



# An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times

José Elias C. Arroyo<sup>a,\*</sup>, Joseph Y.-T. Leung<sup>b</sup>, Ricardo Gonçalves Tavares<sup>a</sup>

<sup>a</sup> Department of Computer Science, Universidade Federal de Viçosa, MG 36570-900, Brazil

<sup>b</sup> Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA



## ARTICLE INFO

### Keywords:

Unrelated parallel batch machines  
Total flow time  
Scheduling  
Iterated greedy  
Local search heuristics  
Meta-heuristics

## ABSTRACT

This paper investigates the problem of scheduling a set of jobs with arbitrary sizes and non-zero release times on a set of unrelated parallel batch machines with different capacities so as to minimize the total flow time of the jobs. The total flow time, defined as the total amount of time that the jobs spend in the system (i.e. the period between the job release dates and its completion times), is one of the most important objectives in scheduling problems, since it can lead to stable utilization of resources and reduction of working-in-process inventory. Motivated by the computational complexity of the problem, a simple and effective iterated greedy (IG) algorithm is proposed to solve it. The IG algorithm uses an efficient greedy heuristic to reconstruct solutions and a local search procedure to further enhance the solution quality. In attempting to obtain optimal solutions for small-medium size instances, a mixed integer programming model for the problem is also presented. The performance of the proposed algorithm is tested on a comprehensive set of small, medium and large benchmark of randomly generated instances, and is compared to three benchmark meta-heuristic algorithms (Discrete Differential Evolution, Ant Colony Optimization and Simulated Annealing) recently proposed for similar parallel batch machine scheduling problems. Experimental results and statistical tests show that the proposed algorithm is significantly superior in performance than the other algorithms.

## 1. Introduction

In classical scheduling problems, generally, the machines process at most one job at a time. However, scheduling on batch processing machines (BPMs), jobs are grouped into batches which are then scheduled on the BPMs. That is, the BPMs allow several jobs to be processed simultaneously in a batch. This type of problem belongs to the parallel-batch (p-batch) scheduling, meaning that the jobs in a batch are processed in parallel (Mönch et al., 2011). In practice, BPMs scheduling problems are encountered in many industries and environments such as semiconductor manufacturing, wafer fabrication, testing of electrical circuits, ship scheduling at navigation locks, steel-making plants, etc (Mönch et al., 2011; Lee et al., 1992; Tang et al., 2014).

In this paper, we consider the problem of scheduling unrelated parallel BPMs. There is a set of  $m$  unrelated parallel BPMs with different capacities. The jobs to be processed have different sizes and non-zero job release times (or arrival times). The release time is the time the job arrives at the system, i.e., the earliest time at which the job can start its processing. Each machine processes each job at a different speed. The jobs are processed without interruption or preemption, that is, once processing starts, no job can be removed from or added into the batch.

The batch release time is equal to the largest release time among all the jobs in the batch and each BPM can process a batch as long as the total size of the jobs in the batch does not exceed the machine capacity. The processing time of a batch is equal to the largest processing time of the jobs in the batch and all jobs in the same batch have the same completion time. The final purpose of the scheduling problem is to obtain a feasible batch schedule for each machine in order to minimize the total flow time, where the flow time of a job is the period between its release time and its completion time.

In real-world manufacturing environments, unrelated parallel machine scheduling is important because most workstations have multiple machines with different speeds for different jobs. Unrelated parallel machines can be considered a generalization of the identical and uniform parallel processing machines. To the best of our knowledge, the problem of scheduling unrelated parallel BPMs with total flow time objective has not been addressed so far. In the parallel batch machine scheduling literature, the makespan is the most commonly used performance measure and is related to maximizing machine or facility utilization. In many industrial processes, the total flow time is also an important objective. This objective measures the quality of service provided to customer or users, and its minimization is equivalent to

\* Corresponding author.

E-mail addresses: [jarroyo@dpi.ufv.br](mailto:jarroyo@dpi.ufv.br) (J.E.C. Arroyo), [leung@njit.edu](mailto:leung@njit.edu) (J.Y.-T. Leung), [ricardo.tavares@ufv.br](mailto:ricardo.tavares@ufv.br) (R.G. Tavares).

minimizing the total job lateness, total waiting time and work-in-process inventory (Chu, 1992). Since the classical single machine scheduling problem to minimize total flow time with different release times is known to be strongly NP-hard (Chu, 1992), the target problem can be also considered strongly NP-hard.

P-batch machine scheduling generally consists of two types of decisions: group the jobs into batches (batch forming), and schedule the batches on the BPMs. Grouping jobs into batches is equivalent to the bin-packing problem which is NP-hard (Garey and Johnson, 1979). For the makespan minimization, the rules First-Fit (FF) and Best-Fit (BF) are generally combined with the Longest Processing Time (LPT) rule (Damodaran and Chang, 2008). The rules FF and BF, initially developed for the bin-packing problem, are used to group the jobs into batches. In the FF rule, the jobs in the sequence are put, one by one, into the first batch with enough space to accommodate it, while in the BF rule, the jobs are put, one by one, into the feasible batch with the smallest residual capacity.

In the literature, there are very few papers that deal with unrelated parallel BPMs and non-identical job sizes. Some studies consider the makespan as objective function. Li et al. (2013b) assumed that the machine capacities are identical and the jobs have zero release times. Arroyo and Leung (2017b) considered identical machine capacities and non-zero job release times. In both papers Li et al. (2013b) and Arroyo and Leung (2017b), deterministic heuristics are presented to minimize the makespan. Arroyo and Leung (2017a) consider machines with different capacities, unequal job release times, and propose an Iterated Greedy (IG) algorithm for makespan minimization.

In this paper we propose the application of the IG algorithm for total flow time minimization in unrelated parallel BPMs with different capacities and unequal job release times. The IG algorithm works by iteratively applying two phases: destruction and construction. In the destruction phase, some jobs are removed from the current schedule, and in the construction phase the removed jobs are reinserted into the schedule using a greedy construction heuristic. To these two phases, an additional local search procedure is incorporated to further enhance the solution quality.

We propose an adaptation of the greedy deterministic heuristic, presented in Arroyo and Leung (2017b), to construct a high quality initial solution for the total flow time minimization problem. Besides that, we propose a strategy to improve the local search procedure. In order to emphasize intensification in the IG algorithm, this strategy considers a maximum distance for the jobs exchange and unnecessary movements are avoided in the search for neighbor solutions.

Since in the literature we have not been able to find any paper dealing with our problem, we cannot compare our IG algorithm directly with any existing algorithm. In order to do the comparison, we revise the meta-heuristic algorithms recently proposed by Zhou et al. (2016), Jia et al. (2017), and Damodaran and Vélez-Gallego (2012), which deals with parallel BPMs scheduling problems similar to ours.

The rest of the paper is organized as follows: In the next section we review the literature on the parallel BPMs scheduling problems. In Section 3, we describe the problem under study and present a mixed integer programming model. In Section 4, we introduce the IG algorithm and describe each of its phases. In Section 5, we describe the implementation of three benchmark meta-heuristic algorithms reported in the literature for similar problems. The computational results of applying the IG algorithm to the problem instances are provided in Section 6, as well as comparisons of the performance against the three benchmark algorithms from the literature. Finally, in Section 7, we conclude this paper and give future directions.

## 2. Literature review

The p-batch scheduling problem was first introduced by Ikura and Gimple (1986). They considered the makespan minimization and presented efficient algorithms to schedule single BPM under the assumption

that job release times and due dates are agreeable. Other pioneer works in this area were presented by Lee et al. (1992) and Uzsoy (1994). Lee et al. (1992) studied the problem of scheduling semiconductor burning operations and presented dynamic programming-based algorithms for minimizing the makespan and the maximum lateness on a single BPM. They also presented heuristics algorithms for a number of problems concerning parallel identical BPMs and provided worst case error bounds. Uzsoy (1994) addressed the problem of scheduling jobs with different sizes on a single BPM to minimize total completion time and makespan. He proved that the problems are NP-hard and several heuristics algorithms were developed, as well as a branch and bound algorithm for the total completion time problem.

Several researchers have been trying to extend the single BPM scheduling problem to parallel BPM environment (Potts and Kovalyov, 2000; Mathirajan and Sivakumar, 2006a; Mönch et al., 2011). In this review we only focus on previous works related to parallel BPMs scheduling problems, especially those of arbitrary job sizes, non-zero job release times and machines with different capacities.

The following authors addressed parallel-identical BPMs with identical capacities, different job sizes and zero job release times. For the problem of minimizing the makespan, Chang et al. (2004) developed a Simulated Annealing (SA) heuristic; Kashan et al. (2008) present a lower bound and proposed a hybrid Genetic Algorithm (GA); Damodaran and Chang (2008) proposed several heuristics based on the rules LPT, FF and BF; Cheng et al. (2013) provided a MIP model and proposed an Ant Colony Optimization (ACO) algorithm; and Jia and Leung (2015) developed a Max-Min Ant System (MMAS) heuristic. For minimizing the makespan and total completion time, Cheng et al. (2012) proposed MIP models, developed a polynomial time algorithm for minimizing the makespan and analyzed the worst case ratios for minimizing total completion time. Koh et al. (2004) and Jia et al. (2016) considered parallel-identical BPMs with incompatible job families. Koh et al. (2004) proposed different heuristics and designed a random key-based GA (RKGA) for the problems of minimizing the makespan and total weighted completion time. Jia et al. (2016) presented a meta-heuristic based on MMAS to minimize the makespan.

Other researchers addressed scheduling problems on parallel-identical BPMs with non-zero job release times (dynamic job arrivals), but jobs with identical sizes. Uzsoy (1995), Mönch et al. (2005), Malve and Uzsoy (2007) and Chiang et al. (2010) addressed parallel-identical BPMs with incompatible job families. Uzsoy (1995) proposed different algorithms to minimize makespan, total weighted completion time and maximum lateness. To minimize the total weighted tardiness, Mönch et al. (2005) applied a GA. To minimize the maximum lateness, Malve and Uzsoy (2007) proposed a family of iterative improvement heuristics and combined them with a random key-based GA. Chiang et al. (2010) proposed a memetic algorithm for minimizing the total weighted tardiness. Bilyk et al. (2014) proposed local search based meta-heuristics (Variable Neighborhood Search (VNS) and Greedy Randomized Adaptive Search Procedure (GRASP)) to minimize the total weighted tardiness on parallel-identical BPMs considering incompatible job families and precedence constraints among the jobs.

Both non-identical job sizes and non-zero job release times were considered by some authors. Mathirajan and Sivakumar (2006b) provided greedy heuristic algorithms to minimize the total weighted tardiness on parallel-identical BPMs with incompatible job families. The makespan minimization was considered by Chung et al. (2009), Damodaran and Vélez-Gallego (2010), Damodaran et al. (2011), Damodaran and Vélez-Gallego (2012). Chung et al. (2009) proposed a MIP model, a MIP-based algorithm and three constructive heuristics. Damodaran and Vélez-Gallego (2010), Damodaran et al. (2011), and Damodaran and Vélez-Gallego (2012) proposed a constructive heuristic, a GRASP heuristic and a SA heuristic, respectively. Ozturk et al. (2012) addressed the makespan minimization problem with equal job processing times and gave a 2-approximation algorithm as well as a MIP model. To minimize the bi-criteria of makespan and maximum tardiness, a Pareto-based ACO

was developed by Xu et al. (2013). Abedi et al. (2015) presented a bi-objective MIP model for parallel-identical BPMs in which different capacity limits are considered. To minimize the objectives makespan and the total weighted earliness and tardiness of jobs, they proposed an  $\epsilon$ -constraint method and two multi-objective heuristics.

The literature that focuses on parallel-identical BPMs with non-identical capacities and makespan minimization is still somewhat limited. Xu and Bean (2007) proposed a MIP model and a GA based on random keys encoding. Wang and Chou (2010) considered non-zero job release times and provided a MIP model and meta-heuristic algorithms based on SA and GA to solve the problem. Damodaran et al. (2012) proposed a Particle Swarm Optimization (PSO) algorithm and Jia et al. (2015) presented a heuristic based on the FF-Decreasing rule as well as a meta-heuristic based on MMAS algorithm. Wang and Leung (2014) considered equal-processing-time jobs and showed that there is no polynomial-time approximation algorithm with an absolute worst-case ratio less than 2, unless  $P = NP$ . They then gave a polynomial-time algorithm with an absolute worst-case ratio of exactly 2. Finally, they proposed a polynomial-time algorithm with asymptotic worst-case ratio no more than  $3/2$ . Very recently, Jia et al. (2017) considered jobs with arbitrary sizes, non-zero job release times and machines with arbitrary capacities. They provided a lower bound and proposed two meta-heuristics based on ACO algorithm to solve the problem, where the first one outperformed the other six meta-heuristic algorithms.

Klemmt et al. (2009), Gokhale and Mathirajan (2014) and Hulett and Damodaran (2015) have considered parallel-identical BPMs with non-identical capacities and the total weighted tardiness minimization. Klemmt et al. (2009) addressed a problem with incompatible job families and unequal job release times. They provided a MIP model and a VNS heuristic. Gokhale and Mathirajan (2014) took into account incompatible job families, unequal release times and allowance for job splitting (splitting of a job is allowed only once, that is, when the capacity of a batch processor exceeds during the batch formation). They have developed a MIP model and heuristic algorithms. Hulett and Damodaran (2015) proposed a PSO algorithm.

We only find two papers dealing with uniform parallel BPMs scheduling problem with different processing speeds and makespan minimization. Li et al. (2013a) considered different job release times and proposed several heuristics based on batching and assignment rules. Very recently, Zhou et al. (2016) included machines with non-identical capacities and arbitrary job sizes, and proposed a Discrete Differential Evolution (DDE) based genetic algorithm.

Few researchers have considered the problem of batch scheduling on unrelated parallel BPMs. Unrelated parallel machine environment is a generalization of the identical and uniform parallel machine environment and is closer to real-world production systems. Li et al. (2013b) and Arroyo and Leung (2017b) considered scheduling unrelated parallel BPMs with identical capacities and non-identical job sizes to minimize the makespan. Li et al. (2013b) considered equal job release times and proposed two groups of deterministic heuristics. In both groups, the rules BF and LPT were used to form batches. The first group of heuristics construct a solution in two phases: grouping the jobs into several feasible batches and scheduling the batches on the set of machines to minimize the makespan. The second group of heuristics also construct a solution in two phases but in a different way. The first phase determines which jobs have to be allocated to which machines and the second phase determines how to schedule jobs on each single BPM. The results presented by Li et al. (2013b) showed that the heuristic of the second group which uses the Earliest Completion Time (ECT) rule in the first phase to allocate the jobs to the machines, outperformed the other presented heuristics. Arroyo and Leung (2017b) considered unequal job release times and proposed three deterministic heuristics based on the rules Earliest Ready Time (ERT), ECT, and FF or BF. The first two heuristics are based on the heuristics proposed by Li et al. (2013b). The third heuristic, different from the heuristics of Li et al. (2013b), constructs a solution in a single phase: grouping the jobs into batches and scheduling the batches on

the machines are done simultaneously. In this heuristic the jobs are first sorted by the ERT rule, and the earliest ready job is iteratively assigned to a batch of the machine that meets the ECT rule. Arroyo and Leung (2017b) showed that the heuristic that uses the FF rule outperforms the other presented heuristics. Arroyo and Leung (2017a) considered scheduling unrelated parallel BPMs with different capacities, non-identical job sizes, unequal job release times and makespan minimization. They proposed an IG algorithm and showed that it performs better than other algorithms, including a GA Wang and Chou (2010), ACO Jia et al. (2015) and SA Damodaran and Vélez-Gallego (2012). Shahidi-Zadeh et al. (2017) studied a scheduling problem on unrelated parallel BPMs with considering the release time and ready time for jobs as well as batch capacity constraints. They propose a multi-objective mixed-integer non-linear programming model and a Harmony Search algorithm to minimize two objectives simultaneously: the makespan, and the sum of tardiness/earliness penalties of jobs and the purchasing cost of machines. Shahvari and Logendran (2017) also studied a bi-objective scheduling problem on unrelated parallel BPMs. They considered operators, in addition to machines, as resources. The operators are assigned for performing the setup and run of batches on machines based on different levels of required skills. Dynamic job release times, non-identical job sizes, machine eligibility and capability for processing jobs, batch capacity limitations and machine-dependent setup time are considered. They proposed a integer programming model and four bi-objective PSO algorithms for simultaneously minimizing the makespan and the production cost including the batch processing cost on both machines and operators along with the earliness and tardiness cost of jobs.

### 3. Problem formulation

In this section, we first present the formal definition of the target problem. Then we formulate the problem as a mixed integer programming (MIP) model.

There is a set of  $n$  jobs,  $J = \{1, 2, \dots, n\}$ , to be processed on a set of  $m$  unrelated parallel BPMs,  $M = \{1, \dots, m\}$ . For each job  $j \in J$ , the size  $s_j$ , the release time  $r_j$  and  $m$  different processing times,  $p_{jk}$  ( $k = 1, \dots, m$ ) are known in advance. Each machine  $k \in M$  has a capacity denoted by  $Q_k$ , and each job does not exceed the largest machine capacity, that is,  $s_j \leq \max_{k \in M} \{Q_k\}, \forall j \in J$ . The total size of all the jobs in a batch cannot exceed the capacity of the machine where the batch is scheduled. Each machine becomes available at time zero and machine idle times are permitted. Once the processing of a batch starts, it cannot be interrupted and other jobs cannot be added into the batch or removed from the batch until the processing is complete. The processing time of a batch is determined by the job with the longest processing time in the batch. Similarly, the release time of a batch is equal to the latest release time of the jobs in the batch. All jobs in a batch have the same start time and completion time. The number of batches on the machine  $k$ , defined by  $nb_k$ , cannot be determined until all the jobs have been grouped. Let  $B_{bk}$  be a batch  $b$  on a machine  $k$  ( $b = 1, \dots, nb_k$ ,  $k = 1, \dots, m$ ). The processing time and release time of batch  $B_{bk}$  are determined by  $P_{bk} = \max_{j \in B_{bk}} \{p_{jk}\}$  and  $R_{bk} = \max_{j \in B_{bk}} \{r_j\}$ , respectively.

The objective of the problem is to group the job set into batches and assign the batches to the unrelated parallel BPMs in order to minimize the total flow time of the schedule, defined by  $TF = \sum_{j=1}^n F_j$ , where  $F_j = C_j - r_j$  and  $C_j$  are the flow time and the completion time of job  $j$ , respectively. Note that, if a job  $j$  is processed in the batch  $B_{bk}$  on machine  $k$ , the completion time of job is  $C_j = S_{bk} + P_{bk}$ , where  $S_{bk}$  is the start time of batch  $B_{bk}$ .

The target problem can be represented as  $R_m | p - \text{batch}, Q_k, s_j, r_j | \sum F_j$  by using a triplet notation  $\alpha/\beta/\gamma$  presented by Graham et al. (1979), where  $R_m$  means  $m$  unrelated parallel batch machines,  $Q_k$  denotes non-identical machine capacities,  $r_j$  and  $s_j$  denote that each job  $j$  has an arbitrary job size and non-zero release time.

A simple lower bound for total flow time can be computed as follows. Without loss of generality, suppose all the  $m$  machines have different

capacities, and the capacities are sorted as follows:  $Q_1 < Q_2 < \dots < Q_m$ . The job set  $J = \{1, \dots, n\}$  can be divided into  $m$  disjoint subsets, denoted by  $J^1, J^2, \dots, J^m$ , where  $J^1 = \{j \in J | s_j \leq Q_1\}$ ,  $J^2 = \{j \in J | Q_1 < s_j \leq Q_2\}, \dots, J^m = \{j \in J | Q_{m-1} < s_j \leq Q_m\}$ . That is, the jobs in  $J^1$  (“small jobs”) can be scheduled on any machine  $k \in M = \{1, \dots, m\}$ , the jobs in  $J^2$  can be scheduled on any machine  $k \in M^2$  (where  $M^2 = \{2, \dots, m\}$ ), and so on, until the jobs in  $J^m$  (“large jobs”) can only be scheduled on machine  $k = m$ . The lower bound  $LB$  is obtained according to Eq. (1):

$$LB = \sum_{j \in J^1} \min_{k \in M} \{p_{jk}\} + \sum_{j \in J^2} \min_{k \in M^2} \{p_{jk}\} + \sum_{j \in J^3} \min_{k \in M^3} \{p_{jk}\} + \dots + \sum_{j \in J^m} p_{jm} \quad (1)$$

Since the flow time of a job is the period between its release time and its completion time, the lower bound  $LB$  is given by the total processing time of the jobs. The period between the job release time and the job start time is not considered. Since each machine processes each job at different processing time, the minimum processing time among the machines that can process a job is considered.

Next, we present a MIP formulation of the  $R_m|p\text{-batch}, Q_k, s_j, r_j|\sum F_j$  problem. The formulation is similar to the MIP model presented by Arroyo and Leung (2017b), however, modifications concerning total flow time become necessary. In this model, the number of batches on each machine is uncertain before constructing a complete solution. In the worst-case, the number of batches formed on a machine  $k$  will be equal to  $n$ . This worst-case occurs when all the jobs are processed on one machine and each job is assigned to one batch. Hence, in the model the number of batches on each machine  $k$  is set to be  $n$  (i.e.  $nb_k = n$ ,  $\forall k = 1, \dots, m$ ). A batch  $B_{bk}$  ( $b = 1, \dots, n$ ;  $k = 1, \dots, m$ ) is open if it contains at least one job; otherwise, it is closed with a processing time of zero.

The following decision variables are used in the model:

- $x_{j,b,k}$ : equal to 1 if job  $j$  is scheduled in the  $b$ th batch on machine  $k$ , otherwise zero.
- $C_j$ : completion time of job  $j$ .
- $S_{b,k}$ : starting time of the  $b$ th batch on machine  $k$ .
- $P_{b,k}$ : processing time of the  $b$ th batch on machine  $k$ .

The mathematical model is:

$$\text{Min} \quad \sum_{j=1}^n (C_j - r_j) \quad (2)$$

$$\text{st.} \quad \sum_{b=1}^n \sum_{k=1}^m x_{j,b,k} = 1, \quad j = 1, \dots, n. \quad (3)$$

$$\sum_{j=1}^n s_j x_{j,b,k} \leq Q_k, \quad b = 1, \dots, n; k = 1, \dots, m. \quad (4)$$

$$S_{b,k} \geq r_j x_{j,b,k}, \quad j = 1, \dots, n; b = 1, \dots, n; k = 1, \dots, m. \quad (5)$$

$$S_{(b+1),k} \geq S_{b,k} + P_{b,k}, \quad b = 1, \dots, n-1; k = 1, \dots, m. \quad (6)$$

$$P_{b,k} \geq p_{j,b,k} x_{j,b,k}, \quad j = 1, \dots, n; b = 1, \dots, n; k = 1, \dots, m. \quad (7)$$

$$C_j \geq S_{b,k} + P_{b,k} - G(1 - x_{j,b,k}), \quad j = 1, \dots, n; b = 1, \dots, n; k = 1, \dots, m. \quad (8)$$

$$C_j \geq 0, \quad S_{b,k} \geq 0, \quad P_{b,k} \geq 0, \quad j = 1, \dots, n; b = 1, \dots, n; k = 1, \dots, m. \quad (9)$$

$$x_{j,b,k} \in \{0, 1\}, \quad j = 1, \dots, n; b = 1, \dots, n; k = 1, \dots, m. \quad (10)$$

The objective function (2) is to minimize the total flow time. Constraint set (3) ensures that each job can only be processed in one batch on one machine. Constraint set (4) guarantees that the total size of all jobs in a batch does not exceed the machine capacity. Constraint sets (5) and (6) determine the starting time of batch  $b$  on machine  $m$  with either the largest job release time in batch  $b$  or the completion time of the previous batch. These constraints ensure that each job can be processed only after it is released and each batch can be started only after the previous one on the machine is finished. Constraint set (7) determines the processing time of batch  $b$  processed on machine  $k$ , which is equal to the longest

job processing time in batch  $b$ . Constraint set (8) determines the job completion times, where  $G$  is a sufficiently large positive number. If a job  $j$  is scheduled in the batch  $b$  on machine  $k$  ( $x_{j,b,k} = 1$ ), its completion time is equal to the batch completion time. Constraints (9) represent non-negativity conditions of the decision variables  $C_j$ ,  $S_{b,k}$  and  $P_{b,k}$ , while constraints (10) ensure that  $x_{j,b,k}$  are binary variables. The parameter  $G$  in (8) is defined as follows:  $G = \sum_{j=1}^n \sum_{k=1}^m p_{jk}$ .

#### 4. The iterated greedy algorithm

Iterated greedy (IG) Hoos and Stützle (2004) is a simple stochastic meta-heuristic that starts from an initial solution and then tries to improve the current solution by iterating over three main phases: destruction, construction and acceptance phases. In the first phase, some elements are randomly removed from the current solution obtaining a partial solution. In the second phase, a greedy construction heuristic is used to reinsert the removed elements in order to form a new complete solution. Next, an acceptance criterion is applied to decide if the new solution substitutes the current solution. To further improve the performance of IG algorithm, a local search phase can be applied to the initial solution and the new solution resulting from the construction phase. These phases are repeated until the stop condition is satisfied. The best obtained solution is returned by the algorithm.

IG has been successfully applied to various scheduling problems (Ruiz and Stützle, 2007; Ying et al., 2009; Rodriguez et al., 2013; Pan and Ruiz, 2014; Benavides and Ritt, 2016). To the best of our knowledge, the first application of IG algorithm for a p-batch scheduling problem on parallel BPMs was proposed by Arroyo and Leung (2017a). These authors considered the makespan minimization. In this paper we develop an IG to solve the problem  $R_m|p\text{-batch}, Q_k, s_j, r_j|\sum F_j$ , where the greedy heuristic used to construct an initial solution, and the local search procedure are different from the IG of (Arroyo and Leung, 2017a). Furthermore, the local search procedure in our IG algorithm is executed periodically because this procedure is an optional phase.

In the following we describe the IG algorithm that we develop for the problem  $R_m|p\text{-batch}, Q_k, s_j, r_j|\sum F_j$ . The general pseudocode of the IG algorithm is presented in Algorithm 1. The algorithm has the following control parameters: the destruction level ( $n_r$ ), probability to accept a worse solution ( $prob$ ), parameter used to activate the local search phase ( $Nls$ ), and parameter that defines the stopping criterion ( $stop\_condition$ ). It is emphasized that the local search procedure is executed every  $Nls$  iterations. All the steps of our IG algorithm are explained in the following subsections.

##### 4.1. Encoding and initialization

In order to implement our IG algorithm, a solution (schedule) is represented by  $m$  sets of batches, one set for each machine:  $BS_1, \dots, BS_m$ . Each set  $BS_k$  is an ordered sequence of  $nb_k$  batches, where  $nb_k$  is a number to be determined ( $k = 1, \dots, m$ ).

The IG algorithm needs an initial solution to start the search process. To rapidly construct an initial feasible solution, we adapt the best greedy deterministic heuristic, proposed by Arroyo and Leung (2017b), for our problem  $R_m|p\text{-batch}, Q_k, s_j, r_j|\sum F_j$ . The basic idea of the greedy heuristic, implemented in this paper, is as follows. Initially, all the jobs should be arranged by using a priority rule, obtaining an ordered job list. Also, the sets  $BS_1, \dots, BS_m$  start empty. Next, the first job of the ordered list is assigned to a batch of the machine that provides the smallest increment in the value of the partial total flow time. If the job fits in no existing batch, a new batch is created on a machine. Even if a feasible batch (batch with sufficient capacity) exists on a machine, the creation of a new batch at the end of the machine is tested. The job assigning is repeated until all the jobs have been assigned to a batch. In the greedy heuristic, grouping the jobs into batches and scheduling the batches on the machines are simultaneously done.

**Algorithm 1 : Iterated-Greedy( $n_r$ , prob, Nls, stop\_condition)**


---

```

1:  $S := \text{Initial\_Solution}();$ 
2:  $S_c := \text{Local\_Search}(S);$ 
3:  $S_{best} := S_c; iteration := 0;$ 
4: while stop_condition do
5:    $S_p := \text{Destruction}(S_c, n_r); //n_r is the destruction level$ 
6:    $S_{new} := \text{Construction}(S_p, n_r);$ 
7:   if iteration mod Nls = 0 then
8:      $S_{new} := \text{Local\_Search}(S_{new});$ 
9:   end if
10:  if  $TF(S_{new}) < TF(S_c)$  then
11:     $S_c := S_{new}; //update the current solution$ 
12:    if  $TF(S_{new}) < TF(S_{best})$  then
13:       $S_{best} := S_{new}; //update the best solution$ 
14:    end if
15:  else
16:    //Acceptance criterion:
17:    if  $\text{random}(0, 1) < \text{prob}$  then
18:       $S_c := S_{new}; //a worse solution is accepted$ 
19:    end if
20:  end if
21:  iteration := iteration + 1;
22: end while
23: Return  $S_{best};$ 

```

---

The first ordered job list is obtained by arranging the jobs in Earliest Ready Time (ERT) (Arroyo and Leung, 2017b) or the Priority Rule for Total Flow time (PRTF) (Chu, 1992). The ERT rule was used in Arroyo and Leung (2017b,a). The PRTF rule was proposed to minimize the total flow time in the single machine scheduling problem with different release times (Chu, 1992). In this paper, the PRTF rule is adapted for the problem under investigation. For each job  $j$ , a priority function  $PRTF(j, \Delta)$  is defined as  $PRTF(j, \Delta) = 2 * \max(\Delta, r_j) + p_j$ , where  $\Delta$  is the availability time of the machines and  $p_j$  is the processing time of job  $j$ . It can be noted that the function  $PRTF(j, \Delta)$  is in fact a combination of the ERT and SPT (Shortest Processing Time) priority rules (Chu, 1992). Since all the machines are available at time zero, we use  $\Delta = 0$ . For the processing time of job  $j$ , we use the minimum  $p_j = \min_{k \in M} \{p_{jk}\}$  or the average  $p_j = \frac{\sum_{k \in M} \{p_{jk}\}}{m}$  processing time. In this paper, the rule PRTF with the minimum and average processing time are called PRTF1 and PRTF2, respectively. In the computational experiments, we compare the performance of the greedy heuristic with each of the three priority rules: ERT, PRTF1 and PRTF2.

A 15-job and 2-machine problem instance (see Table 1) is used to illustrate how an initial schedule is constructed by the greedy heuristic. By applying the PRTF1 rule, the following ordered job list is obtained: {7, 15, 10, 13, 14, 8, 3, 6, 4, 5, 1, 2, 11, 9, 12}. Table 2 shows the first eight steps and the last step of the evolution of the constructive heuristic. The first column of this table contains the next job to be assigned to a batch, the second and third columns contain the formed batches on the machines 1 ( $m_1$ ) and 2 ( $m_2$ ), respectively. The last column presents the total flow time (TF) of the obtained schedule. At the third step we note that, the job 10 (with size  $s_{10} = 3$ ) fits in the batch  $B_1 = \{7, 15\}$  of machine 1 ( $s_7 + s_{15} = 16 < Q_1 = 30$ ). However, to minimize the partial total flow time, the batch  $B_2 = \{10\}$  is created on machine 2. The job 10 is processed with the minimum processing time on machine 2. The final schedule generated by the constructive heuristic is shown in Fig. 1(a). For this schedule, the sets of batches are  $BS_1 = \{B_1, B_3, B_6, B_7\}$  and  $BS_2 = \{B_2, B_4, B_5\}$ , where  $nb_1 = 4$  and  $nb_2 = 3$ . Fig. 1(a) also shows the start and completion times of the batches. The batch release times of the constructed schedule are:  $m_1: R_{B_1} = 9, R_{B_3} = 13, R_{B_6} = 35, R_{B_7} = 47$ ;  $m_2: R_{B_2} = 7, R_{B_4} = 24, R_{B_5} = 52$ . It can be seen that the batches were scheduled in non-decreasing order of their release times, and  $B_1$  and  $B_2$  are the only two batches that start in their release times. In this example

the batches  $B_i$  are enumerated in the order that they were created, and once the machines start the job processing, there are no machine idle times. For large job release times ranges, generally, there are machine idle times.

#### 4.2. Destruction and construction phases

The destruction and construction phases are employed for the solution modification in the IG algorithm, i.e., the two phases are responsible for moving towards new regions in the solution space. The destruction phase is performed in an iterative way, removing a job each time from the current schedule. A given number  $n_r$  of jobs are randomly chosen and removed from the current schedule, where  $n_r$  is a parameter called destruction level. This generates a partial schedule  $S_p$  and the set of removed jobs, which is denoted as  $J_R$ . Apparently, the goal of the destruction phase is not to destroy the current schedule completely, on the contrary, it is desirable that the resulting partial schedule preserves some characteristics of the schedule obtained in the previous iteration. Before starting with the construction phase, the batches on the partial solution  $S_p$  are scheduled in non-decreasing order of their release times, the completion times of batches and the flow times of jobs are recomputed.

The construction phase starts with the partial schedule  $S_p$  and sequentially reassigns the jobs in  $J_R$  (e.g.,  $J_R = \{j_1, \dots, j_{n_r}\}$ ) one at a time, using a greedy insertion heuristic. This heuristic works as follows. The first job  $j_1$  is inserted into all possible existing feasible batches in the partial solution  $S_p$ . The total size of the jobs in the batch should not exceed the capacity of the machine. Also, a new batch is created with job  $j_1$  and is inserted into all possible feasible machines (machines that have capacities greater than or equal to  $s_{j_1}$ ). This batch is inserted into the appropriate position on the machine such that the batches still satisfy the non-decreasing order of their release times. The insertions of jobs into all possible feasible batches and insertions of new batches into the feasible machines generate a number of partial solutions. This number depends on the number of feasible batches and feasible machines. Among these partial solutions, the solution with the minimum total flow time is chosen and kept for the next iteration of the construction phase. Then, the second job  $j_2$  of  $J_R$  is considered and so on until all  $n_r$  jobs are reinserted into the partial solution and a complete schedule is constructed.

To illustrate how the IG algorithm works, consider the 15-job and 2-machine problem instance presented in Table 1. The example shows one iteration of the IG algorithm without considering the local search phase and using  $n_r = 2$  (number of jobs to be removed). Starting from the initial schedule (Fig. 1(a)), a partial schedule is obtained by removing jobs 13 and 3 (see Fig. 1(b)). These two jobs are removed from batches  $B_1$  (machine 1) and  $B_4$  (machine 2), respectively. We can see that by removing jobs 13 and 3, the completion times of all the batches, except batch  $B_2$ , are decreased. If the completion time of a batch is decreased, the completion times of all the jobs in this batch are also reduced, consequently the flow times of these jobs are also reduced. In the construction phase, the jobs 13 and 3 are reinserted, one at a time, in the best position of the partial schedule, that is, a job is assigned to a batch in such a way that the resulting schedule provides the smallest increment in the value of the total flow time. The schedules reached after reinsertion of jobs 13 and 3 are shown in Figs. 1(c) and 1(d), respectively.

By comparing the initial schedule (Fig. 1(a)) and the schedule obtained after the construction phase (Fig. 1(d)), we can see that the total flow time was improved because the completion times of four batches were decreased. For example, the completion time of batch  $B_5$  is decreased from 94 to 85. Then, the flow times of all the jobs in this batch are also reduced, because the completion times of these jobs are equal to the completion time of batch  $B_5$ . It is easily to verify the following Proposition 1.

**Table 1**A data for 15-job, 2-machine problem instance. Machine capacities:  $Q_1 = 30$ ,  $Q_2 = 50$ .

Job $j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$p_{j1}$	54	53	15	60	10	40	14	28	45	42	53	22	21	15	17
$p_{j2}$	22	27	40	11	59	26	52	52	10	20	25	55	55	56	57
$s_j$	5	4	14	6	15	11	1	6	10	3	12	15	3	14	15
$r_j$	32	40	22	33	35	24	8	13	52	7	44	47	9	13	7

**Table 2**

Construction of the initial schedule from the ordered job list {7, 15, 10, 13, 14, 8, 3, 6, 4, 5, 1, 2, 11, 9, 12}.

Assigned job	Machine 1 ( $m_1$ )	Machine 2 ( $m_2$ )	TF
7	$B_1 = \{7\}$	—	14
15	$B_1 = \{7, 15\}$	—	35
10	$B_1 = \{7, 15\}$	$B_2 = \{10\}$	55
13	$B_1 = \{7, 15, 13\}$	$B_2 = \{10\}$	86
14	$B_1 = \{7, 15, 13\}, B_3 = \{14\}$	$B_2 = \{10\}$	118
8	$B_1 = \{7, 15, 13\}, B_3 = \{14, 8\}$	$B_2 = \{10\}$	176
3	$B_1 = \{7, 15, 13\}, B_3 = \{14, 8\}$	$B_2 = \{10\}, B_4 = \{3\}$	221
6	$B_1 = \{7, 15, 13\}, B_3 = \{14, 8\}$	$B_2 = \{10\}, B_4 = \{3, 6\}$	264
...	...	...	...
12	$B_1 = \{7, 15, 13\}, B_3 = \{14, 8\}, B_6 = \{5\}, B_7 = \{12\}$	$B_2 = \{10\}, B_4 = \{3, 6\}, B_5 = \{4, 1, 2, 11, 9\}$	609

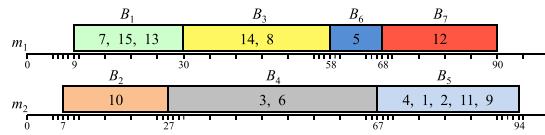
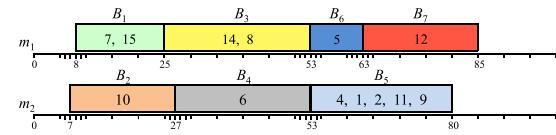
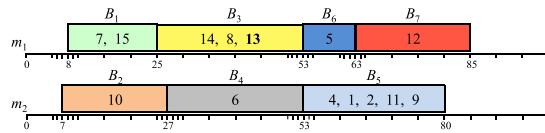
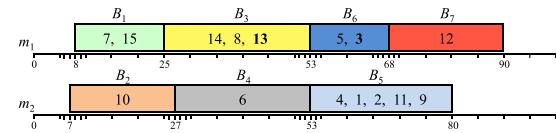
(a) Initial schedule obtained by the greedy heuristic.  $TF = 609$ .(b) Partial schedule obtained in the destruction phase.  $J_R = \{13, 3\}$ .  $TF = 429$ .(c) Construction phase: schedule after reinserting job 13.  $TF = 473$ .(d) Construction phase: schedule after reinserting job 3.  $TF = 529$ .

Fig. 1. (a): Initial schedule. (b), (c), (d): Illustration of the destruction and construction phases of IG algorithm.

**Proposition 1.** The total flow time measure  $\sum_{j=1}^n \{C_j - r_j\}$  decreases only if at least one of the job completion-times decreases. That is,  $\sum_{j=1}^n \{C'_j - r_j\} < \sum_{j=1}^n \{C_j - r_j\}$  if and only if  $C'_j < C_j$  for at least one job  $j$ ,  $1 \leq j \leq n$ .

#### 4.3. Local search phase

Local search (LS) methods are optimization algorithms that start from an initial solution and then try to improve this solution by searching its neighborhood for even better solutions. In the literature, neighborhood-based LS is widely applied method to improve solutions generated by meta-heuristic algorithms. However, the strategy of scanning the entire neighborhood in a LS algorithm usually consumes the largest part of the meta-heuristic (Arroyo and Armentano, 2005). Han et al. (2015) Han et al. (2016) use neighborhood-based LS to further enhance the exploitation of solutions in a blocking flow shop scheduling problem. To reduce the computational effort, they use LS method with a small probability.

In this paper to further improve the performance of our IG algorithm, a LS procedure is applied to the solution just constructed (the initial solution and the solution resulting from the destruction-construction phases). The LS procedure is executed every  $Nls$  iterations, that is, it is executed from time to time ( $Nls$  is a parameter to be determined). The LS is based on job exchange and is similar to that proposed by Arroyo and Leung (2017a). The most important difference is that in Arroyo and Leung (2017a) the jobs to be exchanged are randomly chosen in the

current schedule, that is, each job is randomly selected from a machine and a batch. We have observed that with totally random exchanges, many unnecessary job movements are made and the improvement of the current solution is slower. In this paper, the jobs to be exchanged are randomly chosen from two batches,  $B_a$  and  $B_b$ , of the same machine  $k$  that is randomly chosen. The two batches are randomly chosen in such a way that they keep an adjacent distance less than or equal to  $dist$ , where  $dist$  is an input parameter  $\geq 1$ . To explain better, let us suppose that  $\{B_1, B_2, \dots, B_{nb_k}\}$  is the sequence of batches in the chosen machine  $k$ , where  $nb_k \geq 2$ . The two batches  $B_a$  and  $B_b$  are randomly chosen in machine  $k$ , such that  $1 \leq a \leq nb_k - 1$ , and  $a + 1 \leq b \leq \min\{a + dist, nb_k\}$ . Note that, if  $dist = 1$ , the batches  $B_a$  and  $B_b$  are adjacent in machine  $k$ , that is,  $b - a = 1$ . If  $dist = 2$ , the adjacent distance between the two batches is  $\leq 2$ , that is,  $b - a \leq 2$ . If  $dist = nb_k$ , the batches are randomly chosen in machine  $k$ .

The exchange of the chosen jobs  $j_1 \in B_a$  and  $j_2 \in B_b$  is made if the machine capacity is not exceeded (i.e.  $\sum_{j \in B_a} \{s_j\} - s_{j_1} + s_{j_2} \leq Q_k$  and  $\sum_{j \in B_b} \{s_j\} - s_{j_2} + s_{j_1} \leq Q_k$ ). If the exchange yields a better solution, the new solution  $S'$  is saved and an iteration counter is reset to 0, else this iteration counter is incremented. The LS procedure stops if there is no improvement for  $n$  consecutive iterations (i.e. the iteration counter is

equal to  $n$ ). The pseudocode of the proposed LS procedure is given in Algorithm 2.

---

**Algorithm 2 : Local\_Search( $S$ )**


---

```

1:  $S^* := S$ ;  $counter := 0$ ;
2: while  $counter < n$  do
3:   Select a random machine  $k$  in solution  $S^*$  with at least two batches. Let us suppose that  $\{B_1, B_2, \dots, B_{nb_k}\}$  is the sequence of batches in machine  $k$ , where  $nb_k \geq 2$ ;
4:   Select two random batches  $B_a$  and  $B_b$ , such that  $1 \leq a \leq nb_k - 1$ , and  $a + 1 \leq b \leq \min\{a + dist, nb_k\}$ , where  $dist$  is the distance between  $B_a$  and  $B_b$ ;
5:   Select two random jobs  $j_1$  and  $j_2$  in batches  $B_a$  and  $B_b$ , respectively;
6:   if  $\sum_{j \in B_a} \{s_j\} - s_{j_1} + s_{j_2} \leq Q_k$  and  $\sum_{j \in B_b} \{s_j\} - s_{j_2} + s_{j_1} \leq Q_k$  then
7:      $S' :=$  schedule obtained by exchanging jobs  $j_1$  and  $j_2$  in solution  $S^*$ ;
8:     if  $TF(S') < TF(S^*)$  then
9:        $S^* := S'$ ;
10:       $counter := 0$ ;
11:    end if
12:  end if
13:   $counter := counter + 1$ ;
14: end while
15: Return  $S^*$ ;

```

---

#### 4.4. Acceptance criterion

Acceptance criterion determines whether the new solution  $S_{new}$  is accepted or not as the current solution for the next iteration. As in Arroyo and Leung (2017a), the IG algorithm uses a simple acceptance criterion which accepts worse solutions only with a certain probability  $prop$  ( $0 \leq prop \leq 1$ ). A uniform random number  $rand$  between 0 and 1 is generated. If  $rand < prop$ ,  $S_{new}$  becomes the new current solution, independent of its objective function value. Otherwise,  $S_{new}$  is discarded.  $prop$  is an input parameter to be calibrated. If  $prop = 0$ , the acceptance criteria only accepts solutions with better objective values. However, large values of  $prop$  may produce a higher diversification, and the search process may become slower.

#### 4.5. Decoding procedure

Since a solution is represented by  $m$  sets of batches ( $BS_1, \dots, BS_m$ ), where each set  $BS_k$  contains  $nb_k$  batches ( $k = 1, \dots, m$ ), the decoding procedure transforms this representation into a schedule. That is, this procedure determines the completion times of the batches and computes the total flow time of the schedule ( $TF$ ). The decoding procedure works as follows.

- For each batch  $B_i$  of  $BS_k$  ( $k = 1, \dots, m$ ) determine its completion time:  $CB_i = \max\{CB_{i-1}, R_{B_i}\} + P_i$ , where  $R_{B_i}$  and  $P_i$  are, respectively, the release time and processing time of  $B_i$ , and  $CB_{i-1}$  is the completion time of the immediately preceding batch of  $B_i$ . If  $B_i$  is the first batch of  $BS_k$ ,  $CB_{i-1} = 0$ .
- For each job  $j$  in a batch  $B_i \in BS_k$ , determine its flow time  $F_j = C_j - r_j$ , where  $C_j = CB_i$ .
- Finally, determine  $TF = \sum_{j=1}^n F_j$ .

It is important to mention that in our IG algorithm, the decoding procedure is only used after the destruction phase (when a solution is randomly destructed). For example, in Fig. 1(b) we have a solution obtained after the destruction phase:  $BS_1 = \{B_1, B_3, B_6, B_7\}$  and  $BS_2 = \{B_2, B_4, B_5\}$ , where  $B_1 = \{7, 15\}$ ,  $B_3 = \{14, 8\}$ ,  $B_6 = \{5, \}$  and  $B_7 = \{12\}$  are the batches on machine 1; and  $B_2 = \{10\}$ ,  $B_4 = \{6\}$  and  $B_5 = \{4, 1, 2, 11, 9\}$  are the batches on machine 2. The release time  $R_{B_i}$  and processing time  $P_i$  of each batch  $B_i$  are not necessary to calculate,

because they were determined previously. When a job  $j$  is removed from a batch  $B_i$ ,  $R_{B_i}$  and  $P_i$  are, respectively, equal to the largest release time and largest processing time among all the remaining jobs in the batch. For each machine, the completion times of the batches are calculated in orderly fashion. For example on machine 1, the completion time of batch  $B_1$  is first calculated, then of batches  $B_3$ ,  $B_6$  and  $B_7$ . Note that the completion times of all the batches are  $CB_1 = 25$ ,  $CB_3 = 53$ ,  $CB_6 = 63$ ,  $CB_7 = 85$ ,  $CB_2 = 27$ ,  $CB_4 = 53$  and  $CB_5 = 80$ . Next, the flow times of all the jobs are determined. For example, the flow times of jobs 7 and 15 (in the batch  $B_1$ ) are  $F_7 = CB_1 - r_7 = 25 - 8 = 17$  and  $F_{15} = CB_1 - r_{15} = 25 - 7 = 18$ , respectively (the values of  $r_7$  and  $r_{15}$  are presented in Table 1). Once the flow times of all the jobs are calculated, the total flow time is obtained:  $TF = 429$ .

In the construction phase of the IG algorithm, when a job  $j$  is inserted into a batch  $B_i$  of a machine  $k$ ,  $R_{B_i} = \max\{R_{B_i}, r_j\}$  and  $P_i = \max\{P_i, p_{jk}\}$ . In this case, we only recalculate the completion times of batch  $B_i$  and its subsequent batches:  $B_{i+1}, B_{i+2}, \dots, B_{nb_k}$ , where  $nb_k$  is the number of batches on the machine  $k$ . The completion times of the batches on the other machines are the same.

In the local search phase, neighbor solutions are obtained by jobs exchanges. To change two jobs  $j_1 \in B_a$  and  $j_2 \in B_b$ , first, they should be removed from the batches. Then  $j_1$  and  $j_2$  should be inserted in  $B_b$  and  $B_a$ , respectively. To determine the total flow time of a neighbor solution, we only recalculate the completion times of the batches  $B_a$ ,  $B_b$  and their subsequent batches.

Therefore, the calculation of the total flow time for new obtained solutions is relatively quickly, since it is not necessary to made a complete evaluation.

## 5. Benchmark meta-heuristic algorithms from the literature

To evaluate the efficiency of our IG algorithm, three benchmark meta-heuristic algorithms have been re-implemented to solve the target problem  $R_m|p - batch, Q_k, s_j, r_j| \sum F_j$ . The algorithms are: DDE, ACO and SA proposed, respectively, by Zhou et al. (2016), Jia et al. (2017), and Damodaran and Vélez-Gallego (2012).

To make a fair comparison, all the algorithms use the same stopping condition which is based on the amount of CPU time. Since DDE, ACO and SA were developed to solve different problems, some adaptations of those algorithms were proposed. The following subsections outline the main adaptations done in the three implemented meta-heuristics.

### 5.1. Discrete Differential Evolution (DDE) algorithm

The DDE algorithm presented in Zhou et al. (2016) is a stochastic population-based method in which a solution is constructed from a permutation of the  $n$  jobs (solution encoding). The solution evaluation consists of scheduling the jobs on the parallel BPMs determining the objective function value. Since the original DDE algorithm was proposed to minimize the makespan objective, the solution evaluation procedure uses a constructive method based on the greedy LPT heuristic. For the total flow time objective, we use the greedy constructive heuristic described in Section 4.1. In this heuristic, first, an ordered job list (or job permutation) should be obtained. Then, the batch formation and scheduling are done simultaneously.

In our DDE implementation the initial population is generated as follows. Three permutations are generated by the rules ERT, PRTF1 and PRTF2 (see Section 4.1), respectively. The other permutations in the initial population are generated randomly. All the other steps of the DDE algorithm (mutation, crossover and selection operators) are implemented in the same way as presented in Zhou et al. (2016).

The DDE algorithm has three parameters: population size  $Q$ , mutation scale factor  $R$ , and crossover parameter  $W$ . As in Zhou et al. (2016) the population size is set to  $Q = 40$ .  $R$  and  $W$  were determined experimentally ( $R = 0.3$  and  $W = 0.07$ ).

## 5.2. Ant Colony Optimization (ACO) algorithm

We adapt the best ACO algorithm, proposed by Jia et al. (2017), to solve our problem. At each iteration of ACO algorithm,  $na$  ants are used to construct  $na$  solutions, that is, each ant builds up a solution. A constructive algorithm, named CoS, is used by each ant. This algorithm constructs a batch and immediately schedules the batch on a machine. The constructed batch is scheduled as the last batch on the machine with the smallest completion time. The algorithm CoS finishes when all the  $n$  jobs have been scheduled. Each built solution is improved by a local search algorithm. In our implementation of ACO algorithm, we do the following adaptations.

In the CoS algorithm, to form each batch, two types of candidate job lists are defined. In Jia et al. (2017), the first candidate job list  $CL_k^1$  is formed by the unscheduled jobs that satisfy the capacity constraint of machine  $k$ . In our implementation, to obtain the list  $CL_k^1$ , we use the current best solution found by the ACO algorithm. From this solution,  $CL_k^1$  is formed by the jobs processed on machine  $k$ . It is obvious that in a feasible solution, the jobs satisfy the capacity constraint of the machines. Initially, the current best solution is the solution determined by the greedy constructive heuristic described in Section 4.1.

After determining the first candidate job lists, as in Jia et al. (2017), the machine  $k$  with the smallest completion time and  $CL_k^1 \neq \emptyset$  is selected. Then a batch  $B_b$  to be scheduled on machine  $k$  is constructed.  $B_b$  is initialized with a job randomly selected from  $CL_k^1$ . The second candidate job list  $CL_{kb}^2$  is the set of jobs possibly added to the batch  $B_b$ , and is defined as follows:

$$CL_{kb} = \{j \in U \mid s_j \leq (Q_k - \sum_{i \in B_b} s_i) \text{ and } |R_b - r_j| < dt\}, \quad (11)$$

where  $U$  is the set of unscheduled jobs,  $B_b$  is the batch under construction,  $R_b$  is the actual release time of batch  $B_b$ , and  $dt$  is the maximum difference between the job and the batch release times. Note that,  $CL_{kb}^2$  consists of jobs  $j$  whose sizes are smaller than the residual space of the current batch  $B_b$  on machine  $k$  (i.e.,  $s_j \leq (Q_k - \sum_{i \in B_b} s_i)$ ), and jobs  $j$  with release times not much different than the actual batch release time ( $|R_b - r_j| < dt$ ). In Jia et al. (2017),  $|R_b - r_j| < dt$  is not considered. In the batch construction, to reduce the completion times of the batches, we try to reduce the number of batches and group jobs with similar release times. The use of the condition  $|R_b - r_j| < dt$  in the second candidate job list improves the solution quality of our problem considerably. All the other issues used in the CoS algorithm (e.g. defining the pheromone trails and heuristic information) are the same as presented in Jia et al. (2017).

Through computational experiments, we note that the local search procedure, named LO1, used in the original ACO algorithm does not perform well for our problem. Also, the local search used by our IG algorithm does not work well in the ACO algorithm. Therefore, we propose the following local search procedure to be used in the ACO algorithm. As in Jia et al. (2017), the minimum completion time  $C_{min}$  of all machines in the solution to be improved (current solution) is determined. The batches that complete after  $C_{min}$  are removed and put into a job set  $J'$ . A partial solution is determined. Then, the removed jobs in  $J'$  are reinserted by using the construction phase of the IG algorithm (see Section 4.2), and a new complete solution is obtained. If the current solution is improved, the process is repeated. Otherwise, the local search ends and returns the best found solution.

According to Jia et al. (2017) and other previous works (Jia et al., 2015), the values of the ACO parameters are set as:  $na = 20$  (number of ants)  $\rho = 0.5$  (the pheromone evaporation rate),  $\alpha = 1$  (the relative importance of pheromone trails),  $\beta = 1$  (the relative importance of heuristic information) and  $Q = n$  (used to update the pheromone trails). The parameter  $d$  used to compute the candidate job list  $CL_{kb}^2$  was calibrated. With  $dt = 0.6p_{jk}$  the best results were yielded, where  $p_{jk}$  is the processing time of job  $j$  on the selected machine  $k$  with the smallest completion time.

## 5.3. Simulated annealing

Damodaran and Vélez-Gallego (2012) proposed a SA algorithm in which neighbor solutions are generated by two mechanisms: job interchanges and job insertions. In the first mechanism, two jobs assigned to different batches in the current schedule are interchanged; in the second, one job is moved from one batch to another. While interchanging and exchanging jobs, the capacity of the machine should not be exceeded. The algorithm always attempts to perform a job interchange with probability  $\theta$ , and a job insertion with probability  $1 - \theta$ . For each value of the current temperature  $T_i$ , the algorithm evaluates  $\alpha * n$  neighbor solutions. The new temperature is calculated as  $T_{i+1} = \delta * T_i$ .

In this paper, this SA is applied to the problem under study with the following adaptations: (1) The initial solution is determined by the greedy constructive heuristic described in Section 4.1. (2) The job insertion mechanism is performed by the greedy insertion heuristic used in the construction phase of our IG algorithm. When we use this heuristic, new batches can be created and inserted in the best position of the schedule, obtaining a schedule with the smallest increment in the value of the total flow time. (3) The algorithm stops when a computation time limit is met. If the temperature reaches the zero value or a value close to zero, it is restarted as  $T_{i+1} = 0.5 * T_0$ , where  $T_0$  is the initial temperature.

As in Damodaran and Vélez-Gallego (2012), preliminary experiments were conducted in order to determine the values of the parameters  $\alpha$ ,  $\theta$ , and  $\delta$ . The SA algorithm with the parameter values  $\alpha = 0.6$ ,  $\theta = 0.7$  and  $\delta = 0.9$  yields better results.

## 6. Computational experiments and results

To test the performance of our IG algorithm, it is compared with the three benchmark meta-heuristic algorithms presented in Section 5: DDE (Zhou et al., 2016), ACO (Jia et al., 2017), and SA (Damodaran and Vélez-Gallego, 2012). The algorithms are evaluated on a benchmark set of 1440 problem instances (360 small-medium instances and 1080 large instances). For small-medium instances we compare the performance of IG with respect to the MIP model, presented in Section 3, solved by the commercial optimization solver IBM-ILOG CPLEX 12.6.2, which we will simply refer to as CPLEX.

To compare a group of algorithms or methods and measure the quality of the obtained solutions, we use the relative percentage deviation (RPD) over the best known solution obtained among all the methods. The RPD (response variable) is computed for each method and each instance according to the following Equation:

$$RPD(\%) = 100 \times \frac{TF_{method} - TF_{best}}{TF_{best}}, \quad (12)$$

where  $TF_{best}$  is the minimum total flow time obtained among all the compared methods and  $TF_{method}$  is the total flow time obtained with a given method.

All of the algorithms and the MIP model have been coded in C++ and the experimental results were obtained on a PC with an Intel Xeon 3.5 GHz processor and 64 GB RAM running Ubuntu Linux 14.04. The MIP model was implemented by using the Concert Technology of CPLEX solver.

To make a fair comparison, the stopping condition for all the meta-heuristic algorithms is set to a maximum CPU elapsed time equal to  $0.2 * n$  seconds (Arroyo and Leung, 2017a). Setting the time limit in this way allows more computation effort as the number of jobs increases. CPLEX solver is applied to solve the MIP model with a threshold CPU time of 3600 s for each small-medium instance. That is, if after the established time no optimal solution is obtained, the best current solution (upper bound) is returned by CPLEX. Only CPLEX solver was configured for multi-threaded execution, because this solver use exact methods that need a lot of computational resources.

In the following subsections, we first describe the random instance generation, then we present the parameter setting of our IG algorithm, we analyze the results obtained on small-medium and large instances and finally we present a time analysis of the compared algorithms.

### 6.1. Problem instances

We tested the performance of the algorithms on a set of benchmark instances proposed by Arroyo and Leung (2017a). Each instance consists of the number of jobs ( $n$ ), number of machines ( $m$ ), job sizes ( $s_j$ ), job release times ( $r_j$ ), machine capacities ( $Q_k$ ) and processing time matrix  $p_{jk}$ . The number of jobs is classified into two sets:  $n \in \{20, 30, 40, 50\}$  (small-medium instances), and  $n \in \{100, 150, 200, 250\}$  (large instances). The number of machines is set to be  $m \in \{2, 3\}$  (small-medium instances), and  $m \in \{3, 4, 5\}$  (large instances).

Three types of job sizes are considered. Small, large, and mixed job sizes are uniformly distributed in the ranges  $U[1, 15]$ ,  $U[15, 50]$  and  $U[1, 50]$ , respectively. These ranges of job sizes are denoted by S1, S2, and S3, respectively. Machines with different values of capacity are considered. For the two machine instances ( $m = 2$ ), the capacity of the machines is assumed to be  $Q_k \in \{30, 50\}$ ,  $k = 1, 2$ , that is,  $Q_1 = 30$  and  $Q_2 = 50$  (or  $Q_2 = 30$  and  $Q_1 = 50$ ). For instances with  $m = 3, 4$  and 5, the capacities of the machines are defined as  $Q_k \in \{30, 40, 50\}$ ,  $Q_k \in \{20, 30, 40, 50, 60\}$ , respectively.

Three types of job release times are considered. Specifically, the job release time,  $r_j$ , can be uniformly distributed in the ranges  $U[1, \rho P]$ , where  $\rho = 0.05, 0.1, 0.3$ , and  $P = \frac{\sum_{j=1}^n \sum_{k=1}^m p_{jk}}{m}$ . Small and large release time ranges are generated with  $\rho = 0.05$  and  $\rho = 0.3$ , respectively. The ranges of release times are denoted by R1, R2 and R3, respectively.

The processing times of each job  $j$  on each machine  $k$  ( $p_{jk}$ ) is defined as follows. First,  $m$  disjoint intervals,  $I_i = [a_i, b_i]$ , are defined, were  $a_1 = 10$ ,  $a_i = b_{i-1} + 10$  ( $i = 2, \dots, m$ ), and  $b_i = a_i + 20$  ( $i = 1, \dots, m$ ). For a job  $j$ , the  $m$  processing times ( $p_{jk}$ ,  $k = 1, \dots, m$ ) are uniformly distributed in  $m$  different intervals  $I_i$ , respectively. For each job, the intervals for the machines are selected randomly.

Combining the different factors and levels, there are 72 categories of small-medium instances and 108 categories of large instances. For each category of large instances, there are ten instances; and for each category of small-medium instances, we consider five instances. Therefore, a total of 1080 large instances and 360 small-medium instances are considered.

### 6.2. Calibration of the IG algorithm

In this subsection we first analyze the performance of constructive heuristic (used to determine the initial solution) with each of the three priority rules ERT, PRTF1 and PRTF2 (see Section 4.1). Next, we analyze the parameter  $dist$  used in the local search (LS) procedure.  $dist$  is the distance between the jobs to be exchanged on the same machine. Finally, we calibrate the parameters  $n_r$ ,  $prob$  and  $Nls$  of our IG algorithm.

In order to calibrate these methods, we generate a new set of 200 random instances (calibration instances) according to Section 6.1. We consider instances with 50, 100, 150 and 200 jobs. The results of all experiments done are analyzed by means of the non-parametric Kruskal-Wallis statistical technique using the RPD measure as response variable. In this statistical technique two hypotheses are tested. (i) Null hypothesis: the medians of all the compared methods are equal. (ii) Alternative hypothesis: the median of at least one method is different.

The results of the experiments with the constructive heuristic and the priority rules (ERT, PRTF1 and PRTF2) are shown in Fig. 2(a). This figure presents, for the three constructive heuristics and all the calibration instances, the means plot and Tukey honestly significant difference (HSD) confidence intervals with 95% confidence level from the statistical test. Overlapping intervals indicates that no statistically significant difference exists among the overlapped means. The results of this statistical experiment indicate that there is statistically significant difference between the obtained results at a 95% confidence level. As the obtained  $p$ -value is 0.05, the null hypothesis should be rejected. We can clearly see that there are statistically significant differences among the couple of rules ERT-PRTF1 and PRTF1-PRTF2. The rules ERT and PRTF2 are statistically equivalent. Since PRTF1 is statistically better

than the other rules, this rule will be used to determine the first ordered job list for the greedy constructive heuristic.

Now we present the results of the experiments with the parameter  $dist$  used in the LS procedure. For this parameter, the following values were tested:  $dist \in \{1, 2, 3, 4, 5, 6\}$ . So, we have six LS versions. In this analysis we include two other LS versions, named Rand1 and Rand2. In Rand1, the jobs to be exchanged are randomly chosen on the same machine, that is, parameter  $dist$  is not considered. In Rand2, the jobs to be exchanged are randomly chosen on any machine. Fig. 2(b) shows the means plot and Tukey's HSD intervals at 95% confidence level for all the LS versions. We can see that the LS versions which use  $dist = 3$ ,  $dist = 4$  and  $dist = 5$  are the better versions, and Rand1 and Rand2 are the worst LS versions. Although there is no statistically significant difference between  $dist = 3$ ,  $dist = 4$  and  $dist = 5$ , LS with  $dist = 3$  presents much smaller average RPD. Thus, in the next experiments LS with  $dist = 3$  will be used in the IG algorithm.

Finally, we present the experiments done to tune the parameters for the proposed IG algorithm. There are three parameters that affect the performance measurement of the algorithm: the number of jobs  $n_r$  to be removed of a solution, probability  $prob$  to accept a worse solution, and parameter  $Nls$  used to activate the running of the LS procedure. The parameter  $n_r$  is a percentage of the number of jobs ( $n_r = \lceil d * n \rceil$ , where  $0 < d < 1$ ),  $prob$  is a real number in  $[0, 1]$ , and  $Nls$  is a number of iterations. As in Arroyo and Leung (2017a), the stopping criterion of IG is the computational time:  $0.2 * n$  seconds.

We carry out a full factorial experiment with the parameters  $d$ ,  $prob$  and  $Nls$ . The parameter levels are determined based on the paper (Arroyo and Leung, 2017a) and on some preliminary tests. Each parameter is tested at the following levels:  $d \in \{0.05, 0.08, 0.1, 0.2\}$ ,  $prob \in \{0.0, 0.05, 0.1\}$ ,  $Nls \in \{1, 10, 50, 100, 150, 200\}$ . All combinations of values of the three parameters give 72 versions or configurations of IG algorithm. Each calibration instance was solved five times and the average total flow time value was reported. We measure solution quality by the RPD (determined by Eq. (12)) from the best known solution obtained among all the configurations of IG algorithm. The results of the experiment are analyzed by means of the non-parametric Kruskal-Wallis statistical technique. The results of this statistical experiment indicate that there is statistically significant difference between the obtained results at a 95% confidence level. Fig. 3(a) shows, for the 72 configurations of IG and all the calibration instances, the means plot and Tukey HSD confidence intervals with 95% confidence level from the statistical test. In order to have a clearer picture, Fig. 3(b) only shows the results of 30 configurations of IG which include the better configuration. In this figure, the configurations 25–30 correspond to  $d = 0.08$ ,  $prob = 0.05$  and  $Nls \in \{1, 10, 50, 100, 150, 200\}$ , that is, the parameters of configuration 25 are  $d = 0.08$ ,  $prob = 0.05$  and  $Nls = 1$ , and the parameters of configuration 30 are  $d = 0.08$ ,  $prob = 0.05$  and  $Nls = 200$ . We can clearly see that the variation of parameter  $Nls$  notoriously affects the behavior of the IG algorithm. Generally, the worst results are obtained when the LS procedure is run at each iteration (i.e.,  $Nls = 1$ ). For  $Nls = 10, 50$  and  $100$ , the results are progressively improved (for example, see the configurations 26 to 28, or 32 to 34). For  $Nls = 150$  and  $200$ , the results get worse. We can see that the configurations 34 and 52 present the better results. These configurations correspond to the parameters values ( $d = 0.08$ ,  $prob = 0.1$ ,  $Nls = 100$ ) and ( $d = 0.1$ ,  $prob = 0.1$  and  $Nls = 100$ ), respectively. Although there is no statistically significant difference between these two configurations, the second presents the smallest average RPD. Generally, the group of configurations 49–54 generates better results than the group 31–36. Thus,  $d = 0.1$ ,  $prob = 0.1$  and  $Nls = 100$  will be used for IG algorithm in the next experiments. In the next two subsections, we will test the IG algorithm without the LS procedure, that is, when the LS procedure is not run.

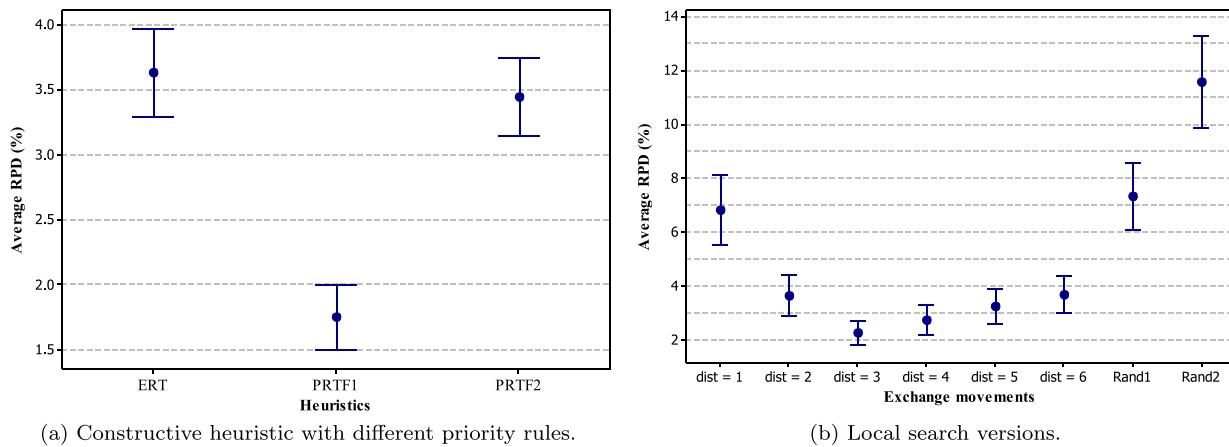


Fig. 2. Mean plot and confidence intervals at 95% confidence.

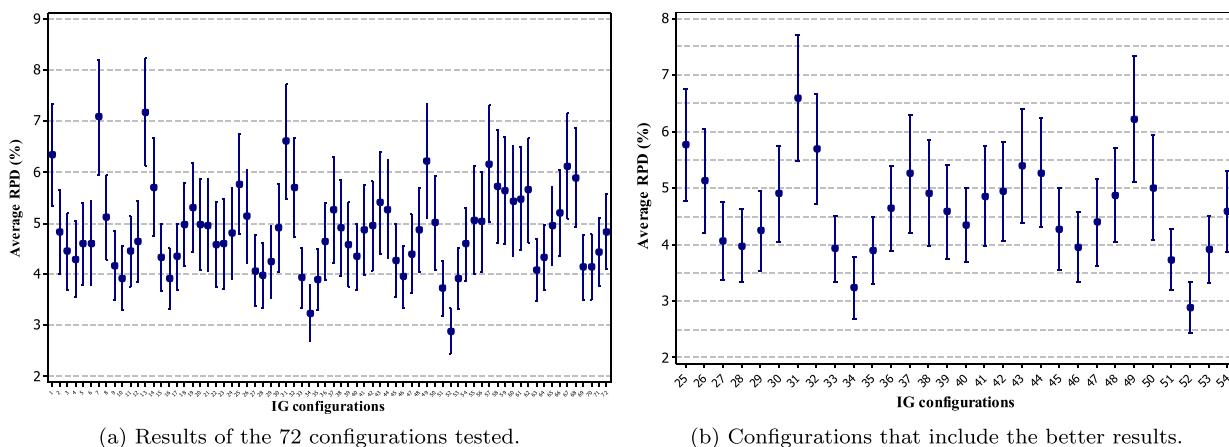


Fig. 3. Mean plot and confidence intervals at 95% confidence level for IG calibration experiment.

### 6.3. Experimental results on small-medium size instances

Since we use the same instances generated by Arroyo and Leung (2017a), the results presented in this and next subsection are organized in similar way as in Arroyo and Leung (2017a).

In this subsection we compare the meta-heuristic algorithms, IG, DDE, ACO and SA, against the CPLEX solver on small-medium instances. Since meta-heuristics are stochastic algorithms, each instance is solved 10 times by each algorithm. In the comparison analysis we consider the average and the best solution for each instance. For a total of 360 small-medium instances, the CPLEX solver was not able to find the optimal solution for any instance after running for one hour. Therefore, the best solution found by CPLEX in one hour of CPU time is applied for comparison. The quality of the solutions determined by the meta-heuristic algorithms and CPLEX is measured by the RPD from the best known solution obtained among all the methods. The RPD is computed for each instance according to Eq. (12).

Table 3 presents the results for  $n = 20$  and  $30$ , and Table 4 presents the results for  $n = 40$  and  $50$ . The average RPDs are grouped by the number of jobs ( $n$ ), number of machines ( $m$ ), and combination of job sizes and job release time ranges (SiRi),  $i = 1, 2, 3$ . The values reported in the tables are the average RPDs over 5 instances. Since each algorithm is run 10 times, for each meta-heuristic algorithm, the Best and Average (Avg) results are reported. That is, two RPD values are computed for each algorithm. The two RPDs are computed by considering the minimum and the average total flow time obtained in 10 runs of the algorithm, respectively. In Tables 3 and 4, we can see that in most groups of instances IG presents the best RPD values. For set of instances with

$n = 20$ , the algorithms are competitive, that is, they present small RPD values. However, for this set of instances SA gives the worst results. For set of instances with  $n = 30$ , the best results are obtained by IG, followed by ACO and DDE. SA and CPLEX presents the worst results. We can note that, for sets of instances with  $n = 40$  and  $n = 50$ , IG maintains its performance, the performance of SA improves, and the performance of DDE worsens. We also can note that the performance of CPLEX worsens significantly when the number of jobs is increased. Generally, CPLEX and the meta-heuristics resent high RPD values for instances with small job sizes (S1) and mixed job sizes (S3), respectively.

To better analyze the results presented in Tables 3 and 4, the Avg results are shown in Figs. 4 and 5, respectively. In these figures, the Avg results of the meta-heuristics algorithms are compared against the best result of CPLEX. Each group of five bars represents the average RPD of the five methods (CPLEX, DDE, ACO, SA and IG) over 10 instances for each category of instances SiRi,  $i = 1, 2, 3$ . Short vertical bars represent good quality solutions. We can see that CPLEX is worse than the meta-heuristics for category S1 of instances (small job sizes and any type of job release times) and for category S3R3 (mixed job sizes and large job release times). For sets of instances with  $n = 30$  and  $n = 40$ , generally CPLEX is better than the meta-heuristics for categories S2R1, S2R2, S2R3, S3R1 and S3R2. Also, ACO obtains better results than IG for instances with  $n = 30$  and categories S2R1, S2R2, S3R1 and S3R2. For all sets of instances with  $n = 50$ , the best results are obtained by IG followed by SA, and DDE and ACO present the worst results.

In order to validate the obtained results for small-medium instances, we apply a statistical analysis using the RPD measure as response variable. In this analysis we include two methods, the deterministic

**Table 3**Average RPDs (%) for instances with  $n = 20$  and  $n = 30$ .

m		$n = 20$								$n = 30$											
		CPLEX		DDE		ACO		SA		IG		CPLEX		DDE		ACO		SA		IG	
		Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg		
2	S1R1	0.23	0.00	0.02	0.00	0.14	0.20	0.26	0.00	0.00	3.06	0.60	1.43	0.19	0.86	0.00	0.05	0.00	0.46		
	S1R2	0.00	0.00	0.07	0.00	0.02	0.00	0.07	0.00	0.00	2.87	0.00	0.47	0.00	0.20	0.00	0.00	0.00	0.08		
	S1R3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.19	0.00	0.72	0.00	0.29	0.23	0.27	0.00	0.18		
	S2R1	0.00	0.00	0.02	0.04	0.23	0.37	0.59	0.00	0.01	0.03	0.83	1.41	0.48	1.05	2.65	3.60	0.02	1.64		
	S2R2	0.24	0.00	0.20	0.09	0.88	1.27	2.05	0.00	0.06	0.59	1.55	2.23	0.64	1.37	2.68	4.09	0.34	1.53		
	S2R3	0.00	0.05	0.97	0.52	1.82	2.93	3.57	0.00	0.48	0.45	3.38	6.08	2.59	4.54	0.56	1.20	0.00	0.71		
	S3R1	0.00	0.00	0.11	0.12	1.50	2.17	3.97	0.00	0.06	0.70	1.07	2.13	0.65	1.37	3.00	5.21	0.00	2.04		
	S3R2	0.00	0.00	0.17	0.00	2.07	3.59	4.90	0.00	0.08	1.18	1.77	3.37	0.53	2.03	3.58	4.76	0.00	2.09		
	S3R3	0.00	0.00	0.23	0.29	0.70	0.67	1.34	0.00	0.09	3.74	3.42	6.61	1.41	4.18	1.70	2.91	0.00	1.25		
	Average	0.15	0.00	0.11	0.08	0.67	0.98	1.52	0.00	0.05	2.03	0.96	2.19	0.46	1.33	1.69	2.48	0.10	1.14		

**Table 4**Average RPDs (%) for instances with  $n = 40$  and  $n = 50$ .

m		$n = 40$								$n = 50$											
		CPLEX		DDE		ACO		SA		IG		CPLEX		DDE		ACO		SA		IG	
		Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg		
2	S1R1	18.9	3.4	6.3	3.0	4.9	0.6	2.0	0.0	0.5	33.4	13.3	16.6	11.4	14.4	1.4	2.6	0.0	1.7		
	S1R2	19.5	2.4	5.1	2.3	4.0	0.2	0.7	0.0	0.2	47.5	9.5	13.6	8.5	12.2	0.0	2.0	0.0	1.7		
	S1R3	5.5	0.3	1.3	0.5	1.2	0.0	0.0	0.0	0.0	91.8	1.8	3.3	2.5	3.3	0.0	0.0	0.0	0.0		
	S2R1	1.8	3.0	4.2	2.1	3.1	2.4	3.2	0.1	1.6	2.8	4.4	6.2	3.5	4.6	1.4	2.6	0.0	1.4		
	S2R2	2.2	4.1	5.8	2.9	4.3	1.3	1.8	0.2	1.3	3.6	7.9	10.2	6.4	8.4	2.1	3.7	0.4	2.0		
	S2R3	1.9	10.4	16.9	9.8	13.8	2.3	4.0	0.0	2.4	11.6	16.7	21.7	15.3	18.6	1.0	2.3	0.1	1.9		
	S3R1	1.1	3.2	5.4	2.0	3.5	5.4	6.8	0.6	3.4	6.5	8.8	11.5	7.3	9.5	3.0	4.0	0.0	2.1		
	S3R2	3.7	6.5	8.5	4.6	6.2	2.8	6.1	0.0	2.4	6.9	14.1	18.7	11.6	15.7	5.5	7.5	0.0	3.5		
	S3R3	6.8	13.5	17.4	11.2	14.9	2.7	4.1	0.0	2.2	15.4	18.2	25.2	18.4	24.1	3.9	5.0	0.0	3.2		
	Average	8.4	4.8	7.5	3.7	5.8	2.3	3.4	0.1	1.7	32.2	9.8	13.2	8.6	11.4	2.4	3.5	0.0	1.9		

heuristic, called FF\_PRTF1, used by the meta-heuristic algorithms to generate initial solutions, and the version of our IG algorithm which does not use the Local Search (LS) procedure, called IG\_nLS. The goal is to analyze the improvement of the meta-heuristics in relation to the heuristic FF\_PRTF1, and show the effectiveness of the LS procedure used in our IG algorithm. For the meta-heuristic algorithms we consider the Avg results in this statistical analysis. The result of the analysis is displayed in Fig. 6. This figure shows the means plot and Tukey's HSD intervals at 95% confidence level. From the perspective of performance, for small-medium instances, IG is statistically the best method, followed by SA. ACO, IG\_nLS and DDE are statistically equivalent, but ACO present the smallest average RPD. CPLEX and FF\_PRTF1 are the worst methods. As can be seen, all the meta-heuristic algorithm improve

significantly over the constructive heuristic FF\_PRTF1, and the LS procedure improves significantly the IG results.

#### 6.4. Experimental results on large size instances

In this subsection we present the comparison of solution quality generated by our IG algorithm and the benchmark algorithms SA, ACO and DDE. The performance of each algorithm is also measured by RPD determined by Eq. (12), where  $FT_{best}$  is the best known solution (total flow time) obtained among all the algorithms. All the 1080 large instances are solved 10 times by each algorithm. We consider the best known solution (reference solution) obtained among all the algorithms

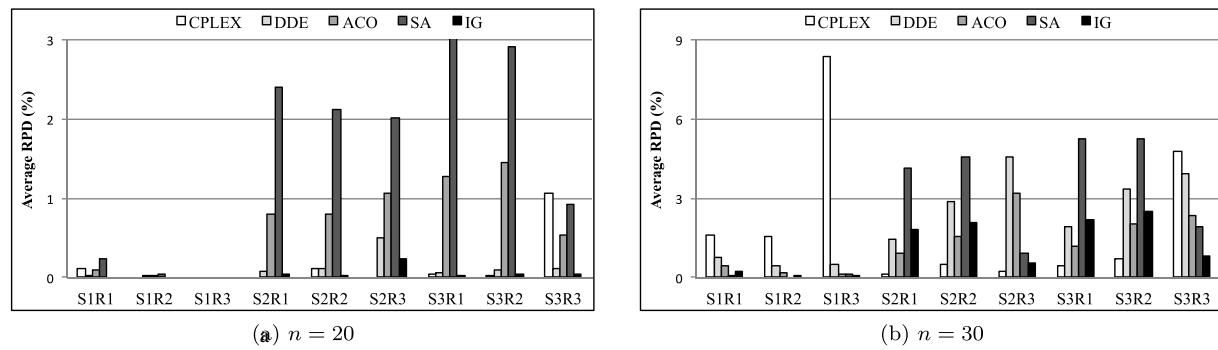


Fig. 4. Comparison of solution quality on instances with 20 and 30 jobs.

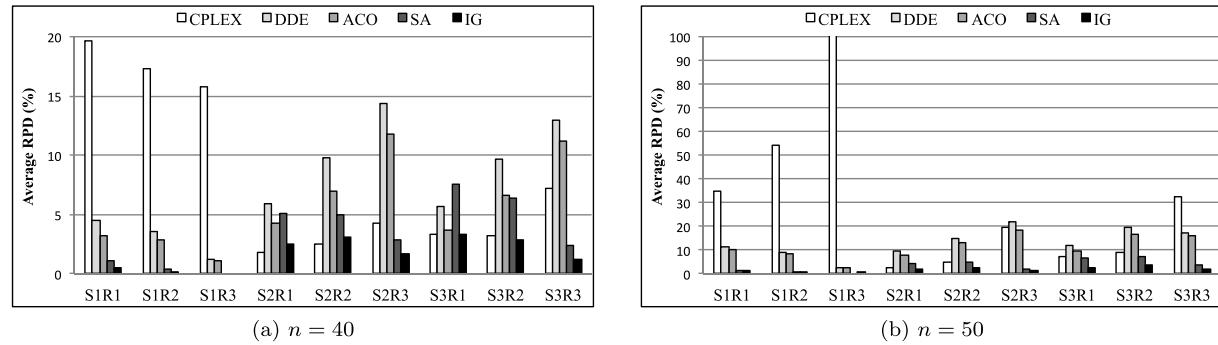


Fig. 5. Comparison of solution quality on instances with 40 and 50 jobs.

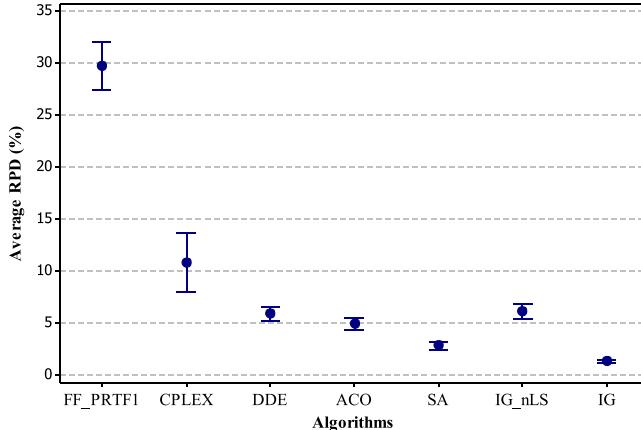


Fig. 6. Mean plot and confidence intervals at 95% confidence level for all the methods and all small-medium instances.

and we use the best and average solution generated by each algorithm to compute the RPD in relation to the best known solution (reference solution) obtained among all the algorithms.

The obtained numerical results (RPDs) are reported in Tables 5 and 6. The average RPDs are grouped by the number of jobs ( $n$ ), number of machines ( $m$ ), and categories SiRi,  $i = 1, 2, 3$ . For each group, 10 instances are tested, and the average RPDs over these 10 instances are presented. For each algorithm the best (*Best*) and average (*Avg*) results are presented. We can clearly observe that IG outperforms the other algorithms, that is, for all groups of large instances (except for group  $n = 200$ ,  $m = 4$ , S2R1), the *Best* and *Avg* values of IG are better than the respective values of the other algorithms. By considering the overall average of the algorithms, SA is better than ACO which in turn is better than DDE. For instances with 200 jobs the algorithms IG, SA, ACO and DDE have overall average RPDs (*Avg* value) of 2.08%,

5.92%, 7.68% and 36.93%, respectively. Also, these algorithm present overall *Best* RPDs of 0.10%, 3.66%, 5.50% and 32.16%, respectively. It can be noted that IG, SA and ACO are much better than DDE. It is possible to see that all the algorithms have low RPD values for groups of instances S1R1, S1R2 and S1R3. That is, for instances with small job sizes, independent of job release times, the solutions found by the meta-heuristics are near the reference solution. All the used instances and the best and average solutions obtained by the algorithms are available at [www.dpi.ufv.br/projetos/scheduling/upbpmFT.htm](http://www.dpi.ufv.br/projetos/scheduling/upbpmFT.htm).

The *Avg* results presented in Tables 5 and 6 are summarized in Figs. 7 and 8, respectively. Each group of four bars represents the average RPD of the four algorithms over 30 instances, for each category of instances SiRi,  $i = 1, 2, 3$ . We can see that for all groups of instances IG presents the shortest vertical bars (low RPDs). The tallest vertical bars (large RPDs) are given by DDE in most groups of instances. We also can see that SA is better than ACO in most groups of instances. All the algorithms present low RPDs for groups of instances S1R1, S1R2 and S1R3. As can be observed, for some group of instances (for example, S1R1, S1R2 and S1R3), when the job release times are increased, the performances of the algorithms improve. This behavior possibly happens because for large job release times, generally, there are machine idle times, several batches are processed with capacity slack, and batch start times coincide with batch release times.

To validate the statistical significance of the observed differences in solution quality, the experimental results are analyzed by means of the non-parametric Kruskal-Wallis statistical technique using the RPD measure as response variable. In this analysis we consider the *Avg* results and also include for comparison the deterministic heuristic FF\_PRTF1 and IG without the LS procedure (IG\_nLS). Fig. 9(a) shows the means plot and Tukey's HSD intervals at 95% confidence level obtained from the statistical analysis on all the large instances. The confidence interval for the means of IG and other algorithms do not overlap, that confirms the statistical significance of the difference between IG and other three algorithms. This means that the proposed IG is statistically better than all other algorithms and by a significant margin. The second

**Table 5**Average RPDs (%) for instances with  $n = 100$  and  $n = 150$ .

m		$n = 100$								$n = 150$							
		DDE		ACO		SA		IG		DDE		ACO		SA		IG	
		Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
3	S1R1	8.94	12.15	6.72	9.15	1.73	3.42	0.00	0.50	15.24	19.56	12.51	14.91	4.51	6.67	0.00	0.89
	S1R2	2.85	3.99	3.55	4.81	0.15	0.57	0.00	0.06	5.56	7.11	5.46	7.20	0.73	1.34	0.00	0.11
	S1R3	0.51	1.08	1.72	2.01	0.02	0.08	0.00	0.08	0.86	0.97	1.65	1.99	0.03	0.08	0.00	0.02
	S2R1	21.65	26.20	8.57	11.01	5.87	7.42	0.00	1.81	35.32	39.86	6.80	9.24	3.32	6.32	0.14	2.57
	S2R2	26.56	33.11	3.69	6.33	2.12	3.76	0.02	2.66	38.59	47.12	3.00	5.50	2.02	5.50	0.11	3.37
	S2R3	19.32	25.41	3.67	5.41	1.65	3.25	0.00	1.30	28.86	34.17	3.29	5.29	3.83	6.19	0.00	1.12
	S3R1	25.76	32.09	16.19	20.05	7.98	10.55	0.00	2.99	44.78	52.14	23.11	26.30	8.38	11.10	0.00	2.42
	S3R2	30.40	39.28	11.59	15.78	6.04	9.32	0.00	3.18	52.74	61.68	15.72	19.59	6.75	10.02	0.00	4.19
	S3R3	9.64	14.01	5.36	7.69	0.58	1.65	0.34	1.21	12.44	16.69	4.37	6.13	0.69	2.02	0.10	1.22
4	S1R1	3.65	6.23	3.15	4.42	0.29	1.01	0.01	0.14	8.49	11.58	5.94	7.14	1.49	2.64	0.00	0.41
	S1R2	1.81	2.88	3.24	3.94	0.00	0.13	0.15	0.18	2.56	3.76	2.46	3.30	0.13	0.43	0.00	0.07
	S1R3	0.24	0.40	0.88	0.95	0.01	0.08	0.02	0.05	0.45	0.50	0.61	0.76	0.01	0.04	0.00	0.01
	S2R1	21.32	25.53	4.03	6.14	4.75	6.29	0.14	2.00	31.72	36.41	3.43	4.93	2.99	5.41	0.23	2.60
	S2R2	26.35	30.81	1.38	2.97	2.86	4.73	0.58	4.15	41.46	46.65	0.58	2.08	4.89	7.66	0.00	5.12
	S2R3	20.06	24.96	0.66	1.86	1.81	3.55	0.16	1.54	27.91	32.14	0.72	2.30	6.47	9.60	0.08	1.71
	S3R1	27.23	31.93	11.35	13.91	8.83	10.57	0.00	3.08	42.21	48.23	10.87	13.99	6.96	10.28	0.00	2.37
	S3R2	25.58	31.16	3.44	5.98	2.81	5.51	0.00	2.89	43.01	50.07	2.99	5.88	3.47	5.85	0.27	3.34
	S3R3	9.00	13.03	2.05	3.11	0.34	1.63	0.28	0.96	15.48	18.50	2.11	3.13	1.30	2.98	0.24	1.27
5	S1R1	2.21	3.55	1.86	2.53	0.09	0.49	0.04	0.11	2.94	4.14	3.40	4.22	0.19	0.65	0.00	0.06
	S1R2	0.53	0.79	1.03	1.26	0.01	0.05	0.02	0.03	1.27	1.63	1.57	1.95	0.02	0.11	0.00	0.00
	S1R3	0.00	0.06	0.39	0.39	0.00	0.00	0.00	0.00	0.30	0.42	0.15	0.16	0.00	0.02	0.00	0.01
	S2R1	35.20	43.16	5.06	7.53	4.41	7.71	0.03	3.12	54.49	65.20	5.20	8.05	3.40	6.32	0.00	3.75
	S2R2	33.38	42.93	2.25	4.41	1.47	4.51	0.23	4.09	63.47	79.01	1.81	4.38	5.18	11.47	0.11	4.48
	S2R3	2.63	4.21	0.69	1.03	0.25	0.94	0.26	0.55	5.37	6.41	0.42	1.03	1.02	1.86	0.14	0.51
	S3R1	32.34	41.26	12.60	16.25	8.21	11.51	0.00	3.53	55.44	64.94	9.27	13.05	3.86	7.28	0.00	2.81
	S3R2	22.78	30.25	2.62	6.03	1.17	3.31	0.25	2.93	35.90	45.29	4.86	8.01	1.75	4.92	0.58	2.72
	S3R3	1.75	3.00	0.38	0.69	0.06	0.32	0.09	0.18	4.49	5.42	0.85	1.41	0.27	1.06	0.11	0.45
Average		15.25	19.39	4.37	6.13	2.35	3.79	0.10	1.60	24.86	29.61	4.93	6.74	2.73	4.73	0.08	1.76

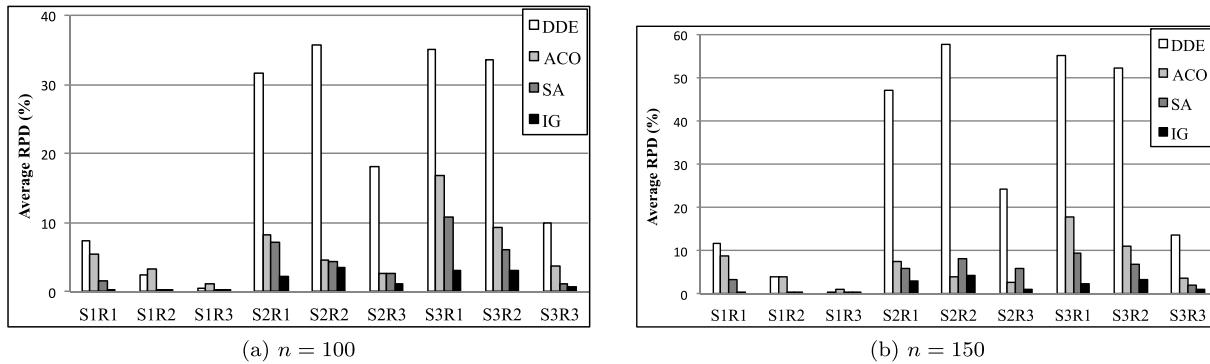


Fig. 7. Comparison of solution quality on instances with 100 and 150 jobs.

best algorithm is SA. ACO and IG\_nLS are statistically equivalent, and DDE is the worst meta-heuristic algorithm. We also can clearly see that the heuristic FF\_PRTF1 is improved by all the meta-heuristic algorithms, and the LS procedure contributes significantly on IG improvement. In order to do a better comparison between SA, ACO and IG\_nLS, the algorithms DDE and FF\_PRTF1 are removed for subsequent analyses and we carry out another statistical test considering only SA, ACO, IG\_nLS and IGA. In Fig. 9(b) we can clearly see that SA is statistically better than ACO and IG\_nLS, and between these last two algorithms there is a statistically significant difference.

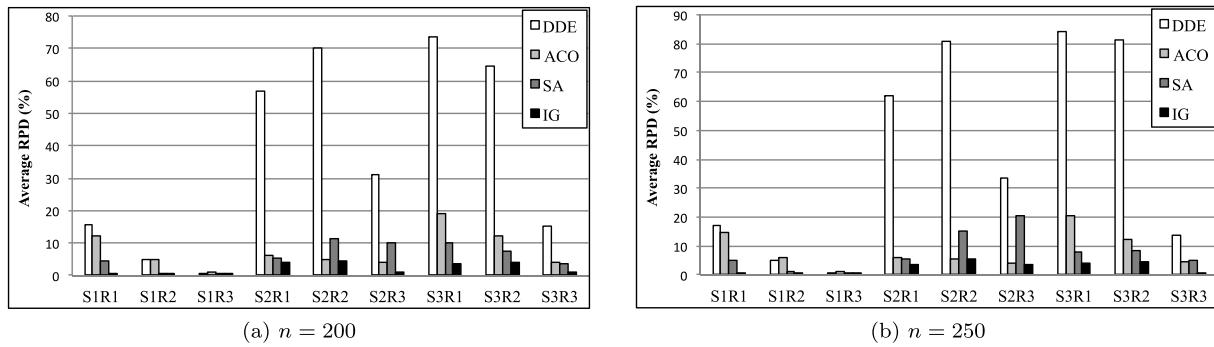
The results obtained by the algorithms are not compared with the lower bound for the total flow time defined by Eq. (1) because it is rather weak.

### 6.5. Time analysis

The experiments presented in previous subsections were done considering solution quality of the meta-heuristic algorithms. To give additional information concerning the computational effort required by the algorithms, we use the time-to-target (TTT) plot analysis (Aixé et al., 2007). A TTT plot is generated by executing an algorithm  $q$  times and measuring the computational time required to reach a solution at least as good as a target solution versus the required CPU time. The running times are sorted in increasing order. The  $i$ th running time  $t_i$  is associated with a probability  $p_i = (i - 0.5)/q$  and the points  $(t_i, p_i)$ ,  $i = 1, \dots, q$ , are plotted. Each plotted point indicates the probability (vertical axis) for the algorithm to achieve the target solution in a given running time

**Table 6**Average RPDs (%) for instances with  $n = 200$  and  $n = 250$ .

m		$n = 200$								$n = 250$							
		DDE		ACO		SA		IG		DDE		ACO		SA		IG	
		Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
3	S1R1	22.82	26.47	17.10	20.68	6.03	8.71	0.00	0.96	25.44	30.01	21.45	25.13	6.69	9.79	0.00	1.05
	S1R2	6.97	7.76	7.07	8.89	0.59	1.53	0.02	0.26	7.01	7.69	8.34	9.59	1.07	1.65	0.00	0.18
	S1R3	1.11	1.31	1.70	1.98	0.01	0.15	0.01	0.03	1.56	1.56	1.46	1.81	0.07	0.22	0.01	0.04
	S2R1	42.46	47.91	8.38	10.87	4.60	6.19	0.09	2.62	44.64	49.42	6.98	9.68	2.58	5.20	0.16	2.84
	S2R2	53.38	59.07	4.90	7.37	4.42	7.62	0.00	3.43	52.35	58.38	3.63	6.72	3.17	5.86	0.07	2.14
	S2R3	38.18	44.85	4.51	7.33	7.53	12.39	0.00	1.26	43.17	50.13	5.38	8.66	15.70	30.96	0.06	8.76
	S3R1	60.94	69.68	25.82	31.63	8.54	11.53	0.00	3.72	67.22	75.77	27.22	33.13	4.69	7.95	0.15	3.28
	S3R2	69.56	79.63	16.28	21.19	5.90	8.53	0.00	5.05	75.48	86.01	14.81	20.25	3.94	5.81	0.00	4.91
	S3R3	19.17	22.37	6.37	8.16	2.59	4.77	0.00	1.31	18.44	19.71	7.21	9.39	2.39	5.96	0.00	0.84
4	S1R1	13.44	15.56	8.45	10.98	2.61	4.10	0.00	0.42	12.81	14.25	10.46	12.53	2.62	3.87	0.00	0.48
	S1R2	3.91	4.72	3.42	4.08	0.27	0.67	0.00	0.09	4.48	4.59	4.63	5.50	0.45	0.96	0.00	0.13
	S1R3	0.41	0.44	0.73	0.89	0.02	0.08	0.04	0.07	0.54	0.54	0.88	1.01	0.06	0.16	0.00	0.04
	S2R1	41.71	45.82	1.71	3.10	1.34	3.50	0.50	4.38	47.05	50.94	0.92	2.98	1.70	4.16	0.20	3.54
	S2R2	50.80	57.37	1.01	3.25	6.69	10.39	0.18	4.79	58.84	65.02	1.70	4.80	10.24	13.34	0.13	5.68
	S2R3	35.35	40.46	1.01	3.06	9.10	14.60	0.00	1.18	39.34	42.83	0.65	2.22	16.19	27.34	0.01	0.98
	S3R1	54.89	62.47	9.92	13.47	6.80	9.05	0.00	2.64	68.55	75.08	8.36	12.43	4.21	7.19	0.00	2.62
	S3R2	56.18	66.06	3.71	7.30	3.54	6.44	0.11	4.22	64.28	72.92	2.67	6.26	2.61	6.70	0.61	4.87
	S3R3	17.95	19.91	2.13	3.17	3.59	5.51	0.00	1.05	17.49	18.75	2.21	3.26	4.59	7.25	0.00	0.76
5	S1R1	4.34	5.15	3.87	4.88	0.33	0.90	0.01	0.12	6.28	6.65	4.80	5.83	0.61	1.33	0.00	0.17
	S1R2	1.66	1.86	1.52	1.90	0.03	0.16	0.00	0.02	2.47	2.48	2.54	2.95	0.09	0.32	0.00	0.05
	S1R3	0.20	0.22	0.39	0.44	0.01	0.02	0.00	0.00	0.41	0.41	0.24	0.36	0.03	0.04	0.00	0.01
	S2R1	68.48	76.49	2.34	5.12	2.06	6.17	1.18	5.62	79.16	85.95	2.62	5.30	2.77	6.63	0.07	3.94
	S2R2	79.58	93.40	1.59	4.45	11.43	16.67	0.14	4.68	104.07	118.96	1.23	5.74	16.20	25.72	0.05	8.08
	S2R3	7.06	7.65	0.97	1.55	1.20	2.77	0.09	0.37	6.78	7.38	0.55	1.30	1.74	3.32	0.05	0.47
	S3R1	75.97	88.75	8.54	11.90	5.46	9.08	0.15	5.14	89.10	101.16	10.89	15.86	5.20	8.17	0.03	5.74
	S3R2	38.25	47.85	4.30	8.25	3.46	7.06	0.00	2.47	70.10	85.36	6.19	10.87	6.25	13.00	0.00	3.60
	S3R3	3.64	3.89	0.86	1.50	0.68	1.27	0.07	0.31	3.18	3.23	0.74	1.28	0.92	1.90	0.01	0.28
Average		32.16	36.93	5.50	7.68	3.66	5.92	0.10	2.08	37.42	42.04	5.88	8.33	4.33	7.59	0.06	2.43

**Fig. 8.** Comparison of solution quality on instances with 200 and 250 jobs.

(horizontal axis). The more to the left is the plot, the better is the corresponding algorithm.

For this analysis we use two instances where all the algorithms IG, SA, IG\_nLS, ACO and DDE found the best known solution (target solution) at least one time. We perform  $q = 120$  runs for each algorithm varying the random number generator seed. The algorithms were made to stop whenever a solution better than or equal to the target solution was found or a maximum computational time of  $0.5 * n$  seconds was reached. Figs. 10(a) and 10(b) displays the TTT plots of the algorithms for two instances with  $n = 100$  and  $n = 200$ , respectively. For the two instances, we can clearly see that the IG algorithm finds solutions as good as the target solution faster than the other algorithms. In Fig. 10(a), we can see that the probabilities of algorithms IG, SA, IG\_nLS, ACO and DDE finding the target solution in 5 s is about 100%, 95%, 85%, 82%

and 0%, respectively. For the first instance, 1.4 s run time is enough for IG to ensure a 100% probability of obtaining the target solution. For the DDE algorithm, 23 s run time is necessary to ensure a 90% probability of obtaining the target solution.

In Fig. 10(b) we can see that the probabilities of IG, SA, IG\_nLS, ACO and DDE finding the target solution in 15 s is about 95%, 73%, 56%, 30% and 0%, respectively. For the DDE algorithm, 50 s run time is necessary to ensure a 90% probability of obtaining the target solution.

This analysis shows that the algorithm IG converges to good solutions with the shortest CPU time. Furthermore, SA converges slightly faster than G\_nLS and ACO, and DDE needs much more time to find good solutions.

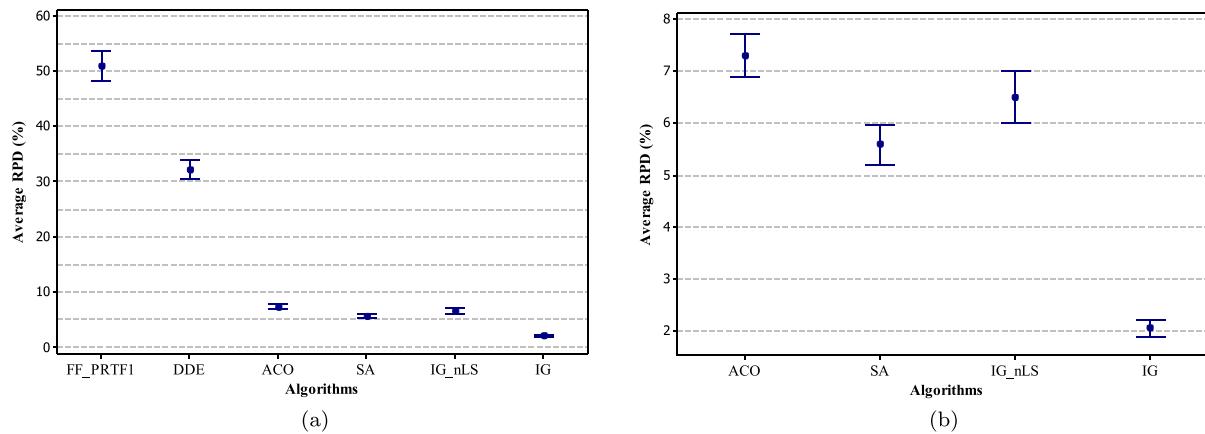
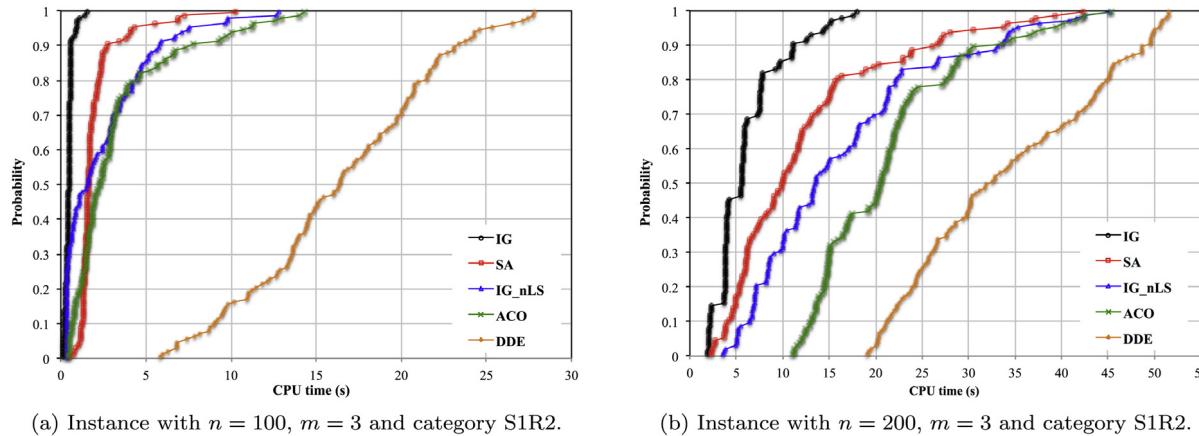


Fig. 9. Mean plot and confidence intervals at 95% confidence level for all algorithms and all large instances.



(a) Instance with  $n = 100, m = 3$  and category S1R2.

(b) Instance with  $n = 200, m = 3$  and category S1R2.

Fig. 10. Time-to-target plots of algorithms IG, SA, IG\_nLS, ACO and DDE.

## 7. Conclusions

In this paper, we investigate a p-batch scheduling problem on unrelated parallel BPMs, that includes jobs with arbitrary job sizes, unequal job release times and arbitrary machine capacities. The objective is to minimize the total flow time. To the best of our knowledge, this problem has not been addressed so far. The main contributions of this work are: (1) a MIP model has been provided for the new problem, (2) an effective Iterated Greedy (IG) algorithm was proposed to solve the problem, (3) new priority rules were tested to build reasonable quality solutions, and a new local search heuristic was proposed to improve solutions, (4) we tested the applicability of other benchmark meta-heuristic algorithms to the problem under study.

A comprehensive set of small-medium and large instances, with different categories of job sizes and job release times, has been employed in order to analyze the performance of the algorithms. For small-medium instances, the algorithms are compared with the exact MIP model which is solved by CPLEX solver. Computational experiments demonstrated the effectiveness of our IG algorithm. It outperforms the other meta-heuristics consistently. The obtained results have been statistically tested. Therefore, these results indicate that the IG meta-heuristic is an attractive alternative for solving p-batch scheduling problems.

Future research is to extend our approach for solving other performance criterion such as total tardiness, and consider other problem characteristics such as setup times and incompatible job families.

## Acknowledgments

The authors would thank the anonymous referees whose suggestions have greatly improved the readability of the paper. The first author

thanks the financial support of CNPq and FAPEMIG, Brazilian research agencies.

## References

- Abedi, M., Seidgar, H., Fazlollahtabar, H., Bijani, R., 2015. Bi-objective optimisation for scheduling the identical parallel batch-processing machines with arbitrary job sizes, unequal job release times and capacity limits. *Int. J. Prod. Res.* 53 (6), 1680–1711.
- Alex, M., Resende, M.G., Ribeiro, C.C., 2007. TTT plots: a perl program to create time-to-target plots. *Optim. Lett.* 1 (4), 355–366.
- Arroyo, J.E.C., Armentano, V.A., 2005. Genetic local search for multi-objective flowshop scheduling problems. *European J. Oper. Res.* 167 (3), 717–738.
- Arroyo, J.E.C., Leung, J.Y.-T., 2017a. An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. *Comput. Ind. Eng.* 105, 84–100.
- Arroyo, J.E.C., Leung, J.Y.-T., 2017b. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. *Comput. Oper. Res.* 78, 117–128.
- Benavides, A.J., Ritt, M., 2016. Two simple and effective heuristics for minimizing the makespan in non-permutation flow shops. *Comput. Oper. Res.* 66, 160–169.
- Bilyk, A., Mönch, L., Almeder, C., 2014. Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. *Comput. Ind. Eng.* 78, 175–185.
- Chang, P.-Y., Damodaran, P., Melouk, S., 2004. Minimizing makespan on parallel batch processing machines. *Int. J. Prod. Res.* 42 (19), 4211–4220.
- Cheng, B., Wang, Q., Yang, S., Hu, X., 2013. An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes. *Appl. Soft Comput.* 13 (2), 765–772.
- Cheng, B., Yang, S., Hu, X., Chen, B., 2012. Minimizing makespan and total completion time for parallel batch processing machines with non-identical job sizes. *Appl. Math. Model.* 36 (7), 3161–3167.
- Chiang, T.-C., Cheng, H.-C., Fu, L.-C., 2010. A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrival. *Comput. Oper. Res.* 37 (12), 2257–2269.

- Chu, C., 1992. Efficient heuristics to minimize total flow time with release dates. *Oper. Res. Lett.* 12 (5), 321–330.
- Chung, S., Tai, Y., Pearn, W., 2009. Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes. *Int. J. Prod. Res.* 47 (18), 5109–5128.
- Damodaran, P., Chang, P.-Y., 2008. Heuristics to minimize makespan of parallel batch processing machines. *Int. J. Adv. Manuf. Technol.* 37 (9–10), 1005–1013.
- Damodaran, P., Diyadawagamage, D.A., Ghrayeb, O., Vélez-Gallego, M.C., 2012. A particle swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines. *Int. J. Adv. Manuf. Technol.* 58 (9–12), 1131–1140.
- Damodaran, P., Vélez-Gallego, M.C., 2010. Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times. *Int. J. Adv. Manuf. Technol.* 49 (9–12), 1119–1128.
- Damodaran, P., Vélez-Gallego, M.C., 2012. A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Syst. Appl.* 39 (1), 1451–1458.
- Damodaran, P., Vélez-Gallego, M.C., Maya, J., 2011. A GRASP approach for makespan minimization on parallel batch processing machines. *J. Intell. Manuf.* 22 (5), 767–777.
- Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-completeness. W. H. Freeman, San Francisco.
- Gokhale, R., Mathirajan, M., 2014. Minimizing total weighted tardiness on heterogeneous batch processors with incompatible job families. *Int. J. Adv. Manuf. Technol.* 70 (9–12), 1563–1578.
- Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.R., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* 5, 287–326.
- Han, Y.-Y., Gong, D., Li, J., Zhang, Y., 2016. Solving the blocking flow shop scheduling problem with makespan using a modified fruit fly optimisation algorithm. *Int. J. Prod. Res.* 54 (22), 6782–6797.
- Han, Y.-Y., Gong, D., Sun, X., 2015. A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. *Eng. Optim.* 47 (7), 927–946.
- Hoos, H.H., Stützle, T., 2004. Stochastic Local Search: Foundations & Applications. Morgan Kaufmann, San Francisco.
- Hulett, M., Damodaran, P., 2015. A particle swarm optimization algorithm for minimizing total weighted tardiness of non-identical parallel batch processing machines. In: IIE Annual Conference. Proceedings. Institute of Industrial Engineers-Publisher, p. 901.
- Ikura, Y., Gimple, M., 1986. Efficient scheduling algorithms for a single batch processing machine. *Oper. Res. Lett.* 5 (2), 61–65.
- Jia, Z.-H., Leung, J.Y.-T., 2015. A meta-heuristic to minimize makespan for parallel batch machines with arbitrary job sizes. *European J. Oper. Res.* 240 (3), 649–665.
- Jia, Z.-H., Li, K., Leung, J.Y.-T., 2015. Effective heuristic for makespan minimization in parallel batch machines with non-identical capacities. *Int. J. Prod. Econ.* 169, 1–10.
- Jia, Z., Li, X., Leung, J.Y.-T., 2017. Minimizing makespan for arbitrary size jobs with release times on p-batch machines with arbitrary capacities. *Future Gener. Comput. Syst.* 67, 22–34.
- Jia, Z.-H., Wang, C., Leung, J.Y.-T., 2016. An ACO algorithm for makespan minimization in parallel batch machines with non-identical job sizes and incompatible job families. *Appl. Soft Comput.* 38, 395–404.
- Kashan, A.H., Karimi, B., Jenabi, M., 2008. A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Comput. Oper. Res.* 35 (4), 1084–1098.
- Klemmt, A., Weigert, G., Almeder, C., Mönch, L., 2009. A comparison of MIP-based decomposition techniques and VNS approaches for batch scheduling problems. In: Proceedings of the 2009 Winter Simulation Conference (WSC). IEEE, pp. 1686–1694.
- Koh, S.-G., Koo, P.-H., Ha, J.-W., Lee, W.S., 2004. Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families. *Int. J. Prod. Res.* 42 (19), 4091–4107.
- Lee, C.-Y., Uzsoy, R., Martin-Vega, L.A., 1992. Efficient algorithms for scheduling semiconductor burn-in operations. *Oper. Res.* 40 (4), 764–775.
- Li, X., Chen, H., Du, B., Tan, Q., 2013a. Heuristics to schedule uniform parallel batch processing machines with dynamic job arrivals. *Int. J. Comput. Integr. Manuf.* 26 (5), 474–486.
- Li, X., Huang, Y., Tan, Q., Chen, H., 2013b. Scheduling unrelated parallel batch processing machines with non-identical job sizes. *Comput. Oper. Res.* 40 (12), 2983–2990.
- Malve, S., Uzsoy, R., 2007. A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Comput. Oper. Res.* 34 (10), 3016–3028.
- Mathirajan, M., Sivakumar, A., 2006a. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *Int. J. Adv. Manuf. Technol.* 29 (9–10), 990–1001.
- Mathirajan, M., Sivakumar, A., 2006b. Minimizing total weighted tardiness on heterogeneous batch processing machines with incompatible job families. *Int. J. Adv. Manuf. Technol.* 28 (9–10), 1038–1047.
- Mönch, L., Balasubramanian, H., Fowler, J.W., Pfund, M.E., 2005. Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Comput. Oper. Res.* 32 (11), 2731–2750.
- Mönch, L., Fowler, J.W., Dauzère-Pérès, S., Mason, S.J., Rose, O., 2011. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *J. Sched.* 14 (6), 583–599.
- Ozturk, O., Espinouse, M.-L., Mascolo, M.D., Gouin, A., 2012. Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates. *Int. J. Prod. Res.* 50 (20), 6022–6035.
- Pan, Q.-K., Ruiz, R., 2014. An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega* 44, 41–50.
- Potts, C.N., Kovalyov, M.Y., 2000. Scheduling with batching: A review. *European J. Oper. Res.* 120 (2), 228–249.
- Rodriguez, F.J., Lozano, M., Blum, C., García-Martínez, C., 2013. An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Comput. Oper. Res.* 40 (7), 1829–1841.
- Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European J. Oper. Res.* 177 (3), 2033–2049.
- Shahidi-Zadeh, B., Tavakkoli-Moghaddam, R., Taheri-Moghadam, A., Rastgar, I., 2017. Solving a bi-objective unrelated parallel batch processing machines scheduling problem: A comparison study. *Comput. Oper. Res.* 88, 71–90.
- Shahvari, O., Logendran, R., 2017. A bi-objective batch processing problem with dual-resources on unrelated-parallel machines. *Appl. Soft Comput.* 61, 174–192.
- Tang, L., Gong, H., Liu, J., Li, F., 2014. Bicriteria scheduling on a single batching machine with job transportation and deterioration considerations. *Nav. Res. Logist.* 61 (4), 269–285.
- Uzsoy, R., 1994. Scheduling a single batch processing machine with non-identical job sizes. *Int. J. Prod. Res.* 32 (7), 1615–1635.
- Uzsoy, R., 1995. Scheduling batch processing machines with incompatible job families. *Int. J. Prod. Res.* 33 (10), 2685–2708.
- Wang, H.-M., Chou, F.-D., 2010. Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics. *Expert Syst. Appl.* 37 (2), 1510–1521.
- Wang, J.-Q., Leung, J.Y.-T., 2014. Scheduling jobs with equal-processing-time on parallel machines with non-identical capacities to minimize makespan. *Int. J. Prod. Econ.* 156, 325–331.
- Xu, S., Bean, J.C., 2007. A genetic algorithm for scheduling parallel non-identical batch processing machines. In: 2007 IEEE Symposium on Computational Intelligence in Scheduling. IEEE, pp. 143–150.
- Xu, R., Chen, H., Li, X., 2013. A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system. *Int. J. Prod. Econ.* 145 (1), 371–386.
- Ying, K.-C., Lin, S.-W., Huang, C.-Y., 2009. Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. *Expert Syst. Appl.* 36 (3), 7087–7092.
- Zhou, S., Liu, M., Chen, H., Li, X., 2016. An effective discrete differential evolution algorithm for scheduling uniform parallel batch processing machines with non-identical capacities and arbitrary job sizes. *Int. J. Prod. Econ.* 179, 1–11.