European Journal of Operational Research 251 (2016) 419–431

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Order acceptance and scheduling problems in two-machine flow shops: New mixed integer programming formulations

Rasul Esmaeilbeigi, Parisa Charkhgard*, Hadi Charkhgard

School of Mathematical and Physical Sciences, The University of Newcastle, Australia

ARTICLE INFO

Article history:
Received 1 August 2014
Accepted 30 November 2015
Available online 4 December 2015

Keywords:
Order acceptance
Scheduling
Mixed integer programming
Preprocessing
Valid inequalities

ABSTRACT

We present two new mixed integer programming formulations for the order acceptance and scheduling problem in two machine flow shops. Solving this optimization problem is challenging because two types of decisions must be made simultaneously: which orders to be accepted for processing and how to schedule them. To speed up the solution procedure, we present several techniques such as preprocessing and valid inequalities. An extensive computational study, using different instances, demonstrates the efficacy of the new formulations in comparison to some previous ones found in the relevant literature.

1. Introduction

Many manufacturing companies use Make-To-Order (MTO) production systems. In MTO systems, planning for the manufacture of a product will begin only when a customer order is received. The main advantage of these systems is that they give rise to low finished goods inventories. However, these systems have a significant disadvantage in that the lead time for the fulfillment of orders may result in significant financial loss for companies because of the loss of business due to production limitations. As a consequence, to remain competitive, companies employing these systems must decrease their order delivery times. This can be achieved by employing an accurate production plan that determines which orders should be accepted and how they should be scheduled. The solution to the Order Acceptance and Scheduling Problem (OASP) is an important step in the development of such a plan.

OASPs have been studied extensively over the past 20 years and a number of different versions of these problems exist. We refer interested readers to the literature survey by Slotnick (2011) for details. Versions of OASPs in which the objective functions maximize the total net revenue, i.e. the difference between sum of revenues and total weighted tardiness or lateness, have been studied by many authors in a single-machine environment. Slotnick and Morton (1996) are believed to be the first researchers who addressed this problem under the assumption of static arrivals, meaning that all jobs are assumed to be available at time zero. They proposed two heuristic algorithms and a Branch and Bound (B&B) technique to solve the problem in this case.

Later, Ghosh (1997) proved that an OASP with lateness penalties is NP-hard. He also presented two pseudo-polynomial time dynamic programming algorithms, and a polynomial-time approximation scheme in order to solve the problem. Slotnick and Morton (2007) considered tardiness related penalties instead of lateness penalties. They developed a B&B algorithm and a number of heuristics to solve this problem exactly with at most 10 jobs in about 6000 seconds on average. As far as we know, the largest instances of OASP with tardiness related penalties in a single-machine environment were solved by Nobibon and Leus (2011). They proposed two Mixed Integer Linear Programming (MILP) formulations and could solve instances of the problem with at most 50 jobs to optimality within two hours using the IBM ILOG CPLEX Optimizer (see http://www-01.ibm.com/software/info/ilog). In order to compute high quality solutions for large size instances of the problem, Rom and Slotnick (2009) developed a genetic algorithm. They showed that while their proposed approach is slower than the available heuristics appearing in the relevant literature, it generates solutions of higher quality.

Oğuz, Salman, and Yalçın (2010) added more assumptions to the OASP with tardiness related penalties in a single machine environment. They considered release dates for each job and sequence dependent setup times. They gave a MILP formulation of the problem and could solve instances of the problem with at most 15 jobs to optimality. To compute high quality solutions for larger size instances of the problem, they also developed three heuristics. Later, Cesaret, Oğuz, and Salman (2012) developed a tabu search algorithm for this problem. They showed that their proposed algorithm is faster and can provide solutions with higher quality when compared with previous

* Corresponding author. Tel.: +61 2 4926 1326.
E-mail address: parisa.charkhgard@uon.edu.au (P. Charkhgard).

http://dx.doi.org/10.1016/j.ejor.2015.11.036
0377-2217/

heuristics. Lin and Ying (2013) introduced a new artificial bee colony based algorithm to solve this problem. Their experimental results indicated that their proposed heuristic is competitive with the algorithm by Cesaret et al. (2012).

There are also a few studies about the OASP with tardiness related penalties in an m-machine permutation flow shop environment. For example, Xiao, Zhang, Zhao, and Kaku (2012) developed a partial optimization based simulated annealing algorithm to solve instances of the problem. Later, Lin and Ying (2015) presented a multi-initiator simulated annealing algorithm, and showed that the new heuristic outperforms the algorithm by Xiao et al. (2012). Recently, Lei and Guo (2015) addressed the biobjective version of the problem where the objectives are minimization of the makespan and maximization of the total net revenue. To solve instances of the problem, they employed a parallel neighborhood search algorithm and compared it with a tabu search and a variable neighborhood search algorithm.

In this paper, we consider the OASP in a 2-Machine Flow shop environment (OASP-2MF) which is recently addressed by Wang, Xie, and Cheng (2013a) and Wang, Xie, and Cheng (2013b). In Wang et al. (2013b), the authors tried to generalize the work of Slotnick and Morton (2007). They introduced two MILP formulations which could solve instances of the problem with up to 13 jobs within a one hour time limit using CPLEX. In addition, they proposed a B&B algorithm which simultaneously took into account job selection and scheduling and benefited from the use of some dominance rules in the pruning procedure. They showed that their purpose-built solver can solve larger sized instances of the problem with up to 20 jobs within an hour. In Wang et al. (2013a), the authors developed a modified artificial bee colony algorithm to compute good solutions for even larger instances of the problem.

The main contribution of our research is the development of two new MILP formulations for the OASP-2MF. In addition, to speed up the solution procedure, we present several enhancements (cuts and preprocessing techniques) which can reduce the size of the problem significantly and make the formulations stronger. Our new formulations have the following three desirable characteristics:

- The number of variables and constraints in these formulations is quadratically bounded by the number of jobs. Some previous researchers have developed time-indexed formulations for single-machine or 2-machine flow shop versions of the OASP, but the size of these formulations can increase dramatically if processing times are large.
- They outperform previous formulations even before applying the enhancements.
- CPLEX can solve instances of the problem that are 5 times larger than those solved by the purpose-built solver which is developed in Wang et al. (2013b).

We compare our new formulations after applying the enhancements and show that one of them performs far better than the other. Using our best formulation, CPLEX can achieve an optimality gap of less than 2 percent, on average, within 1800 seconds, even for instances of OASP-2MF with as many as 100 jobs.

The rest of the paper is organized as follows. In Section 2, we review some preliminary notation and results. In Section 3, we introduce two new MILP formulations for OASP-2MF. In Section 4, we discuss enhancements to make the formulations stronger. In Section 5, we report the results of a comprehensive computational study. Finally, in Section 6, we give some concluding remarks.

2. Preliminaries

In an OASP-2MF two decisions must be made at the same time: which orders to be accepted for processing and how to schedule them. We assume that the set of orders (*jobs*) is known in advance. Due to the flow shop structure of the problem, each job can be processed on machine 2 at some time after its processing on machine 1 has been completed.

We denote the set of jobs by $N = \{1, 2, \ldots, n\}$. The revenue and processing times of each job $i \in N$ on machines 1 and 2 are denoted by $u_i \in \mathbb{Z}^+$, $p_i^1 \in \mathbb{Z}^+$ and $p_i^2 \in \mathbb{Z}^+$ (where we use $\mathbb{Z}^+$ to denote the set of positive integers), respectively. We sometimes sort the processing times on each machine from small to large. We use $p_{[i]}^1$ and $p_{[i]}^2$ to denote the processing times in the ith position of the sorted lists, for machines 1 and 2, respectively. We denote the completion time of job $i \in N$ by C_i. We assume that each job $i \in N$ has a due date, denoted by $d_i \in \mathbb{Z}^+$, and that there is a delay penalty, denoted by $w_i \in \mathbb{Z}^+$, for each unit of the completion time which exceeds d_i, i.e., $C_i - d_i$ (note that due dates are positive integers). Also, there is no reward or penalty for early delivery. We sometimes refer to the delay time of the job $i \in N$ as its *tardiness*, and denote it by T_i, i.e., $T_i = \max\{0, C_i - d_i\}$. The *net revenue*, i.e., the difference between the revenue and the delay cost, of each job $i \in N$ is defined by $\pi_i := u_i - w_i \cdot T_i$. Furthermore, we assume that the goal is to maximize the total net revenue in the OASP-2MF.

Observe that, in an optimal solution, if job $i \in N$ is accepted, then $\pi_i \geq 0$. Moreover, $\pi_i = u_i$ if job $i \in N$ is accepted and fulfilled before its due date, and $\pi_i < u_i$ if job $i \in N$ is accepted and fulfilled after its due date. We sometimes refer to $u_i - \pi_i$ (or equivalently $w_i \cdot T_i$) as the *tardiness penalty* for job $i \in N$, if it is accepted. The following propositions provide the basis for the development of the different MILP formulations and enhancements in this paper. Note that Propositions 1 and 3 are straight forward to prove, and they are known results in the literature of the classical 2-machine flow shop problem (see for instance Baker, 1974; Kim, 1993). Therefore, we have omitted their proofs. Moreover, it should be mentioned that Wang et al. (2013b) used Proposition 1 to validate their MILP formulations.

Proposition 1. *For accepted jobs, there is an optimal schedule in which each job is processed on both machines in the same sequence.*

Proposition 2. *In an optimal schedule, $C_i^U := \frac{u_i}{w_i} + d_i$ is an upper bound for the completion time of an accepted job $i \in N$.*

Proof. Suppose that the assertion is not true. Therefore, there must exist a job $i \in N$ in an optimal schedule whose completion time C_i is strictly larger than C_i^U. The net revenue of job i is

$$\pi_i = u_i - w_i \cdot \max\{0, (C_i - d_i)\}.$$

Because $C_i^U < C_i$,

$$\pi_i < u_i - w_i \cdot \max\{0, (C_i^U - d_i)\}$$
$$= u_i - w_i \cdot \max\left\{0, \left(\frac{u_i}{w_i} + d_i - d_i\right)\right\} = 0.$$

This contradicts our assumption that the schedule is optimal since we can improve the total net revenue by simply rejecting job i. □

Note that Proposition 2 is independent of machine environment and processing restrictions.

Proposition 3. *Let S be the list of accepted jobs for a schedule and suppose that the jobs will be processed in the same order as they appear in the list. Let $J_k \in S$ be the job which is allocated to position k in the list S where $1 \leq k \leq |S|$ and $|S|$ is the number of elements of S. Then*

$$C_{J_k}^L := \max\left\{\sum_{1\leq q\leq k} p_{J_q}^1 + p_{J_k}^2, \sum_{1\leq q\leq k} p_{J_q}^2 + p_{J_1}^1\right\}$$

is a lower bound for the completion time of job J_k.

3. New formulations

In this section, we describe two new MILP formulations for the OASP-2MF. We then compare them with two Previous Formulations

(PF1 and PF2) proposed by Wang et al. (2013b) in terms of size complexity and the number of *disjunctive constraints* (meaning those with a big M parameter). The validity of both of these formulations is confirmed by Proposition 1.

New Formulation 1 (NF1): In order to describe the model, we first define some sets of decision variables. For each job $i \in N$, we use a binary decision variable to indicate whether the job is accepted or rejected,

$$y_i := \begin{cases} 1 & \text{If job } i \text{ is accepted,} \\ 0 & \text{Otherwise.} \end{cases}$$

For each pair of jobs $(i, j) \in N^2$ with $i \neq j$, we define a binary decision variable to indicate which one is to proceed earlier,

$$z_{ij} := \begin{cases} 1 & \text{If jobs } i \text{ and } j \text{ are accepted, and job } j \\ & \quad \text{is processed after job } i, \\ 0 & \text{Otherwise.} \end{cases}$$

We also use continuous decision variables C_i and T_i to represent the completion time and tardiness, respectively, of each job $i \in N$. Using these decision variables, our first formulation of the OASP-2MF can be expressed as follows:

$$\max \sum_{i \in N} (u_i \cdot y_i - w_i \cdot T_i) \tag{1}$$

$$\text{subject to } z_{ij} + z_{ji} \leq y_i \quad \forall i, j \in N \text{ with } i \neq j \tag{2}$$

$$z_{ij} + z_{ji} \geq y_i + y_j - 1 \quad \forall i, j \in N \text{ with } i < j \tag{3}$$

$$C_i + p_j^2 + (z_{ij} - 1) \cdot M \leq C_j \quad \forall i, j \in N \text{ with } i \neq j \tag{4}$$

$$\sum_{i \in N \setminus \{j\}} p_i^1 \cdot z_{ij} + (p_j^1 + p_j^2) \cdot y_j \leq C_j \quad \forall j \in N \tag{5}$$

$$T_i \geq C_i - d_i \quad \forall i \in N \tag{6}$$

$$y_i \in \{0, 1\} \quad \forall i \in N \tag{7}$$

$$z_{ij} \in \{0, 1\} \quad \forall i, j \in N \text{ with } i \neq j \tag{8}$$

$$C_i, T_i \geq 0 \quad \forall i \in N. \tag{9}$$

The objective function maximizes the total net revenue. Constraint (2) guarantees that each job $i \in N$ is accepted if it is processed before or after another job $j \in N \setminus \{i\}$. Constraint (3) ensures that if jobs $i, j \in N$ are accepted, then either job i is processed after job j or vice versa. Constraint (4) states that if job $i \in N$ precedes job $j \in N \setminus \{i\}$, then the completion time of job j should be no shorter than the sum of the completion time of job i and the processing time of job j on machine 2. In these constraints, M is a sufficiently large positive number. Constraint (5) implies that if job $j \in N$ is accepted, then its completion time is no shorter than the sum of the processing times of the jobs which should be processed before job j on machine 1 plus the processing times of job j on machines 1 and 2. Constraint (6) captures the tardiness of each job $i \in N$. Finally, constraints (7)–(9) specify the domains of each decision variable. Observe that in an optimal solution, if a job $i \in N$ is rejected then $T_i = 0$, and so $C_i \leq d_i$.

Before explaining the second formulation, we make some comments about constraint (4). First, it is not difficult to see that constraint (4) implies that $z_{ij} + z_{ji} \leq 1$ for all $i, j \in N$ with $i \neq j$. This is because, for $i \neq j$, if jobs $i, j \in N$ are accepted, then either $C_i < C_j$ or $C_j < C_i$ (note that $p_j^2 \in \mathbb{Z}^+$). This implies that the formulation is valid even without constraint (2). However, we include constraint (2) because they can be interpreted as a simple valid inequality for the model and our computational results show that this can improve the run time of CPLEX. Secondly, we can safely set $M = \sum_{i=1}^{n} (p_i^1 + p_i^2)$. Later, we show that it might be possible to find an even smaller positive value for M.

Finally, constraint (4) plays an important role in enforcing a *transitive relationship* between the decision variables z_{ij}. Let $a, b, c \in N$ be three jobs such that $a \neq b \neq c$. It is easy to see that if $z_{ab} = 1$ and $z_{bc} = 1$, then z_{ac} must also be equal to one. To enforce such a relationship between binary decision variables, researchers usually add some inequalities to their models. For instance, Dyer and Wolsey (1990); Nemhauser and Savelsbergh (1992); Unlu and Mason (2010), and Reisi-Nafchi and Moslehi (2015) added the constraint

$$z_{ab} + z_{bc} + z_{ca} \leq 2$$

to their models while some other researchers, such as Chudak and Hochbaum (1999), added the constraint

$$z_{ab} \leq z_{ac} + z_{cb}$$

to their models in order to ensure transitivity of their decision variables. Proposition 4 shows that we do not need to add any of the above logical constraints to NF1 because constraints (3) and (4) imply the existence of a transitive relationship between our decision variables.

Proposition 4. *Constraints (3) and (4) imply the existence of a transitive relationship between the decision variables z_{ij} where $i, j \in N$ and $i \neq j$.*

Proof. Let $a, b, c \in N$ be three accepted jobs such that $a \neq b \neq c$. Suppose that $z_{ab} = 1$ and $z_{bc} = 1$. We prove z_{ac} must be exactly one.

It is easy to see from constraint (4) that since $p_j^2 \in \mathbb{Z}^+$ (that is, $p_j^2 \geq 1$) and $z_{ab} = z_{bc} = 1$, we must have $C_a + 1 \leq C_b$ and $C_b + 1 \leq C_c$. Consequently, $C_a + 2 \leq C_c$. Since a and c are accepted jobs, constraints (3) and (4), or equivalently constraints (2) and (3), imply that exactly one of z_{ac} and z_{ca} must be equal to one.

Suppose that z_{ca} is equal to one. Constraint (4) implies that $C_c + 1 \leq C_a$ which gives a contradiction. Therefore, z_{ac} must be equal to one. □

New Formulation 2 (NF2): In order to describe the model, we first define some new sets of decision variables. Let $Q := \{1, 2, \ldots, n\}$ be the set of positions (in the sequencing list) to which jobs can be assigned. For each job $i \in N$ and for each position $k \in Q$, we define a binary decision variable x_{ik} which indicates whether the job is assigned to that position, that is,

$$x_{ik} := \begin{cases} 1 & \text{If job } i \text{ is accepted and assigned to position } k, \\ 0 & \text{Otherwise.} \end{cases}$$

For each position $k \in Q$, we define a continuous decision variable C'_k which stores the completion time of the job allocated to position k. Using these decision variables, our second formulation of the OASP-2MF can be expressed as follows:

$$\max \sum_{i \in N} (u_i \cdot y_i - w_i \cdot T_i) \tag{1}$$

$$\text{subject to } \sum_{k \in Q} x_{ik} = y_i \quad \forall i \in N \tag{10}$$

$$\sum_{i \in N} x_{ik} \leq 1 \quad \forall k \in Q \tag{11}$$

$$C'_{k+1} \geq C'_k + \sum_{i \in N} p_i^2 \cdot x_{i,k+1} \quad \forall k \in Q \setminus \{n\} \tag{12}$$

$$C'_k \geq \sum_{i \in N} \sum_{\substack{k' \in Q, \\ k' \leq k}} p_i^1 \cdot x_{ik'} + \sum_{i \in N} p_i^2 \cdot x_{ik} \quad \forall k \in Q \tag{13}$$

$$T_i \geq C'_k - d_i + M \cdot (x_{ik} - 1) \quad \forall i \in N \text{ and } k \in Q \tag{14}$$

$$y_i \in \{0, 1\} \quad \forall i \in N \tag{15}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in N \text{ and } k \in Q \tag{16}$$

$$C'_k \geq 0 \quad \forall k \in Q \tag{17}$$

$$T_i \geq 0 \quad \forall i \in N. \tag{18}$$

The objective function in this formulation also maximizes the total net revenue. Constraint (10) ensures that each accepted job is allocated to exactly one position (evidently, the rejected jobs will not be allocated to any position). Constraint (11) states that each position cannot be assigned to more than one job. Constraint (12) guarantees that the job in the (k+1)th position cannot be completed in a time any shorter than the sum of the time it takes to complete the job in position k and the processing time for the (k+1)th job on machine 2. Constraint (13) implies that the job allocated to position k cannot be completed in a shorter time than the sum of the processing times on machine 1 of the jobs in the earlier positions added to the processing times for the job in position k on machines 1 and 2. Constraint (14) captures the tardiness of each job $i \in N$. Note that we can safely set $M = \sum_{i=1}^{n} (p_i^1 + p_i^2)$ in these constraints. Finally, constraints (15)–(18) define the domains of each decision variable. Observe that in an optimal solution, if a job $i \in N$ is rejected then $T_i = 0$. Also, for position $k \in Q$, $C'_k \geq C'_{k'}$ for all $k' \in \{1, \ldots, k-1\}$.

It should be mentioned that in NF2, constraint (10) can be written as $\Sigma_{k \in Q} x_{ik} \leq 1$ and hence the binary (auxiliary) variable y_i for $i \in N$ can be removed from the model (we can substitute $\Sigma_{k \in Q} x_{ik}$ for y_i in the objective function). However, we have observed that the presence of these variables increases the efficiency of our algorithm during our computational experiments and so we have chosen to include these variables in the expression of our model. One reason for this increased efficiency could be the new branching strategy which is defined by introducing the new (auxiliary) variables y_i for $i \in N$ that influence the value of $\Sigma_{k \in Q} x_{ik}$ in the B&B search tree.

In NF2, empty positions are allowed to appear between two consecutive jobs. This means that the formulation is symmetric. An integer linear program is symmetric if its variables can be permuted without changing the structure of the problem (Margot, 2010). In Section 4.3 we break the symmetry of NF2 by using a set of valid inequalities.

Next, we compare NF1 and NF2 with PF1 and PF2 in terms of size complexity. Interested readers may refer to Appendix to see PF1 and PF2 in detail. Note that PF2 is a time-indexed formulation, and so the number of binary variables and constraints of this formulation is highly dependent on the processing times. As a consequence, to evaluate the size complexity of PF2, we should know the number of breakpoints in time defined for each job $i \in N$, because that indicates how many time-indexed variables exist in the model. In PF2, for each job $i \in N$, the number of breakpoints in time for job $i \in N$ is $\Gamma - p_i^1 - p_i^2 + 1$, where the parameter Γ is an upper bound for the number of breakpoints. Suppose that the processing times are generated randomly and independently from the discrete uniform distribution on the interval $[1, L]$ where $L \in \mathbb{Z}^+$. If $\tilde{N} := \{i \in N : p_i^1 \geq p_i^2\}$ be the set of jobs whose processing times on machine 1 are not less than their processing times on machine 2, then Γ is defined by

$$\Gamma = \sum_{i \in \tilde{N}} p_i^1 + \sum_{i \in N \setminus \tilde{N}} p_i^2 + L.$$

Proposition 5. *If $p_i^1, p_i^2 \in \mathbb{Z}^+$ for $i \in N$ are generated randomly and independently from the discrete uniform distribution on $[1, L]$, then the expected value $E[\Gamma]$ of Γ is given by*

$$E[\Gamma] = \frac{n(4L-1)(L+1)}{6L} + L.$$

Proof. If $X_i := \max(p_i^1, p_i^2)$, then we certainly have $\Gamma = \sum_{i \in N} X_i + L$. Consequently, because the processing times are generated randomly and independently,

$$E[\Gamma] = E\left[\sum_{i \in N} X_i + L\right] = nE[X] + L.$$

Let $F_{X_i}(x)$ be the cumulative distribution function of X_i for each $i \in N$, where $x \in [1, L]$ is an integer. Then

$$F_{X_i}(x) = Pr(X_i \leq x) = Pr(p_i^1 \leq x) \cdot Pr(p_i^2 \leq x) = \left(\frac{x}{L}\right)^2.$$

So,

$$Pr(X_i = x) = F_{X_i}(x) - F_{X_i}(x-1) = \frac{2x-1}{L^2}.$$

Therefore,

$$E[X] = E[X_i] = \sum_{x=1}^{L} x \cdot Pr(X_i = x) = \frac{(4L-1)(L+1)}{6L}$$

and our result follows. □

Table 1
The comparison of the formulations.

Model	NBVs	NCVs	NCs	NDCs
PF1	$2n^2$	$2n$	$n^3 + 2n^2 + 2n - 1$	$2n^2 - 2n$
PF2	$n\Gamma$ *	$2n$	$n^2\Gamma + 2n\Gamma + 2n$ *	$n^2\Gamma + n\Gamma$ *
NF1	n^2	$2n$	$2.5n^2 - 0.5n$	$n^2 - n$
NF2	$n^2 + n$	$2n$	$n^2 + 4n - 1$	n^2

* Approximate quantities.

Table 1 summarizes the Number of Binary Variables (NBVs), the Number of Continuous Variables (NCVs), the Number of Constraints (NCs) and the Number of Disjunctive Constraints (NDCs) required by each model to formulate an OASP-2MF with n jobs.

As can be seen from Table 1, the NBVs and NDCs of the PF1, NF1 and NF2 are bounded by $O(n^2)$. However, the NBVs and NDCs of PF2 are bounded by $O(n\Gamma)$ and $O(n^2\Gamma)$, respectively. Moreover, for NF1 and NF2, the NCs is bounded by $O(n^2)$. By contrast, the NCs is bounded by $O(n^3)$ and $O(n^2\Gamma)$ for the PF1 and PF2, respectively. The NCVs is linearly bounded in the number of jobs for all formulations.

Tables 2 –4 give numerical values which show the differences between the formulations for different problem sizes when processing times are generated randomly from the intervals [1, 10] and [1, 100]. Note that the value of Γ is problem dependent and we cannot compute it without knowing the values of the processing times. However, Proposition 5 allows the expected value of Γ to be computed if the processing times are generated randomly from a discrete uniform distribution. So we have used $E[\Gamma]$ instead of Γ in the tables. It is evident from the tables that the numbers of variables and constraints for NF1 and NF2 are significantly smaller than the numbers of variables and constraints for PF1 and, especially, PF2. Note that PF2 is a time-indexed formulation.

4. Enhancements

In this section, we provide some techniques which can possibly make NF1 and NF2 stronger and reduce their size complexity dramatically. Some of these techniques can be applied to only one of the formulations, but the others can be used with both formulations.

4.1. Moderating big-M coefficients

LP relaxation of a MILP formulation with disjunctive sets may be improved significantly by moderating its big-M parameters. In NF1 and NF2, constraints (4) and (14) contain big-M parameters. As was

Table 2
The comparison of the formulations in terms of NBVs.

Example	Problem size			Formulations				
	n	E[Γ] ($p_i^1, p_i^2 \in [1, 10]$)	E[Γ] ($p_i^1, p_i^2 \in [1, 100]$)	PF1	PF2 * ($p_i^1, p_i^2 \in [1, 10]$)	PF2 * ($p_i^1, p_i^2 \in [1, 100]$)	NF1	NF2
1	12	95.8	906.0	288	1150	10,872	144	156
2	25	188.8	1,779.1	1250	4719	44,478	625	650
3	50	367.5	3458.3	5000	18,375	172,913	2500	2550
4	100	725.0	6816.5	20,000	72,500	681,650	10,000	10,100

* Approximate averages of NBVs.

Table 3
The comparison of the formulations in terms of NCs.

Example	Problem size			Formulations				
	n	E[Γ] ($p_i^1, p_i^2 \in [1, 10]$)	E[Γ] ($p_i^1, p_i^2 \in [1, 100]$)	PF1	PF2 * ($p_i^1, p_i^2 \in [1, 10]$)	PF2 * ($p_i^1, p_i^2 \in [1, 100]$)	NF1	NF2
1	12	95.8	906.0	2039	16,118	152,229	354	191
2	25	188.8	1779.1	16,924	127,456	1,200,959	1550	724
3	50	367.5	3458.3	130,099	955,600	8,991,550	6225	2699
4	100	725.0	6816.5	1,020,199	7,395,200	69,528,500	24,950	10,399

* Approximate averages of NCs.

Table 4
The comparison of the formulations in terms of NDCs.

Example	Problem size			Formulations				
	n	E[Γ] ($p_i^1, p_i^2 \in [1, 10]$)	E[Γ] ($p_i^1, p_i^2 \in [1, 100]$)	PF1	PF2 * ($p_i^1, p_i^2 \in [1, 10]$)	PF2 * ($p_i^1, p_i^2 \in [1, 100]$)	NF1	NF2
1	12	95.8	906.0	264	14,945	141,333	132	144
2	25	188.8	1779.1	1200	122,688	1,156,431	600	625
3	50	367.5	3458.3	4900	937,125	8,818,538	2450	2500
4	100	725.0	6816.5	19,800	7,322,500	68,846,650	9900	10,000

* Approximate averages of NDCs.

discussed previously, one simple way to moderate the big-M parameters for these formulations is to set $M = \sum_{i=1}^{n}(p_i^1 + p_i^2)$ in both formulations. However, this is not the smallest trivial value that can be assigned to M. Proposition 2 allows us to compute smaller values for M. In NF1, we can use $\hat{M}_{ij} := \lfloor C_i^U \rfloor + p_j^2$ where $i, j \in N$ and $i \neq j$ to replace M in constraint (4) and obtain the new constraints:

$$C_i + p_j^2 + (z_{ij} - 1) \cdot \hat{M}_{ij} \leq C_j \quad \forall i, j \in N \text{ and } i \neq j.$$

It is easy to see that $\hat{M}_{ij}$ is a suitable value for M because if $z_{ij} = 0$, the inequality $C_i - \lfloor C_i^U \rfloor \leq C_j$ holds. Note that we use $\lfloor C_i^U \rfloor$ instead of C_i^U because we have assumed that all parameters in the problem are positive integers. So, the completion time of each accepted job should also be a positive integer. In NF2, we can substitute $\check{M}_i := \max_{j \in N} \lfloor C_j^U \rfloor - d_i$ where $i \in N$ for M in constraint (14). This yields the new constraints:

$$T_i \geq C'_k - d_i + \check{M}_i \cdot (x_{ik} - 1) \quad \forall i \in N \text{ and } k \in Q.$$

It is easy to see that $\check{M}_i$ is a suitable value to replace M because if $x_{ik} = 0$, the inequality $C'_k - \max_{j \in N} \lfloor C_j^U \rfloor \leq T_i$ holds.

To illustrate the differences between M, $\hat{M}_{ij}$ and $\check{M}_i$, we compare their expected values, i.e., respectively $E(\hat{M}_{ij})$, $E(\check{M}_i)$, and $E(M)$, in an example with n jobs. We assume that due dates are generated randomly from a uniform distribution on the interval $[2L, nL]$, but that all the other parameters are generated uniformly from the interval $[1, L]$. It is not hard to see that $E(\frac{u_i}{w_i}) \approx \ln(\sqrt{L})$, so $E(\hat{M}_{ij}) \approx \frac{L}{2}(n + 3) + \ln(\sqrt{L})$. Moreover, because $\max_{j \in N} \lfloor C_j^U \rfloor \leq L(n + 1)$, we see that $E(\check{M}_i) \leq \frac{1}{2}nL$. However, $E(M) = n(L + 1)$ which is much larger than both of the calculated values for $E(\hat{M}_{ij})$ and $E(\check{M}_i)$.

4.2. Valid inequalities for NF1

Knapsack constraints have long been studied in operations research (see, for instance, Atamtürk & Savelsbergh, 2005). If modern commercial MIP solvers identify these inequalities in the model, they can possibly make the formulation stronger by generating some additional valid inequalities such as *cover cuts* (see, for instance, Avella, Boccia, & Vasilyev, 2012; Kaparis & Letchford, 2010). To exploit this ability of commercial solvers, we propose to add the following two knapsack constraints to NF1:

$$\sum_{i \in N \setminus \{j\}} p_i^1 \cdot z_{ij} \leq \left(\lfloor C_j^U \rfloor - p_j^1 - p_j^2\right) \cdot y_j \quad \forall j \in N. \tag{19}$$

$$\sum_{i \in N \setminus \{j\}} p_i^2 \cdot z_{ij} \leq \left(\lfloor C_j^U \rfloor - p_{[1]}^1 - p_j^2\right) \cdot y_j \quad \forall j \in N. \tag{20}$$

Note that the validity of these knapsack constraints for NF1 is ensured by Propositions 2 and 3. In other words, we may deduce from Propositions 2 and 3 that if $y_j = 1$ where $j \in N$, then

$$\max\left(\sum_{i \in N \setminus \{j\}} p_i^1 \cdot z_{ij} + p_j^1 + p_j^2, \sum_{i \in N \setminus \{j\}} p_i^2 \cdot z_{ij} + p_{[1]}^1 + p_j^2\right) \leq \lfloor C_j^U \rfloor.$$

This inequality is valid because the left hand side of each inequality gives a lower bound for the completion time.

To see how cover cuts can be generated by commercial solvers, consider constraint (19) and job $j \in N$. A set $K_j \subseteq N \setminus \{j\}$ is called a *cover* if

$$\sum_{i \in K_j} p_i^1 > \left(\lfloor C_j^U \rfloor - p_j^1 - p_j^2\right).$$

For any cover K_j, the cover cut $\sum_{i\in K_j} z_{ij} \le |K_j| - 1$ is a valid inequality for NF1 and can be added to the formulation.

Proposition 2 also allows us to add some valid bounding inequalities for the completion time and tardiness of each job $i \in N$ to the formulation of NF1:

$$T_i \le \lfloor C_i^U \rfloor - d_i \quad \forall i \in N \tag{21}$$

$$C_i \le \lfloor C_i^U \rfloor \quad \forall i \in N. \tag{22}$$

Adding bounding inequalities is useful because they allow commercial MIP solvers to eliminate some of the variables in their preprocessing phase. For instance, in inequality (21), we may place bounds on the tardiness values for each job. As a consequence, if $\lfloor C_i^U \rfloor = d_i$, commercial solvers can set T_i equal to zero and eliminate this variable from the mathematical formulation.

4.3. Valid inequalities for NF2

We first discuss some valid inequalities which were introduced by Nobibon and Leus (2011) to solve the OASP in a single-machine environment, namely

$$\sum_{i=1}^{n} x_{i,k+1} \le \sum_{i=1}^{n} x_{ik} \quad \forall k \in Q\backslash\{n\}. \tag{23}$$

These inequalities are useful because they break the symmetry in the formulation by removing any empty positions that appear between two consecutive jobs. In other words, these constraints ensure that position $k+1$ is empty if position k is empty. Interested readers may refer to Sherali and Smith (2001) for the application of such hierarchical constraints in breaking the symmetry.

Valid inequality (23) can be implemented differently by introducing binary auxiliary variables v_k for all $k \in Q$ and adding the following constraints:

$$\sum_{i=1}^{n} x_{ik} = v_k \quad \forall k \in Q \tag{24}$$

$$v_{k+1} \le v_k \quad \forall k \in Q\backslash\{n\}. \tag{25}$$

As will be shown in Section 5, this implementation results in a significant improvement in the runtime for two main reasons. First, the number of nonzero elements in the coefficient matrix of the constraints is sharply reduced by $2n^2 - 5n + 2$. This is because constraint (25) has simpler representation in comparison with constraint (23). Moreover, the addition of constraint (24) to the formulation makes constraint (11) redundant and allows their removal. Second, the introduction of the auxiliary variables v_k for all $k \in Q$ provides a new type of branching strategy which influences the value of $\sum_{i=1}^{n} x_{ik}$ in the B&B search tree. More precisely, by introducing an auxiliary variable v_k, we can branch on the position k, and decide whether it is empty or not. It is evident from constraints (24) and (25) that branching on position k can be effective since if $v_k = 0$ then $x_{ik'} = 0$ for each $i \in \{1, \ldots, n\}$ and $k' \in \{k, \ldots, n\}$.

Note that introducing auxiliary variables is quite common, both in the contexts of constraint programming and of integer programming, because they can guide branching schemes. However, as we have shown, they can also be useful in decreasing the number of nonzero elements in the matrix of coefficients.

Another type of valid inequalities which can be added to NF2 is bounding inequalities such as

$$T_i \le \lfloor C_i^U \rfloor - d_i \quad \forall i \in N \tag{26}$$

$$C'_k \le \sum_{i\in N} \lfloor C_i^U \rfloor \cdot x_{ik} + C_{max}^U \cdot (1 - v_k) \quad \forall k \in Q \tag{27}$$

where $C_{max}^U := \max_{i\in N} \lfloor C_i^U \rfloor$. These inequalities are similar to the ones we added to NF1. Evidently, for a position k, inequality (27) might be active only if a job is assigned to position k, and it implies that the completion time of the job allocated to position k cannot exceed its upper bound. Note that we can also add similar knapsack inequalities to the ones proposed for NF1 to NF2 as follows:

$$\sum_{i\in N} \sum_{\substack{q\in Q,\\ q\le k}} p_i^1 \cdot x_{iq} + \sum_{i\in N} p_i^2 \cdot x_{ik} \le \sum_{i\in N} \lfloor C_i^U \rfloor \cdot x_{ik} + C_{max}^U \cdot (1 - v_k)$$
$$\forall k \in Q$$
$$\sum_{i\in N} \sum_{\substack{q\in Q,\\ q\le k}} p_i^2 \cdot x_{iq} + \sum_{i\in N} p_i^1 \cdot x_{i1} \le \sum_{i\in N} \lfloor C_i^U \rfloor \cdot x_{ik} + C_{max}^U \cdot (1 - v_k)$$
$$\forall k \in Q.$$

However, these knapsack inequalities do not improve the runtime of NF2 when used in conjunction with constraint (27). This is mainly because constraint (27) is stronger than the knapsack inequalities. Both the knapsack constraints and constraint (27) have the same right hand sides, but the left hand sides of the knapsack inequalities give a lower bound for C'_k (see Proposition 3).

Note that constraint (27) imposes a relationship between the binary variables that are used in the knapsack inequalities (see the right hand sides) for NF2. This is not the case for NF1, and so there is no relationship between the knapsack inequalities for NF1 and constraint (22). Thus, knapsack inequalities can improve efficiency for NF1, even in the presence of the bounding constraint (22).

4.4. Preprocessing for NF2

We will now describe two preprocessing techniques which can be applied to NF2. We also describe how these techniques can be used in practice. The development of these techniques was based on the fact that we are able to compute a lower bound, denoted by C_{iq}^L, for the completion time of each job $i \in N$ if it is located in position $q \in Q$. Consequently, if the obtained lower bound $C_{iq}^L \ge C_i^U$, we can safely set $x_{iq} = 0$. Since we have assumed that all parameters for the problem are positive integers, we can use $\lceil C_{iq}^L \rceil \ge C_i^U$ instead of $C_{iq}^L \ge C_i^U$ as a condition to set $x_{iq} = 0$. In other words, to reduce the size of the problem, if $\lceil C_{iq}^L \rceil \ge C_i^U$, then the variable x_{iq} should not be generated for NF2. Note that in general, we cannot use $\lceil C_{iq}^L \rceil \ge \lfloor C_i^U \rfloor$ instead of $\lceil C_{iq}^L \rceil \ge C_i^U$ as a condition to set $x_{iq} = 0$.

Suppose that $C_i^U \in \mathbb{Z}$. If $\lceil C_{iq}^L \rceil = C_i^U$, then we can safely set $x_{iq} = 0$. If job i is completed at time C_i^U, then its net revenue is zero, i.e., $\pi_i = 0$. Therefore, setting $x_{iq} = 0$ does not make the objective value worse. However, if $C_i^U \in \mathbb{R}\backslash\mathbb{Z}$ and $\lceil C_{iq}^L \rceil = \lfloor C_i^U \rfloor$, we cannot set $x_{iq} = 0$. This is because if the job is completed at time $\lfloor C_i^U \rfloor$, then its net revenue is strictly positive, $\pi_i > 0$. Therefore, setting $x_{iq} = 0$ may cut off global optimal solutions of the OASP-2MF.

Next, we introduce some notation to facilitate the presentation and discussion of our preprocessing techniques. Let $A_i := \{q \in Q : \lceil C_{iq}^L \rceil \ge C_i^U\}$ be the set of *impermissible positions* for job $i \in N$. Similarly, we denote the set of *impermissible jobs* for position $q \in Q$ by $B_q := \{i \in N : \lceil C_{iq}^L \rceil \ge C_i^U\}$. Evidently, before beginning our preprocessing procedure, these sets are empty. However, in each iteration of the preprocessing procedure, we update them. Observe that the elements of B_1 cannot be assigned to any position. As a consequence, they should be rejected straight away.

The First Preprocessing Technique (PRE1): By Proposition 3, regardless of which job is allocated to position $q \in Q$, we are able to compute a lower bound, denoted by C_q^L, for its completion time. This can be done simply by computing

$$C_q^L = \max\left\{\sum_{k=1}^{q} p_{[k]}^1 + p_{[q]}^2, \sum_{k=1}^{q} p_{[k]}^2 + p_{[1]}^1\right\}.$$

Consequently, we can set $C^L_{iq} = C^L_q$ for all $i \in N$ and update the elements of sets A_i and B_q.

It is easy to see that a better lower bound may be computed if we compute C^L_{iq} directly instead of C^L_q. This can be done simply by fixing $i \in N$ to position $q \in Q$. Let $\{\tilde{p}^1_{[1]}, \ldots, \tilde{p}^1_{[n-1]}\} := \{p^1_{[1]}, \ldots, p^1_{[n]}\} \backslash \{p^1_i\}$ and $\{\tilde{p}^2_{[1]}, \ldots, \tilde{p}^2_{[n-1]}\} := \{p^2_{[1]}, \ldots, p^2_{[n]}\} \backslash \{p^2_i\}$ be the sets of sorted preprocessing times (from small to large) when job $i \in N$ is eliminated. The value C^L_{iq} can be computed as follows:

$$C^L_{iq} = p^2_i + \max\left\{\sum_{k=1}^{q-1} \tilde{p}^1_{[k]} + p^1_i, \sum_{k=1}^{q-1} \tilde{p}^2_{[k]} + p^1_{[1]}\right\}.$$

Because computing C^L_{iq} provides us with a better lower bound than C^L_q, we use it in this paper. Note that the main advantage of using processing technique PRE1 is its speed because its implementation does not require any significant computational effort. We later show in Section 5 that PRE1 works well and its quality is just a little bit worse than the preprocessing technique that is introduced next.

The Second Preprocessing Technique (PRE2): Another way of computing C^L_q, regardless of which job is allocated to position $q \in Q$ is to modify NF2. Let $H_q := \{1, \ldots, q\}$ be the set of positions in Q, up to and including $q \in Q$. To compute C^L_q, we assume that for all $k \in H_q \backslash \{q\}$, C^L_k has been previously computed and that the sets A_i and B_k have been updated at each step. The lower bound C^L_q can be obtained by solving the following optimization problem:

$$C^L_q = \min\ C'_q \tag{28}$$

$$\text{subject to} \sum_{i \in N \backslash B_k} x_{ik} = 1 \quad \forall k \in H_q \tag{29}$$

$$\sum_{k \in H_q \backslash A_i} x_{ik} \leq 1 \quad \forall i \in N \backslash B_1 \tag{30}$$

$$C'_{k+1} \geq C'_k + \sum_{i \in N \backslash B_{k+1}} p^2_i \cdot x_{i,k+1} \quad \forall k \in H_q \backslash \{q\} \tag{31}$$

$$C'_k \geq \sum_{i \in N \backslash B_1} \sum_{\substack{k' \in H_q \backslash A_i, \\ k' \leq k}} p^1_i \cdot x_{ik'} + \sum_{i \in N \backslash B_k} p^2_i \cdot x_{ik} \quad \forall k \in H_q \tag{32}$$

$$C'_k \leq \sum_{i \in N \backslash B_k} \lfloor C^U_i \rfloor \cdot x_{ik} \quad \forall k \in H_q \tag{33}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in N \backslash B_1 \text{ and } k \in H_q \tag{34}$$

$$C'_k \geq 0 \quad \forall k \in H_q. \tag{35}$$

The objective is to minimize C'_q, the completion time of the job which should be assigned to position $q \in Q$. Constraint (29) ensures that all positions are filled. Constraint (30) guarantees that each job is allocated to at most one position. Constraints (31) and (32) are equivalent to constraints (12) and (13) in NF2, respectively. Constraint (33) ensures that the completion time of the job allocated to a position is not more than its upper bound. Finally, constraints (34) and (35) determine the domains of the variables.

Note that the presented optimization problem minimizes the make span of all allocated jobs in H_q. Therefore, the objective value gives a lower bound on the completion time of the job that has been allocated to position $q \in Q$.

As was discussed for PRE1, after computing C^L_q, we can set $C^L_{iq} = C^L_q$ for all $i \in N$ and update the elements of sets A_i and B_q. It is easy to see that, if the optimization problem is infeasible for C^L_q, then we can set $B_k = \{1, \ldots, n\}$ for all $k \in \{q, q+1, \ldots, n\}$ and add the elements of the set $\{q, q+1, \ldots, n\}$ to A_i for all $i \in N$.

Observe that the number of optimization problems that need to be solved is at most n. Unfortunately, this is too time consuming because each of these problems is, itself, a MILP. Consequently, to speed up the procedure, we solve the LP-relaxation of these optimization problems in this paper.

Table 5
The data for the example.

Job	1	2	3	4	5
u_i	2	5	6	3	2
w_i	5	10	9	2	8
p^1_i	6	10	7	3	7
p^2_i	1	4	5	7	6
d_i	24	23	14	15	35
C^U_i	24.40	23.50	14.67	16.50	35.25

Note that, in a similar way to the one discussed for PRE1, we can compute C^L_{iq} directly rather than C^L_q. This can be done simply by adding the constraint $x_{iq} = 1$ to the optimization problem. However, in this situation, the number of optimization problems that would then need to be solved is bounded above by n^2. Unfortunately, this is not a practical approach because it is too time consuming even if we solve only the LP-relaxations.

An example: In order to illustrate how the preprocessing techniques work, we consider a simple example which was introduced by Wang et al. (2013b). The parameters for the problem are given in Table 5. The upper bound on the completion time of each job is given in the last row of the table.

The output of algorithms PRE1 and PRE2 for the given example is shown in Table 6 where column *EV(percent)* gives the percentage of eliminated x_{ik} variables, i.e., $EV = \frac{\sum_{i \in N} |A_i|}{n^2} \times 100$. As can be seen from the table, both preprocessing techniques have roughly the same effect. Technique PRE1 could remove 44 percent and PRE2 could omit 40 percent of variables. The difference between PRE1 and PRE2 is only because of Position 2. Here PRE1 could recognize that Job 3 was unable to be allocated to it, but PRE2 could not.

Note that because C^L_{iq} is directly computed in PRE1, rather than computing C^L_q as in PRE2, PRE1 could eliminate more variables in this particular example. However, as we will show in our later computational experiments, PRE2 performs better for larger size instances because it considers processing times on both machines simultaneously (in the optimization problem) to compute a lower bound.

5. Computational results

To evaluate the performance of the proposed formulations and improvement techniques, we conducted an extensive computational study. We used the C++ programming language to code the formulations and preprocessing techniques and used CPLEX 12.4 to solve the formulations. All the experiments were run on a computer with a single-core 2.5 gigahertz Intel processor and 4.0 gigabyte RAM.

We generated 7 classes of instances denoted by *CLn* (where n is the number of jobs) including *CL*10, *CL*20, *CL*40, *CL*60, *CL*80, *CL*100 and *CL*150. The largest class was only used to compare the performance of the preprocessing techniques for NF2. Each class contained 9 subclasses and each subclass had 10 randomly generated instances. Subclasses were defined in such a way that they contained instances with different characteristics. Each subclass was characterized by parameters τ (average tardiness factor) and R (relative range factor) which controlled the due dates. Each of these parameters could take values from the set {0.3, 0.6, 0.9}. Processing times and delay penalties were generated from a discrete uniform distribution on the interval [1, 10]. Moreover, revenues were drawn from a log-normal distribution with mean 0 and standard deviation 1 (all numbers were rounded up to the smallest following integer). The due date for job $i \in N$ was generated randomly from a discrete uniform distribution on the interval

$$\left[\max\{P(1-\tau-R/2), p^1_i + p^2_i\}, \max\{P(1-\tau+R/2), p^1_i + p^2_i\}\right].$$

Table 6
The output of preprocessing methods PRE1 and PRE2 for the example.

Preprocessing		Position q					*EV (percent)*
		1	2	3	4	5	
PRE1	C^L_{1q}	7	10	17	24	**34**	44
	C^L_{2q}	14	17	23	**30**	**37**	
	C^L_{3q}	12	**15**	**21**	**28**	**38**	
	C^L_{4q}	10	16	**23**	**30**	**40**	
	C^L_{5q}	13	16	22	29	**39**	
	B_q	ϕ	{3}	{3, 4}	{2, 3, 4}	{1, 2, 3, 4, 5}	
PRE2	C^L_q	7	11	17	24	∞^*	40
	B_q	ϕ	ϕ	{3, 4}	{2, 3, 4}	{1, 2, 3, 4, 5}	

* No feasible solution found and the position is always empty.

where $P = \sum_{i=1}^{n} p_i^1 + p_{[1]}^2$. As is mentioned by Nobibon and Leus (2011) and Wang et al. (2013b), generating due dates from this interval guarantees that no jobs are consistently tardy.

To show the effectiveness of the proposed enhancements, we added them to the formulations in stages. We considered three different levels of improvements for NF1:

- $NF1^I$: The original formulation of NF1;
- $NF1^{II}$: NF1 when big-M coefficients were moderated;
- $NF1^{III}$: $NF1^{II}$ with all valid inequalities incorporated.

Six different levels of improvements were considered for NF2:

- $NF2^I$: The original formulation (NF2).
- $NF2^{II}$: NF2 with valid inequality (23).
- $NF2^{III}$: NF2 with valid inequalities (24) and (25).
- $NF2^{IV}$: $NF2^{III}$ when big-M coefficients were moderated.
- $NF2^V$: $NF2^{IV}$ with preprocessing and valid inequality (26).
- $NF2^{VI}$: $NF2^V$ with valid inequality (27).

To quantify the performance of each level of the improved formulation, we report the average computational time in seconds (*Avg.Time*). Note that we imposed a time limit of 1800 seconds for CPLEX. For cases in which some instances could not be solved to optimality within the time limit, the average optimality gap (*Avg.Gap*) is reported. Where suitable, the average objective value of LP-relaxation (*Avg.LP Obj*) and the average number of searched nodes (*Avg.#Node*) are also reported. To demonstrate the performance of the preprocessing techniques, we report the average percentage of eliminated x_{ik} variables (*Avg.EV*) as well as the average computational time in seconds.

5.1. Performance of the new formulations without enhancements

We now compare $NF1^I$ and $NF2^I$ with PF1 and PF2 to show that the new formulations perform better than the previous ones even without using any enhancements. To do the comparison, we only use class *CL*10. This is because PF1 and PF2 perform poorly on larger classes. For instance, we generated some instances with 15 jobs, but PF1 and PF2 were unable to solve most of them within the time limit. Table 7 reports the average solution times and LP-relaxation values of all formulations, to enable their comparison. Using the data in this table, we make the following comments and observations.

(1) Instances with $\tau = 0.3$ are harder to solve than others for PF1, PF2 and $NF1^I$. One reason for this is that when τ is smaller, fewer jobs are expected to be rejected.
(2) $NF1^I$ has the best LP-relaxation values. Evidently, the LP-relaxation value of $NF1^I$ gets better as τ increases. Note that PF2 is a time-indexed formulation. Among the different formulations of scheduling problems presented in the literature, time-indexed formulations are well known for their high quality LP-relaxation objective values (see, for instance, van den Akker, Hurkens, & Savelsbergh, 2000). However, PF2 is around 16.1 percent weaker than $NF1^I$ on average. This is mainly because PF2 contains more disjunctive constraints (see Section 3).
(3) There is a significant difference in solution times between the new and the previous formulations. Of course, this is not surprising since the number of constraints and binary variables associated with the new formulations is far smaller. Because $NF1^I$ is stronger than the other formulations, it could solve all instances more quickly, taking less than one second on average in this experiment. It can be seen that, on average, $NF1^I$ is around 7, 70 and 128 times faster than $NF2^I$, PF1, and PF2, respectively.

Note that we only tested the formulations with problem instances in which the processing times of the jobs were drawn from [1,10]. However, it is worth mentioning that by increasing the processing times, time-indexed formulation performance is expected to get even worse. The reason for this is that the numbers of variables and constraints in the time-indexed formulation are dependent on the processing times. However, this does not have a considerable effect on the performance of the other formulations. Note that because the due dates were generated in terms of processing times, the parameter characteristics did not change dramatically for the other formulations when the processing times increased. This phenomenon was also observed by Wang et al. (2013b). They mentioned that PF1 is robust with respect to the distribution range of the processing times, but that is not the case for PF2 (which is a time-indexed formulation), meaning that it can only solve instances with fewer jobs as processing times increase. They reported that when processing times are generated from the interval [1,100], PF2 can only solve problem instances with up to six jobs within one hour.

5.2. Effect of enhancements on NF1

We now compare $NF1^I$, $NF1^{II}$, and $NF1^{III}$ on class *CL*20 to show the importance of the proposed enhancements. The results are reported in Table 8. Based on the data in this table, we make some observations and comments.

(1) All formulations display similar behavior for different values of τ and R. In other words, instances with larger values of τ are more easily solved by NF1. Note that τ affects the mean of the due dates, but R mainly affects their variance. So, when τ increases, it is expected that more orders will be rejected. Consequently, the NF1 formulation shows better performance for those instances. Evidently, when $\tau = 0.3$, the instances increase in ease of solution as R increases. However, the opposite occurs when $\tau = 0.9$. Following this pattern, the instances with a medium value of R (e.g. $R = 0.6$) are easier to be solved when $\tau = 0.6$.

Table 7
Performance of the formulations on *CL*10.

τ	R	PF1		PF2		$NF1^I$		$NF2^I$	
		Avg. time (seconds)	Avg. LP Obj	Avg. time (seconds)	Avg. LP Obj	Avg. time (seconds)	Avg. LP Obj	Avg. time (seconds)	Avg. LP Obj
0.3	0.3	62.1	22.3	152.9	22.3	0.9	22.2	1.5	22.3
	0.6	22.3	21.1	57.9	21.1	0.4	21.0	1.6	21.1
	0.9	20.5	22.4	22.9	22.4	0.2	22.3	1.0	22.4
AVG		**35.0**	**21.9**	**77.9**	**21.9**	**0.5**	**21.8**	**1.4**	**21.9**
0.6	0.3	20.6	21.3	16.2	21.3	0.3	17.2	3.5	21.3
	0.6	16.7	26.5	26.7	26.5	0.3	23.9	2.3	26.5
	0.9	13.9	20.6	17.0	20.6	0.2	19.2	1.9	20.6
AVG		**17.1**	**22.8**	**20.0**	**22.8**	**0.3**	**20.1**	**2.6**	**22.8**
0.9	0.3	3.9	22.4	15.8	22.3	0.1	13.0	1.8	22.4
	0.6	10.9	20.9	13.4	20.9	0.2	14.7	2.6	20.9
	0.9	17.6	23.7	22.4	23.7	0.3	19.6	3.5	23.7
AVG		**10.8**	**22.3**	**17.2**	**22.3**	**0.2**	**15.8**	**2.6**	**22.3**
Total AVG		**20.9**	**22.4**	**38.4**	**22.3**	**0.3**	**19.2**	**2.2**	**22.4**

Table 8
Comparing variants of the first formulation on *CL*20.

τ	R	$NF1^I$		$NF1^{II}$		$NF1^{III}$	
		Avg. time (seconds)	Avg. gap (percent)	Avg. time (seconds)	Avg. gap (percent)	Avg. time (seconds)	Avg. gap (percent)
0.3	0.3	1800.0	9.735	1800.0	9.400	1707.7	8.850
	0.6	614.1	1.097	614.0	1.097	976.3	0.963
	0.9	248.0	0.313	247.6	0.313	65.6	0.000
AVG		**887.4**	**3.715**	**887.2**	**3.603**	**916.5**	**3.271**
0.6	0.3	430.6	0.256	430.6	0.256	198.9	0.000
	0.6	33.1	0.000	33.0	0.000	14.4	0.000
	0.9	272.9	0.256	273.0	0.256	90.0	0.000
AVG		**245.6**	**0.171**	**245.5**	**0.171**	**101.1**	**0.000**
0.9	0.3	1.2	0.000	1.2	0.000	1.3	0.000
	0.6	5.3	0.000	5.3	0.000	3.1	0.000
	0.9	63.0	0.000	63.0	0.000	56.5	0.000
AVG		**23.1**	**0.000**	**23.1**	**0.000**	**20.3**	**0.000**
Total AVG		**385.4**	**1.295**	**385.3**	**1.258**	**346.0**	**1.090**

(2) There is no significant difference between the runtimes of $NF1^I$ and $NF1^{II}$ on average. So, moderating big-M values is not very effective for NF1.
(3) $NF1^{III}$ outperforms $NF1^I$ and $NF1^{II}$ in terms of runtime and the average gap in the time limit. It should be mentioned that 15 out of the 90 instances could not be solved to optimality by $NF1^I$ and $NF1^{II}$. However, using valid inequalities reduced the number of unsolved problems from 15 to 11.

5.3. Preprocessing techniques

Before comparing different variants of NF2, we discuss the performance of preprocessing methods PRE1 and PRE2 on the *CL*20, *CL*80 and *CL*150 instances. The experimental results for these cases are reported in Table 9. From the table, we make some observations and comments.

(1) In general, for both techniques, the average number of eliminated variables is larger when τ is larger. That is because when τ increases, the time window of the due dates shifts forward. As a result, more orders will be rejected.
(2) In general, for both techniques, when $\tau = 0.3$, the average EV becomes larger as R increases. However, when $\tau = 0.9$, it is the other way around.
(3) The average computational time of PRE2 is smaller for larger τ-values. This is again a consequence of the fact that more orders will be rejected when τ increases. So, a smaller number of optimization problems will need to be solved for PRE2. Note that it is not surprising that the average computational time of PRE2 is increased for larger classes of instances because more optimization problems need to be solved.
(4) As the number of jobs increases, PRE2 performs better than PRE1 in terms of eliminated variables. In class *CL*20, the average EVs for both techniques is 47.4 percent. However, for class *CL*150, PRE2 is around 4 percent better than PRE1 on average. Of course this is costly in terms of efficiency and, on average, we need to spend around 270 seconds of valuable computational time eliminating variables.

For the remainder of this paper, we only use PRE2 in our computational experiments because it eliminates more variables on average. However, PRE1 is faster, so, it may be useful in heuristic or metaheuristic approaches.

5.4. Effect of enhancements on NF2

We first compare $NF2^I$, $NF2^{II}$, $NF2^{III}$ and $NF2^{IV}$ on *CL*20 instances. The results are given in Table 10. We do not show the values of the average optimality gap for $NF2^{II}$, $NF2^{III}$ and $NF2^{IV}$ since they could solve all the instances to optimality. Based on the data in the table, we make the following observations and comments.

(1) The average optimality gap and runtime of $NF2^I$ increase as parameter τ increases. This is completely the opposite of what we observed for NF1. So, unlike the NF1, larger τ-values make problem instances harder to solve using NF2.

Table 9
Performance of preprocessing techniques.

τ	R	*CL*20			*CL*80			*CL*150		
		PRE1	PRE2		PRE1	PRE2		PRE1	PRE2	
		Avg. EV (percent)	Avg. time (seconds)	Avg. EV (percent)	Avg. EV (percent)	Avg. time (seconds)	Avg. EV (percent)	Avg. EV (percent)	Avg. time (seconds)	Avg. EV (percent)
0.3	0.3	26.7	0.6	26.3	20.3	15.9	21.1	19.3	409.9	20.9
	0.6	24.8	0.5	24.3	20.1	21.8	21.1	19.9	560.1	21.7
	0.9	31.0	0.5	30.6	22.7	21.0	24.1	21.8	587.9	23.8
AVG		**27.5**	**0.5**	**27.0**	**21.0**	**19.6**	**22.1**	**20.3**	**519.3**	**22.1**
0.6	0.3	48.9	0.5	50.6	42.4	5.1	46.3	41.7	128.3	47.2
	0.6	52.0	0.6	53.1	42.8	7.9	45.8	42.3	228.1	46.1
	0.9	43.2	0.7	42.5	41.2	11.7	43.9	41.2	349.2	44.9
AVG		**48.0**	**0.6**	**48.7**	**42.1**	**8.2**	**45.3**	**41.7**	**235.2**	**46.0**
0.9	0.3	76.3	0.3	74.8	71.8	0.8	75.4	71.3	13.5	76.3
	0.6	66.5	0.5	66.0	63.1	2.4	67.3	61.2	47.8	66.9
	0.9	57.7	0.5	58.2	54.4	4.9	58.2	54.3	108.3	59.4
AVG		**66.8**	**0.4**	**66.3**	**63.1**	**2.7**	**67.0**	**62.3**	**56.5**	**67.5**
Total AVG		**47.4**	**0.5**	**47.4**	**42.1**	**10.2**	**44.8**	**41.4**	**270.4**	**45.2**

Table 10
Comparing four different variants of NF2 on *CL*20.

τ	R	$NF2^{I}$			$NF2^{II}$		$NF2^{III}$		$NF2^{IV}$	
		Avg. time (seconds)	Avg. gap (percent)	Avg. node No	Avg. time (seconds)	Avg. node No	Avg. time (seconds)	Avg. node No	Avg. time (seconds)	Avg. node No
0.3	0.3	27.0	0.000	16103.0	7.2	4250.5	4.7	2357.7	3.4	737.6
	0.6	37.8	0.000	14366.9	7.0	3514.5	5.2	2565.9	3.0	846.7
	0.9	479.0	0.667	140209.6	44.5	16742.6	23.3	18098.3	33.3	18486.4
AVG		**181.3**	**0.222**	**56893.2**	**19.6**	**8169.2**	**11.1**	**7674.0**	**13.2**	**6690.2**
0.6	0.3	1518.9	7.451	952840.3	112.8	38247.0	34.5	38179.5	6.2	1716.5
	0.6	1353.1	7.845	624096.7	80.3	27009.6	33.4	35802.8	14.9	7306.8
	0.9	660.9	1.485	141140.6	43.5	26779.9	25.2	26935.1	12.7	5451.8
AVG		**1177.6**	**5.594**	**572692.5**	**78.9**	**30678.8**	**31.1**	**33639.1**	**11.3**	**4825.0**
0.9	0.3	1498.3	44.788	1423641.0	56.6	24873.2	46.0	51191.7	1.8	722.3
	0.6	1707.9	30.565	1210049.0	68.1	21304.3	57.4	73761.6	5.4	1539.6
	0.9	1479.3	8.008	814708.2	82.5	22825.6	47.9	50420.5	7.7	1008.6
AVG		**1561.9**	**27.787**	**1149466.1**	**69.1**	**23001.0**	**50.4**	**58457.9**	**4.9**	**1090.2**
Total AVG		**973.6**	**11.201**	**593017.3**	**55.8**	**20616.4**	**30.8**	**33257.0**	**9.8**	**4201.8**

(2) The main issue with $NF2^{I}$ is symmetry. Consequently, many nodes in the B&B tree needed to be investigated by CPLEX. The result was that 38 out of 90 instances could not be solved to optimality by $NF2^{I}$ within the time limit. However, after breaking symmetry in $NF2^{II}$ by adding constraint (23) to NF2, the number of investigated nodes was decreased by about 96.5 percent. This resulted in a significant improvement in the runtime so that we were able to solve all instances to optimality in less than one minute, on average.

(3) The number of nodes which are solved by $NF2^{III}$ is around 60 percent more than the number solved by $NF2^{II}$. However, its runtime is around half of the runtime of $NF2^{I}$ on average. As mentioned previously, when we include constraints (24) and (25) in place of constraint (23), constraint (11) should be removed from the NF2. Moreover, constraint (25) have much simpler structure in comparison with constraint (23). Consequently, the number nonzero elements in the matrix of coefficients decreases sharply. So, it is not surprising that, on average, $NF2^{II}$ explores 369 nodes per second, but $NF2^{III}$ explores 1080 nodes per second;

(4) By contrast to our observation for *NF*1, moderating big-M values has a significant impact on *NF*2. The number of explored nodes and runtime are decreased by about 87.4 percent and 68.2 percent, on average, respectively, when $NF2^{IV}$ is implemented rather than $NF2^{III}$.

Before showing that NF2 can be improved even further, we now compare NF1 and NF2. From the data in Tables 8 and 10, it is evident that if we compare the basic models of NF1 and NF2, i.e., $NF1^{I}$ and $NF2^{I}$, then NF1 outperforms NF2. On average, $NF1^{I}$ solves all instances in around 385.4 seconds (with an average gap of 1.295 percent), but $NF2^{I}$ solves them in about 973.6 seconds (with an average gap of 11.201 percent). However, the situation is completely different for the improved version of these formulations, i.e., $NF1^{III}$ and $NF2^{IV}$. On average, $NF1^{III}$ solves all instances in about 346.0 seconds (with an average gap of 1.090 percent), but $NF2^{IV}$ solves them in around 9.8 seconds (with an average gap of 0.000 percent). As a result, the difference between $NF2^{IV}$ and $NF1^{III}$ is significant enough to say that the improved NF2 is a better formulation.

Next we show that NF2 can be improved even further by using preprocessing and adding valid bounding inequalities. Table 11 shows the data for the implementation of $NF2^{IV}$, $NF2^{V}$ and $NF2^{VI}$ on *CL*40. $NF2^{IV}$ could solve all instances of class *CL*20 in less than 10 seconds on average. However, when we doubled the number of jobs, its runtime increased by a factor of more than 62, on average, and it could not solve all instances to optimality within the time limit. Results for $NF2^{V}$ showed that preprocessing can reduce the runtime by a factor of 6 on average. It can also decrease the average optimality gap from 0.949 percent to 0.099 percent.

Note that, as we discussed previously, preprocessing of commercial solvers can eliminate some of the variables if we add valid

Table 11
Comparing $NF2^{IV}$, $NF2^{V}$ and $NF2^{VI}$ on $CL40$.

τ	R	$NF2^{IV}$		$NF2^{V}$		$NF2^{VI}$	
		Avg. time (seconds)	Avg. gap (percent)	Avg. time (seconds)	Avg. gap (percent)	Avg. time (seconds)	Avg. gap (percent)
0.3	0.3	24.5	0.000	5.4	0.000	2.0	0.000
	0.6	240.5	0.163	11.8	0.000	8.7	0.000
	0.9	189.4	0.000	16.1	0.000	18.3	0.000
AVG		**151.4**	**0.054**	**11.1**	**0.000**	**9.7**	**0.000**
0.6	0.3	159.9	0.000	15.3	0.000	7.5	0.000
	0.6	1585.0	1.699	264.3	0.000	109.7	0.000
	0.9	1231.5	3.150	449.3	0.889	446.9	0.508
AVG		**992.1**	**1.616**	**243.0**	**0.296**	**188.0**	**0.169**
0.9	0.3	108.9	0.000	7.8	0.000	4.1	0.000
	0.6	985.2	0.874	31.7	0.000	19.2	0.000
	0.9	1137.2	2.659	189.7	0.000	108.5	0.000
AVG		**743.7**	**1.178**	**76.4**	**0.000**	**43.9**	**0.000**
Total AVG		**629.1**	**0.949**	**110.2**	**0.099**	**80.5**	**0.056**

Table 12
Performance of $NF2^{VI}$.

τ	R	10	20	30	60		80		100	
		Avg. time (seconds)	Avg. time (seconds)	Avg. time (seconds)	Avg. time (seconds)	Avg. gap (percent)	Avg. time (seconds)	Avg. gap (percent)	Avg. time (seconds)	Avg. gap (percent)
0.3	0.3	0.1	0.4	2.2	13.5	0.000	76.9	0.000	432.2	0.044
	0.6	0.2	0.6	2.4	217.3	0.082	423.6	0.000	1479.0	0.431
	0.9	0.2	3.9	4.1	338.0	0.066	939.8	0.360	1714.7	0.891
AVG		**0.2**	**1.6**	**2.9**	**189.6**	**0.049**	**480.1**	**0.120**	**1208.6**	**0.455**
0.6	0.3	0.1	1.4	5.3	166.3	0.000	278.3	0.000	1205.3	0.271
	0.6	0.1	2.0	13.7	926.8	0.419	1630.0	0.964	1632.4	1.466
	0.9	0.2	2.1	62.4	1475.3	2.166	1800.0	2.344	1800.0	5.145
AVG		**0.1**	**1.8**	**27.1**	**856.1**	**0.862**	**1236.1**	**1.103**	**1545.9**	**2.294**
0.9	0.3	0.1	0.3	1.0	430.0	0.273	1050.7	0.864	1626.7	2.916
	0.6	0.1	0.6	3.4	824.2	1.597	1800.0	4.082	1676.6	2.854
	0.9	0.1	0.9	7.3	1119.1	0.572	1712.1	2.664	1800.0	3.717
AVG		**0.1**	**0.6**	**3.9**	**791.1**	**0.814**	**1520.9**	**2.536**	**1701.1**	**3.163**
Total AVG		**0.1**	**1.3**	**11.3**	**612.3**	**0.575**	**1079.1**	**1.253**	**1485.2**	**1.971**

inequality (26). Therefore, the role of these valid inequalities is the same as for PRE2, meaning that they are developed for eliminating variables. That is why we add PRE2 and valid inequality (26) at the same time to $NF2^{IV}$ to obtain $NF2^{V}$ for the model. The results can be improved even further by using $NF2^{VI}$. Evidently, when compared with $NF2^{V}$, the runtime is improved by around 26.9 percent and the optimality gap is decreased from 0.099 percent to 0.056 percent, on average, by using $NF2^{VI}$.

5.5. Overall performance of the best formulation

In Table 12, we report the results of the implementation of the best formulation , i.e., $NF2^{VI}$, on different classes of instances. It can be seen that instances of small size are solved easily without any difficulty, and larger instances are solved with a reasonable optimality gap within a half-hour time limit by using this model. On average, instances of class $CL100$ are solved with a 1.971 percent optimality gap within the time limit. As is evident from the data in the table, the subclass with $\tau = 0.6$ and $R = 0.9$ contains the most difficult instances which have been solved. When $\tau = 0.6$ and $R = 0.9$, none of the instances of classes $CL80$ and $CL100$ could be solved to optimality within the time limit. However, the average optimality gap never exceeded 5.2 percent even in these instances.

It is also interesting to compare the results of $NF2^{VI}$ and the B&B proposed by Wang et al. (2013b). Wang et al. (2013b) showed that their proposed B&B algorithm can solve instances with 20 jobs in 253.5 seconds, on average. However, CPLEX can use $NF2^{VI}$ to solve such instances easily in no more than 1.3 seconds with a computer that has similar characteristics to the one used by Wang et al. (2013b). As a result, we can almost improve the runtime by a factor of 200 for instances with n=20.

6. Conclusion

In this study, we developed two different mathematical formulations for the order acceptance and scheduling problem in two-machine flow shops. The complexity sizes of the new formulations are significantly smaller than previous models proposed by Wang et al. (2013b). We developed several techniques such as preprocessing and valid inequalities to improve each of these formulations.

It was shown that the new formulations perform much better than the previous ones even without using any enhancement techniques. Comparing the basic versions of our proposed formulations with each other shows that the first formulation is better than the second one. However, after the addition of various enhancements, the second formulation performs far better than the first one. Wang et al. (2013b) were able to solve instances up to 20 jobs with their purpose-built algorithm within an hour. However, we were able to solve instances up to five times larger than their investigated instances on a similar computer within half an hour using CPLEX and the improved version of our second formulation.

We hope that the simplicity and performance of our new mathematical formulations encourage (more) researchers to consider developing new MILP formulations or enhancement techniques for

the order acceptance and scheduling problem (in two-machine flow shops). It is worth mentioning that in this study, we only tried to improve our new MILP formulations. However, in regard to future research, it may be interesting to explore the MILP formulations by Wang et al. (2013b) (especially PF2 since it is a time-indexed formulation), and trying to improve them by reformulating the constraints or developing big-M moderation, valid inequalities, and preprocessing techniques for them.

Appendix

Previous Formulation 1 (PF1):

Let $N := \{1, 2, \ldots, n\}$ be the set of jobs, and $Q := \{1, 2, \ldots, n\}$ be the set of positions (in the sequencing list) to which jobs can be assigned. To describe PF1, Wang et al. (2013b) define 5 sets of decision variables:

- For each job $i \in N$, they introduce a binary decision variable to indicate whether the job is accepted or rejected,

$$y_i := \begin{cases} 1 & \text{If job } i \text{ is accepted,} \\ 0 & \text{Otherwise;} \end{cases}$$

- For each job $i \in N$ and for each position $k \in Q$, they define a binary decision variable x_{ik} to indicate whether the job is assigned to that position, that is,

$$x_{ik} := \begin{cases} 1 & \text{If job } i \text{ is accepted and assigned to position } k, \\ 0 & \text{Otherwise;} \end{cases}$$

- For each pair of jobs $(i, j) \in N^2$ with $i \neq j$, they introduce a binary decision variable to indicate which one is processed immediately after the other,

$$z'_{ij} := \begin{cases} 1 & \text{If jobs } i \text{ and } j \text{ are accepted, and job } j \text{ is processed} \\ & \quad \textit{immediately} \text{ after job } i, \\ 0 & \text{Otherwise;} \end{cases}$$

- For each job $i \in N$, they introduce a continuous decision variable C_i to indicate its completion time.
- For each job $i \in N$, they introduce a continuous decision variable T_i to indicate its tardiness.

Using these decision variables, PF1 can be expressed as follows:

$$\max \sum_{i \in N} (u_i \cdot y_i - w_i \cdot T_i) \tag{A.1}$$

subject to:

$$\sum_{k \in Q} x_{ik} = y_i \quad \forall i \in N \tag{A.2}$$

$$\sum_{i \in N} x_{ik} \leq 1 \quad \forall k \in Q \tag{A.3}$$

$$z'_{ij} \leq y_i,\ z'_{ij} \leq y_j \quad \forall i, j \in N \text{ with } i \neq j \tag{A.4}$$

$$x_{ik} + x_{j,k+1} \leq z'_{ij} + 1 \quad \forall i, j \in N, i \neq j, \text{ and } k \in Q \backslash \{n\} \tag{A.5}$$

$$C_i + p_j^2 \cdot y_j + (z'_{ij} - 1) \cdot M \leq C_j \quad \forall i, j \in N \text{ with } i \neq j \tag{A.6}$$

$$(p_j^1 + p_j^2) \cdot x_{j1} \leq C_j \quad \forall j \in N \tag{A.7}$$

$$\begin{aligned} &\textstyle\sum_{\substack{k' \in Q, \\ k' \leq k}} \sum_{i \in N} p_i^1 \cdot x_{ik'} + p_j^1 \cdot x_{j,k+1} + p_j^2 \cdot y_j \\ &\quad + (y_j - 1) \cdot M \leq C_j + M \cdot \textstyle\sum_{\substack{k' \in Q, \\ k' \leq k}} x_{jk'} \end{aligned} \quad \forall j \in N, \text{ and } k \in Q \backslash \{n\} \tag{A.8}$$

$$T_i \geq C_i - d_i \quad \forall i \in N \tag{A.9}$$

$$\sum_{i \in N} x_{i,k+1} \leq \sum_{i \in N} x_{ik} \quad \forall k \in Q \backslash \{n\} \tag{A.10}$$

$$y_i \in \{0, 1\} \quad \forall i \in N \tag{A.11}$$

$$z'_{ij} \in \{0, 1\} \quad \forall i, j \in N \text{ with } i \neq j \tag{A.12}$$

$$x_{ik} \in \{0, 1\} \quad \forall i \in N \text{ and } k \in Q \tag{A.13}$$

$$C_i, T_i \geq 0 \quad \forall i \in N. \tag{A.14}$$

In PF1, the objective function maximizes the total net revenue. Constraint (A.2) ensures that each accepted job is allocated to exactly one position (evidently, the rejected jobs will not be allocated to any position). Constraint (A.3) implies that a given position can accommodate at most one job. Constraint (A.4) states that both jobs i and j must be accepted when $z'_{ij} = 1$. Constraint (A.5) states that if jobs i and j are accepted and job j is immediately processed after job i, then $z'_{ij} = 1$. Constraint (A.6) implies that if job j is preceded by job i, then the completion time of job j should be no earlier than the sum of the completion time of job i and the processing time of job j on machine 2. Constraint (A.7) dictates that if job j is accepted and processed in the first position of a schedule, then the completion time of job j is no earlier than the sum of its processing times on machines 1 and 2. Constraint (A.8) dictates that if job j is accepted and processed in the kth position (where $2 \leq k \leq n$) of a schedule, then the completion time of job j is no earlier than the sum of the processing times of the jobs processed before job j on machine 1 and the processing times of job j on machines 1 and 2. Constraint (A.9) captures the tardiness of each job $i \in N$. Inequality (A.10) removes any empty positions that appear between two consecutive jobs. Constraints (A.11)–(A.14) specify the domains of each decision variable. In (A.6) and (A.8), M is a large positive number, e.g., $M = \sum_{i \in N} (p_i^1 + p_i^2)$.

Previous Formulation 2 (PF2)

PF2 is a time-indexed formulation and to describe it, Wang et al. (2013b) define three sets of variables. The first two sets include only continuous decision variables to indicate the completion time and tardiness of each job, i.e., $\{C_1, \ldots, C_n\}$ and $\{T_1, \ldots, T_n\}$. The last set includes (binary) time-indexed variables:

$$x'_{it} := \begin{cases} 1 & \text{If job } i \in N \text{ is accepted and begin to be processed} \\ & \quad \text{at time } t \in \mathbb{T} \text{ on machine 2,} \\ 0 & \text{Otherwise;} \end{cases}$$

where $\mathbb{T} := \{p_i^1, \ldots, \Gamma - p_i^2\}$, $\Gamma := \sum_{i \in \tilde{N}} p_i^1 + \sum_{i \in N \backslash \tilde{N}} p_i^2 + L$, $\tilde{N} := \{i \in N : p_i^1 \geq p_i^2\}$, and L is the length of the interval from which the processing times of the problem instance are randomly generated.

Using these decision variables, PF2 can be expressed as follows:

$$\max \sum_{i \in N} \left[\left(\sum_{t \in \mathbb{T}} x'_{it} \right) \cdot u_i - w_i \cdot T_i \right] \tag{A.15}$$

subject to:

$$\sum_{t \in \mathbb{T}} x'_{it} \leq 1 \quad \forall i \in N \tag{A.16}$$

$$(x'_{it} - 1) \cdot M + \sum_{t'=t+1}^{t+p_i^2-1} x'_{it'} + \sum_{j \neq i} \sum_{t'=t}^{t+p_i^2-1} x'_{jt'} \leq 0 \quad \forall i \in N \text{ and } \forall t \in \mathbb{T} \tag{A.17}$$

$$(x'_{it} - 1) \cdot M + \sum_{j \in N} \sum_{t'=p_j^1}^{t} x'_{jt'} p_j^1 \leq t \quad \forall i \in N \text{ and } \forall t \in \mathbb{T} \tag{A.18}$$

$$C_i \geq \sum_{t'=t}^{\Gamma - p_i^2} x'_{it'} \cdot (t' + p_i^2) \quad \forall i \in N \text{ and } \forall t \in \mathbb{T} \tag{A.19}$$

$$T_i \geq C_i - d_i \quad \forall i \in N \tag{A.20}$$

$$\left(\sum_{p_j^1 \leq t' \leq t} x'_{jt'} - 1 \right) \cdot M + C_j + x'_{it} \cdot p_i^2 \leq (1 - x'_{it}) \cdot M + C_i \quad \forall i, j \in N, i \neq j, \text{ and } \forall t \in \mathbb{T} \tag{A.21}$$

$$x'_{it} \in \{0, 1\} \quad \forall i \in N \text{ and } \forall t \in \mathbb{T} \tag{A.22}$$

$$C_i, T_i \geq 0 \quad \forall i \in N. \tag{A.23}$$

In PF2, the objective function maximizes the total net revenue. Constraint (A.16) states that an accepted job is processed exactly once on machine 2. Constrain (A.17) implies that if processing of job i begins at time t on machine 2 then the other jobs cannot be processed within the time window $[t, t + p_i^2)$ on machine 2 . Constraint (A.18) states that processing of job i on machine 2 can begin at time t only if processing of all the preceding jobs on machine 1 finish before time t. Constraint (A.19) implies that the completion time of job i is no earlier than $t' + p_i^2$ (where $(t \leq t' \leq \Gamma - p_i^2)$) if it begins to be processed at time t' on machine 2. Constraint (A.20) captures the tardiness of each job $i \in N$. Inequality (A.21) implies that if job i is accepted and begins to be processed at time t, then the completion time of job i is no earlier than the sum of its processing time on machine 2 and the completion time of its preceding job j on machine 2. Constraints (A.22)–(A.23) specify the domains of each decision variable. In (A.17), (A.18), and (A.21), M is a large positive number, e.g., $M = \sum_{i \in N}(p_i^1 + p_i^2)$.

References

Atamtürk, A., & Savelsbergh, M. (2005). Integer-programming software systems. *Annals of Operations Research, 140*(1), 67–124.

Avella, P., Boccia, M., & Vasilyev, I. (2012). Computational testing of a separation procedure for the knapsack set with a single continuous variable. *INFORMS Journal on Computing, 24*(1), 165–171.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. John Wiley & Sons.

Cesaret, B., Oğuz, C., & Salman, F. S. (2012). A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research, 39*(6), 1197–1205.

Chudak, F. A., & Hochbaum, D. S. (1999). A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters, 25*(5), 199–204.

Dyer, M. E., & Wolsey, L. A. (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics, 26*(23), 255–270.

Ghosh, J. B. (1997). Job selection in a heavily loaded shop. *Computers & Operations Research, 24*(2), 141–145.

Kaparis, K., & Letchford, A. (2010). Separation algorithms for 0–1 knapsack polytopes. *Mathematical Programming, 124*(1–2), 69–91.

Kim, Y.-D. (1993). A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. *Computers & Operations Research, 20*(4), 391–401. http://dx.doi.org/10.1016/0305-0548(93)90083-U.

Lei, D., & Guo, X. (2015). A parallel neighborhood search for order acceptance and scheduling in flow shop environment. *International Journal of Production Economics, 165*(0), 12–18. http://dx.doi.org/10.1016/j.ijpe.2015.03.013.

Lin, S.-W., & Ying, K.-C. (2013). Increasing the total net revenue for single machine order acceptance and scheduling problems using an artificial bee colony algorithm. *Journal of Operational Research Society, 64*(2), 293–311.

Lin, S.-W., & Ying, K.-C. (2015). Order acceptance and scheduling to maximize total net revenue in permutation flowshops with weighted tardiness. *Applied Soft Computing, 30*(0), 462–474. http://dx.doi.org/10.1016/j.asoc.2015.01.069.

Margot, F. (2010). Symmetry in integer linear programming. In M. Jnger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, & L. A. Wolsey (Eds.), *50 years of integer programming 1958–2008* (pp. 647–686). Springer Berlin Heidelberg. doi:10.1007/978-3-540-68279-0_17.

Nemhauser, G., & Savelsbergh, M. (1992). A cutting plane algorithm for the single machine scheduling problem with release times. In M. Akgül, H. Hamacher, & S. Tüfeki (Eds.), *Combinatorial optimization: vol. 82* (pp. 63–83). Springer Berlin Heidelberg. NATO ASI Series.

Nobibon, F. T., & Leus, R. (2011). Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research, 38*(1), 367–378.

Oğuz, C., Salman, F. S., & Yalçın, Z. B. (2010). Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics, 125*(1), 200–211.

Reisi-Nafchi, M., & Moslehi, G. (2015). A hybrid genetic and linear programming algorithm for two-agent order acceptance and scheduling problem. *Applied Soft Computing, 33*(0), 37–47. http://dx.doi.org/10.1016/j.asoc.2015.04.027.

Rom, W. O., & Slotnick, S. A. (2009). Order acceptance using genetic algorithms. *Computers & Operations Research, 36*(6), 1758–1767.

Sherali, H. D., & Smith, J. C. (2001). Improving discrete model representations via symmetry considerations. *Management Science, 47*(10), 1396–1407. doi:10.1287/mnsc.47.10.1396.10265.

Slotnick, S. A. (2011). Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research, 212*(1), 1–11.

Slotnick, S. A., & Morton, T. E. (1996). Selecting jobs for a heavily loaded shop with lateness penalties. *Computers & Operations Research, 23*(2), 131–140.

Slotnick, S. A., & Morton, T. E. (2007). Order acceptance with weighted tardiness. *Computers & Operations Research, 34*(10), 3029–3042.

Unlu, Y., & Mason, S. J. (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering, 58*(4), 785–800.

van den Akker, J., Hurkens, C., & Savelsbergh, M. (2000). Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing, 12*(2), 111–124.

Wang, X., Xie, X., & Cheng, T. (2013a). A modified artificial bee colony algorithm for order acceptance in two-machine flow shops. *International Journal of Production Economics, 141*(1), 14–23.

Wang, X., Xie, X., & Cheng, T. (2013b). Order acceptance and scheduling in a two-machine flowshop. *International Journal of Production Economics, 141*(1), 366–376.

Xiao, Y.-Y., Zhang, R.-Q., Zhao, Q.-H., & Kaku, I. (2012). Permutation flow shop scheduling with order acceptance and weighted tardiness. *Applied Mathematics and Computation, 218*(15), 7911–7926. http://dx.doi.org/10.1016/j.amc.2012.01.073.