A Discrete Time Exact Solution Approach for a Complex Hybrid Flowshop Scheduling Problem with limited Wait Constraints

Céline Gicquel, Laura Hege, Michel Minoux, W. van Canneyt

A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints

C. Gicquel[1a], L. Hege[a], M. Minoux[b], W. van Canneyt

[a] *Laboratoire de Genie Industriel, Grande Voie des Vignes, 92290 Chatenay-Malabry, France*
[b] *Laboratoire d'Informatique de Paris 6, 4 place Jussieu, 75005 Paris, France*

Abstract

We study a real-world complex hybrid flow-shop scheduling problem arising from a bio-process industry. There are a variety of constraints to be taken into account, in particular zero intermediate capacity and limited waiting time between processing stages. We propose an exact solution approach for this optimization problem, based on a discrete time representation and a mixed-integer linear programming formulation. The proposed solution algorithm makes use of a new family of valid inequalities exploiting the fact that a limited waiting time is imposed on jobs between two successive production stages. The results of our computational experiments confirm that the proposed method produces good feasible schedules for industrial instances.

Keywords:
Hybrid flow-shop scheduling, batch scheduling, limited-wait constraints, discrete-time model, mixed-integer linear programming, valid inequalities

1. Introduction

Hybrid flow-shop scheduling (HFS) problems arise in manufacturing environments in which a set of jobs are to be processed following the same production flow in a series of stages, each stage comprising several parallel machines. Hence, the main decisions in operating such a system consist in

[1]Corresponding author. E-mail: celine.gicquel@ecp.fr . Tel: (+33) 1 41 13 16 30. Fax: (+33) 1 41 13 12 72.

assigning jobs to machines and in scheduling jobs on machines at each stage of the production process while optimizing a given objective function.

As mentioned in [1], in the standard form of the HFS problem, machines at a given stage are identical, a machine can process only one operation at a time, each job has to be processed on exactly one machine at each stage, setup and removal times are negligible, buffer capacities between stages are unlimited and there is no restriction on the waiting time of a job between two stages.

In the present paper, we investigate a complex hybrid flow-shop scheduling problem arising from a bio-process industry. This problem significantly deviates from the standard form recalled above due to the presence of several additional constraints and assumptions to be taken into account, in particular:

- a job may be processed simultaneously by more than one machine at some stages of the production process, i.e. there exist multiprocessor tasks.
- there is no buffer capacity between stages so that if no machine is available at the next stage, a completed job is "blocked" in the previous stage and must wait on the machine which processed it.
- due to unstable intermediate products, there is a limited waiting time between operations of a given job ("no-wait" or "limited-wait" constraints).
- time-consuming setup and removal operations are required before and after the processing of a job, leading to positive setup and removal times.

The main contributions of this paper can be summarized as follows. First, we report a real-world complex hybrid flow-shop scheduling problem found in a bio-process industry, thus making a step toward reducing the gap between the theory and practice of scheduling mentioned in [2] and [3]. Second, we present an exact solution approach based on discrete time representation and mixed-integer linear programming. The proposed solution algorithm makes use of a new family of strong valid inequalities exploiting the fact that a limited waiting time is imposed on jobs between two successive production stages. As far as we know, this is the first time the issue of valid inequalities is

addressed in the context of limited waiting time constraints. Computational experiments confirm that the proposed method consistently produces good feasible schedules for industrial-size instances involving up to 35 jobs.

The rest of this paper is organized as follows. We provide in section 2 a detailed review of the related literature. In section 3, we describe the industrial scheduling problem in more detail. We then develop in section 4 the proposed exact solution approach and report in section 5 numerical results based on industrial data. Finally, we provide some concluding remarks in section 6.

2. Literature review

2.1. Hybrid flow-shop scheduling

Recent literature reviews on the HFS problem can be found in [1] and [3]. Both papers use the standard $\alpha|\beta|\gamma$ structure to describe the scheduling problem with respect to shop configuration (α), job constrains (β) and objective functions (γ). References are then classified according to the characteristics of the production system and/or the type of approach used to solve the optimization problem. The reader is referred to these papers for comprehensive surveys on the HFS literature. In the present literature review, we focus on those papers in which one of the complicating features mentioned in the introduction is considered.

The situation where an operation at a certain stage requires more than one machine ("multiprocessor task") is addressed among others in [4] and [5]. The authors of [6], [7] and [8] study the case of limited buffer capacities between production stages which leads to a "blocking scheduling problem" where a completed job remains on a machine and blocks it until a downstream machine becomes available. Contributions on hybrid flow-shop scheduling under no-wait constraints can be found for instance in [9] and [10]. We point out here that the case of *limited-wait constraints* has only received limited attention: noticeable exceptions can be found in [11], [12] and [13]. Finally, HFS with positive setup and/or removal times is studied e.g. in [14]. However, as mentioned in [2], even if there are several papers addressing realistic extensions of the hybrid flow-shop scheduling, very few papers consider several complicating features jointly. Moreover, as can be seen from the detailed survey presented in [1], there seems to be no previous attempt in the Operations Research literature to solve the variant of the HFS problem studied in the present paper, i.e. the HFS problem with multiprocessor tasks, zero

buffer capacity, limited waiting time between consecutive operations of a job and positive setup and removal times.

There is a wide variety of solution approaches for the HFS problem. The authors of [1] propose a classification into three broad classes: exact methods, heuristics and metaheuristics. Exact solution approaches for the HFS problem mostly rely on problem-specific Branch & Bound algorithms where nodes correspond to partial schedules and lower bounds are computed by exploiting specific properties of the HFS problem. But, as can be seen from the surveys presented in [1] and [15], research in this area has focused on simplified versions of the problem by considering either problems with a restricted number of processing stages (typically 2) or problems close to the standard form of the HFS. Moreover, existing Branch & Bound algorithms appear to be limited to situations where the objective is to minimize makespan or mean flow time. In the present paper, we handle a HFS problem involving a general number of processing steps, several complicating job constraints and a more complex criterion, namely total weighted tardiness. This is why we did not consider the development of an exact solution approach based on a specific Branch & Bound algorithm but rather chose to use a standard Branch & Bound algorithm with a tight mixed-integer programming linear formulation.

2.2. Batch scheduling in chemical processes

A variant of the HFS problem has been widely studied in the context of chemical process industries where the problem is often referred to as the "scheduling problem for multi-product multi-stage batch plants". Such plants usually produce similar chemical compounds by performing on each batch of product an identical sequence of processing operations defined by the product recipe. Each batch to be processed corresponds to a job in the HFS problem and has to be assigned to a production unit and scheduled on this unit at each stage of the production process.

Recent reviews on the chemical engineering literature on short-term scheduling of batch processes are provided in [16] and [17]. One of the main features used in these papers to distinguish among scheduling problems is the so-called *inventory storage policy*. Namely, unlike discrete manufacturing industries, chemical industries are commonly characterized by the use of buffer tanks to store intermediate liquid or powder products between production stages. Moreover, it is usually impossible to mix in the same storage tank different batches of the same product. The authors of [18] propose a classification of

storage policies in multi-stage batch plants based on the available storage capacity and the amount of time a batch can wait after its processing at a given stage before undergoing the next production operation. They identify three types of storage policies: (1) unlimited intermediate storage, (2) limited intermediate storage (i.e. there is a limited number of available storage tanks which can be either dedicated to a given type of product or shared among all products), (3) no intermediate storage (i.e. there is no intermediate storage tank but a batch, once processed by a production unit, can wait in this unit until a unit in the next stage becomes free to receive it). In terms of the time a batch can wait between two production operations, the authors of [18] distinguish between unlimited and limited waiting time policies and classify the "zero-wait" constraint as a special case of the limited waiting time policy. Contributions on multi-stage batch plants scheduling with limited intermediate storage capacity and/or zero waiting time can also be found among others in [19], [20], [21], [22]. In what follows, we study a multi-product multi-stage batch scheduling problem with no intermediate storage capacity and non-zero limited waiting time between operations. To the best of our knowledge, the case of non-zero limited waiting time has only been considered in [18] where, in contrast with the model investigated in the present paper, an approach based on a continuous-time formulation is proposed.

Optimization models for sequential batch scheduling can be broadly classified into two main groups based on the time representation. Discrete time models rely on a division of the scheduling horizon into a number of time intervals of identical duration and allow events such as the beginning or ending of the processing of a batch on a unit to happen only at the boundaries of these time intervals. Examples of work based on such an approach can be found in [23], [24], [25], [26] and [27]. One of the main advantages of discrete time models is that scheduling constraints have to be monitored only at specific and known time points. This reduces the problem complexity and makes the model structure simpler and easier to solve, particularly when limitations on renewable resources availability or on storage capacity must be taken into account. However, to achieve a suitable accuracy of the solution, it is usually needed to use a sufficiently small time interval, resulting in large-size combinatorial optimization problems. In contrast, continuous time models avoid an external discretization of the scheduling horizon and allow events to take place at any time instant. A first type of approach following this direction is based on the concept of time-slots which stands for a set of predefined time

intervals with unknown durations. The production schedule is obtained by assigning a batch to be processed to each time slot on each production unit. The authors of [21], [28] and [29] among others use slot-based models to solve short-term batch scheduling problems. A second type of approach is based on sequence-based models where the batch sequence on each unit is given by a set of binary variables defining precedence relations between batches. Contributions on sequence-based models can be found for instance in [18], [30] and [31]. Continuous-time models usually lead to smaller combinatorial optimization problems as compared to discrete time models. However resource constraints (such as the limitations on manpower availability or on storage capacity arising in our problem) are difficult to handle with continuous time formulations. Namely, as mentioned in [17], they require "the definition of more complicated constraints involving many big-M terms, which tends to increase the model complexity and the integrality gap and may negatively impact on the capabilities of the method". This is why we propose a discrete time representation to model and solve the batch scheduling problem under study.

3. Problem description

We now provide a detailed description of the industrial case study which provided the initial motivation for this work. The optimization problem consists in scheduling production for a bio-process in which fermentation techniques are used.

Bio-processes are processes that use living cells or microorganisms (bacteria, yeasts, fungi...) to obtain products such as antibiotics, antibodies or enzymes. Bio-processes often involve a processing step called fermentation. Fermentation consists in placing the living organisms together with nutrients in an appropriate environment where temperature, pressure and oxygen content are controlled so that their metabolism produces the expected material. These biochemical reactions result in the formation of many undesired by-products. Therefore, after the fermentation is completed, it is necessary to carry out recovery operations to separate the desired end product from the other residues and isolate it in its pure form. Optimization of production scheduling in such bio-processes have been studied e.g. in [32] for penicillin production and in [33] for intra-cellular enzyme production.

In the considered case study, the production process comprises four production stages: (1) fermentation step 1, (2) fermentation step 2, (3) broth

preparation, (4) recovery. The first step of fermentation consists in placing the microorganisms in a small tank (called a fermenter) where they start growing and reproducing. The cells are then transferred in a larger fermenter, containing previously sterilized nutrients and water. During this second step of the fermentation, the cells continue to grow and multiply, while producing the expected end product. When the fermentation is complete, the mixture of cells, nutrients, residues and end product, called the broth, is transferred in a preparation tank where it undergoes some transformations prior to recovery operations. The final step is the recovery where the desired end product is separated from the broth, using mainly filtration and purification techniques.

There is a variety of aspects that need to be considered when developing scheduling models for the above specific batch process. The main features are as follows:

1. This is a multi-product multi-stage sequential process. Production is represented by a predefined number of fixed-size batches which have to be processed under the same production flow according to similar recipes. There are four production stages with several identical production units at each stage. Recipes differ only with respect to the value of processing times and production unit requirements.

2. There is no intermediate storage capacity so that after being processed at a given stage, a batch is either transferred immediately to a production unit in the next stage or must wait in the current production unit until one becomes available at the next production stage. Moreover, waiting times between processing stages are limited. More precisely, in the application considered here, there are no-wait constraints between stages 1-2 and stages 2-3 and limited-wait constraints between stages 3-4.

3. There are some multiprocessor tasks at stage 3. Namely, depending on its size, a batch may require simultaneously one or two preparation tanks during the broth preparation.

4. There are sequence-independent setup and removal times at stages 1 and 2. Indeed, fermenters need to be sterilized and filled up with the appropriate cell culture medium prior to processing of a new batch and some cleaning operations are required in fermenters after the batch has been transferred to the next production stage. Moreover, there also are sequence-independent removal times at stage 3 due to the fact that during recovery operations, the volume of a batch is progressively transferred from the preparation tanks to the recovery unit. More precisely, the preparation tanks in which the batch is processed remain occupied during its recovery and are freed only when the

filtration process for the entire batch has been completed.

5. The setup of a fermenter at stage 2 of the production process requires a number of difficult manual operations to be carried out by the plant staff, especially during the first period of setup. Due to manpower restrictions, it is possible to start the setup of at most one fermenter in stage 2 during any specified time period.

The batch scheduling problem can thus be stated as follows: given a set of batches with specified due-dates, we seek to determine an assignment of batches to units and a sequence of the processing of the batches in units at each stage so that all operating constraints are satisfied and total weighted tardiness is minimized. According to the $\alpha|\beta|\gamma$ notation used in [1] and [3], the scheduling problem considered in the present paper can be noted: $FF_4, ((PM^{(i)})_{i=1}^4)|block,\ no-wait, size_{j3}, ST_{SI}, R_{SI}|\overline{T}^w$.

4. An exact solution approach

We now discuss a mathematical programming model to describe and solve the batch scheduling problem stated in section 3. We assume here that time is uniformly discretized, i.e. that the horizon is divided into planning periods of identical duration and that events such as the beginning or the end of the processing of a batch occur only at the beginning of a planning period. Moreover, we make use of the fact that identical parallel production units are available at each stage to reduce the model complexity. Namely, we do not treat the detailed assignment of batches to units at each stage s but rather use an aggregate representation of production capacity and consumption at stage s. This yields the detailed MILP formulation described in the next subsection.

4.1. MILP formulation

We wish to schedule production of a number of batches indexed $b = 1...B$ over a scheduling horizon divided into $t = 1...T$ periods of identical duration. Each batch is to be processed according to a production flow involving $s = 1...S$ processing stages, with U_s identical production units available at stage s. We denote M the amount of available manpower.

Parameters relative to batch b are denoted as follows:

- DD_b: due-date of batch b,
- w_b: weight of batch b,
- k_{bs}: number of units needed to process batch b at stage s ($k_{bs} > 1$ for

multiprocessor tasks),
- α_{bs}^{τ}: amount of manpower needed by batch b at stage s τ periods after the beginning of setup operations for this batch,
- σ_{bs}: setup time required on the production units of stage s before batch b can be transferred to stage s,
- π_{bs}: production time of batch b at stage s,
- ρ_{bs}: removal time (due to cleaning or transfer operations) needed on stage s after batch b has been transferred to stage $s+1$,
- δ_{bs}: maximum amount of time a batch can wait after completion of its processing at stage s before being transferred to stage $s+1$.
We use the following binary decision variables:

$$y_{bs}^{t} = \begin{cases} 1 & \text{if processing of batch } b \text{ at stage } s \text{ starts in period } t \\ 0 & \text{otherwise.} \end{cases}$$

Using these notation, the batch scheduling problem can be formulated as follows:

$$min \sum_{b=1}^{B} w_b \sum_{t=DD_b-\pi_{bs}+1}^{T} (t+\pi_{bS}-DD_b) y_{bS}^{t} \tag{1}$$

$$\sum_{t=1}^{T} y_{bs}^{t} = 1 \qquad \forall b, \forall s \tag{2}$$

$$\sum_{b=1}^{B} k_{bs} \Big[\sum_{\tau=1}^{t+\sigma_{bs}} y_{bs}^{\tau} - \sum_{\tau=1}^{t-\rho_{bs}} y_{b,s+1}^{\tau} \Big] \leq U_s \qquad \forall s = 1...S-1, \forall t \tag{3}$$

$$\sum_{b=1}^{B} k_{bS} \Big[\sum_{\tau=t-\pi_{bS}-\rho_{bS}+1}^{t+\sigma_{bS}} y_{bS}^{\tau} \Big] \leq U_S \qquad \forall t \tag{4}$$

$$\sum_{s=1}^{S} \sum_{b=1}^{B} \sum_{\tau=1}^{\sigma_{bs}+\pi_{bs}+\rho_{bs}} \alpha_{bs}^{\tau} y_{b,s}^{t-\tau+1+\sigma_{bs}} \leq M \qquad \forall t \tag{5}$$

$$y_{bs}^{t} + \sum_{\tau=1}^{t+\pi_{bs}} y_{b,s+1}^{\tau} \leq 1 \qquad \forall b, \forall s, \forall t \tag{6}$$

$$y_{bs}^{t} = y_{b,s+1}^{t+\pi_{bs}} \qquad \forall (b,s) \text{ s.t. } \delta_{bs} = 0, \forall t \tag{7}$$

$$y_{bs}^{t} + \sum_{\tau=t+\pi_{bs}+\delta_{bs}}^{T} y_{b,s+1}^{\tau} \leq 1 \qquad \forall (b,s) \text{ s.t. } \delta_{bs} > 0, \forall t \tag{8}$$

$$y_{bs}^t \in \{0,1\} \qquad \forall b, \forall s, \forall t \qquad (9)$$

The objective, minimizing the total weighted tardiness, is expressed by (1). Constraints (2) ensure that processing of batch b at stage s starts only once during the scheduling horizon. Constraints (3)-(4) are production capacity constraints: they state that the number of units required at stage s in period t cannot be greater than the total number of units available U_s. Note that a batch b requires k_s production units at stage s over the set of periods between the beginning of setup operations (i.e. σ_{bs} periods before its processing at stage s starts) and the end of removal operations (i.e. ρ_{bs} periods after it has been transferred to stage $s+1$ or its processing at stage S is finished). Constraints (5) ensure that, in each period, the total demand on manpower is bounded by the available amount of manpower. Constraints (6) guarantee that every batch b undergoes operations following the prescribed recipe, i.e. that processing of batch b at stage $s+1$ cannot begin before processing at stage s is completed. Constraints (7) guarantee that no-wait restrictions are respected. Similarly, (8) guarantee that the maximum waiting time between operations at stages s and $s+1$ is respected. The binary character of variables y_{bs}^t is enforced by (9).

4.2. Valid inequalities

The formulation proposed in subsection 4.1 enables us to solve exactly only small instances of the scheduling problems. A possible explanation for this lies in the observation that the linear relaxation of (1)-(9) only provides a poor approximation to the exact optimal integer solution values. In order to circumvent this difficulty, we investigate a way of strengthening the formulation based on the use of a new family of valid inequalities. More precisely, we exploit a feature of the considered scheduling problem which has received limited attention in the literature, namely the presence of *limited-wait constraints*, to derive the following valid inequalities.

Proposition 1
All solutions of (1)-(9) satisfy:

$$\forall (b,s) \text{ s.t. } \delta_{bs} > 0, \forall (t_1,t_2) \text{ s.t. } t_2 \geq t_1$$
$$\sum_{\tau=t_1}^{t_2} y_{bs}^\tau + \sum_{\tau=1}^{t_1+\pi_{bs}} y_{b,s+1}^\tau + \sum_{\tau=t_2+\pi_{bs}+\delta_{bs}}^{T} y_{b,s+1}^\tau \leq 1 \qquad (10)$$

Moreover, (10) can be interpreted as clique constraints associated with maximal cliques in an underlying incompatibility graph. □

Before proceeding to the proof, we briefly explain the idea underlying (10). We consider a batch b at stage s where the waiting time before transfer to stage $s+1$ is assumed to be bounded by δ_{bs}. Inequalities (10) state that starting the processing of batch b at stage s within the time interval $[t_1, t_2]$ is incompatible with starting the processing of batch b at stage $s+1$ before period $t_1 + \pi_{bs}$ or after period $t_2 + \pi_{bs} + \delta_{bs}$. In other words, if the starting date of processing of batch b at stage s is fixed within time interval $[t_1, t_2]$, there is a rather tight time window within which processing of batch b at stage $s+1$ may start, namely $[t_1 + \pi_{bs} + 1; t_2 + \pi_{bs} + \delta_{bs} - 1]$.

Proof

We now provide the proof for proposition 1. We arbitrarily choose a batch b and a stage s such that $\delta_{bs} > 0$ as well as two time periods (t_1, t_2) s.t. $t_2 \geq t_1$ and show that the corresponding inequality of type (10) is valid.
The proof relies on the fact that there are a number of explicit or implicit binary exclusion constraints in formulation (1)-(9) which corresponds to the edges of an underlying incompatibility graph.
We first derive from formulation (1)-(9) the following binary exclusion constraints:

$$y^t_{bs} + y^\tau_{bs} \leq 1 \qquad \forall t, \forall \tau \qquad (11)$$

$$y^t_{b,s+1} + y^\tau_{b,s+1} \leq 1 \qquad \forall t, \forall \tau \qquad (12)$$

$$y^t_{bs} + y^\tau_{b,s+1} \leq 1 \qquad \forall t \text{ s.t. } t \geq t_1, \forall \tau \text{ s.t. } \tau \leq t_1 + \pi_{bs} \qquad (13)$$

$$y^t_{bs} + y^\tau_{b,s+1} \leq 1 \qquad \forall t \text{ s.t. } t \leq t_2, \forall \tau \text{ s.t. } \tau \leq t_2 + \pi_{bs} + \delta_{bs} \qquad (14)$$

Indeed, (11) and (12) are induced by (2), (13) are induced by (6) and (14) by (8).
Now we consider the incompatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ defined as follows:
- there is a node $v = (s, t) \in \mathcal{V}$ for each binary decision variable $y^t_{bs}, t = 1..T$ and a node $v = (s+1, t) \in \mathcal{V}$ for each binary decision variable $y^t_{b,s+1}, t = 1..T$.
- there is an edge $a \in \mathcal{A}$ between any two nodes if the corresponding decisions are incompatible, i.e. if in system (11)-(14), there is a binary exclusion constraint linking the corresponding decision variables.

A set $C \subset \mathcal{V}$ is called a *clique* if each pair of nodes in C is connected by an edge. A *maximal clique* is a clique which is not properly contained in another clique. Each maximal clique in $\mathcal{G}$ gives rise to a valid inequality called a *maximal clique constraint* stating that the sum of corresponding binary variables should be less than or equal to 1.
We consider the node set $C_s = \{(s, t_1), ..., (s, t_2)\}$: thanks to constraints (11), we know that there is an edge between any two nodes of C_s, i.e. C_s is a clique. Similarly, the node set $C_{s+1} = \{(s+1, 1), ..., (s+1, t_1 + \pi_{bs}), (s+1, t_2 + \pi_{bs} + \delta_{bs}), ..., (s+1, T)\}$ is a clique (see constraints (12)).
We first show that the node set $C_s \cup C_{s+1}$ is a clique.
Namely, let $v = (s, \tau), \tau \in [t_1; t_2]$ be an arbitrarily chosen node in C_s and $v' = (s+1, \theta), \theta \in [1; t_1 + \pi_{bs}] \cup [t_2 + \pi_{bs} + \delta_{bs}; T]$ be an arbitrarily chosen node in C_{s+1}:
- either $\theta \in [1; t_1 + \pi_{bs}]$ and there is an edge between v and v' thanks to constraints (13),
- or $\theta \in [t_2 + \pi_{bs} + \delta_{bs}; T]$ and there is an edge between v and v' thanks to constraints (14).
Thus, there is an edge between any pair of nodes (v, v') such that $v \in C_s$ and $v' \in C_{s+1}$: $C_s \cup C_{s+1}$ is a clique.
We now show that the node set $C_s \cup C_{s+1}$ is a *maximal clique*. Namely, let us consider a node w not included in $C_s \cup C_{s+1}$:
- either $w = (s, \tau)$ with $\tau < t_1$: $C_s \cup C_{s+1} \cup \{w\}$ is not a clique as w is not linked to node $(s+1, \tau + \pi_{bs} + 1)$ which belongs to $C_s \cup C_{s+1}$,
- or $w = (s, \tau)$ with $\tau > t_2$: $C_s \cup C_{s+1} \cup \{w\}$ is not a clique as w is not linked to node $(s+1, \tau + \pi_{bs} + \delta_{bs} - 1)$ which belongs to $C_s \cup C_{s+1}$,
- or $w = (s+1, \tau)$ with $\tau \in [t_1 + \pi_{bs} + 1; t_2 + \pi_{bs} + 1]$: $C_s \cup C_{s+1} \cup \{w\}$ is not a clique as w is not linked to node $(s, \tau - \pi_{bs} - 1)$ which belongs to $C_s \cup C_{s+1}$,
- or $w = (s+1, \tau)$ with $\tau \in$: $[t_1 + \pi_{bs} + \delta_{bs} - 1; t_2 + \pi_{bs} + \delta_{bs} - 1]$: $C_s \cup C_{s+1} \cup \{w\}$ is not a clique as w is not linked to node $(s, \tau - \pi_{bs} - \delta_{bs} + 1)$ which belongs to $C_s \cup C_{s+1}$.
As a consequence, $C_s \cup C_{s+1}$ is not contained in another clique, i.e. it is a maximal clique providing the multiple exclusion constraint (10). □

We note that the maximality property is important since this guarantees that inequalities (10) are the strongest possible clique constraints based on (11)-(14). The computational results displayed in section 5 will confirm that significant strengthening of the linear relaxation is obtained when adding these constraints to the model.

4.3. Cut & Branch algorithm

The number of valid inequalities of type (10) grows very fast with problem size. Namely, for each batch b and each stage s such that $\delta_{bs} > 0$, there are 2^T valid inequalities of type (10). Therefore, it is not possible to include all of them in the formulation. They can however be generated as needed according to a cutting-plane strategy. This is what we did in our implementation of the solution procedure. More precisely, we devised the following cutting-plane algorithm (CPA) which we use at the root node of the Branch & Bound search tree to generate violated valid inequalities of type (10) and add them to the initial formulation (1)-(9).

Algorithm (CPA)

Step 1. Solve the linear relaxation of the problem and denote $\tilde{y}$ the optimal solution of the relaxed problem.

Step 2. For each batch b and each stage s such that $\delta_{bs} > 0$,
- for $t_1 = 1..T$, for $t_2 = 1...T$, compute:
  $LHS(t_1, t_2) = \sum_{\tau=t_1}^{t_2} \tilde{y}_{bs}^{\tau} + \sum_{\tau=1}^{t_1+\pi_{bs}} \tilde{y}_{b,s+1}^{\tau} + \sum_{\tau=t_2+\pi_{bs}+\delta_{bs}}^{T} \tilde{y}_{b,s+1}^{\tau}$.
- let $nc = 0$ and $test = 1$.
- while $nc < K$ and $test = 1$,
  - look for $V^{max} = LHS(t_1^{max}, t_2^{max}) = max_{(t_1,t_2)} LHS(t_1, t_2)$.
  - if $V^{max} > 1$,
    1) add the valid inequality corresponding to (t_1^{max}, t_2^{max}) to formulation (1)-(9).
    2) let $LHS(t_1^{max}, \tau) = LHS(\tau, t_2^{max}) = 0$ for $\tau = 1...T$.
    3) let $nc = nc + 1$.
  - else, let $test = 0$.

Step 3. If no violated inequalities are found in step 2, stop. Otherwise go back to step 1.

Algorithm (CPA) is intended to generate at each iteration the K most violated inequalities of type (10) for each batch b and each stage s such that $\delta_{bs} > 0$. However, preliminary computational experiments showed that, if nothing is done to prevent it, there is a rather high probability that a large number of similar cutting planes will be generated. This is explained by the fact that a valid inequality of type (10) is often violated by the current

continuous solution due to the values of a small subset of the $T + \delta_{bs} - 1$ variables involved in its expression.

For instance, we often have the case where, for the valid inequality corresponding to (t_1^{max}, t_2^{max}), $LHS(t_1^{max}, t_2^{max}) > 1$ due to the fact that $\sum_{\tau=t_1^{max}}^{t_2^{max}} y_{bs}^{\tau} + \sum_{\tau=1}^{t_1^{max}+\pi_{bs}} y_{b,s+1}^{\tau} > 1$ and $\sum_{\tau=t_2^{max}+\pi_{bs}+\delta_{bs}}^{T} y_{b,s+1}^{\tau} = 0$. In this situation, any valid inequality (t_1^{max}, θ_2) such that $\theta_2 > t_2^{max}$ will also be violated with a value $LHS(t_1^{max}, \theta_2) = LHS(t_1^{max}, t_2^{max})$ and might belong to the K cutting-planes generated during the current iteration of algorithm (CPA). This will lead to the unnecessary generation of a large number of similar cutting planes, each one having a rather low contribution to closing the integrality gap.

To avoid this, during a given iteration of (CPA), if a violated valid inequality is generated for the pair (t_1, t_2), we prevent the algorithm from generating any other cutting plane corresponding to (t_1, τ) or (τ, t_2). This is achieved by setting to 0 the corresponding entries in table LHS.

5. Computational results

We discuss the results of some computational experiments carried out to evaluate the impact of the formulation enhancements proposed in subsection 4.2 and to estimate the size of the largest instances of the problem that could be solved to optimality within reasonable computation time limit.

5.1. Test instances

We use the data from our industrial case study to create instances of various size.

Time is discretized in periods of identical duration (typically 4 hours) and the scheduling horizon comprises 2 to 4 weeks. The plant configuration is the same for all instances: there are $S = 4$ processing stages with $U_1 = 2$, $U_2 = 5$, $U_3 = 5$ and $U_4 = 1$.

The plant produces $P = 6$ different kinds of end products, i.e. there are $P = 6$ different recipes. Each recipe p corresponds to a given set of values for setup, processing and removal times (σ_s, π_s, ρ_s) as well as for production unit requirement (κ_s). Table 1 provides a detailed description of the recipes where all time values are expressed as integer multiples of time periods.

There are no-wait constraints after stages 1 and 2 (i.e. $\forall b, \delta_{b1} = \delta_{b2} = 0$) and limited-wait constraints after stage 3 (i.e. $\forall b, \delta_{b3} = 9$). Manpower availability is restricted to $M = 1$ team and manpower consumption arises

Table 1: Data relative to recipes

p	σ_1	π_1	ρ_1	κ_1	σ_2	π_2	ρ_2	κ_2	σ_3	π_3	ρ_3	κ_3	σ_4	π_4	ρ_4	κ_4
1	2	5	1	1	3	24	2	1	0	3	4	2	0	4	0	1
2	1	1	1	1	2	4	2	1	0	2	2	1	0	2	0	1
3	2	5	1	1	2	19	2	1	0	2	2	2	0	2	0	1
4	2	2	1	1	2	15	3	1	0	2	2	1	0	2	0	1
5	2	2	1	1	3	13	2	1	0	2	4	1	0	4	0	1
6	2	2	1	1	3	15	2	1	0	2	2	1	0	2	0	1

only during the first period of setup operations at stage 2 of the process. We thus have: $\alpha^{\tau}_{bs} = 1$ if $s = 2$ and $\tau = 1$, $\alpha^{\tau}_{bs} = 0$ otherwise.

The instances tested differ with respect to the following characteristics:

- the length of the scheduling horizon: $T \in \{100, 120, 140, 160\}$.
- the number of batches involved in the scheduling problem: $B \in \{20, 25, 30, 35\}$.
- the batch parameters. For each batch b, a recipe p is randomly chosen from a discrete uniform $DU(1, P)$ distribution. Due-dates occur every 20 periods, i.e. $\forall b, DD_b \in \{20, 40, 60, 80, 100, 120, 140, 160\}$. For each batch b, a due-date DD_b is randomly chosen from a discrete uniform distribution over the set of possible due-dates and a weight w_b is randomly chosen from a discrete uniform $DU(1, 2)$ distribution.

We generated two sets of instances:

- a set (set A) of 20 instances of various size to get a first evaluation of the proposed solution approach,
- a larger set (set B) of 50 instances involving 30 batches and 140 periods to confirm the conclusions of the first part of the computational study.

Table 2 provides for each set of instances the number of planning periods in the scheduling horizon T, the number of batches B as well as the resulting number of binary variables Var and constraints $Const$ in the initial formulation (1)-(9).

Table 2: Data relative to instances

Instances	T	B	Var	Const
A1-A5	100	20	4000	4580
A6-A10	120	25	6000	6640
A11-A15	140	30	8400	9340
A16-A20	160	35	11200	12320
B1-B50	140	30	8400	9340

5.2. Results for set A instances

We solved each instance of set A with a standard MILP software (CPLEX 11.1) using either the initial formulation (1)-(9) or the strengthened formulation obtained after running algorithm CPA. All tests were run on a Pentium 4 (2.8 GHz) with 504 Mo of RAM, running under Windows XP. We used the default settings of CPLEX MILP solver. This means that some cutting planes, among which are clique cuts, cover cuts and Gomory fractional cuts, are added automatically to the model.

The computational results obtained with the two formulations are displayed in table 3. We provide:

- Cut_{CPA}: the number of violated valid inequalities of type (10) generated by algorithm CPA and added to the strengthened formulation.
- Cut_{STD}: the number of cutting-planes generated by CPLEX standard cutting-plane algorithm at the root node of the Branch and Bound tree.
- Gap_0: the initial gap, i.e. the relative difference between the lower bound provided by the linear relaxation of the problem and the best integer solution found after 4 hours of computation. We use the value of the lower bound obtained after the cutting-plane generation carried out by CPLEX algorithm at the root node of the Branch & Bound tree has stopped.
- $Nodes$: the number of nodes of the search tree explored before a guaranteed optimal solution is found or the time limit of 4 hours is reached.
- CPU_{IP}: the time in seconds needed to find a guaranteed optimal solution when one has been found. CPU_{IP} comprises the time needed to strengthen the formulation using algorithm CPA.

- Gap_F: the gap between the best lower bound found and the best integer feasible solution obtained after 4 hours of computation.

Results from table 3 show that using the strengthened formulation enables us to significantly enhance the efficiency of the Branch and Bound procedure embedded in CPLEX. Namely:

- for small size instances (I1-I10), the average computation time to obtain a guaranteed optimal solution is significantly reduced from 2368s with the initial formulation to 850s with the strengthened formulation.

- for industrial size instances (I11-I20), the number of instances for which a guaranteed optimal solution could be found within 4 hours of computation is increased from 4 with the initial formulation to 7 with the strengthened formulation. Moreover, the average remaining gap after 4 hours of computation is decreased from 6.5% with the initial formulation to 2.7% with the strengthened formulation.

The main explanatory factor for this lies in the observation that the lower bounds provided by the strengthened formulation appears to be stronger that the ones provided by the initial formulation. Indeed, the initial gap Gap_0 is on average 8.5% with the initial formulation as compared to 6.1% with the strengthened formulation. As a consequence, the number of nodes explored by the Branch and Bound procedure before a guaranteed optimal solution is obtained or the computation time limits are reached is divided on average by a factor of 4.6 (from 3010 to 655).

However, in 5 out of the 20 instances of set A (A4, A5, A6, A14 and A20), the computation time needed to obtain a guaranteed optimal solution is greater with the strengthened formulation than with the initial formulation. This might be explained by the fact that the number of cutting planes generated by algorithm CPA is much larger that the number of cutting planes generated by the standard CPLEX algorithm. As a consequence, the linear program to be solved at each node of the Branch & Bound tree is larger and its resolution requires more time. In most cases, this increase of computation time at each node of the Branch & Bound tree is compensated by a significant decrease in the number of nodes to be explored before optimality is reached. But for a small subset of instances (especially the smaller ones), this compensation does not occur and the computation time with the strengthened formulation is larger than with the initial formulation.

	Initial formulation					Strengthened formulation					
	Cut_{STD}	Gap_0	$Nodes$	CPU_{IP}	Gap_F	Cut_{CPA}	Cut_{STD}	Gap_0	$Nodes$	CPU_{IP}	Gap_F
A1	172	8.0	1796	577	0	1508	8	6.5	434	493	0
A2	175	16.7	2460	1148	0	1258	9	14.1	817	987	0
A3	42	0	0	290	0	1292	33	0	0	289	0
A4	109	0	0	119	0	1349	16	0	0	409	0
A5	247	1.0	3	120	0	1621	5	0.5	0	166	0
A6	95	0	0	928	0	1284	17	0	0	1391	0
A7	244	7.3	3494	3864	0	2000	12	5.9	684	2162	0
A8	199	3.1	469	790	0	1852	10	2.4	6	403	0
A9	202	2.0	472	3780	0	1573	13	1.4	24	512	0
A10	374	4.4	27554	12063	0	2416	11	2.9	836	1692	0
A11	214	4.5	1059	8342	0	2253	33	2.9	114	1741	0
A12	598	6.3	7421	14400	5.2	3766	4	1.8	1879	5833	0
A13	249	3.5	207	1600	0	2293	14	2.2	1	559	0
A14	138	48.5	1096	3323	0	1208	16	47.8	862	5472	0
A15	401	12.4	9103	14400	11.6	2507	7	11.9	2950	14400	9.4
A16	451	8.0	2848	14400	7.4	4418	18	2.6	3703	8143	0
A17	679	14.6	1554	14400	13.5	4720	9	12.1	373	14400	11.9
A18	411	17.6	618	14400	16.8	3115	15	5.8	420	14400	5.2
A19	334	11.1	60	14400	10.8	2323	27	1.5	5	7314	0
A20	327	0	0	12356	0	2608	34	0	10	12765	0

Table 3: Results for set A instances

To make sure that the formulation strengthening obtained thanks to algorithm CPA has a statistically significant positive impact on the computation time and solution quality, we carried out additional computational experiments. We considered a larger set of industrial size instances and used a statistical test to check wether the observed improvements are significant.

5.3. Results for set B instances

The second set of instances comprises 50 industrial size instances involving 30 batches and 140 periods. The results obtained with the initial and strengthened formulations are displayed in Table 4 and 5. We first provide:

- $\#Feas$: the number of instances for which a feasible solution could be obtained within 4 hours of computation,
- $\#Opt$: the number of instances for which a guaranteed optimal solution could be obtained within 4 hours of computation.

Moreover, for each instance, we computed:

- $\Delta Gap_0 = Gap_0(IF) - Gap_0(SF)$ the difference between the initial gaps obtained with the initial and the strengthened formulations,
- $\Delta Nodes = Nodes(IF) - Nodes(SF)$ the difference between the numbers of nodes explored with the initial and the strengthened formulations,
- $\Delta CPU_{IP} = CPU_{IP}(IF) - CPU_{IP}(SF)$ the difference between the computation times with the initial and the strengthened formulations,
- $\Delta Gap_F = Gap_F(IF) - Gap_F(SF)$ the difference between the final gaps obtained with the initial and the strengthened formulations.

For each of these performance measures, we provide in table 5 the mean, minimum and maximum values.

Results from table 4 confirm that using the strengthened formulation enables us to significantly enhance the efficiency of the Branch and Bound procedure embedded in CPLEX. Namely:

- the number of instances that could be solved to proven optimality is doubled while using the strengthened formulation,

Table 4: Results for set B instances

	Initial formulation	Strengthened formulation
#Feas	45	49
#Opt	10	20

Table 5: Results for set B instances

	Mean value	Min/Max values
ΔGap_0	11.9	[-0.05; 98.6]
$\Delta Nodes$	2952	[0; 11345]
ΔCPU_{IP}	3043	[-2150;13181]
ΔGap_F	12.45	[-1.61;100]

- the mean decrease in computation time (CPU_{IP}) obtained while using the strengthened formulation is 3043s,
- the mean decrease in the final gap (Gap_F) obtained while using the strengthened formulation is 12.45%.

Moreover, we conducted a t-test (see e.g. [34]) to check whether the observed mean values of ΔCPU_{IP} and ΔGap_F are statistically significantly different from 0.

The results of these tests indicate that:

- the mean decrease in the computation time obtained while using the strengthened formulation belongs to the interval [1834; 4252] with a 95% probability.
- the mean decrease in the final gap obtained while using the strengthened formulation belongs to the interval [4.9%; 19.9%] with a 95% probability.

This confirms that the use of algorithm CPA to strengthen the initial formulation leads to a statistically significant improvement of the computational efficiency of the Branch and Bound procedure embedded in CPLEX solver.

6. Conclusion

We investigated a scheduling problem arising from a bio-process industry. The problem is a complex variant of the hybrid flow-shop problem in which there are multiprocessor tasks, zero buffer capacity, limited waiting time between consecutive operations of a job and positive setup and removal times. We proposed a mixed-integer linear programming formulation based on a discrete time representation. We then derived a new family of strong valid inequalities exploiting the fact that a limited waiting time is imposed on jobs between two successive production stages and used these valid inequalities to devise a Cut & Branch algorithm. The results of our computational experiments confirm that the proposed method consistently provides good feasible schedules for industrial size instances with a reasonable computation effort.

Among the possible research directions suggested by the present work, it might be worth investigating the development of a heuristic solution approach so as to be able to provide good feasible schedules for industrial size instances within a shorter computation time.

References

[1] Ruiz R, Vazquez-Rodriguez J. The hybrid flow shop scheduling problem. European Journal of Operational Research 2010;205:1–18.

[2] Ruiz R, Serifoglu F, Urlings T. Modeling realistic hybrid flowshop scheduling problems. Computers & Operations Research 2008;35:1151–75.

[3] Ribas I, Leisten R, Framinan J. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research 2010;37:1439–54.

[4] Oguz C, Ercan M, Edwin Cheng T, Fung Y. Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. European Journal of Operational Research 2003;110:390–403.

[5] Ying K, Lin S. Scheduling multistage hybrid flowshops with multiprocessor tasks by an effective heuristic. International Journal of Production Research 2009;13(1):3525–38.

[6] Sawik T. An exact approach for batch scheduling in flexible flow lines with limited intermediate buffers. Mathematical & Computer Modelling 2002;36:461–71.

[7] Wardono B, Fathi Y. A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. European Journal of Operational Research 2004;155:380–401.

[8] Wang X, Tang L. A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers. Computers & Operations Research 2009;36:907–18.

[9] Grabowski J, Pempera J. Sequencing of jobs in some production system. European Journal of Operational Research 2000;125(3):535–50.

[10] Wang Z, Xing W, Bai F. No-wait flexible flowshop scheduling with no-idle machines. Operations Research Letters 2005;33(6):609–14.

[11] Yang D, Chern M. A two-machine flowshop scheduling problem with limited waiting time constraints. Computers & Industrial Engineering 1995;28(1):63–70.

[12] Su L. A hybrid two-stage flowshop with limited waiting time constraints. Computers & Industrial Engineering 2003;47:409–24.

[13] Akkerman R, van Donk D, Gaalman G. Influence of capacity- and time-constraint intermediate storage in two-stage food production systems. International Journal of Production Research 2007;45(13):2955–73.

[14] Low C. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. Computers & Operations Research 2005;32:2013–25.

[15] Kis K, Pesch E. A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. European Journal of Operational Research 2005;164:592–608.

[16] Floudas C, Lin X. Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review. Computers & Chemical Engineering 2004;28:2109–29.

[17] Mendez C, Cerda J, Grossmann I, Harjunkoski I, Fahl M. State-of-the-art review of optimization methods for short-term scheduling of batch processes. Computers & Chemical Engineering 2006;30:913–46.

[18] Sundaramoorthy A, Maravelias C. Modeling of storage in batching and scheduling of multistage processes. Computers & Chemical Engineering 2006;30:913–46.

[19] Moon S, Park S, Kook Lee W. Mixed-integer linear programming model for short-term scheduling of a special class of multipurpose batch plants. Industrial & Engineering Chemistry Research 1999;38:2144–50.

[20] Lee D, Vassiliadis V, Park J. List-based treshold-accepting algorithm for zero-wait scheduling of multiproduct batch plants. Industrial & Engineering Chemistry Research 2002;41:6579–88.

[21] Liu Y, Karimi I. Scheduling multistage batch plants with parallel units and no interstage storage. Computers & Chemical Engineering 2008;32:671–93.

[22] Liu B, Wang L, Qian B, Jin Y. An effective hybrid particle swarm optimization for batch scheduling of polypropylene processes. Computers & Chemical Engineering 2010;34:518–28.

[23] Kondili E, Pantelides C, Sargent R. A general algorithm for short-term scheduling of batch operations-I. MILP forumation. Computers & Chemichal Engineering 1993;17(2):211–27.

[24] Shah N, Pantelides C, Sargent R. A general algorithm for short-term scheduling of batch operations-II. Computational issues. Computers & Chemical Engineering 1993;17(2):229–44.

[25] Blomer F, Gunther H. LP-based heuristics for scheduling chemical batch processes. International Journal of Production Economics 2000;38(5):1029–2051.

[26] Maravelias C, Grossmann I. Minimization of the makespan with a discrete-time state-task network formulation. Industrial & Engineering Chemistry Research 2003;42:6252–7.

[27] Burkard R, Hatzl J. Review, entensions and computational comparison of MILP formulations for scheduling of batch processes. Computers & Chemical Engineering 2005;29:1752–69.

[28] Moon S, Hrymak A. New MILP models for scheduling of multiproduct batch plants under zero-wait policy. Industrial & Engineering Chemistry Research 1996;35:3458–69.

[29] Lamba N, Karimi I. Scheduling parallel production lines with resource constraints. 1. Model formulation. Industrial & Engineering Chemistry Research 2002;41:779–89.

[30] Cera J, Henning G, Grossmann I. A mixed-integer linear programming model for short-term scheduling of single-stage multiproduct batch plants with parallel lines. Industrial & Engineering Chemistry Research 1997;36:1695–707.

[31] Mendez C, Cerda J. Optimal scheduling of batch plants satisfying multiple product orders with different due-dates. Computers & Chemical Engineering 2000;24:2223–45.

[32] Yuan J, Xue Y, Hu K, Wu H, Jia Q. On-line application oriented optimal scheduling for penicillin fed-batch fermentation. Chemical Engineering & Processing 2009;48:651–8.

[33] Samsatli N, Shah N. An optimization based design procedure for biochemical processes. Part II: detailed scheduling. Food and Bioproducts Processing: Transactions of the Institution of Chemical Engineers, Part C 1996;74(4):232–42.

[34] Crawley M. Statistics: an introduction using R. Chichester, UK: John Wiley & Sons; 2005.