathcal{R}_i^\rho}$ is called an *assignment* of operations i to processing units k. By $p_i(x)$ and $c_{ij}(x)$ we denote the processing time of operation i and the changeover time from operation i to operation j given assignment x, where we suppose that $p_0(x) = p_{n+1}(x) = 0$ and $c_{0i}(x) = c_{i(n+1)}(x) = 0$ for all $i \in \widetilde{V}$.

Given a resource $k \in \mathcal{R}^\rho$, a schedule S and an assignment x, let $P_k(S, x)$ designate the set of all pairs (i, j) such that $i \neq j$, $x_{ik} = x_{jk} = 1$, and $S_i \leq S_j$. Schedule S is called *process-feasible* with respect to assignment x if no two operations i and j overlap on a processing unit, i.e.,

$$S_j \geq S_i + p_i(x) + c_{ij}(x) \quad (k \in \mathcal{R}^\rho,\ (i, j) \in P_k(S, x)) \tag{10}$$

Now we turn to the **storage facilities**, which can be modeled as so-called *cumulative resources* (cf. Neumann and Schwindt 2002). For each storage facility we introduce one cumulative resource keeping its inventory. Let $\mathcal{R}^\gamma$ be the set of all cumulative resources. For each $k \in \mathcal{R}^\gamma$, a minimum inventory $\underline{R}_k$ (safety stock) and a maximum inventory $\overline{R}_k$ (storage capacity) are given. Assuming that each product π is stocked in a dedicated storage facility k and that no safety stocks are prescribed we obtain $\underline{R}_k = 0$ and $\overline{R}_k = \sigma_\pi$ for all $k \in \mathcal{R}^\gamma$, where σ_π is the storage capacity for product π (with $\sigma_\pi = 0$ if $\pi \in \Pi^p$). Each operation $i \in V$ has a demand r_{ik} for resource $k \in \mathcal{R}^\gamma$. If $r_{ik} > 0$, the inventory of resource k is replenished by r_{ik} units at time $S_i + p_i(x)$. If $r_{ik} < 0$, the inventory is depleted by $-r_{ik}$ units at time S_i. r_{0k} represents the initial stock level of resource k. Suppose that operation i corresponds to an execution of task τ and that resource k is dedicated to product π. The demand of operation i for resource k then is $r_{ik} = \alpha_{\tau\pi}\beta_\tau$.

Let $V_k^+ := \{i \in V \mid r_{ik} > 0\}$ and $V_k^- := \{i \in V \mid r_{ik} < 0\}$ be the sets of operations replenishing and depleting, respectively, the inventory of resource $k \in \mathcal{R}^\gamma$. Schedule S is said to be *storage-feasible* with respect to

assignment x if

$$\underline{R}_k \leq \sum_{i \in V_k^+ : S_i + p_i(x) \leq t} r_{ik} + \sum_{i \in V_k^- : S_i \leq t} r_{ik} \leq \overline{R}_k \qquad (k \in \mathcal{R}^\gamma,\ t \geq 0) \quad (11)$$

Usually, **temporal constraints** of the type $S_j - S_i \geq \delta_{ij}(x)$ for $(i,j) \in E$ with $E \subseteq V \times V$ have to be taken into account as well. The right-hand side $\delta_{ij}(x)$ is a *minimum time lag* between the start of operations i and j. If $\delta_{ij}(x) < 0$, then $-\delta_{ij}(x)$ can be interpreted as a *maximum time lag* between the start of operations j and i. In case of $\delta_{ij}(x) = p_i(x)$, the corresponding temporal constraint is referred to as a *precedence constraint*, and a time lag $\delta_{0i}(x)$ is called a *release date* for operation i. For each operation $i \in \widetilde{V}$ we set $\delta_{0i}(x) := 0$ and $\delta_{i(n+1)}(x) := p_i(x)$. Further time lags may be generated by applying constraint propagation techniques detecting temporal constraints that are satisfied by at least one optimal solution to the batch-scheduling problem. For example, for two operations i, j with $i < j$ belonging to the same task τ we can introduce the time lag $\delta_{ij}(x) = 0$, without loss of optimality.

Based on time lags $\delta_{ij}(x)$ for $(i,j) \in E$ we can compute *distances* $d_{ij}(x)$ between any two operations $i, j \in V$. Distances $d_{ij}(x)$ coincide with the minimum time lags between operations i and j that are implied by the prescribed time lags (see e.g., Neumann et al. 2003, Sect. 1.3). Given an assignment x, a schedule S satisfying

$$S_j \geq S_i + \delta_{ij}(x) \qquad ((i,j) \in E) \quad (12)$$

is called *time-feasible* with respect to x.

A schedule which is time-, process-, and storage-feasible with respect to a given assignment x is called *feasible* with respect to x. A pair (S, x) is a feasible solution to the cyclic batch-scheduling problem if x is an assignment and S is a feasible schedule with respect to x. The cyclic batch-scheduling problem consists in finding a feasible solution (S, x) with minimum makespan S_{n+1}, i.e.,

$$\text{(C-BSP)} \quad \begin{cases} \text{Minimize} & S_{n+1} \\ \text{subject to} & (9) \text{ to } (12) \\ & S_0 = 0 \\ & x_{ik} \in \{0,1\} \quad (i \in \widetilde{V},\ k \in \mathcal{R}_i^\rho) \end{cases}$$

3.2 Priority-Rule Based Method

The basic idea of the priority-rule based solution method is as follows. At first, we choose an assignment x of operations to processing units, where

we balance the workload to be processed on alternative processing units by using a simple greedy heuristic. The method then consists of two phases. During the first phase, we relax the storage-capacity constraints. Using a serial schedule-generation scheme, the operations are iteratively scheduled on the processing units in such a way that the inventory does not fall below the safety stock at any point in time. Based on the resulting schedule, precedence constraints between replenishing and depleting operations are introduced according to a FIFO strategy. Those precedence constraints ensure that the material-availability constraints are always satisfied. In the second phase, which again applies the serial schedule-generation scheme, the operations are scheduled subject to the storage-capacity and the precedence constraints introduced.

In the remainder of this subsection we explain the schedule-generation scheme of the first phase in more detail. We then briefly sketch the modifications needed for using the scheme in the second phase. Since assignment x has been fixed, we omit x in the notation of the processing times and time lags.

Let $Pred(j)$ be the set of predecessors of node j with respect to the strict order $\{(i,j) \in V \times V \mid d_{ij} \geq 0 \text{ and } d_{ji} < 0\}$. It holds that $i \in Pred(j)$ precisely if operation i must be started no later than operation j but conversely, operation j may be started after operation i. Moreover, let $\mathcal{C}$ be the *completed set* of operations i already scheduled in prior iterations and let $S^{\mathcal{C}} := (S_i)_{i \in \mathcal{C}}$ be the *partial schedule* constructed. We say that an operation $j \notin \mathcal{C}$ is *eligible* for being scheduled if (i) all of its predecessors have been scheduled, i.e., $Pred(j) \subseteq \mathcal{C}$ and (ii), there is no cumulative resource k whose inventory level falls below the safety stock after the completion of all operations from set $\mathcal{C} \cup \{j\}$, i.e., $r_k(S^{\mathcal{C}}, \infty) + r_{jk} \geq \underline{R}_k$ for all $k \in \mathcal{R}^\gamma$.

The procedure is now as follows (see Algorithm 1). At first, we initialize the earliest and latest start times ES_i and LS_i for all $i \in V$. In each iteration of the schedule-generation scheme we then determine the set $\mathcal{E}$ of eligible operations j, select one operation $j^* \in \mathcal{E}$ according to priority indices $\pi(j)$, determine the earliest feasible start time $t^* \geq ES_{j^*}$ for operation j^*, schedule j^* at time t^*, and update the earliest and latest start times of the operations i not yet scheduled. Starting with partial schedule $S^{\mathcal{C}}$ where $\mathcal{C} = \{0\}$ and $S_0 = 0$ we perform those steps until all operations have been scheduled, i.e., until $\mathcal{C} = V$.

Sometimes it may happen that due to maximum time lags between scheduled operations $i \in \mathcal{C}$ and the operation j^* selected, the latest start time LS_{j^*} of j^* is strictly smaller than time t^*. Then no feasible start time can be found for operation j^*, and $S^{\mathcal{C}}$ cannot be extended to a feasible schedule. In this case, we perform the following unscheduling step. At

first, we determine the set $\mathcal{U} = \{i \in \mathcal{C} \mid LS_j^* = S_i - d_{j^*i}\}$ of all operations i that must be delayed for being able to schedule j^* at time t^*. Then, we increase the earliest start times of operations i from set $\mathcal{U}$ by adding the release dates $\delta_{0i} = S_i + t^* - LS_{j^*}$, update the distances d_{ij} accordingly, and restart the scheduling procedure. In the implementation shown in Algorithm 1, the number u of unscheduling steps is limited by some upper bound $\overline{u}$.

Algorithm 1: Schedule-generation scheme of phase 1

$u := 0$;
2: $S_0 := 0$, $\mathcal{C} := \{0\}$;
for all $i \in V$ **do** (* initialize ES_i and LS_i *)
  $ES_i := d_{0i}$, $LS_i := -d_{i0}$;
while $\mathcal{C} \neq V$ **do**
  $\mathcal{E} := \{j \in V \setminus \mathcal{C} \mid Pred(j) \subseteq \mathcal{C},\ r_k(S^{\mathcal{C}}, \infty) + r_{jk} \geq \underline{R}_k \text{ for all } k \in \mathcal{R}^\gamma\}$;
  $j^* := \min\{j \in \mathcal{E} \mid \pi(j) = \text{ext}_{h \in \mathcal{E}} \pi(h)\}$;
  $t' := \min\{t \geq ES_{j^*} \mid r_k(S^{\mathcal{C}}, \tau) + r_{j^*k} \geq \underline{R}_k \text{ for all } k \in \mathcal{R}^\gamma,\ \tau \geq t\}$;
  $t^* := \min\{S_{j^*} \geq t' \mid S^{\mathcal{C} \cup \{j^*\}} \text{ is process-feasible}\}$;
  if $t^* > LS_{j^*}$ **then** (* unschedule and restart *)
    $u := u + 1$;
    if $u > \overline{u}$ **then** terminate;
    $\mathcal{U} := \{i \in \mathcal{C} \mid LS_{j^*} = S_i - d_{j^*i}\}$;
    for all $i \in \mathcal{U}$ **do** $d_{0i} := S_i + t^* - LS_{j^*}$;
    update distances d_{ij} for all $i, j \in V$ and **goto** line 2;
  else (* schedule j^* at time t^* *)
    $S_{j^*} := t^*$, $\mathcal{C} := \mathcal{C} \cup \{j^*\}$;
    for all $j \in V \setminus \mathcal{C}$ **do** (* update ES_j and LS_j *)
      $ES_j := \max(ES_j, S_{j^*} + d_{j^*j})$;
      $LS_j := \min(LS_j, S_{j^*} - d_{jj^*})$;
return S;

After having obtained a time- and process-feasible schedule satisfying the material-availability constraints, we link producing and consuming operations according to a FIFO strategy. This means that for each $k \in \mathcal{R}^\gamma$ we iterate the replenishing operations $i \in V_k^+$ according to nondecreasing completion times $S_i + p_i$ and allot the corresponding r_{ik} units to depleting operations $j \in V_k^-$ in the order of nondecreasing start times S_j. For each pair $(i, j) \in V_k^+ \times V_k^-$ for which j consumes units produced by i, we introduce a precedence constraint between i and j by setting $\delta_{ij} := \max(\delta_{ij}, p_i)$. Subsequently, we update the distances d_{ij} and proceed with the second phase of our procedure.

When during the second phase we deal with storage-capacity instead of material-availability constraints, we define the eligible set to be $\mathcal{E} := \{j \in V \setminus \mathcal{C} \mid Pred(j) \subseteq \mathcal{C},\ r_k(S^{\mathcal{C}}, \infty) + r_{jk} \leq \overline{R}_k \text{ for all } k \in \mathcal{R}^\gamma\}$. In the definition of $\mathcal{E}$, we use the predecessor sets $Pred(j)$ from the first phase in order to allow the scheduling of depleting operations before the replenishing operations allotted to them have been added to the partial schedule. The earliest storage-feasible start time of operation j^* is now given by $t' := \min\{t \geq ES_{j^*} \mid r_k(S^{\mathcal{C}}, \tau) + r_{jk} \leq \overline{R}_k \text{ for all } k \in \mathcal{R}^\gamma,\ \tau \geq t + p_{j^*}\}$. In this way, we ensure that any partial schedule $S^{\mathcal{C}}$ is feasible.

3.3 Concatenation

For generating the complete production schedule we proceed as follows. The (sub-)schedule S computed by the priority-rule based method defines precedence relationships between the operations i, j of one cycle being executed on the same processing unit or producing and consuming the same product. Those precedence relationships are translated into time lags δ_{ij}, which ensure that no resource conflict can occur when left- or right-shifting the operations. More precisely, for each pair of operations (i, j) with $S_j \geq S_i + p_i(x) + c_{ij}(x)$ and $x_{ik} = x_{jk} = 1$ for some $k \in \mathcal{R}^\rho$ we introduce the time lag $\delta_{ij} = p_i(x) + c_{ij}(x)$ preventing the overlapping of i and j. For pairs (i, j) with $S_j \geq S_i + p_i(x)$ and $r_{ik} > 0$, $r_{jk} < 0$ for some $k \in \mathcal{R}^\gamma$, the time lags $\delta_{ij} = p_i(x)$ guarantee the availability of the intermediate stocked in resource k. Eventually, we add the time lag $\delta_{ij} = -p_j(x)$ for each pair (i, j) with $S_j + p_j(x) \geq S_i$ and $r_{ik} < 0$, $r_{jk} > 0$ for some $k \in \mathcal{R}^\gamma$, to avoid an excess of the storage capacity of resource k. Moreover, the completion time of the last operation that is processed on a processing unit defines a release date δ_{0i} for the changeover to the first operation i on that unit in the next execution of the subschedule. Analogously, the last change in the inventory level of an intermediate gives rise to a release date δ_{0i} for the first operation i that subsequently produces or consumes that intermediate.

The start and completion times of the operations in the first cycle equal those of subschedule S. For computing the start and completion times of the operations in the next cycle, we solve a temporal scheduling problem which consists in computing an earliest schedule for those operations subject to the precedence relationships between and the release dates for the operations. As it is well-known, this temporal scheduling problem can be solved efficiently by longest path calculations. By iteratively concatenating the ν subschedule copies in this way, we finally obtain the production schedule sought.

4 Experimental Performance Analysis

We have compared the new heuristic to the decomposition approach by Gentner et al. (2004). The performance analysis was based on a test set introduced in Gentner (2005), which has been constructed by varying the primary requirements for final products in the case study of Kallrath (2002). For each instance, we have computed a solution to the cyclic batching problem using Frontline Systems' Solver package. The subschedules have been computed by a randomized multi-pass version of the priority-rule based method presented in Section 3. The tests have been performed on an 800 MHz Pentium III personal computer. The results for the decomposition approach have been reported in Gentner (2005) and refer to a 1400 MHz Pentium IV personal computer.

The results obtained for the 13 problem instances are shown in Table 1. For each problem instance the new method is able to find a markedly better solution. Especially for larger problem instances, the required CPU time is significantly smaller than for the decomposition approach. Having prescribed an upper bound of $\overline{\varepsilon} = 100$ batches, about 75 seconds are required for solving the cyclic batching problem. The priority-rule based method has been stopped after 15 seconds of CPU time. The concatenation has always required less than one second of CPU time.

Table 1: Computational results

Instance	Gentner (2005) Makespan	Gentner (2005) t_{cpu} [s]	This paper # batches	This paper Makespan	This paper t_{cpu} [s]
WeKa0_1	352	38	176	264	89
WeKa0_2	474	53	264	390	89
WeKa0_3	612	120	352	516	89
WeKa0_4	738	209	440	642	89
WeKa0_5	906	178	528	768	89
WeKa0_6	1046	215	616	894	90
WeKa0_7	1199	323	704	1020	91
WeKa0_8	1334	281	792	1146	91
WeKa0_9	1548	399	880	1272	91
WeKa0_10	1740	431	968	1398	91
WeKa0_15	2123	644	1408	2028	91
WeKa0_20	2899	1500	1848	2658	91
WeKa0_30	4416	5235	2728	3918	92

5 Conclusions

We have considered a short-term planning problem of batch production in the process industries. We have proposed a heuristic solution method for solving large-scale instances of this problem, consisting of the three steps cyclic batching, cyclic batch-scheduling, and concatenation. Because each of those steps has to be performed only once, the computational requirements of the heuristic are moderate. In an experimental performance analysis, we have shown that the new method clearly outperforms the best solution approach known from literature.

An important area of future research is, for example, the design of efficient solution methods for the case of continuous production, where the production and consumption rates of products are decision variables as well. Moreover, procedures for robust and reactive short-term planning should be developed, which are able to cope with uncertainty with respect to planning data like primary requirements, processing times, or yields.

References

[1] Blömer F, Günther HO (1998) Scheduling of a multi-product batch process in the chemical industry. Computers in Industry 36:245–259

[2] Brucker P, Hurink J (2000) Solving a chemical batch scheduling problem by local search. Annals of Operations Research 96:17–36

[3] Castro P, Barbosa-Póvoa AP, Matos H (2001) An improved RTN continuous-time formulation for the short-term scheduling of multipurpose batch plants. Industrial & Engineering Chemistry Research 40:2059–2068

[4] Gentner K (2005) Dekompositionsverfahren für die ressourcenbeschränkte Projektplanung. Shaker Verlag, Aachen

[5] Gentner K, Neumann K, Schwindt C, Trautmann N (2004) Batch production scheduling in the process industries. In: Leung JYT (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton, pp. 48/1–48/21

[6] Ierapetritou MG, Floudas CA (1998) Effective continuous-time formulation for short-term scheduling. 1. Multipurpose batch processes, Industrial & Engineering Chemistry Research 37:4341–4359

[7] Kallrath J (2002) Planning and scheduling in the process industry. OR Spectrum 24:219–250

[8] Kondili E, Pantelides CC, Sargent RWH (1993) A general algorithm for short-term scheduling of batch operations: I. MILP Formulation. Computers and Chemical Engineering 17:211–227

[9] Maravelias CT, Grossmann IE (2004) A hybrid MILP/CP decomposition approach for the continuous time scheduling of multipurpose batch plants. Computers and Chemical Engineering 28:1921–1949

[10] Neumann K, Schwindt C (2002) Project scheduling with inventory constraints. Mathematical Methods of Operations Research 56:513–533

[11] Neumann K, Schwindt C, Trautmann N (2002) Advanced production scheduling for batch plants in process industries. OR Spectrum 24:251–279

[12] Neumann K, Schwindt C, Zimmermann J (2003) Project Scheduling with Time Windows and Scarce Resources. Springer, Berlin

[13] Schwindt C, Trautmann N (2004) A priority-rule based method for batch production scheduling in the process industries. In: Ahr D, Fahrion R, Oswald M, Reinelt G (eds) Operations Research Proceedings 2003. Springer, Berlin, pp. 111–118

[14] Trautmann N (2005) Operative Planung der Chargenproduktion. Deutscher Universitäts-Verlag, Wiesbaden