KU LEUVEN

FACULTEIT ECONOMIE EN BEDRIJFSWETENSCHAPPEN

Scheduling hybrid flow shops with time windows

Yang F, Leus R.

KBI_1816

Scheduling hybrid flow shops with time windows

Fan Yang*, Roel Leus†
ORSTAT, Faculty of Economics and Business, KU Leuven
Naamsestraat 69, 3000 Leuven, Belgium

Abstract

Hybrid flow shops can be encountered in various industrial settings. In this paper we develop methods for scheduling hybrid flow shops with hard time windows. Specifically, we study a two-stage hybrid flow shop scheduling problem with time windows to minimize the total weighted completion times. Each stage consists of one or more identical parallel machines, and each job visits two processing stages in series. Finding a feasible schedule with hard time windows is a challenging task in this setting, because it is NP-complete in the strong sense even for a single machine in a single stage. We propose two matheuristics to find an initial feasible solution by local branching. We also develop two schedule improvement procedures, one based on stage-by-stage decomposition, and one using adapted local branching. The performance of our methods is validated via extensive computational experiments.

Keywords: hybrid flow shop, scheduling, time windows, matheuristic, local branching

1 Introduction

A hybrid flow shop consists of two or more processing stages, where each stage contains one or more machines in parallel. This type of production layout is used very frequently in the manufacturing industry, especially in process industries, for instance in the production of food, beverages, chemicals, textile, glass, pulp, rubber, plastic, and fabricated metal. Due to this wide occurrence, the optimization of the scheduling of hybrid flow shops has attracted the attention from many scholars (e.g., for the glass industry (Almada-Lobo et al., 2008; Liu et al., 2017), consumer goods (Baumann and Trautmann, 2013), potash (Schulze et al., 2016), steel (Pan, 2016), and electronics (Bang and

*ORCID: 0000-0001-5239-6762.
†Corresponding author. E-mail address: Roel.Leus@kuleuven.be, tel.: +32 16 32 69 67, ORCID: 0000-0002-9215-3914.

Kim, 2011; Shahvari and Logendran, 2018)). Apart from pure manufacturing settings, the two-stage hybrid flow shop has also been used as a model for other planning environments, such as the integration of production and delivery (e.g., concrete production (Garcia and Lozano, 2005), food catering (Chen and Vairaktarakis, 2005), perishable foods (Viergutz and Knust, 2014)), and quay crane and yard truck scheduling for inbound containers (Kaveshgar and Huynh, 2015). As a result, hybrid flow shop scheduling occupies an important role in contemporary operations management.

For any company, moreover, the assurance of on-time completion and delivery of customer orders is a key factor in its daily operations (Garcia and Lozano, 2005; Berghman and Leus, 2015). During operational scheduling, this assurance can be modelled as a time window constraint for each order, each with its own release date and deadline. Related studies on scheduling with time windows are scarce, and most previous literature has focused on a single machine environment (Pan and Shi, 2005; Jula and Kones, 2013; Davari et al., 2016). Chamnanlor et al. (2014) investigate a re-entrant hybrid flow shop, where the time window controls the time spent by a job in the different processing stages. To the best of our knowledge, there is no related work on general hybrid flow shop scheduling with hard time windows. There are, however, a number of articles that consider the minimization of tardiness (Garcia and Lozano, 2005; Huang and Yang, 2008; Berghman and Leus, 2015) instead of the use of hard time windows, which is an understandable choice as obtaining even simply a feasible solution is difficult with hard time windows, especially for large-sized instances.

In this paper, we study the minimization of the total weighted completion times in a two-stage hybrid flow shop, where each job must be processed between its ready time and deadline. Following the standard three-field notation of Graham et al. (1979), with additions by Haouari et al. (2006) and Pinedo (2016), our problem can be denoted as $F2(P)|r_i, \bar{d}_i|\sum w_i C_i$. Our main contributions are twofold: firstly, two matheuristics are developed to produce an initial feasible solution; both are based on the concept of local branching. Secondly, we propose two schedule improvement procedures, one based on stage-by-stage decomposition, and one using local branching. The performance of our methods is validated via extensive computational experiments.

The remainder of this paper is structured as follows. Section 2 summarizes the related literature. In Section 3 we provide a formal problem statement and a time-indexed formulation. Section 4

introduces local branching, which is essential to most of the algorithms proposed. In Section 5 we present two matheuristics to obtain an initial feasible solution, and in Section 6 we describe two improvement methods. Section 7 contains the computational results, and Section 8 provides a summary and conclusions.

2 Literature review

Comprehensive surveys of hybrid flow shop scheduling can be found in Quadt and Kuhn (2007), Ribas et al. (2010) and Ruiz and Vazquez-Rodriguez (2010). In this section we first review the complexity of scheduling hybrid flow shops and of scheduling with time windows. Subsequently, we provide an overview of the solution methods that have been used to date for hybrid flow shop scheduling.

Hybrid flow shop scheduling is NP-hard in the strong sense for most objective functions, since it directly generalizes the classic flow shop and the parallel machine setting. For instance, problems $F3||C_{\max}$ and $F2||\sum C_i$ are both NP-hard in strong sense (Garey et al., 1976). Yet, there also exist specific variants that can be addressed in polynomial time (Hall et al., 2000; Guirchoun et al., 2005). Moreover, almost every scheduling problem with hard time windows (with release dates or deadlines) seems to be difficult. Lenstra et al. (1977), for instance, show that $1|r_i|\sum w_iC_i$ is strongly NP-hard, and finding a feasible solution for $1|r_i,\bar{d}_i|-$ is strongly NP-complete (Garey et al., 1976). From these results, we conclude that problem $F2(P)|r_i,\bar{d}_i|\sum w_iC_i$ is also NP-hard in the strong sense.

Probably due to the inherent complexity of hybrid flow shops, only few articles work with exact methods. The earliest branch-and-bound (B&B) algorithm was introduced by Brah and Hunsucker (1991) for the problem $F(P)||C_{\max}$, where the schedule for each machine at every stage is a sequence of jobs; the overall problem is tackled stage by stage. The B&B method solves instances with up to eight jobs and two stages. Azizoglu et al. (2001) propose a novel branching scheme, where the schedule of each stage is represented by a sequence of jobs and the jobs are assigned to the earliest available machine by sequence order. The performance of their scheme is better than Brah and Hunsucker's, but the instance size that can be solved is still very limited (up to 15 jobs). Haouari et al. (2006) design a specific B&B algorithm for the problem $F2(P)||C_{\max}$,

including tight lower and upper bounding procedures at each node and a feasibility and adjustment procedure. The algorithm can solve rather large instances (with up to 1000 jobs) in reasonable runtimes. Wang et al. (2015) propose a B&B algorithm for a two-stage hybrid flow shop with no-wait constraints, capable of handling up to 20 jobs. Other B&B approaches can also be found in the literature (see, for instance, Néron et al., 2001; Lee and Kim, 2004; Allaoui and Artiba, 2006; Hadda et al., 2014). The instance size that B&B methods can solve is typically quite small, but the methods can also serve as the basis for other algorithms, such as beam search and A-star search. A notable exception is the work by Haouari et al. (2006), who also solve larger instances, but we cannot equal their performance in this paper because the weighted completion time objective does not allow for equally strong lower bounds, and since finding a feasible solution is NP-hard, it is almost impossible to compute an upper bound by a fast heuristic at each node.

Mixed-integer programming (MIP) is another way to obtain exact solutions, especially in real-world environments, since a variety of hard and complicated constraints often arise in practical settings, which are typically easily incorporated in a MIP formulation. Sawik (2001) investigates an assembly line of printed wiring board with finite buffers and job blocking. He describes a MIP formulation for makespan minimization, which can handle jobs with different product types. Berghman et al. (2014) examine a practical case of dock assignment, where trailers park at a parking zone and are assigned to warehouse gates during a given time period for loading or unloading. The transportation between gates and the parking zone is performed by tractors. They model this setting as a three-stage hybrid flow shop, and various MIP formulations are examined. The best performance is achieved by a time-indexed formulation, which can solve medium-sized instances to optimality. Further practical studies can be found in Gicquel et al. (2012) and Simpson and Abakarov (2013).

In light of the difficulty of scheduling hybrid flow shops, heuristic methods are typically a convenient and efficient choice. Heuristics can be classified into two categories (Quadt and Kuhn, 2007): "while holistic approaches consider the complete scheduling problem in an integrated way, decomposition approaches divide the problem with respect to the production stages, the individual jobs, or the sub-problems to be solved (batching, loading, and sequencing)." Most holistic heuristics are based on local search and meta-heuristics. Pan et al. (2017), for instance, present an iterated

search with greedy and local search to minimize earliness and tardiness in a hybrid flow shop. Belaid et al. (2012) use a greedy algorithm, a novel ant colony optimization with features of simulated annealing (taken from T'kindt et al. (2002)) and a dedicated heuristic to solve a practical case in the shampoo industry. For further examples of holistic heuristics, we refer to Komaki et al. (2016), Lei and Guo (2016) and Liu et al. (2017). Decomposition strategies have also been used to break up the scheduling of a hybrid flow shop into independent stages. Bang and Kim (2011), for instance, study a semiconductor wafer probing problem. To minimize total tardiness, they employ a bottleneck-focused technique, which first constructs a schedule on the bottleneck workstation and subsequently considers the other stages. Similar strategies can be found in Shahvari and Logendran (2018) and Tan et al. (2018).

Hybrid flow shops occur widely in current-day manufacturing, and are often enriched with complex and practical constraints such as limited buffers (Baumann and Trautmann, 2013; Liu et al., 2017), no buffers (Grabowski and Pempera, 2000; Gicquel et al., 2012), limited-waiting restrictions (Gicquel et al., 2012), no-wait constraints (Grabowski and Pempera, 2000; Berghman et al., 2014; Berghman and Leus, 2015), machine eligibility constraints (Liu et al., 2017; Shahvari and Logendran, 2018), re-entrant flows (Chamnanlor et al., 2014; Schulze et al., 2016), batch processing (Liu et al., 2017; Tan et al., 2018), release dates (Shahvari and Logendran, 2018), sequence-dependent setup times (Bang and Kim, 2011), dedicated machines (Hadda et al., 2014), and so on. These technical constraints often arise with multiple stages (Almada-Lobo et al., 2008; Gicquel et al., 2012; Baumann and Trautmann, 2013; Chamnanlor et al., 2014) and large-scale instances (Almada-Lobo et al., 2008; Baumann and Trautmann, 2013; Schulze et al., 2016; Liu et al., 2017), resulting in a complicated and difficult problem. Chamnanlor et al. (2014), for example, solve a case of hard-disk production with 17 stages. Baumann and Trautmann (2013) study a case of consumer goods with four stages, 59 intermediates and 203 final products. Almada-Lobo et al. (2008) handle the mass production of glass containers, where each production line outputs thousands of products per minute. We conclude that the development of efficient approaches for finding high-quality solutions to large hybrid flow shop instances is a valuable area of study.

3 Problem description

We schedule a set $N = \{1, \ldots, n\}$ of n independent jobs in a two-stage hybrid flow shop, where m_1 and m_2 identical parallel machines constitute stages 1 and 2, respectively. The machines in each stage are gathered in sets $M_1 = \{1, \ldots, m_1\}$ and $M_2 = \{1, \ldots, m_2\}$. Each job i has associated processing times at each stage denoted by p_{i1} and p_{i2}, a release date r_i, a deadline $\bar{d}_i$, and a weight w_i. Each job is assigned to a machine at each stage, while each machine can handle at most one job at a time. Each processing stage for each job is executed without preemption, machine breakdowns are not considered, and the buffer capacity between the two stages is infinite. The scheduling objective is to minimize the total weighted completion time of the final processing stage, which is a criterion that often occurs in industrial scheduling (Panwalkar et al., 1973; Tang and Wang, 2010; Simpson and Abakarov, 2013).

Below, we provide a mathematical programming model M0 to describe the problem more clearly. Berghman et al. (2014) test and compare an assignment-based formulation, a flow formulation and a time-indexed formulation for the problem $Pm||\sum w_i C_i$. They observe that the time-indexed formulation has the best performance by far; for instances with 15 jobs and five machines, for example, the time-indexed formulation only needs an average of around one second, while the other two formulations spend more than one hour per instance. Based on this result, we propose a time-indexed formulation for our problem, where time is divided into periods, with period t starting at time $t-1$ and ending at time t. The planning horizon is $H = \{\min_{i\in N}\{r_i\}, \ldots, \max_{i\in N}\{\bar{d}_i - p_{i2} + 1\}\}$. We also define $r_{i1} = r_i$, $r_{i2} = r_i + p_{i1}$, $\tau_{i1} = \bar{d}_i - p_{i2} - p_{i1} + 1$ and $\tau_{i2} = \bar{d}_i - p_{i2} + 1$. Let H_{ij} be the set of potential start periods of each job i at stage j, i.e., $H_{i1} = \{r_{i1}, r_{i1}+1, \ldots, \tau_{i1}\}$ and $H_{i2} = \{r_{i2}, \ldots, \tau_{i2}\}$. The formulation uses binary decision variables $x_{ijt} \in \{0, 1\}$, for $i \in N$, $j = 1, 2$ and $t \in H_{ij}$, where $x_{ijt} = 1$ if job i starts at period t in stage j, and $x_{ijt} = 0$ otherwise. The time-indexed model M0 is as follows:

$$\min \quad \sum_{i\in N} w_i \sum_{t\in H_{i2}} x_{i2t}(t + p_{i2} - 1) \tag{1}$$

subject to

$$\sum_{i\in N} \sum_{k=\max\{t-p_{ij}+1, r_{ij}\}}^{\min\{t,\tau_{ij}\}} x_{ijk} \le m_j, \qquad \forall j = 1, 2,\ \forall t \in H \tag{2}$$

$$\sum_{t \in H_{ij}} x_{ijt} = 1, \qquad \forall i \in N,\ \forall j = 1, 2 \tag{3}$$

$$\sum_{t \in H_{i2}} t x_{i2t} - \sum_{t \in H_{i1}} t x_{i1t} \geq p_{i1}, \qquad \forall i \in N \tag{4}$$

Constraints (2) state that the number of jobs processed in any period cannot exceed the machine capacity, constraints (3) imply that each job is scheduled exactly once in every stage, and constraints (4) ensure precedence constraints between two stages. The latter constraints can be replaced by the following set of stronger constraints (Christofides et al., 1987):

$$\sum_{k=t}^{\tau_{i1}} x_{i1k} + \sum_{k=r_{i2}}^{\min\{\tau_{i2}, t+p_{i1}-1\}} x_{i2k} \leq 1, \qquad \forall i \in N,\ \forall t \in H_{i1}. \tag{5}$$

Constraint set (5) also increases the size of the formulation, and it therefore does not automatically improve the computational performance; its application should depend on the specific environment (see Berghman et al., 2014; Berghman and Leus, 2015). In preliminary computational experiments, we have found that the equation set (5) tends to lead to models that are so large that the solver can simply not handle them anymore, and so we will use the inequalities (4) throughout the text.

Example 1 *Consider an example instance with the job data in Table 1.*

Table 1: Job data

Job i	p_{i1}	p_{i2}	w_i	r_i	$\bar{d}_i$
1	7	9	4	20	44
2	13	67	8	5	140
3	52	21	2	4	170
4	72	58	5	24	194
5	56	62	6	0	223
6	53	20	1	1	157

Table 2: An optimal schedule

Job	S_1	C_1	S_2	C_2
1	20	26	27	35
2	5	17	18	84
3	27	78	85	105
4	56	127	128	185
5	0	55	56	117
6	79	131	132	151

We let $m_1 = m_2 = 2$. An optimal schedule is described in Table 2, where the columns S_i and C_i contain the starting and completion period in stage i for each job ($i = 1, 2$). The model M0 for this instance contains 792 binary variables and 344 constraints; the optimal objective value is 2800. The corresponding Gantt chart is shown in Figure 1.

4 Local branching

The model M0 in Section 3 is a 0-1 integer programming model with many binary variables x_{ijt}, and the bottleneck to its solution is the efficient search of this large feasible set of binary variables. In this section we briefly introduce local branching (Fischetti and Lodi, 2003), which is a search

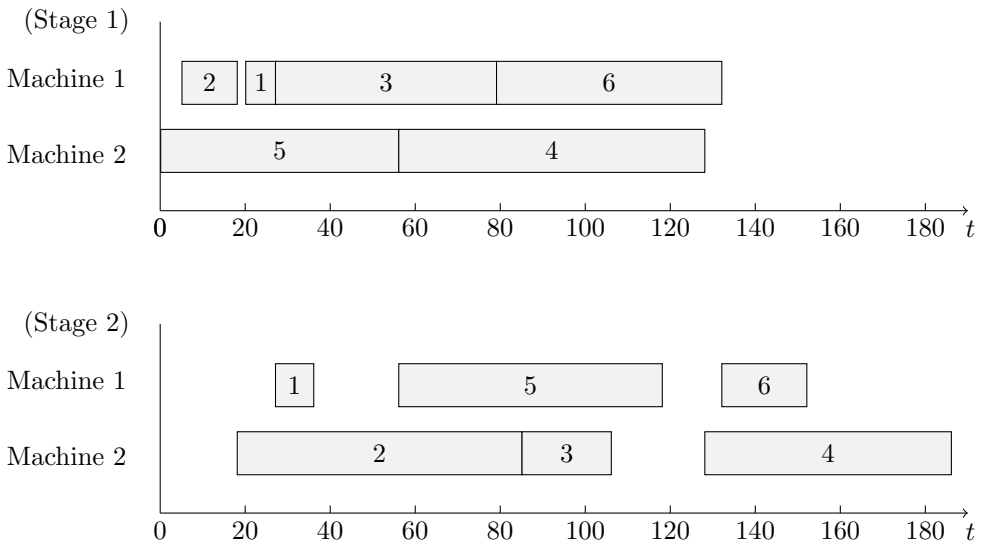

Figure 1: Gantt chart of the optimal schedule for Example 1

method that uses a regular solver to improve a given solution locally and iteratively. In Sections 5 and 6, local branching will prove to be a useful tool in the development of a number of matheuristics for the scheduling problem at hand.

The basic idea of local branching is to employ a regular solver as a black-box "tactical" tool to explore the proper solution subspaces that are defined and controlled at a "strategic" level by an external branching structure. Unlike a "standard" branching scheme, local branching considers an imbalanced tree where one branch has a small feasible set and can be solved by the solver, whereas the other branch has a considerably larger feasible set that is difficult to explore completely. After solving the small branch, the larger one is branched into two imbalanced descendants again. The basic scheme is illustrated in Figure 2. As usual, the incumbent solution is updated each time a better solution is encountered.

We denote a feasible solution to model M0 by x. Let $x^{(0)}$ be an initial solution and $\bar{S} = \{x_{ijt} | x^{(0)}_{ijt} = 1\}$. Then we can confine the search space to contain only solutions x that respect the following constraint, with r a predetermined radius:

$$|x - x^{(0)}| = \sum_{x_{ijk} \in \bar{S}} (1 - x_{ijk}) + \sum_{x_{ijk} \notin \bar{S}} x_{ijk} \leq r, \tag{6}$$

where the two terms in the left-hand side of the inequality count the number of binary variables that, compared to $x^{(0)}$, "flip" their value either from 1 to 0 or from 0 to 1, respectively. If the cardinality of $\bar{S}$ of any feasible solution is a constant (which is the case for model M0), then the

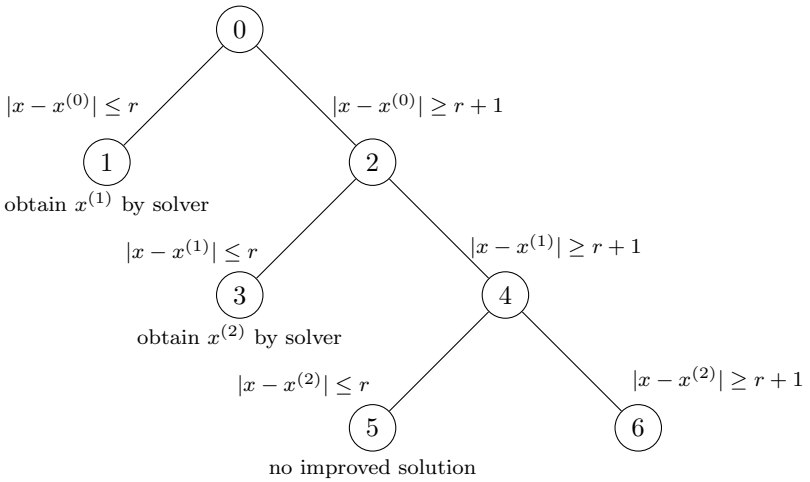

Figure 2: Illustration of the tree structure for local branching

constraint (6) can be replaced by

$$\sum_{x_{ijt} \in \bar{S}} (1 - x_{ijt}) \leq r', \tag{7}$$

with $r' = r/2$. Constraint (6) or (7) confines the solution domain to a hypercube centered at $x^{(0)}$, which defines the smaller branch. In the larger branch, we impose

$$|x - x^{(0)}| \geq r + 1. \tag{8}$$

The smaller branch (node 1), i.e. the original model with the constraint (6), can be solved by a regular solver. The optimal solution is denoted by $x^{(1)}$. Subsequently, the larger branch (node 2) can be branched into a smaller branch (node 3) with the constraint $|x - x^{(1)}| \leq r$, and a larger branch (node 4) with the constraint $|x - x^{(1)}| \geq r + 1$. Nodes 3 and 4 also include the constraint (8), since it defines their parent node 2. Subsequently, we can use a solver to solve node 3. If there is no improved solution (node 5 in Figure 2), this situation leads to node 6 with constraint $|x - x^{(2)}| \geq r + 1$, and the branching procedure is halted. More algorithmic details of the local branching scheme can be found in Fischetti and Lodi (2003), who also state that the resulting node 6 can then typically be solved be a solver.

5 Initialization

In this section, we first describe how to produce an initial schedule while ignoring the deadline constraints. Subsequently, two matheuristics are proposed to find a feasible solution based on an input reference solution that has violated deadlines. The first one is referred to as *infeasible schedule*

repair (ISR), and employs local branching directly. The second approach is called *feasible schedule search by artificial variables* (FSSA) and is adapted from Fischetti and Lodi (2008); the procedure relaxes constraints violated by the input schedule by means of artificial variables in model M0, and then repairs the infeasibility by local branching. Another potential approach would be to change the job deadlines $\bar{d}_i$ into due dates d_i and minimize the total tardiness: when the total tardiness is zero then the schedule is feasible. The corresponding problem $F2(P)|r_i, d_i|\sum T_i$ is NP-hard in the strong sense, however, and an optimal solution procedure for this problem takes a prohibitive amount of time, which is why we do not follow this approach in this paper. Improvements in the objective function can be achieved using the methods in Section 6.

5.1 Initial schedule generation

We first introduce a schedule representation and generation scheme. Subsequently we describe a *job window heuristic* (JWH), which aims to improve the schedule quality by iteratively and locally optimizing a sub-sequence of the schedule representation within a varying job window.

5.1.1 Schedule representation and generation

We represent a schedule by means of a pair (η_1, η_2) of ordered job lists (below also referred to as "sequences" or "permutations"), one for each stage. In detail, $\eta_1 = (\ell_{11}, \ell_{21}, \ldots, \ell_{n1})$ and $\eta_2 = (\ell_{12}, \ell_{22}, \ldots, \ell_{n2})$, where ℓ_{ij} is the i-th listed job at stage j, with $j = 1, 2$. A schedule representation is transformed into a schedule using the so-called serial generation scheme (Kolisch, 1996), which iteratively selects the next job in the list and schedules it as early as possible. The scheme takes machine capacities, ready times and stage precedence into account, but not the deadlines. If there exists a feasible schedule, then a list pair exists that generates such a schedule. A stronger result also holds: with a regular objective function (i.e., non-decreasing with task completion times), there always exists a list pair that leads to an optimal schedule by the serial generation scheme. The optimal schedule in Example 1, for instance, can be obtained from the list pair $((5, 2, 1, 3, 4, 6), (2, 1, 5, 3, 4, 6))$.

Example 2 *Consider the instance described in Example 1. Applying the serial schedule generation scheme to the list pair* $((1, 2, 3, 4, 5, 6), (1, 5, 4, 3, 2, 6))$ *leads to the schedule in Table 3. This schedule is infeasible because jobs 2, 3 and 6 violate their deadlines (printed in boldface).*

Table 3: The schedule generated in Example 2

Jobs	S_1	C_1	S_2	C_2	$\bar{d}_i$
1	20	26	27	35	44
2	5	17	178	244	**140**
3	18	69	157	177	**170**
4	27	98	99	156	194
5	70	125	126	187	223
6	99	151	188	207	**157**

Many choices can be made for the initial list pair, for instance by ordering the jobs in non-decreasing order of their deadlines. Another possible way to generate a list pair is by solving the linear relaxation of model M0 and then ordering the jobs in non-decreasing order of their *average starting time* in each stage. The main drawback of this method is that solving the linear relaxation is time-consuming, especially for large instances. To illustrate, suppose that we have the following values for a given job $i \in N$ at stage $j = 1$ or 2: $x_{ij4} = x_{ij5} = 0.3$ and $x_{ij7} = 0.4$, then the corresponding average starting period is $4 \times 0.3 + 5 \times 0.3 + 7 \times 0.4 = 5.5$.

5.1.2 Job window heuristic

Our job window heuristic employs a strategy of job-based decomposition, similar to Debels and Vanhoucke (2007), Della Croce et al. (2014) and Davari et al. (2016). The procedure iteratively and locally improves a given schedule within a varying job window. Afterwards, if the schedule is still infeasible, it can be repaired by ISR (Section 5.2) or FSSA (Section 5.3).

Given a job sequence σ, a position r in the sequence and a size parameter h, let $\sigma(r, h)$ be the set of jobs that are in consecutive positions r, $r+1$,..., $\min\{r+h-1, n\}$ of the sequence σ. We call this sub-sequence a *job window*. A job window gives rise to a new problem $P|r'_{ij}, \bar{d}'_{ij}|\sum w_i C_{ij}$ with at most h jobs, where $r'_{ij}, \bar{d}'_{ij}$ and C_{ij} are the release date, deadline and completion time of job $i \in \sigma(r, h)$ at stage j (the window pertains to one stage only). For the first stage, we set $r'_{i1} = r_i$ and $\bar{d}'_{i1} = \bar{d}_i - p_{i2}$, and for the second stage we let $r'_{i2} = C_{i1} + 1$ and $\bar{d}'_{i2} = \bar{d}_i$. Let H'_{ij} be the set of potential start periods of job i at stage j, i.e., $H'_{i1} = \{r'_{i1}, r'_{i1}+1, \ldots, \bar{d}'_{i1} - p_{i1} + 1\}$ and $H'_{i2} = \{r'_{i2}, \ldots, \bar{d}'_{i2} - p_{i2} + 1\}$. For a given stage $j = 1$ or 2, we solve this problem with the following time-indexed formulation:

$$\min \quad \sum_{i \in \sigma(r,h)} w_i \sum_{t \in H'_{ij}} x_{ijt}(t + p_{ij} - 1) \tag{9}$$

subject to

$$\sum_{i\in\sigma(r,h)}\sum_{k=\max\{t-p_{ij}+1,r'_{ij}\}}^{\min\{t,\tau_{ij}\}} x_{ijk} \le m_a(t), \qquad \forall t \in H' \tag{10}$$

$$\sum_{t\in H'_{ij}} x_{ijt} = 1, \qquad \forall i \in \sigma(r,h). \tag{11}$$

The objective is to minimize the total weighted completion times of the jobs in job window $\sigma(r,h)$ at stage j. Constraints (10) impose the machine capacity constraints, with total planning horizon $H' = \{\min_{i\in\sigma(r,h)}\{r'_{ij}\},\ldots,\max_{i\in\sigma(r,j)}\{\bar{d'_{ij}} - p_{ij} + 1\}\}$. Value $m_a(t)$ is the number of available machines at time period t, taking into account the scheduling decisions made for all the previous job windows. Constraints (11) state that each each job is scheduled exactly once. If the formulation is infeasible, we generate a tentative (infeasible) partial schedule based on the job order in σ and the serial generation scheme, and we continue to the next job window. Instead of MIP, B&B (Davari et al., 2016) and meta-heuristics (Debels and Vanhoucke, 2007) are other possible choices as a search tool. We have chosen for MIP because it can guarantee optimal solutions, and it can handle larger job windows than the B&B method (to the best of our knowledge, there is no efficient B&B method for $P|r'_{ij},\bar{d'_{ij}}|\sum w_iC_{ij}$ in the literature). A MIP model is also easily extensible to include additional technical constraints.

We apply this procedure stage by stage, where the search starts with the first job window of sequence σ at stage 1. Each job window is considered one by one in the order of σ. After stage 1, we update the job completion times and we run the procedure for stage 2. Davari et al. (2016) choose a job window randomly, but this is not straightforward to implement in a hybrid flow shop. If the final schedule is still infeasible, we will use ISR or FSSA (see below) to attempt to achieve feasibility.

Example 3 *Consider Example 2. We start with list pair* $\sigma = ((1,2,3,4,5,6),(1,5,4,3,2,6))$*, and we select the width of the job window as* $h = 3$*. We first optimize sub-problem* $(1,2,3)$ *in stage 1, followed by* $(4,5,6)$*; in this second run, we take the schedule for* $(1,2,3)$ *into account. At the second stage we solve the sub-problem for* $(1,5,4)$ *and subsequently* $(3,2,6)$*, in which the jobs cannot start before their completion in stage 1. The final schedule is represented in the Table 4, with corresponding job sequences* $((3,2,1,5,4,6),(1,5,4,3,2,6))$*. We see that jobs 2 and 6 still violate their deadlines, but we have achieved an improvement compared to Example 2.*

Table 4: The schedule generated in Example 3

Jobs	S_1	C_1	S_2	C_2	$\bar{d}_i$
1	20	26	27	35	44
2	5	17	166	232	**140**
3	4	55	145	165	170
4	56	127	128	185	194
5	27	82	83	144	223
6	83	135	186	205	**157**

5.2 Infeasible schedule repair

Given an initial infeasible schedule $\bar{x}$, we define set $\bar{S} = \{x_{ijt} | \bar{x}_{ijt} = 1, t \in H_{ij}, i \in N, j = 1, 2\}$ and we impose a constraint to delimit a search space for local branching as

$$\sum_{x_{ijk} \in \bar{S}} (1 - x_{ijk}) + \sum_{x_{ijk} \notin \bar{S}} x_{ijk} \leq r, \tag{12}$$

where r is the search radius, and we implement local branching with model M0 and constraint (12). If the model in the smaller branch is infeasible, then we branch as shown in Figure 3 with $x^{(0)} = \bar{x}$, i.e., we reverse the constraint such that the left-hand side of equation (12) is greater than or equal to $r + 1$. Subsequently, the radius is increased by Δ_r^{ISR} to $r_1 = r + \Delta_r^{ISR}$, and we solve a new smaller branch. When the radius is greater than or equal to $|\bar{S}| + 2n$ and the model in the smaller branch is still infeasible, then the corresponding instance is not feasible. We halt local branching as soon as we find a feasible solution. In Figure 3, the procedure is interrupted when we obtain the feasible schedule $x^{(1)}$ in node 3.

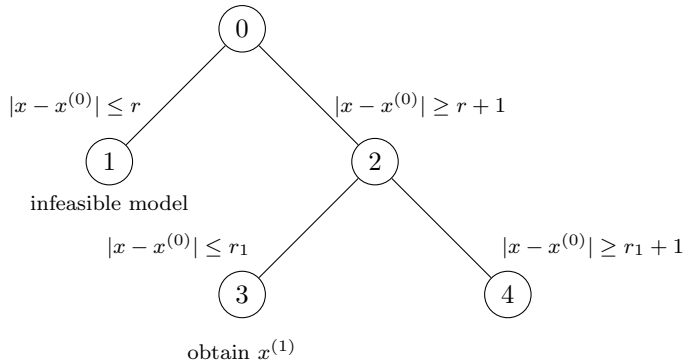

Figure 3: Branching tree for ISR

Example 4 *Consider the infeasible schedule of Example 3. We define set* $\bar{S} = \{x_{1,1,20}, x_{1,2,27}, x_{2,1,5}, x_{3,1,4}, x_{3,2,145}, x_{4,1,56}, x_{4,2,128}, x_{5,1,27}, x_{5,2,83}, x_{6,1,83}\}$. *The set* $\bar{S}$ *does not include* x*-variables for*

jobs 2 and 6 at stage 2 because the corresponding starting periods are outside of the sets H_{22} and H_{62}. We set the initial search radius $r = 6$. The repaired feasible schedule is described in Table 5, with objective value 3444; this schedule is already obtained in the node corresponding with node 1 in Figure 3, so in the first smaller branch.

Table 5: The schedule of Example 4

Jobs	S_1	C_1	S_2	C_2	$\bar{d}_i$
1	20	26	27	35	44
2	5	17	18	84	140
3	4	55	92	112	170
4	56	127	128	185	194
5	27	82	160	221	223
6	83	135	138	157	157

5.3 Feasible schedule search by artificial variables

For an initial infeasible schedule $\bar{x}$, let Γ_1, Γ_2 and Γ_3 be the subsets of constraints from constraint sets (1), (2) and (3) that are violated by $\bar{x}$ in model M0. We relax the constraints in Γ_l, $l = 1, 2, 3$, as follows:

$$\sum_{i\in N}\sum_{\ell=\max\{t-p_{ij}+1,r_{ij}\}}^{\min\{t,\tau_{ij}\}} x_{ij\ell} - \delta_{1k} \le m_j, \qquad \forall k \in \Gamma_1 \tag{13}$$

$$\sum_{t\in H_{ij}} x_{ijt} + \delta_{2k} - \delta'_{2k} = 1, \qquad \forall k \in \Gamma_2 \tag{14}$$

$$\sum_{t\in H_{i2}} t x_{i2t} - \sum_{t\in H_{i1}} t x_{i1t} + \delta_{3k} \ge p_{i1}, \qquad \forall k \in \Gamma_3 \tag{15}$$

where $\delta_{lk} \ge 0$, $l = 1, 2, 3$, and $\delta'_{2k} \ge 0$ are nonnegative continuous artificial variables. We replace the original objective in M0 by

$$\min \quad \sum_{k\in\Gamma_1} \delta_{1k} + \sum_{k\in\Gamma_2} (\delta_{2k} + \delta'_{2k}) + \sum_{k\in\Gamma_3} \delta_{3k}. \tag{16}$$

Once this objective is zero, we have obtained a feasible solution.

With the same definition of $\bar{S}$ as in Section 5.2, we also incorporate the local branching constraint (12). Specifically, we implement local branching with constraints (13), (14), (15), (12), and all the remaining constraints that are not violated in M0 to minimize objective function (16).

Compared to Fischetti and Lodi (2008), which was the main inspiration for our algorithm, FSSA skips a feasibility-pump procedure (Fischetti et al., 2005), which needs to solve a linear relaxation and is time-consuming. We also minimize the sum of continuous artificial variables,

which is easier than using binary indicator variables when solving a MIP (see Fischetti and Lodi (2008) for more details). If the final result of local branching has an objective greater than zero, then the instance is infeasible. One drawback of FSSA is that the original objective function is disregarded, thus the feasible solution obtained might be arbitrarily bad.

Example 5 *Consider the infeasible schedule of Example 3. As discussed in Example 4, the deadlines of jobs 2 and 6 are not respected, and some of the constraints (2) and (3) in model M0 are violated. The feasible schedule generated by FSSA is shown in Table 5, with objective value 3112.*

Table 6: The schedule of Example 5

Jobs	S_1	C_1	S_2	C_2	$\bar{d}_i$
1	20	26	31	39	44
2	5	17	29	95	140
3	97	148	150	170	170
4	56	127	128	185	194
5	0	55	75	136	223
6	32	84	96	115	157

6 Improvements

In this section we introduce two improvement methods that can be applied to an initial feasible solution. The first method is a stage-by-stage decomposition (SD) and uses a regular solver for improvements. The second procedures is an adapted form of local branching (AL). Both of the procedures improve the initial solution by a MIP solver, which allows to keep the solution feasible. Meta-heuristics are an alternative, but would require significant attention to ensure that no infeasible solutions are generated during the random search. A computational comparison of the two improvement methods SD and AL can be found in Section 7.

6.1 Stage-by-stage decomposition

In the stage-by-stage decomposition SD, we divide the original problem into two parallel machine scheduling problems with time windows, and we optimize each stage in succession. Consider an initial feasible solution $\bar{x}$ composed of $\bar{x}_1$ and $\bar{x}_2$, which correspond with schedules for stages 1 and 2, respectively. For stage 1, set $\tilde{d_{i1}} = \bar{S_{i2}} - 1$ for all $i \in N$, where $\bar{S_{i2}}$ is the starting time of job i at stage 2 in $\bar{x}$. This leads to a problem $Pm_1|r_i, \tilde{d_{i1}}|\sum w_i C_{i1}$, where r_i, $\tilde{d_{i1}}$ and C_{i1} are the

release date, deadline and completion time of job i at stage 1. We solve this problem with a MIP solver and we provide the initial schedule $\bar{x}_1$ as the starting solution to the solver, which helps to find a feasible solution quickly. Similarly, we set $\tilde{r}_{i2} = C_{i1} + 1$ for $i \in N$, and solve the problem $Pm_2|\tilde{r}_{i2}, \bar{d}_i| \sum w_i C_{i2}$, where $\tilde{r}_{i2}$, $\bar{d}_i$ and C_{i2} are the release date, deadline and completion time of job i at stage 2; again we give $\bar{x}_2$ as starting solution to the solver.

Obtaining an optimal solution at stage 1 may be time-consuming, and might leave only little time for stage 2. We can therefore impose a time limit on the first stage. When $m_1 = m_2$, we impose a time limit of $\left(\frac{2}{3}\right)(T - T_{ini})$ in the first stage, where T and T_{ini} are the total time limit and the initialization time, respectively. Otherwise, $m_1 \neq m_2$ and there is a *bottleneck stage*, namely the stage with the lowest number of machines. If m_B and m_N are the machine numbers in the bottleneck and the non-bottleneck stage, then we impose the following limits on the runtime:

$$T_B = \frac{2m_N - m_B}{2m_N}(T - T_{ini}), \quad T_N = \frac{m_B}{2m_N}(T - T_{ini}),$$

where T_B is the time for the bottleneck stage and T_N for the non-bottleneck stage.

6.2 Adapted local branching

Local branching is another possibility for improving a given starting solution. We have made some adaptations to the general framework in function of the particular problem setting of hybrid flow shops with time windows.

A starting solution is provided to the solver in each node of the search tree, which is necessary especially for instances with tight time windows, where finding a feasible solution is quite difficult. The standard local branching constraint and the reference solution normally simply control the definition of the search space. If we set a small search radius, a feasible solution is easily found based on the reference solution, but the solution improvement procedure will be slow. In order to speed up the improvements achieved by the solver, we set the initial solution from the initialization stage as the starting solution for the first node, and other nodes use the solution from the previous level in the search tree.

We also impose a time limit T_{node} and a stop criterion based on the gap G_{node} between upper bound and lower bound for each node, because of the difficulty of convergence for some instances. Once the time limit or the gap threshold is reached, then we backtrack to the parent node and

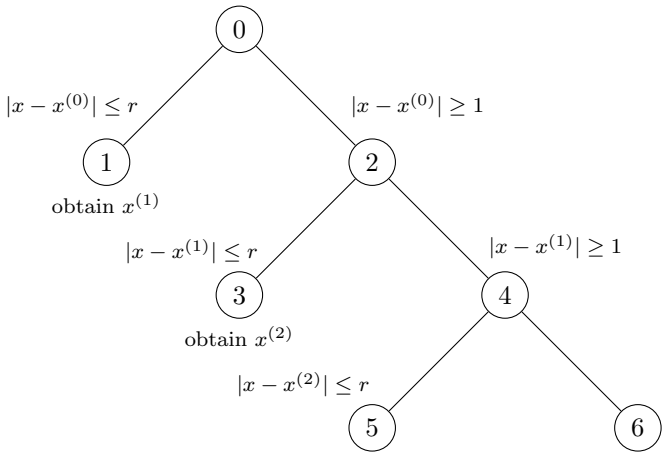

Figure 4: Illustration of adapted local branching

create a new smaller branch. We illustrate the procedure in Figure 4. Solution $x^{(0)}$ is the reference (input) solution. At node 1, we solve model M0 with local branching constraint $|x - x^{(0)}| \leq r$, and afterwards we reverse the branching to $|x - x^{(0)}| \geq 1$, effectively rendering the solution $x^{(0)}$ "tabu," as in tabu search. Subsequently, $x^{(1)}$ serves as starting solution at node 3, in which we impose $|x - x^{(1)}| \leq r$. In case the obtained solution $x^{(1)}$ in node 1 equals the starting solution then the solver will not receive a starting solution in node 3, since $x^{(1)}$ then violates the constraint $|x - x^{(0)}| \geq 1$. If we do not find a feasible solution in a node, we reduce the radius by $\triangle_r^{AL}$ and increase the time limit by $\triangle_t$, because a smaller solution space is easier to explore. The search procedure will halt when for a predetermined time T^{AL} or a predetermined number of rounds (number of small branches) K^{AL} there is no improved solution, and then return the best solution found. In our implementation, we have actually opted for the local branching constraint (7) rather than (6) because this turns out to lead to slightly better results here.

7 Computational results

In this section we report the results of a number of experiments to evaluate the effectiveness of the proposed methods. All algorithms are coded in C++, and Gurobi 8.0.1 is used to solve the MIP models. All computational results are obtained on a laptop with Intel core i5 2.3GHz processor, 4GB of RAM, and running under Windows 10. In Section 7.1 we describe the experimental setup. Section 7.2 contains a comparison between the initialization methods. Section 7.3 compares the performance of the matheuristics on small instances, in Section 7.4 we provide a comparison of the

different procedures on large instances.

7.1 Experimental setup

To the best of our knowledge, there are no relevant benchmark instances available in the literature for the problem at hand; we therefore generate our own instance sets. For each job $i \in N$, the processing times p_{ij}, $j = 1, 2$, are drawn (as integers) from a discrete uniform distribution on $[1, 100]$, which is a very common choice in the literature, and an integer weight w_i is generated from the discrete uniform distribution on $[1, 10]$. We generate time windows using a scheme adapted from Davari et al. (2016). Release dates r_i are generated from a discrete uniform distribution on the interval $[0, \phi P_1]$, where $P_1 = \sum_{i \in N} p_{i1}$. Deadlines $\bar{d}_i$ are sampled from the discrete uniform distribution on $[r_i + p_{i1} + p_{i2}, r_i + p_{i1} + p_{i2} + \rho P_2]$, where $P_2 = \sum_{i \in N} p_{i2}$. We consider six scenarios for the combination of values for ϕ and ρ, namely $(\phi, \rho) \in \{(0.1, 0.6), (0.3, 0.6), (0.2, 0.5), (0.2, 0.8), (0.3, 0.5), (0.5, 0.3)\}$.

Two problem sets are generated. Set I contains small instances with $n = 20$ and 30, which will allow to assess the gap from the optimal solution. We consider the following choices for the machine layout (m_1, m_2) of the flow shop: $(1, 4)$, $(2, 2)$, $(2, 4)$ and $(3, 2)$. A time limit of 600 seconds is imposed for each instance in Set I. Set II consists of larger instances with $n = 60$, 90 and 120, and has three different machine layouts, namely $(2, 2)$, $(2, 4)$ and $(4, 2)$. We test each procedure with a time limit T of 600, 900 or 1200 seconds, according to the problem size, which is a similar choice as Della Croce et al. (2014). For each combination of job number, scenario and machine layout we construct 10 instances.

7.2 Comparison of the initialization methods

Table 7 compares several priority rules based on the number of instances for which no feasible solution is produced (column "NoS") although the instance itself is feasible, and the average number of tardy jobs "Tar." The second column of the table shows how many instances are feasible, out of the total number. We have tested the earliest-deadline-first (ED) rule, a random priority rule (RP), and ordering in the average starting time of the linear relaxation (ALP). The ED rule orders the jobs in increasing $\bar{d}_i$ for both stages. If deadlines are equal, priority is given to the job with the lower release date. We also report the performance of the job window heuristic JWH

Table 7: Comparison of initialization methods

n	Fea/Total	ED		RP		ALP			JWH		
		NoS	Tar	NoS	Tar	Time	NoS	Tar	Time	NoS	Tar
20	157/240	83	1.95	157	15.19	1.84	128	2.10	1.35	15	0.19
30	165/240	86	4.40	164	24.33	5.46	145	3.75	2.96	18	0.43
60	174/180	84	6.98	172	51.08	27.71	168	9.70	10.91	6	0.41
90	176/180	90	10.90	176	79.00	76.37	173	15.24	24.87	4	0.39
120	180/180	102	12.23	180	106.69	193.74	179	21.59	44.68	3	0.03
Total		445		849			793			46	

Table 8: Comparison between ISR and FSSA

n	Fea/Total	JWH + ISR			JWH + FSSA		
		Time	NoS	Obj	Time	NoS	Obj
20	157/240	1.77	0	38222	1.74	0	38345
30	164/240	4.91	2	78679	3.38	3	79901
60	172/180	12.82	2	280105	12.23	1	280662
90	176/180	26.79	2	618169	29.05	0	618956
120	180/180	49.40	0	1078339	47.68	0	1078662
Total			6			4	

with the ED ordering as input; we set the job window width $h = 10$. The columns labeled "Time" contain the average runtime for ALP and JWH; the runtime for ED and RP is not mentioned because it is very low. For ALP, when the time for solving the LP exceeds the time limit, which happens for some instances, then these instances are not included in the computation of average Time and Tar.

From Table 7, we see that JWH can find feasible solutions to most feasible instances in a reasonable time; the performance of ALP is rather disappointing, especially given its high runtimes. We therefore use the output of JWH as the input for ISR and FSSA; Table 8 shows the results. In ISR, we set the search radius $r = \max\{2a, 20\}$ and $\triangle_r^{ISR} = \lfloor 0.5 * \max\{2a, 20\} \rfloor$, where a is the number of jobs violating their deadlines in $\bar{x}$. Based on Fischetti and Lodi (2008) and our own experiments, at each node of FSSA, we use an adaptive radius r as $1.2 * \lfloor (|\Gamma_1| + |\Gamma_2| + |\Gamma_3|) \rfloor$, where $|\Gamma_1| + |\Gamma_2| + |\Gamma_3|$ is the number of constraints violated by the current solution. If $|\Gamma_1| + |\Gamma_2| + |\Gamma_3|$ is less than 20 then $r = 20$.

In Table 8, "Obj" are the average objective values. FSSA solves slightly more instances than ISR but the initial objective values are a bit higher, which is expectable because FSSA pursues a different objective. We give priority to initializing the highest number of instances, and we therefore select JWH + FSSA as initialization method. Below, we combine JWH + FSSA with

the two improvement methods SD and AL (referred to as "JFS" and "JFA," respectively), and we compare with the time-indexed formulation.

7.3 Results for small instances (Set I)

Tables 9–11 show the results of JFS, JFA and the optimal solutions by the time-indexed formulation M0. In JFA, we set the initial search radius $r' = 30$, and the parameters $\triangle_r^{AL} = 5$ and $\triangle_t = 30$. In case no solution is found in a node then we let $r' := \max\{r' - \triangle_r^{AL}, 10\}$. The time limit for each node is $3n$, and the gap threshold G_{node} is 0.05. Let $T^{AL} = T - T_{ini}$, where T_{ini} is the initialization time, and $K^{AL} = 3$. Subsets of 10 instances are identified based on the job number n, the scenario for (ϕ, ρ) (value "S," numbered from 1 to 6), and the machine layout (m_1, m_2); for each subset, the column "Fea" indicates the number of feasible instances (out of 10). As before, "NoS" refers to the number of feasible instances for which no feasible solution is found. "Time" is the average runtime computed only for the feasible instances; when an instance is infeasible then this is typically recognized very quickly by M0 if the LP is also infeasible, which is almost always the case. The values for "Gap" report the average gap in the objective value (expressed as a percentage) between the corresponding method and the output of model "M0" within the time limit, as follows: Gap $= \frac{Z_{method} - Z_{\mathrm{M0}}}{Z_{\mathrm{M0}}} \times 100$, where Z_{method} is the objective found by the method and Z_{M0} is the objective found by model M0; if no feasible solution is found within the time limit, then we take Z_{M0} as the LP bound. The number of instances for which a guaranteed optimal solution is found, is reported as "OPT."

We observe that the balance between stages affects the feasibility of the instance; the layout $(m_1, m_2) = (1, 4)$, for example, only has infeasible instances, due to the capacity imbalance between the two stages. Obviously, the tightness of the time windows (determined mainly by the parameter ρ) also influences the probability of infeasibility.

Not all methods are able to obtain feasible solutions in all instances, especially for tight time windows (especially scenarios 1 and 3). The time-indexed formulation in particular struggles for some settings. In Table 9, for example, M0 does not produce feasible solutions for five instances with 30 jobs of scenario 1 in layout $(2, 2)$, and in Table 10 model M0 has seven unsolved instances with 30 jobs of scenario 3 in layout $(2, 2)$. Also, the time-indexed formulation cannot reach an optimal solution for some small instances, probably due to the high number of binary variables. In

Table 9, model M0 cannot find any optimal solution for the layout $(2, 2)$ with 30 jobs in scenarios 1 and 2.

We also find that JFA has a better performance than JFS in most instances. This might be explained by the fact that JFA can iteratively improve any given starting solution, while JFS is more dependent on the starting solution, especially with tight time windows, and an early start of jobs in stage 2 leads to even tighter time windows in stage 1, thus leaving only little margin for improvement in a stage-based decomposition.

Table 9: Results for scenarios 1 and 2 of Set I

n	(S, m_1, m_2)	Fea	JFS		JFA		M0		
			NoS	Gap	NoS	Gap	OPT	NoS	Time
20	(1,1,4)	0	-	-	-	-	-		-
20	(1,2,2)	4	0	3.92	0	1.39	3	0	211.76
20	(1,2,4)	5	0	9.85	0	2.11	4	0	130.85
20	(1,3,2)	8	0	7.11	0	0.61	4	0	304.35
20	(2,1,4)	1	0	22.64	0	2.96	1	0	103.77
20	(2,2,2)	10	0	4.21	0	−0.87	7	0	369.59
20	(2,2,4)	10	0	8.06	0	1.25	10	0	123.48
20	(2,3,2)	10	0	3.99	0	0.65	10	0	30.98
30	(1,1,4)	0	-	-	-	-	-		-
30	(1,2,2)	6	0	5.58	0	4.36	0	5	400.74
30	(1,2,4)	8	0	7.30	0	−3.39	7	0	275.32
30	(1,3,2)	7	0	2.27	0	2.26	1	0	588.14
30	(2,1,4)	0	-	-	-	-	-		-
30	(2,2,2)	10	0	−5.20	0	−9.68	0	4	600.00
30	(2,2,4)	10	0	13.36	0	2.95	7	1	396.61
30	(2,3,2)	10	0	−1.49	0	−1.59	7	0	284.06

Table 10: Results for scenarios 3 and 4 of Set I

n	(S, m_1, m_2)	Fea	JFS		JFA		M0		
			NoS	Gap	NoS	Gap	OPT	NoS	Time
20	(3,1,4)	0	-	-	-	-	-		-
20	(3,2,2)	8	0	6.62	0	1.58	4	0	404.97
20	(3,2,4)	8	0	9.04	0	0.99	8	0	73.83
20	(3,3,2)	9	0	3.95	0	1.20	8	0	105.30
20	(4,1,4)	2	0	9.87	0	2.31	2	0	43.43
20	(4,2,2)	9	0	6.56	0	1.69	3	0	464.88
20	(4,2,4)	9	0	11.29	0	1.51	9	0	63.47
20	(4,3,2)	9	0	4.20	0	1.06	8	0	136.58
30	(3,1,4)	0	-	-	-	-	-	0	-
30	(3,2,2)	8	2	13.39	2	10.08	0	7	600.00
30	(3,2,4)	8	1	14.88	1	2.58	6	0	362.96
30	(3,3,2)	10	0	1.32	0	−1.20	4	0	483.68
30	(4,1,4)	1	0	13.86	0	4.35	0	0	600
30	(4,2,2)	10	0	−0.02	0	−6.81	0	1	600.00
30	(4,2,4)	10	0	10.69	0	−1.65	6	0	412.94
30	(4,3,2)	10	0	−1.68	0	−3.89	8	0	289.39

Table 11: Results for scenarios 5 and 6 of Set I

n	(S, m_1, m_2)	Fea	JFS		JFA		M0		
			NoS	Gap	NoS	Gap	OPT	NoS	Time
20	(5,1,4)	0	-	-	-	-	-		-
20	(5,2,2)	10	0	6.77	0	1.92	4	1	447.29
20	(5,2,4)	10	0	6.58	0	1.76	10	0	36.92
20	(5,3,2)	10	0	3.76	0	0.90	10	0	38.04
20	(6,1,4)	0	-	-	-	-	-		-
20	(6,2,2)	7	0	1.61	0	1.23	5	0	206.23
20	(6,2,4)	9	0	1.67	0	1.21	9	0	23.85
20	(6,3,2)	9	0	0.35	0	0.38	9	0	9.33
30	(5,1,4)	0	-	-	-	-	-		-
30	(5,2,2)	9	0	5.16	0	−0.76	0	4	600.00
30	(5,2,4)	9	0	14.81	0	3.54	7	0	296.48
30	(5,3,2)	9	0	2.89	0	2.21	7	0	272.04
30	(6,1,4)	0	-	-	-	-	-		-
30	(6,2,2)	9	0	2.52	0	1.84	4	2	358.85
30	(6,2,4)	10	0	3.68	0	2.64	10	0	121.31
30	(6,3,2)	10	0	0.55	0	0.76	10	0	46.72

7.4 Results for large instances (Set II)

Tables 12–14 report the results of JFS, JFA and the MIP formulation M0 on Set II. The gap is computed as Gap $= \frac{Z_{method} - Z_{\text{M0}}}{Z_{\text{M0}}} \times 100$, where Z_{M0} is the LP bound from model M0.

For the large instances, tighter time windows lead to significantly more difficult instances. For scenario 1 in Table 12, for example, the time-indexed formulation solves only 14 out of 84 feasible instances, while for the scenario 2 (with looser time windows), this amounts to 65 out of 88 feasible instances. Looser time windows appear to be more difficult to schedule to guaranteed optimality, however: although an initial feasible solution is easily found, improvements are more difficult, and this is probably due at least in part to the higher number of variables in the corresponding time-indexed formulation. In scenario 4 of Table 13, for example, some gaps for M0 are larger than 100 percent.

Again, the machine layout also strongly affects the results. Model M0 has the most difficulties with finding feasible solutions for the layout $(2, 2)$, where the capacity is the most restrictive. Based on the average gaps, it seems to be easier to find high-quality solutions for the layout $(4, 2)$ rather than for $(2, 4)$, probably because with $(4, 2)$ the shop can handle more jobs at the first stage, which decreases idle time on the machines at the second stage.

Similarly to Set I, JFA performs better than JFS for most cases in Set II. In scenario 6, however, the performance of JFS is quite close to JFA's, and even better for some cases. We suspect that this

Table 12: Results for scenarios 1 and 2 of Set II

n	(S, m_1, m_2)	Fea	JFS		JFA		M0	
			NoS	Gap	NoS	Gap	NoS	Gap
60	(1,2,2)	9	0	18.18	0	22.28	9	-
60	(1,2,4)	9	0	24.31	0	9.61	9	-
60	(1,4,2)	10	0	5.24	0	6.84	4	30.42
60	(2,2,2)	9	0	16.97	0	15.87	2	61.64
60	(2,2,4)	10	0	21.30	0	7.56	0	62.08
60	(2,4,2)	9	0	1.24	0	2.79	0	2.57
90	(1,2,2)	8	0	24.32	0	31.16	8	-
90	(1,2,4)	8	0	32.84	0	15.42	5	76.72
90	(1,4,2)	10	0	19.24	0	17.81	6	5.71
90	(2,2,2)	10	0	21.23	0	14.60	8	74.44
90	(2,2,4)	10	0	25.80	0	8.14	2	3.99
90	(2,4,2)	10	0	8.29	0	4.33	0	2.84
120	(1,2,2)	10	0	29.98	0	32.28	10	-
120	(1,2,4)	10	0	33.69	0	20.14	10	-
120	(1,4,2)	10	0	34.95	0	15.86	9	94.04
120	(2,2,2)	10	0	20.85	0	19.53	8	79.75
120	(2,2,4)	10	0	24.97	0	10.13	3	79.19
120	(2,4,2)	10	0	12.08	0	6.08	0	4.03

Table 13: Results for scenarios 3 and 4 of Set II

n	(S, m_1, m_2)	Fea	JFS		JFA		M0	
			NoS	Gap	NoS	Gap	NoS	Gap
60	(3,2,2)	9	0	17.26	0	19.12	9	-
60	(3,2,4)	9	0	20.88	0	8.85	9	-
60	(3,4,2)	10	0	2.32	0	5.90	7	5.4
60	(4,2,2)	10	0	21.10	0	20.77	0	76.59
60	(4,2,4)	10	0	27.34	0	9.12	1	77.3
60	(4,4,2)	10	0	2.96	0	4.20	0	36.46
90	(3,2,2)	10	0	19.52	0	21.52	10	-
90	(3,2,4)	10	0	25.07	0	13.86	10	-
90	(3,4,2)	10	0	19.60	0	6.93	8	2.80
90	(4,2,2)	10	0	28.52	0	17.77	3	109.89
90	(4,2,4)	10	0	33.08	0	9.24	1	70.32
90	(4,4,2)	10	0	22.58	0	6.03	0	13.9
120	(3,2,2)	10	0	27.33	0	25.33	10	-
120	(3,2,4)	10	0	25.07	0	16.03	10	-
120	(3,4,2)	10	0	23.42	0	10.79	9	5.09
120	(4,2,2)	10	0	29.13	0	24.95	2	115.96
120	(4,2,4)	10	0	31.85	0	13.22	2	117.29
120	(4,4,2)	10	0	22.46	0	9.06	0	81.85

Table 14: Results for scenarios 5 and 6 of Set II

n	(S, m_1, m_2)	Fea	JFS		JFA		M0	
			NoS	Gap	NoS	Gap	NoS	Gap
60	(5,2,2)	10	0	15.25	0	16.56	8	77.93
60	(5,2,4)	10	0	19.84	0	6.82	8	73.82
60	(5,4,2)	10	0	1.12	0	3.19	2	19.59
60	(6,2,2)	9	1	4.80	1	7.38	8	22.47
60	(6,2,4)	10	0	5.52	0	5.05	4	0.56
60	(6,4,2)	9	0	0.46	0	1.95	0	0.72
90	(5,2,2)	10	0	20.04	0	15.82	9	69.72
90	(5,2,4)	10	0	21.98	0	9.10	8	6.30
90	(5,4,2)	10	0	11.26	0	5.40	2	18.06
90	(6,2,2)	10	0	6.58	0	8.24	9	29.63
90	(6,2,4)	10	0	7.47	0	6.07	8	7.79
90	(6,4,2)	10	0	0.35	0	1.41	1	0.61
120	(5,2,2)	10	0	23.21	0	22.24	10	-
120	(5,2,4)	10	0	21.50	0	11.01	10	-
120	(5,4,2)	10	0	15.74	0	7.21	4	3.58
120	(6,2,2)	10	0	6.66	0	12.97	9	3.89
120	(6,2,4)	10	0	8.23	0	8.05	1	1.96
120	(6,4,2)	10	0	2.57	0	3.17	0	1.15

is the case because the release dates of scenario 6 are distributed widely, which means that jobs can often be scheduled quite close to their release dates. This renders these instances relatively easy, and model M0 also has a good performance in scenario 6. Moreover, M0 has a smaller gap than JFS and JFA in some cases, e.g., case (3,4,2) with 90 jobs in Table 13. More generally, model M0 does not find a feasible solution for many harder instances, however.

8 Conclusion

In this paper we have studied the scheduling of a hybrid flow shop with hard time windows to minimize the total weighted completion times. Based on our literature review, we find that this problem has not really been studied before, although the problem statement is very natural, and we have indicated close connections with a wide range of industrial cases. We have developed two initialization and two improvement methods for the scheduling problem, which can be combined to lead to a number of different matheuristic strategies. Our computational experiments show that the resulting matheuristics are competitive with a regular solver for small instances, and they outperform the solver for medium-sized and large instances.

For future work in this area, several research directions can be explored. First of all, other optimization methods can be considered and hybridized. Various meta-heuristic procedures can

be plugged into the job window heuristic, for instance. The incorporation of a number of practical factors, such as restrictive storage policies (with limited buffers or waiting times) and production characteristics (such as uniform or unrelated parallel machines), is also evident, as is the extension to more than two production stages. Finally, industrial case studies could be used to validate the applicability of the current models and algorithms.

References

H. Allaoui and A. Artiba. Scheduling two-stage hybrid flow shop with availability constraints. *Computers & Operations Research*, 33(5):1399–1419, 2006.

B. Almada-Lobo, J. F. Oliveira, and M. A. Carravilla. Production planning and scheduling in the glass container industry: A VNS approach. *International Journal of Production Economics*, 114 (1):363–375, 2008.

M. Azizoglu, E. Cakmak, and S. Kondakci. A flexible flowshop problem with total flow time minimization. *European Journal of Operational Research*, 132(3):528–538, 2001.

J. Y. Bang and Y. D. Kim. Scheduling algorithms for a semiconductor probing facility. *Computers & Operations Research*, 38(3):666–673, 2011.

P. Baumann and N. Trautmann. A continuous-time MILP model for short-term scheduling of make-and-pack production processes. *International Journal of Production Research*, 51(6):1707–1727, 2013.

R. Belaid, V. T'kindt, and C. Esswein. Scheduling batches in flowshop with limited buffers in the shampoo industry. *European Journal of Operational Research*, 223(2):560–572, 2012.

L. Berghman and R. Leus. Practical solutions for a dock assignment problem with trailer transportation. *European Journal of Operational Research*, 246(3):787–799, 2015.

L. Berghman, R. Leus, and F. C. R. Spieksma. Optimal solutions for a dock assignment problem with trailer transportation. *Annals of Operations Research*, 213(1):1–23, 2014.

S. A. Brah and J. L. Hunsucker. Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 51(1):88–99, 1991.

C. Chamnanlor, K. Sethanan, C. F. Chien, and M. Gen. Re-entrant flow shop scheduling problem with time windows using hybrid genetic algorithm based on auto-tuning strategy. *International Journal of Production Research*, 52(9):2612–2629, 2014.

Z. L. Chen and G. L. Vairaktarakis. Integrated scheduling of production and distribution operations. *Management Science*, 51(4):614–628, 2005.

N. Christofides, R. Alvarez-Valdes, and J. M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3): 262–273, 1987.

M. Davari, E. Demeulemeester, R. Leus, and F. Talla Nobibon. Exact algorithms for single-machine scheduling with time windows and precedence constraints. *Journal of Scheduling*, 19(3):309–334, 2016.

D. Debels and M. Vanhoucke. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457–469, 2007.

F. Della Croce, A. Grosso, and F. Salassa. A matheuristic approach for the two-machine total completion time flow shop problem. *Annals of Operations Research*, 213(1):67–78, 2014.

M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.

M. Fischetti and A. Lodi. Repairing MIP infeasibility through local branching. *Computers & Operations Research*, 35(5):1436–1445, 2008.

M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1): 91–104, 2005.

J. M. Garcia and S. Lozano. Production and delivery scheduling problem with time windows. *Computers & Industrial Engineering*, 48(4):733–742, 2005.

M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

C. Gicquel, L. Hege, M. Minoux, and W. Van Canneyt. A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints. *Computers & Operations Research*, 39(3):629–636, 2012.

J. Grabowski and J. Pempera. Sequencing of jobs in some production system. *European Journal of Operational Research*, 125(3):535–550, 2000.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

S. Guirchoun, P. Martineau, and J. C. Billaut. Total completion time minimization in a computer system with a server and two parallel processors. *Computers & Operations Research*, 32(3): 599–611, 2005.

H. Hadda, N. Dridi, and S. Hajri-Gabouj. Exact resolution of the two-stage hybrid flow shop with dedicated machines. *Optimization Letters*, 8(8):2329–2339, 2014.

N. G. Hall, C. N. Potts, and C. Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102(3):223–243, 2000.

M. Haouari, L. Hidri, and A. Gharbi. Optimal scheduling of a two-stage hybrid flow shop. *Mathematical Methods of Operations Research*, 64(1):107–124, 2006.

R.-H. Huang and C.-L. Yang. Ant colony system for job shop scheduling with time windows. *The International Journal of Advanced Manufacturing Technology*, 39(1-2):151–157, 2008.

P. Jula and I. Kones. Continuous-time algorithms for scheduling a single machine with sequence-dependent setup times and time window constraints in coordinated chains. *International Journal of Production Research*, 51(12):3654–3670, 2013.

N. Kaveshgar and N. Huynh. Integrated quay crane and yard truck scheduling for unloading inbound containers. *International Journal of Production Economics*, 159:168–177, 2015.

R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333, 1996.

G. Komaki, E. Teymourian, and V. Kayvanfar. Minimising makespan in the two-stage assembly hybrid flow shop scheduling problem using artificial immune systems. *International Journal of Production Research*, 54(4):963–983, 2016.

G. C. Lee and Y. D. Kim. A branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness. *International Journal of Production Research*, 42(22): 4731–4743, 2004.

D. Lei and X. Guo. Hybrid flow shop scheduling with not-all-machines options via local search with controlled deterioration. *Computers & Operations Research*, 65:76–82, 2016.

J. K. Lenstra, A. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

M. Liu, X. Yang, J. Zhang, and C. Chu. Scheduling a tempered glass manufacturing system: a three-stage hybrid flow shop model. *International Journal of Production Research*, 55(20): 6084–6107, 2017.

E. Néron, P. Baptiste, and J. N. Gupta. Solving hybrid flow shop problem using energetic reasoning and global operations. *Omega*, 29(6):501–511, 2001.

Q. K. Pan. An effective co-evolutionary artificial bee colony algorithm for steelmaking-continuous casting scheduling. *European Journal of Operational Research*, 250(3):702–714, 2016.

Q. K. Pan, R. Ruiz, and P. Alfaro-Fernandez. Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*, 80: 50–60, 2017.

Y. Pan and L. Shi. Dual constrained single machine sequencing to minimize total weighted completion time. *IEEE Transactions on Automation Science and Engineering*, 2(4):344–357, 2005.

S. Panwalkar, R. Dudek, and M. Smith. Sequencing research and the industrial scheduling problem. In *Symposium on the Theory of Scheduling and its Applications*, volume 86, pages 29–38. Springer Berlin, 1973.

M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016.

D. Quadt and H. Kuhn. A taxonomy of flexible flow line scheduling procedures. *European Journal of Operational Research*, 178(3):686–698, 2007.

I. Ribas, R. Leisten, and J. M. Framinan. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, 37(8):1439–1454, 2010.

R. Ruiz and J. A. Vazquez-Rodriguez. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18, 2010.

T. Sawik. Mixed integer programming for scheduling surface mount technology lines. *International Journal of Production Research*, 39(14):3219–3235, 2001.

M. Schulze, J. Rieck, C. Seifi, and J. Zimmermann. Machine scheduling in underground mining: an application in the potash industry. *OR Spectrum*, 38(2):365–403, 2016.

O. Shahvari and R. Logendran. A comparison of two stage-based hybrid algorithms for a batch scheduling problem in hybrid flow shop with learning effect. *International Journal of Production Economics*, 195:227–248, 2018.

R. Simpson and A. Abakarov. Mixed-integer linear programming models for batch sterilization of packaged-foods plants. *Journal of Scheduling*, 16(1):59–68, 2013.

Y. Tan, L. Mönch, and J. W. Fowler. A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines. *Journal of Scheduling*, 21(2):209–226, 2018.

L. Tang and X. Wang. An improved particle swarm optimization algorithm for the hybrid flowshop scheduling to minimize total weighted completion time in process industry. *IEEE Transactions on Control Systems Technology*, 18(6):1303–1314, 2010.

V. T'kindt, N. Monmarché, F. Tercinet, and D. Laügt. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research*, 142(2):250–257, 2002.

C. Viergutz and S. Knust. Integrated production and distribution scheduling with lifespan constraints. *Annals of Operations Research*, 213(1):293–318, 2014.

S. Wang, M. Liu, and C. Chu. A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling. *International Journal of Production Research*, 53(4):1143–1167, 2015.

FACULTY OF ECONOMICS AND BUSINESS
Naamsestraat 69 bus 3500
3000 LEUVEN, BELGIË
tel. + 32 16 32 66 12
fax + 32 16 32 67 91
info@econ.kuleuven.be
www.econ.kuleuven.be