Computers and Operations Research 113 (2020) 104793

Contents lists available at ScienceDirect

Computers and Operations Research

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

A genetic algorithm for scheduling open shops with sequence-dependent setup times

Levi R. Abreu, Jesus O. Cunha, Bruno A. Prata*, Jose M. Framinan

Department of Industrial Engineering, Federal University of Ceará, Campus do Pici, Bl. 714, Fortaleza 60440554, Brazil

ARTICLE INFO

Article history:
Received 25 March 2019
Revised 26 August 2019
Accepted 29 August 2019
Available online 3 September 2019

Keywords:
Production scheduling
Combinatorial optimization
Taguchi method
Metaheuristics

ABSTRACT

In the open shop scheduling problem with sequence-dependent setup times, there is no established order for the processing of the jobs, which leads to a large number of possible solutions for the scheduling problem. Furthermore, there is a setup time between two consecutive operations, which depends on the job previously processed. In this work we propose a hybrid genetic algorithm for the OSSP with sequence-dependent setup times and total completion time minimization as objective function. Our proposal uses two novel constructive heuristics which are combined for the generation of the initial population. We carry out an extensive computational experience using problem instances taken from the related literature to evaluate the performance of the proposed algorithms as compared to existing heuristics for the problem. The quality of the solutions and the CPU time required are used as performance criteria. Among them, the genetic algorithm with direct decoding presents a smaller value for the average relative percentage deviation in comparison with the electromagnetic algorithm proposed by Naderi et al. (2011). The computational results prove the excellent performance of the proposed metaheuristic for the tested instances, resulting in the most efficient algorithm so-far for the problem under consideration.

1. Introduction

The open shop scheduling problem (OSSP) is a combinatorial optimization problem widely studied in the last few decades since the seminal paper of Gonzalez and Sahni (1976). In the OSSP, a set of n jobs has to be processed in a set of m machines, so each job must visit all the machines. However, in the open shop layout there are no predefined routes for the jobs in the machines, so the processing route for each job is to be determined in the scheduling process (Framinan et al., 2014). Therefore, the number of feasible solutions for this problem is larger than for other layouts, such as, for example, parallel machines, flow shop, or job shop production environments. However, some constraints apply in the OSSP, such as the fact that the processing of the jobs in the machines occurs in distinct moments, i.e. a given job cannot be processed concomitantly in more than one machine. Furthermore, in the non-preemptive case of the OSSP, the processing of the jobs cannot be stopped until the end of the task.

The OSSP has a number of industrial applications, such as plastic molding, chemical processes, oil industry, and food production, whereas in the service sector is used to model medical care services, vehicle maintenance, and telecommunications (Gonzalez and Sahni, 1976; Lin et al., 2008; Naderi et al., 2010; 2012). For some authors, the open shop layout is assumed implicitly as the operational scheduling discipline in Flexible Manufacturing Systems (FMS) (Kusiak, 1985; Ben-Arieh and Dror, 1991), which in turn may be used within the smart manufacturing system envisioned by Industry 4.0 (Yin et al., 2018). It is not then surprising that, nowadays, the OSSP is an active and promising research topic in the production scheduling area (Anand and Panneerselvam, 2016).

Traditionally, setup times are considered as a part of processing times. However, this assumption leads to problems in the scheduling process because the completion times of jobs could be reduced substantially with explicit consideration of setup times (Allahverdi et al., 2008). Taking into consideration explicitly the setup times, a given production sequencing problem can be solved more realistically.

In this paper, we focus on the OSSP with the objective of minimising the sum of the completion times, which is usually mentioned as a relevant objective for today's manufacturing environments (see e.g. Liu and Reeves, 2001), leading to an even use of resources, fast turnaround of jobs, and low work-in-process (Rajendran and Ziegler, 1997). We also assume that a setup time exists whenever a machine has to process a job, and that the length of the setup time depends on the previous job that has been processed in the machine. Despite the theoretical and practical importance of the open shop with explicit setup times, the

* Corresponding author.
E-mail address: baprata@ufc.br (B.A. Prata).

https://doi.org/10.1016/j.cor.2019.104793
0305-0548/

contributions in the literature are rather limited. This limitation is highlighted in some recent literature reviews on the scheduling problems with setup times/costs (Allahverdi, 2015) and on the open shop scheduling problems (Anand and Panneerselvam, 2016).

Since the problem considered is NP-hard (Garey and Johnson, 1979) for most usually-considered objectives save the most trivial cases (e.g. one-machine open shop), the main contributions on the OSSP focus on approximate methods obtaining good solutions within a reasonable computation effort. Next, we discuss the main contributions related to OSSP with setups.

Strusevich (1993) proposes an $O(n)$ algorithm for minimizing the makespan for the two-machine OSSP with setup, processing and removal times separated. Low et al. (2003) study this variant of the problem for m machines, proposing an integer linear programming model as well as a simulated annealing metaheuristic to get high-quality solutions. Low and Yeh (2009) propose a hybrid genetic algorithm for the above-mentioned variant, outperforming a Genetic Algorithm (GA), a Simulated Annealing (SA), and a Tabu Search (TS) in all evaluated test problems.

Roshanaei et al. (2010) present a non-preemptive open shop scheduling problem with sequence dependent setup times to minimize makespan. The above-mentioned authors present the following algorithms: two new metaheuristics called multi-neighborhood search simulated annealing and hybrid simulated annealing, two constructive heuristics named LTMPT and LTRMPT and three additional metaheuristics: variable neighboorhood search, simulated annealing, and genetic algorithm. These algorithms are evaluated in a set of randomly generated test instances. Cankaya et al. (2019) present a mixed-integer programming model and a constraint programming model for the open-shop scheduling problem with sequence-dependent post-setup times to minimize the makespan. This problem arises in chemical tanker scheduling in ports, and a case study in the Port of Houston is presented. Their computational experience shows that the integer linear programming model is better for short-term decisions, and the constraint programming model is better for long-term decisions. Zhang et al. (2019) address a multiprocessor open shop scheduling environment to solve a large-scale health examination scheduling problem. The objective function is the minimization of the total completion time. A mixed-integer linear programming problem is proposed, as well as three metaheuristics: a simulated annealing algorithm, a genetic algorithm, and a hybrid particle swarm optimization. A set of test instances was carefully generated for the assessment of the proposed approaches, and the genetic algorithm outperformed all the other evaluated methods.

Regarding other objectives than the makespan, Naderi et al. (2011) present an integer linear programming model for the open shop scheduling problem with sequence-dependent setup times to minimize total completion time. In view of the complexity of the variant under study, the above-mentioned authors propose an electromagnetism-like algorithm denoted EH in the following. Their computational experience shows that the proposed model and EH were more effective than the other algorithms tested.

Noori-Darvish et al. (2012) present an open shop scheduling problem with sequence-dependent setup times, fuzzy processing times and fuzzy due dates. A bi-objective possibilistic mixed-integer linear programming model is presented. To obtain the Pareto front approximations for small-sized instances, these authors present an interactive fuzzy multi-objective decision-making approach. To solve medium-sized and large-sized instances, a multi-objective particle swarm optimization (MOPSO) algorithm is proposed.

Despite the existing contributions –namely the EH algorithm by Naderi et al. (2011)–, we believe that there is room for developing more efficient solution procedures for the problem. Our paper aims at presenting a hybrid genetic algorithm for the OSSP with sequence-dependent setup times and total completion time minimization as objective. For an efficient generation of the initial population, the GA is hybridized with two new constructive heuristics proposed in this paper, which yield good solutions in negligible computation times. We also evaluate the performance of three decoding schemes, thus investigating the efficiency of different solution encoding policies. In order to determine the best configuration of the algorithms, we perform a Taguchi design of experiments (see e.g. Phadke, 1995). We carry out an extensive computational experimentation using the well-known open shop test instances proposed by Guéret and Prins (1998), Taillard (1993) and Bruckner et al. (1997). The results shows that our proposal outperforms the existing procedures for the problem.

The remainder of this paper is organized as follows: in Section 2, the scheduling problem treated in this paper is described; in Section 3, the proposed genetic algorithm is presented; in Section 4, we explain the experimental design, while in Section 5 we discuss the results of the computational experiments. Finally, in Section 6 we draw some conclusions and suggestions for future works.

Table 1
Processing times for open shop example.

$i\backslash j$	1	2	3
1	661	168	171
2	250	489	505
3	333	343	324

Table 2
Setup times for open shop example.

$i\backslash j\backslash k$	M1			M2			M3		
	1	2	3	1	2	3	1	2	3
1	250	484	425	471	467	77	39	488	135
2	485	191	170	156	6	176	182	423	240
3	406	485	475	380	456	284	100	274	240

2. Problem description

Since in the open shop environment there is no fixed route for the jobs, the number of feasible solutions increases substantially with the instance size. An additional difficulty resides in the number of possible redundant solutions that can occur with the permutation encoding. In this representation, distinct permutations could represent the same solution. Thus, how to filter the redundant solutions plays a key role in a given algorithm. Also, the explicit consideration of setup times is another additional problem, once the job previously processed impacts in the machine setup time for the next operations.

Let us consider n jobs that must be processed in m machines, in which each job operation has a processing time associated without route constraints, i.e. the jobs can visit the machines in any order. We present an illustrative example with three jobs and three machines, with the processing times taken from the test problem GP03-01 by Guéret and Prins (1999) and presented in Table 1, while the setup times –presented in Table 2– have been genereted randomly. The usual encoding for the problem is a vector representing the order in which the operations are performed (Khuri and Miryala, 1999). In Table 3, the operations to be performed in an instance with 3 jobs and 3 machines are presented. Considering the instance in Table 1, the sequence $s = (6, 1, 8, 4, 7, 3, 9, 5, 2)$ returns a solution with a makespan of 2040 time units, as shown in Fig. 1.

Table 3
Operations for the presented instance.

Operation	1	2	3	4	5	6	7	8	9
Machine	1	1	1	2	2	2	3	3	3
Job	1	2	3	1	2	3	1	2	3

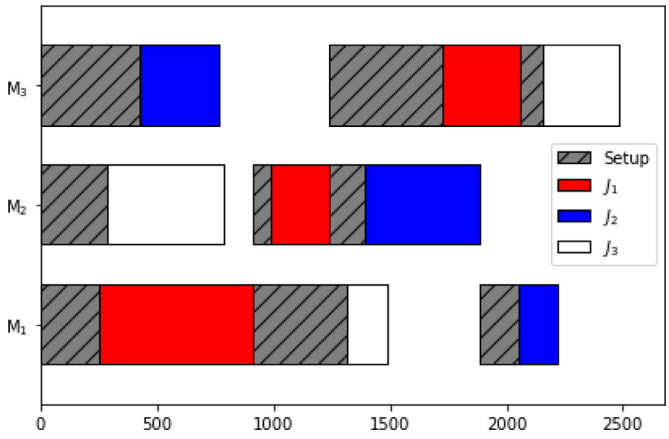

Fig. 1. Gantt chart for the presented solution.

In our problem, the setup time of a job depends on the previously processed job. Thereby, the setup times for each job are represented by an element of matrix s_{ijk} so that the setup times are variable for each job and each machine, as well as sequence-dependent. For this problem, we propose the lower bound (LB) given by Eq. (1):

$$LB = \sum_{i=1}^{M} \sum_{j=1}^{N} \left(p_{ij} + \min_{k=1,..N} \{s_{ijk}\} \right) \quad (1)$$

This lower bound would be used as an ingredient for the heuristics proposed in Section 3.2 for the initialisation of the population of the Genetic Algorithm, and to measure the quality of the solutions in Section 5.

As mentioned earlier, the problem under consideration is NP-hard, therefore it is of interest to develop efficient approximate methods providing good solutions with a reasonable computational effort. In the next section, we propose a new metaheuristic based on Genetic Algorithms to address this issue.

3. A new genetic algorithm

For the resolution of the problem under study, we propose a new genetic algorithm with knowledge of the problem domain, aiming at presenting high-quality solutions within admissible computation times. Some specific constructive heuristics are proposed to generate the initial population. Several genetic operators are considered and the selection of the best algorithm is based on a Taguchi experimental design, as detailed in the next section. In addition, we present some filters for redundant solutions in the mutation operator as well as in the local search, as presented by Naderi et al. (2010).

The proposed genetic algorithm presents the following operators: generation of the initial population, fitness evaluation of the generated solutions, parent selection, crossover, and mutation. A restart procedure is applied to avoid the premature convergence of the population. After all generations, a 2-opt local search algorithm is applied in the best-generated offspring. The pseudocode of the proposed genetic algorithm is described in Algorithm 1, whereas in Fig. 2 we present a flowchart of the algorithm. Their main elements are discussed in the next subsections.

Data: p, $setup$, $popsize$, $MaxGen$, selection(.), crossover(.), mutation(.), $pmut$, restart(.), 2-optLocalSearch(.)
Result: A sequence $\Pi := (\pi_{11}, \pi_{12}, ..., \pi_{mn})$
$pop \leftarrow$ CreatePopulation($popsize$);
$fitness \leftarrow$ CalculateFitness(pop, p, $setup$);
while $k \leq MaxGen$ **do**
  $parent1$, $parent2 \leftarrow$ selection(pop, $fitness$);
  $offspring \leftarrow$ crossover($parent1$, $parent2$);
  if $random() \leq pmut$ **then**
    $offspring \leftarrow$ mutation($offspring$);
  end
  if *the criterion for performing the restart is satisfied* **then**
    pop, $fitness \leftarrow$ restart(pop, $fitness$);
  end
  pop, $fitness \leftarrow$ replacement($offspring$);
  $k \leftarrow k + 1$;
end
$\Pi \leftarrow$ the best solution $\in pop$;
$\Pi \leftarrow$ 2-optLocalSearch(Π) ;

Algorithm 1: Pseudo-code of the proposed GA.

3.1. Encoding and decoding schemes

For the fitness evaluation, we consider a permutation encoding, as described in Section 2. A permutation list is a linear order list of all the operations. With this encoding, the operators developed must use some filters in order to avoid redundant solutions. We present three forms of decoding a given solution:

Direct decoding (D): In this decoding, the permutation is sequenced from the beginning to the end of the list and the jobs are allocated in the machines, as presented by Naderi et al. (2011).

Indirect decoding (I): In this decoding, the next operation to be sequenced is always the operations with the shortest start time, as described by Naderi et al. (2010).

Indirect decoding with setups (IS): In this new decoding scheme, the next operation to be sequenced is always the operations with the shortest start time. The start time considers the setup time necessary for each operation, taking into account the last job schedule in a given machine. Algorithm 2 describes the algorithm

Data: Π
Result: A sequence $S := (\pi_{11}, \pi_{12}, ..., \pi_{mn})$ with encoding scheme
$S \leftarrow \emptyset$;
$U \leftarrow \Pi$;
$M \leftarrow$ list with time accumulated in each machine;
$J \leftarrow$ list with time accumulated in each job;
$s_{ij} \leftarrow$ start time for processing of operation π_{ij} **while** $\|U\| > 0$ **do**
  $y \leftarrow \min_{(i,j) \in U} \left\{ \max \left\{ M_i, J_j \right\} + setup_{ijk} \right\}$; `// k is the index of the last job allocated to machine i`
  $R = \{\pi_{ij} | s_{ij} = y, \pi_{ij} \in U\}$;
  $\pi_{zd} \leftarrow$ operation in earliest relative position $\in R$;
  $U \leftarrow U - \{\pi_{zd}\}$;
  $S \leftarrow S + \{\pi_{zd}\}$;
  update J and M with time of π_{zd} operation;
  update s_{ij} based on J and M times, $\forall i \in M, j \in N$;
end

Algorithm 2: Indirect decoding with setups (IS) scheme procedure.

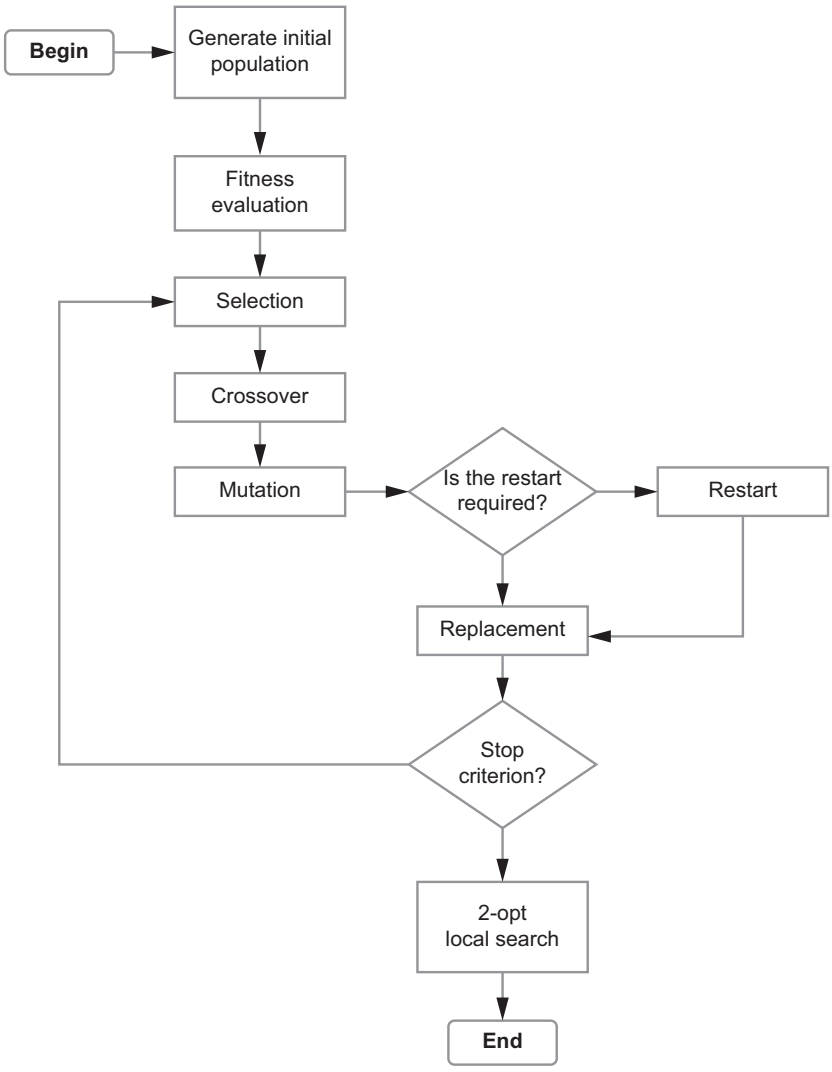

Fig. 2. Flowchart of the proposed hybrid genetic algorithm.

for this decoding. As it can be seen, the new decoding scheme always allocates the operation with the shortest start time, considering the accumulated times in the machines and the job to be sequenced. In addition, the scheme considers the cost of setup involved in the machine in relation to the last job allocated and the next one to be sequenced.

3.2. Generation of the initial population

The method for generating the initial population is described as follows: 25 % of the solutions are generated with a new constructive heuristic proposed for the problem, which is denoted as BICH-MIH, which is the result of combining two heuristics specifically designed for the problem. The remainder of the initial population is randomly generated. Next, we describe the constructive heuristics employed.

3.2.1. Bounded Insertion Constructive Heuristic (BICH)

The BICH type of heuristic was first proposed by Fernandez-Viagas and Framinan (2015) for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness. Basically, this type of heuristic iteratively constructs a solution by appending an unscheduled job at the end of a partial sequence of unscheduled jobs. The key is to obtain a suitable indicator to select the proper unscheduled job to be appended. Obviously, in order to obtain good solutions using this heuristic, the indicator has to be problem-specific, so we cannot adapt the heuristic in a straightforward manner. Furthermore, the sequence for our problem consists of operations and not jobs. However, BICH seems to be an interesting technique for the OSSP because the idea of the heuristic is to use a mechanism to limit the number of solutions to be explored, so it could be very useful for the OSSP as the number of feasible solutions is very large.

BICH obtains a sequence $\Pi := (\pi_{11}, \pi_{13}, ..., \pi_{mn})$ starting with Π as an empty sequence and adding one operation in each iteration o $(o = 1, \ldots, n \cdot m)$ as follows. We select the machine k which finishes first the processing of the operations in the partial sequence Π. Then, among the unscheduled operations corresponding to machine k, we pick the operation π_{kj} yielding the minimum value of the sum of the two following terms: the (partial) sum of the completion times of Π after the appending of π_{kj} plus the lower bound of the remaining unscheduled operations, according to Eq. (1). The idea is to pick an indicator that weighs the contribution of selecting a specific operation and also the contribution of the non-selected operations. The pseudo code of the proposed BICH is presented in Algorithm 3. In this figure, TCT is the total completion time, LB is a function that computes the (partial) lower bound, and p is a sample instance.

Data: TCT(.), LB(.), p
Result: A sequence $\Pi := (\pi_{11}, \pi_{12}, ..., \pi_{mn})$
$\Pi \leftarrow \{\}$;
$P \leftarrow$ copy(p);
$M \leftarrow$ list with time cumulative in each machine;
$J \leftarrow$ list with time cumulative in each job;
$\Omega_k \leftarrow$ list with the jobs allocated to machine k, $\forall k \in \{1, ..., m\}$;
while $\|\Pi\| < n \times m$ **do**
  machine $k \leftarrow \underset{i\in\{1,...,m\}}{\text{argmin}}\, M_i$job $j \leftarrow \underset{j\in\{1,...,n\}, j\notin\Omega_k}{\text{argmin}}$ $TCT(\Pi \cup \{\pi_{ki}\}, p)$ + LB(π_{ki}, P)$\Pi \leftarrow \Pi \cup \{\pi_{kj}\}$;
  $\Omega_k \leftarrow \Omega_k \cup \{j\}$;
  $P_{kj} \leftarrow 0$;
  update J and M with time of π_{kj} operation;
end

Algorithm 3: Pseudo-code of the BICH.

3.2.2. Minimal Idleness Heuristic (MIH)

Priority rules for the OSSP are usually based on LPT algorithms that sort operations in decreasing order. Taking into account that the open shop presents certain similarities with the parallel machine scheduling environment, allocating the jobs with the longest processing times is an interesting strategy. However, this strategy can lead to a high idleness of the machines. While in the parallel machines environment, this idleness is zero, in the open shop it can increase the completion times of a given solution.

Along these lines, we propose a new constructive procedure, not considered previously in the literature, that takes into consideration the idleness minimization. Solutions with low values of cumulative processing times will usually present low completion times. A measure for idleness can be the accumulated processing times for the jobs and machines. If the cumulative processing time for a given job is greater than the cumulative processing time for a given machine, it means that the machine will wait until the job is finished, and consequently, it can be allocated to the current machine. If the situation is the opposite, this job was already processed in another machine and its processing in the current machine will not result in idleness. Φ_{ij} represents the idleness generated by the allocation of job j to machine i, taking into account that it is allocated after job l in machine i, therefore inducing the corresponding setup s_{ijl}. Eq. (2) presents how to compute the idleness indicator Φ_{ij}, while Algorithm 4 shows the pseudocode of MIH.

$$\Phi_{ij} = \begin{cases} J_j - M_i + s_{ijl}, & \text{if } J_j > M_i \\ s_{ijl}, & \text{otherwise} \end{cases} \tag{2}$$

3.2.3. Constructive heuristic BICH-MIH

For the customer order scheduling problem with total completion time objective, Framinan and Perez-Gonzalez (2017) present a constructive heuristic in which there is a look-ahead mechanism for evaluating not only the potential contribution of the candidate orders to the objective function, but also an estimation of the contribution to the objective function of the non-scheduled orders. This look-ahead mechanism has been also applied to other scheduling problems. On the basis of such reasoning, we propose a constructive algorithm for our problem that takes into consideration the contribution of an operation for the total completion time as well as to the idleness. We adopt a weight aggregation function for combining the two objectives, i.e. total completion time minimization and idleness minimization. Aiming at weighing these two criteria, an α value is selected, in which a value close to 1 indicates

Data: p
Result: A sequence $\Pi := (\pi_{11}, \pi_{12}, ..., \pi_{mn})$
$\Pi \leftarrow \emptyset$;
$M \leftarrow$ list with time cumulative in each machine;
$J \leftarrow$ list with time cumulative in each job;
$\Omega_k \leftarrow$ list with the jobs allocated to machine k, $\forall k \in \{1, ..., m\}$;
while $\|\Pi\| < n \times m$ **do**
  machine $k \leftarrow \underset{i\in\{1,...,m\}}{\text{argmin}}\, M_i$; `// index from machine finishing first`
  job $j \leftarrow \underset{j\in\{1,...,n\}, j\notin\Omega_k}{\text{argmin}}\, \Phi_{kj}$; `// the best job not allocated in machine k, through the MIH rule.`
  $\Pi \leftarrow \Pi \cup \{\pi_{kj}\}$;
  $\Omega_k \leftarrow \Omega_k \cup \{j\}$;
  update J and M with time of π_{kj} operation;
end

Algorithm 4: Pseudo-code of the MIH.

greater importance for the idleness and a value close to 0 indicates greater importance for the completion time.

More specifically, let Ψ_{ij} be a performance indicator for the insertion of the operation π_{ij} in the permutation, α the weight of the expected contribution for the idleness minimization and Φ_{ij} the expected contribution for the idleness minimization. The objective function is the total completion time (TCT), LB is a function that calculates the (partial) lower bound, and p and *setup* is a sample instance. The performance indicator Ψ_{ij} can be computed as follows:

$$\Psi_{ij} = (1-\alpha) \times (\text{TCT}(\Pi \cup \{\pi_{ij}\}, p, setup) + \text{LB}(\pi_{ij}, P, Setup)) + \alpha \times \Phi_{ij} \tag{3}$$

The BICH-MIH heuristic is detailed in Algorithm 5.

Data: TCT(.), LB(.), p, $setup$, α
Result: A sequence $\Pi := (\pi_{11}, \pi_{12}, ..., \pi_{mn})$
$\Pi \leftarrow \emptyset$;
$P \leftarrow$ copy(p);
$Setup \leftarrow$ copy($setup$);
$M \leftarrow$ list with time cumulative in each machine;
$J \leftarrow$ list with time cumulative in each job;
$\Omega_k \leftarrow$ list with the jobs allocated to machine k, $\forall k \in \{1, ..., m\}$;
while $\|\Pi\| < n \times m$ **do**
  machine $k \leftarrow \underset{i\in\{1,...,m\}}{\text{argmin}}\, M_i$; `// index from machine finishing first.`
  job $j \leftarrow \underset{j\in\{1,...,n\}, j\notin\Omega_k}{\text{argmin}}\, \Psi_{kj}$; `// the best job not allocated in machine k, through the combined approach rule with` α`.`
  $\Pi \leftarrow \Pi \cup \{\pi_{kj}\}$;
  $\Omega_k \leftarrow \Omega_k \cup \{j\}$;
  $P_{kj} \leftarrow 0$;
  update J and M with time of π_{kj} operation;
end

Algorithm 5: Pseudo-code of the BICH-MIH heuristic

3.3. Selection procedures

We present three selection operators taking into consideration both the randomness and the quality of the solution. In all the pre-

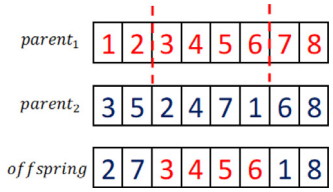

Fig. 3. Example of OX crossover.

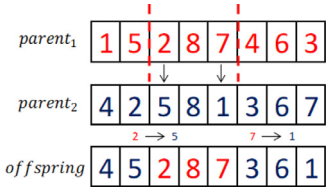

Fig. 4. Example of PMX crossover.

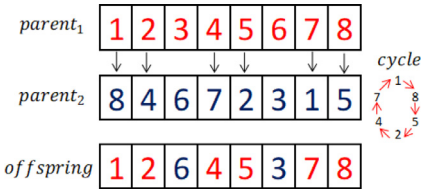

Fig. 5. Example of CX crossover.

sented selection procedures, the genetic operator aims at selecting two parents, called $parent_1$ and $parent_2$, for subsequent crossover operation.

3.3.1. Tournament

The selection based on the binary tournament randomly selects four individuals in the current population, performing a pairwise comparison. Let $parent_1$ be the best of two first solutions and $parent_2$ the best of the two last solutions. Thus, the selection procedure combines randomness and selective pressure.

3.3.2. Roulette

For each individual we compute the inverse of the objective function $F_i = \frac{1}{TCT_i}$. After that, we calculate the selection probability for each solution in the current population, which is its fitness in relation to the total, i.e. the probability of selecting individual i is given by $prob_i = \frac{F_i}{\sum F_i}$.

3.3.3. Rank

This operator is quite similar to the roulette operator, differing by the consideration of normalized probabilities. The best solution in the current population takes a value equals to the population size, the second best solution takes the value of the best solution minus one, and so on.

3.4. Crossover operators

The crossover operators were selected in view of the encoding using a permutation list, i.e. all the operators apply to a permutation of coded operations. The proposed crossover operators present a scheme for generating only feasible solutions. We consider three types of crossover: CX, PMX, and OX, which are described below.

3.4.1. Order crossover – OX

The OX crossover randomly selects two cutoff points from $parent_1$ and adds this information in the generated *offspring*. The remaining operations are added in the order in which they appear in $parent_2$, thus always generating a feasible offspring. Fig. 3 illustrates an example for OX crossover.

3.4.2. Partially Matched Crossover – PMX

PMX crossover randomly selects two cutoff point from $parent_1$ and the *offspring* integrally inherits the generated partial sequence. Finally, each remaining operation is obtained from $parent_2$. However, when an unfeasible sequence is generated, some exchanges are performed for the correction of the solution. More specifically, the operations from $parent_2$ that would generate an infeasible solution (operations already inserted in the offspring from $parent_1$) are inserted in the remaining positions of the offspring, taking into consideration the sequence of $parent_2$. Fig. 4 illustrates an example.

3.4.3. Cycle Crossover – CX

CX crossover generates an offspring based on the cycle of the operations from its parents, building solutions that preserve their absolute positions. With the generated cycle, *offspring* receives operations from $parent_1$ and the remaining operations from $parent_2$. Fig. 5 illustrates an example for this crossover.

3.5. Mutation operator

The mutation of a generated offspring is based on a swap movement. We generate a random number, uniformly distributed between 0 and 1. If this number is less than or equal to a given probability *pmut*, which is a parameter of the genetic algorithm, a random exchange between two operations is performed.

3.6. Replacement of parents by offspring

The population update may be based on three criteria: the classic genetic algorithm with elitism (simple), the hill climbing criterion (HC), and Metropolis criterion from simulated annealing (SA). These criteria adopt elitism: the algorithm preserves the solution with the best value of the objective function in the next generation.

3.6.1. Simple replacement

In a standard GA, the generated *offspring* is usually inserted in the current population replacing the solution with the worst value of the objective function. This strategy can lead the search for local optima (Gendreau and Potvin, 2010).

3.6.2. HC replacement

In this strategy the generated *offspring* is inserted in the current population only if this solution is better than its parents, such as in the well-known hill climbing algorithm. In this way, the population diversity is maintained since the offspring updates solutions with a significative similarity.

3.6.3. Simulated annealing replacement

In this strategy, we insert the generated *offspring* according to the well-known Metropolis criterion. If the generated *offspring* is better than its parents, we insert it in the current population. Otherwise, we insert the *offspring* on the basis of a given acceptance probability which depends on temperature. This temperature is high at the beginning of the search, being reduced over the generations. The offspring acceptance probability follows a Boltzmann distribution, such as in a simulated annealing algorithm:

$$Criterion = e^{-\Delta/t} \tag{4}$$

in which Δ means the total completion time (TCT) difference between the generated offspring and the best parent, i.e.:

$$\Delta = TCT(offspring) - TCT(parent_{best}) \tag{5}$$

It is important to highlight that, in the first generations, the probability of accepting solutions with worse values of the objective function is greater, ensuring the diversity of the population. In

the last generations, this probability is substantially reduced, leading to the intensification of the search.

Hence, we guarantee a replacement mechanism with randomness and selective pressure, combining strategies of diversification and intensification and preventing that the search is stuck in local optima (Gendreau and Potvin, 2010). Initial temperature ($T0$) receives a high value ($T0 = 100$) which is reduced over the generations, with a geometric decay rate (the decay rate is set to 0.95 in the experiments.).

3.7. Restart

According to Prata (2015), the restart operator aims at preventing the premature convergence of the search and leads to better results. This operator is controlled by a parameter *MaxGen*, which is a percentage of the maximum number of generations. The algorithm restarts the current population if there is no improvement after $0.3 \times MaxGen$ generations. The algorithm also restarts the temperature, in the case of simulated annealing replacement operator. In both cases, the algorithm preserves the best individual in the current population (elitism assumption). In this way, the best solution will distribute good genetic material for the new solutions, leading the algorithm to promising areas of the search space.

3.8. Local search

The local search adopted in this proposal is a 2-opt local search with the best improvement strategy, as presented in Naderi et al. (2011). The improvement procedure presents a mechanism for the search space reduction by means of a filter that avoids the generation of redundant solutions. The local search is executed in the best offspring generated by the proposed GA. On the basis of preliminary computational experiments, we have observed that the inclusion of the local search has led to substantially better solutions.

4. Experimental design

In this section we describe the design of the experiments carried out to select the best parameters of the algorithm proposed in Section 3. In the next sections, we present the indicators used for the evaluations, describe the test instances, and discuss the selection of the parameters of the algorithm.

4.1. Indicator for the evaluation

The statistic used in the analysis of the computational experiments is the gap for instance i between the solution sol_{ik} obtained by method k and the lower bound of the instance (LB_i), as presented in the Eq. (6). Thus, for each instance we calculate the relative percentage deviation (RPD) as the relative distance to the lower bound.

$$RPD_{ik} = \frac{sol_{ik} - LB_i}{LB_i} \cdot 100 \tag{6}$$

4.2. Description of test instances

In the open shop scheduling literature, test instances have been published by Guéret and Prins (1998), Taillard (1993), and Bruckner et al. (1997). For the test problems proposed by Guéret and Prins (1998), the processing time matrices are squared, because the number of jobs is equal to the number of machines. The sets of instances are divided in $m \in$ [3, 4, 5, 6, 7, 8, 9, 10]. The processing times are uniformly distributed with U[1,999]. For each set of instances there are 10 different test problems, totalyzing 80 instances. For the test problems proposed by Taillard (1993), the processing time matrices are similar to Guéret and Prins (1998), and the sets of instances are divided in n, $m \in$ {4, 5, 7, 10, 15, 20} with random values uniformly distributed between 1 and 100 for the processing times. For each size, 10 instances are randomly generated, totalyzing 60 instances. Finally, the test problems by Bruckner et al. (1997) are generated in a similar way, with random values uniformly distributed between 1 and 500 and 6 sets of jobs n, $m \in$ {3, 4, 5, 6, 7, 8}, totalyzing 60 instances.

Regarding the setup times, the instances of Guéret and Prins (1998), Taillard (1993), and Bruckner et al. (1997) do not provide this information. Therefore, as in Naderi et al. (2011), the setup times are randomly generated, with a distribution U[1,499] for the first five test problems of each set of instances and U[500,999] for the last five test problems of each set of instances.

Table 4
Factors and levels for Taguchi experimental design.

Factors	Levels		
	1	2	3
MaxGen	150	300	500
popsize	20	50	100
pmut	0.01	0.02	0.05
Selection	Tournament	Roulette	Rank
Crossover	OX	PMX	CX
Replacement	Simple	HC	SA
Do restart	True	False	–

4.3. Taguchi experimental design

Since the proposed GA presents different parameters and operators, we evaluate their influence of its performance by means of an experimental design approach. In our design of experiments, we consider the parameters presented in Table 4.

Since we analyse seven factors (each one with two or three levels) and we evaluate three distinct decoding schemes, a complete evaluation of the different parameters and operators of the algorithms is prohibitive. We adopt a Taguchi experimental design (Phadke, 1995) with 7 factors and 18 tests. The notation is L18(2∧1 3∧6), in which one factor presents two levels and six factors present 3 levels. Table 5 describes each test.

For the Taguchi experimental design, we consider a subset of the test intances. We randomly select a sample of 30 test problems among the 192 available test problems, taking into consideration the sizes $n \in$ {3, 4, 5, 6, 7, 8}.

Fig. 6 describes the main effects of the analyzed factors using the Average Relative Percentage Deviation (ARPD). The GA variants were run 10 times for each test instance, and the average results were tallied. It is possible to check that the restart operator, as well as the CX crossover operator, result in a significative improvement in the quality of the solutions.

In view of the results of the Taguchi experimental design, the best parametrization for the GA is considering tournament selection, CX crossover, $pmut = 0.05$, hill climbing replacement, restart procedure and $MaxGen = 100$. Table 6 presents the analysis of variance (ANOVA) (see e.g. Montgomery, 2017) for the ARPD. In this case, it can be seen that only restart and crossover operators are statistically significant.

5. Results and discussion

The above mentioned algorithms were implemented in the Intel®Distribution for Python* integrated development environment (https://software.intel.com/en-us/distribution-for-python) and were run in C with Cyton library (http://cython.org/)

Table 5
All experiments for L18(2∧1 3∧6).

Trial	Do restart	Popsize	Selection	Crossover	pmut	Replacement	MaxGen
1	True	20	Tournament	OX	0.01	Simple	150
2	True	20	Roulette	PMX	0.02	HC	300
3	True	20	Rank	CX	0.05	SA	500
4	True	50	Tournament	OX	0.02	HC	500
5	True	50	Roulette	PMX	0.05	SA	150
6	True	50	Rank	CX	0.01	Simple	300
7	True	100	Tournament	PMX	0.01	SA	500
8	True	100	Roulette	CX	0.02	Simple	150
9	True	100	Rank	OX	0.05	HC	300
10	False	20	Tournament	CX	0.05	HC	150
11	False	20	Roulette	OX	0.01	SA	300
12	False	20	Rank	PMX	0.02	Simple	500
13	False	50	Tournament	PMX	0.05	Simple	300
14	False	50	Roulette	CX	0.01	HC	500
15	False	50	Rank	OX	0.02	SA	150
16	False	100	Tournament	CX	0.02	SA	300
17	False	100	Roulette	OX	0.05	Simple	500
18	False	100	Rank	PMX	0.01	HC	150

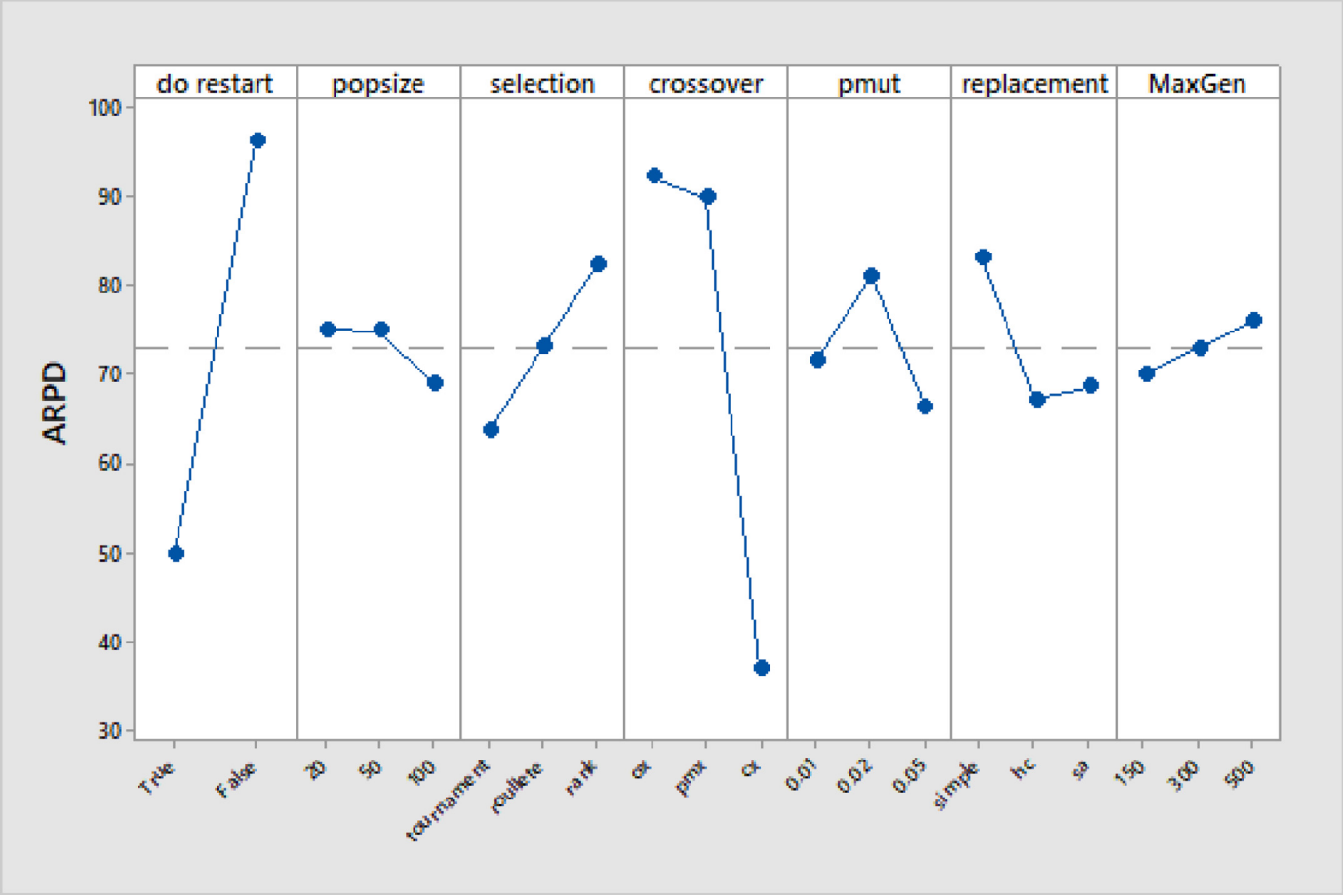

Fig. 6. Efectcs plots for diference levels of the crontrolled factors.

(Behnel et al., 2011). The computational experience was performed on a PC with Intel Core i7-4771 CPU 3.50 GHz and 8GB memory. The algorithms implemented are the following:

- Bounded Insertion Constructive Heuristic (BICH);
- Minimal Insertion Heuristic (MIH);
- GA(D): proposed genetic algorithm with direct decoding;
- GA(I): proposed genetic algorithm with indirect decoding;
- GA(IS): proposed genetic algorithm with indirect decoding with setups;
- Electromagnetic Heuristic (EH) proposed by Naderi et al. (2011), the state-of-art algorithm for OSSP with setup times.

For comparison purposes, we also consider the results of the Mixed Integer Linear Programming (MILP) model presented by Naderi et al. (2011), which was modeled and run in the same computer using IBM ILOG CPLEX version 12.8, with a time limit of 3600s. Regarding the metaheuristics under comparison, each algorithm is run 10 times for each instance, and the average results

Table 6
Variance test for ARPD in parameters of GA ($\alpha = 0.05$).

Font	GL	SQ Seq	SQ (Aj.)	QM (Aj.)	F	$p - value$
Do restart	1	9719.9	9719.9	9719.90	9.61	**0.036**
Popsize	2	141.2	141.2	70.60	0.07	0.934
Selection	2	1037.8	1037.8	518.90	0.51	0.633
Crossover	2	11725.1	11725.1	5862.55	5.80	**0.046**
pmut	2	661.0	661.0	330.48	0.33	0.739
Replacement	2	947.4	947.4	473.69	0.47	0.656
MaxGen	2	100.9	100.9	50.47	0.05	0.952
Errors	4	4043.8	4043.8	1010.94		
Total	17	28377.0				

are tallied. Table 7 illustrates the ARPD values for each class of instances, divided into low and high setup times.

In view of the results obtained, it can be seen that the metaheuristics outperform the constructive heuristics for all the instances. With respect to setup times it is clear that the instances with high setup distributions present solutions with smaller values of ARPD.

In order to verify wether the previous differences in RPD values are statistically significant, we apply an ANOVA. The so-obtained p-values are very close to zero, and in Fig. 7 we show the means plot with confidence intervals ($\alpha = 0.05$) for all methods proposed. We can clearly see that there are statistically significant differences between the average RPD values among all the methods.

With respect to the MILP model, it is clearly outperformed by MIH, GA(D), GA(IS), and EH even if the CPU effort of the former is much higher. Regarding constructive heuristics, BICH presents the worst results. Although MIH is outperformed by GA(D) and EH, the quality of the solutions obtained by MIH is similar to those yielded by the metaheuristics, with much smaller computation times. Therefore, this algorithm can be an interesting approach for solving real-world problems in an industrial environment if solutions are required within negligible computational effort. With respect to the metaheuristics tested, GA(D) presents the best overall results. More specifically, it outperforms EH, which is the best so-far method for the problem under study.

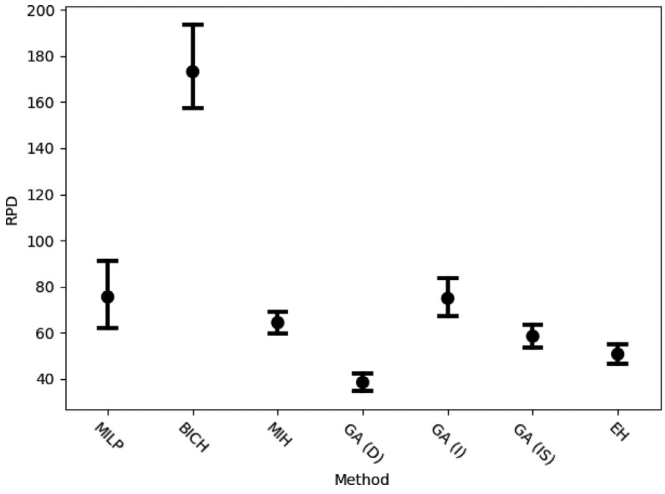

Fig. 7. 95 % confidence interval for ARPD in all methods.

In order to further ascertain the results obtained by the metaheuristics, we checked whether the previous differences in the mean of RPD values are statistically significant. We apply ANOVA and present HSD Tukey intervals ($\alpha = 0.05$) for all the combinations of methods in Fig. 8. In this figure, if the difference between the means of the RPD of the methods crosses the vertical line, then this difference cannot be considered significant in statistical terms.

As it can be seen, there are statistically significant differences between the average RPD values among all the algorithms analyzed. Considering GA (D) and EH, the diference is significant and GA(D) presents better results than EH. Thereby, the method proposed by Naderi et al. (2011) is outperformed in terms of the quality of solutions. Comparing the quality of the solutions of the three different decoding schemes, it can be seen that GA(D) obtains the best results, and that the results obtained by GA(IS) are

Table 7
ARPD values for all methods in each set of instance and setup type.

Method	MILP		BICH		MIH		GA (D)		GA (I)		GA (IS)		EH	
Setup	Low	High	Low	High	Low	High	Low	High	Low	High	Low	High	Low	High
Guéret and Prins														
GP03	21.51	13.81	52.56	44.03	34.90	24.93	21.51	13.81	31.41	29.36	35.61	26.95	21.51	13.81
GP04	32.49	16.76	69.34	59.02	51.00	25.31	26.51	8.29	49.21	19.73	54.79	16.51	32.49	16.76
GP05	31.91	19.85	106.84	72.35	53.09	27.56	29.77	9.01	50.69	24.55	53.24	20.50	34.42	21.05
GP06	46.98	29.03	158.78	92.11	65.44	33.49	39.78	13.26	61.91	26.65	62.95	23.21	53.80	28.35
GP07	57.68	33.28	175.76	110.75	65.78	30.99	41.13	15.20	69.01	28.38	71.69	24.07	56.44	28.58
GP08	65.93	56.19	193.52	130.50	78.07	33.84	48.69	15.13	84.58	30.06	64.04	24.65	67.07	30.13
GP09	97.12	51.76	255.96	129.64	85.15	32.68	57.99	13.02	102.85	31.83	76.56	25.88	77.14	29.70
GP10	146.96	109.66	304.98	152.38	90.89	33.97	64.10	12.62	121.15	34.44	83.15	23.90	82.24	31.08
Taillard														
tai_4x4	43.29	16.38	99.34	52.14	72.59	23.38	35.59	8.76	73.20	20.23	60.25	19.72	43.63	16.38
tai_5x5	51.27	18.82	156.01	77.61	96.77	34.81	59.87	10.84	84.50	30.05	83.99	23.77	60.66	24.84
tai_7x7	80.93	37.52	234.90	101.52	103.68	34.37	67.89	17.53	118.03	31.37	104.04	24.56	82.67	31.02
tai_10x10	274.96	108.45	325.42	138.91	120.58	36.83	83.22	15.36	174.00	34.90	108.18	24.88	105.13	32.47
tai_15x15	566.93	3.27	506.75	211.46	128.43	34.59	91.82	17.43	231.21	39.89	120.76	22.12	111.34	32.83
tai_20x20	127.59	3.14	707.71	261.01	124.05	32.08	95.31	13.04	280.47	42.11	133.01	21.37	119.95	30.31
Brucker														
j3	59.17	68.02	64.74	89.70	39.27	48.87	26.51	26.93	36.73	43.19	44.21	70.31	26.51	26.93
j4	50.24	62.65	99.22	104.21	57.60	67.75	31.27	36.10	69.08	66.27	49.71	64.87	37.01	38.66
j5	64.61	56.94	122.93	160.35	85.85	90.36	48.70	47.21	77.96	82.01	66.55	70.46	54.37	52.33
j6	62.57	61.40	183.98	157.52	93.01	90.27	62.03	52.76	92.73	96.88	77.39	93.27	72.62	64.95
j7	83.19	79.30	220.70	221.43	105.68	103.34	64.79	68.11	110.16	111.88	100.47	99.41	82.66	82.52
j8	113.00	99.05	234.95	267.89	108.99	104.08	71.98	72.22	135.80	138.88	107.15	102.72	91.48	95.39
Min	21.51	3.14	52.56	44.03	34.90	23.38	21.51	8.29	31.41	19.73	35.61	16.51	21.51	13.81
Avg	103.92	47.26	213.72	131.73	83.04	47.18	53.42	24.33	102.74	48.13	77.89	41.16	65.66	36.40
Max	566.93	109.66	707.71	267.89	128.43	104.08	95.31	72.22	280.47	138.88	133.01	102.72	119.95	95.39

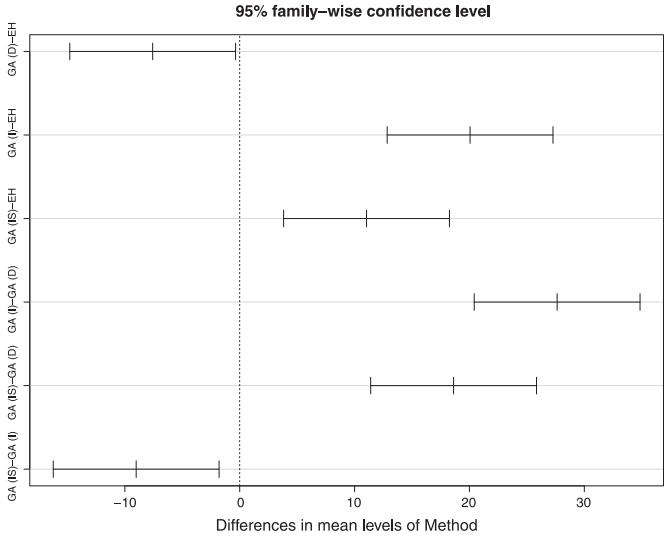

Fig. 8. Tukey HSD intervals at the 95% confidence level for the analyzed methods.

better than those obtained by GA(I). This means that the IS proposed decoding scheme outperforms the classic indirect decoding by Naderi et al. (2011), even if both are outperformed by GA(D).

Regarding the computational effort required by the different methods, Table 8 shows the average computational times (in seconds) for all considered methods for each instance size and setup type. As the computational times for the constructive heuristics are negligible (less than 1 second), we did not report them. In view of the results, the following points can be highlighted: The MILP model and the GA(IS) present the largest computational times, while GA(D) outperforms all the other metaheuristics. Aiming at comparing the effect of the instance size in the computational effort, Fig. 9 illustrates the average computational times for each instance size and setup type. EH performs similarly to GA(D) for the large-sized instances, and shows a worse performance for the small-sized ones. For the latter instances, the difference between the low and high setup types is not relevant, while for the large-sized ones it is statistically significant.

Finally, the trade-off between the quality of the solutions obtained by each method and its computational requirements can be assessed by presenting in Fig. 10 a Pareto front of the different methods with respect to their Average RPD (ARPD) and Average Computational Times (ARCT) to compare their relative efficiency. As it can be seen, the three algorithms proposed in this paper (the two constructive heuristics BICH and MIH, and the GA) constitute efficient methods for the problem under consideration. It is also worth noting the efficiency of GA(D) as compared to the other decoding methods. We believe that the reason for the excellent performance of the direct decoding method lies in the fact that the objective function is computed using exactly the same sequence in which the operations appear in the solution, therefore the improvement obtained by the GA is a consequence of an improvement in the objective function, since there is no need to change the sequence of the orders (in other words, to perform a decoding). In contrast, when a solution is decoded using an indirect operator – as in GA(I) or GA(IS) – the sequence of the operations is changed, and the setup values are modified because they are sequence-dependent. This alteration deteriorates the performance of the GA. Furthermore, the need to perform a coding-decoding process also increases the cpu time requirements, as it can be seen in Table 8.

6. Conclusions and future research

In this paper, we address the open shop shop scheduling problem with sequence dependent setup times and with the objective function of minimizing the total completion time. We develop a hybrid genetic algorithm that incorporates two new constructive heuristics as well as three different decoding schemes.

A series of computational experiments are carried out in order to evaluate the performance of the proposed algorithms. We use the relative deviation percentage as an indicator of the quality of the solutions, and the average computation time as an indicator

Table 8
Average computational times for all methods in each set of instance and setup type.

Method	MILP		GA (D)		GA (I)		GA (IS)		EH	
Setup	Low	High	Low	High	Low	High	Low	High	Low	High
Guéret and Prins										
GP03	0.04	0.05	0.13	0.13	0.19	0.19	0.24	0.24	8.07	8.09
GP04	1.77	8.29	0.30	0.30	0.54	0.53	0.67	0.68	12.86	12.98
GP05	319.57	4306.29	0.67	0.67	1.39	1.36	1.85	1.82	16.47	16.67
GP06	3602.16	3600.93	1.42	1.43	3.22	3.27	4.64	4.58	23.48	23.80
GP07	3600.42	3600.64	2.87	2.86	6.94	6.95	10.89	10.91	30.44	30.60
GP08	3600.30	3600.62	5.48	5.36	15.18	14.23	23.75	23.64	37.71	36.94
GP09	3600.23	3600.23	9.79	9.66	26.49	29.87	49.14	48.22	46.75	45.45
GP10	3600.71	3600.14	16.94	16.47	55.49	50.85	93.75	92.35	54.00	53.95
Taillard										
tai_4x4	0.92	6.33	0.30	0.30	0.54	0.54	0.68	0.66	11.13	11.31
tai_5x5	271.31	3601.25	0.67	0.67	1.33	1.34	1.83	1.83	17.07	16.75
tai_7x7	3600.20	3601.14	2.91	2.84	5.57	5.42	10.88	10.81	30.26	30.20
tai_10x10	3600.08	3600.09	16.85	16.43	40.02	38.50	93.56	92.52	55.94	55.43
tai_15x15	3600.16	3600.08	28.68	40.48	449.46	401.94	1170.39	1165.56	117.63	113.60
tai_20x20	3600.13	3600.16	615.96	605.27	3863.24	3647.43	8418.71	8686.47	214.09	209.99
Brucker										
j3	0.02	0.02	0.11	0.12	0.17	0.17	0.21	0.23	8.25	8.13
j4	0.11	0.09	0.27	0.28	0.43	0.45	0.60	0.62	12.18	12.51
j5	10.75	10.09	0.63	0.62	1.14	1.12	1.77	1.75	17.34	16.34
j6	3044.00	2055.83	1.38	1.40	2.40	2.60	4.59	4.60	22.97	21.84
j7	3600.48	3600.32	2.78	2.81	5.63	5.70	10.89	10.91	28.68	31.26
j8	1250.81	3600.22	5.48	5.52	11.62	11.25	24.51	24.60	35.89	37.08
Min	0.02	0.02	0.11	0.12	0.17	0.17	0.21	0.23	8.07	8.09
Avg	1947.82	2361.56	35.68	35.68	224.55	211.19	496.18	509.15	40.06	39.65
Max	3602.16	4306.29	615.96	605.27	3863.24	3647.43	8418.71	8686.47	214.09	209.99

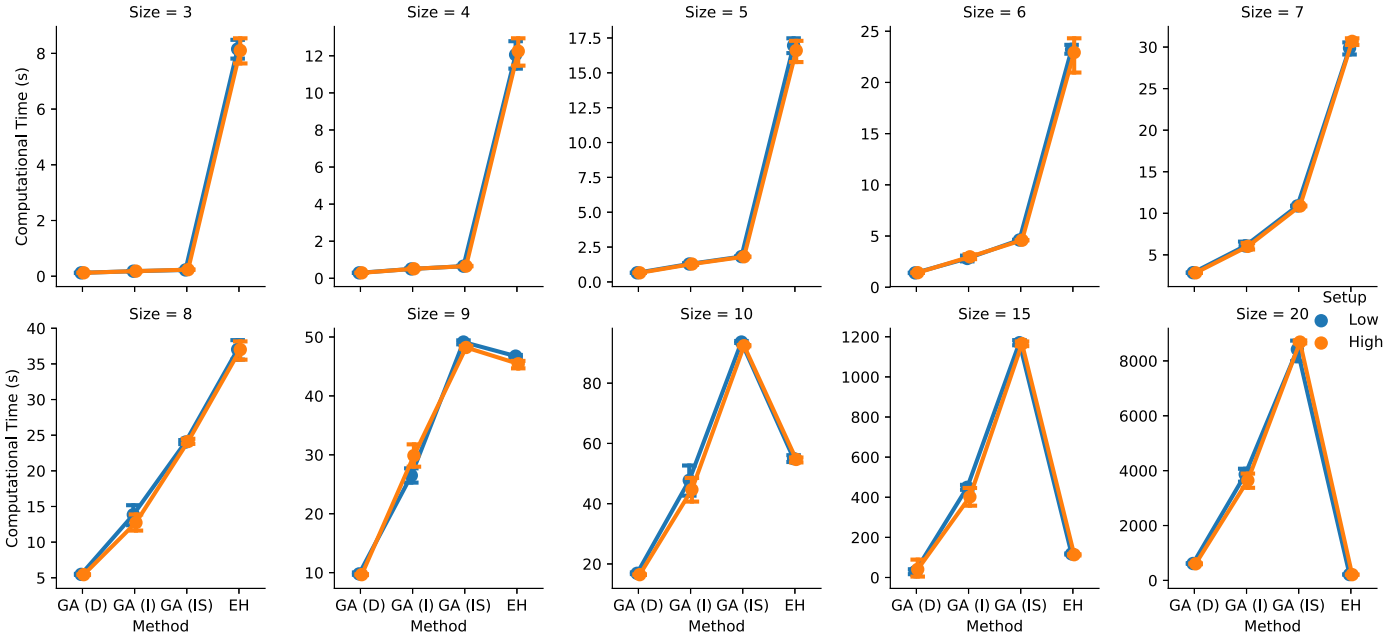

Fig. 9. 95 % confidence interval for Computational time in best methods in each size of instance.

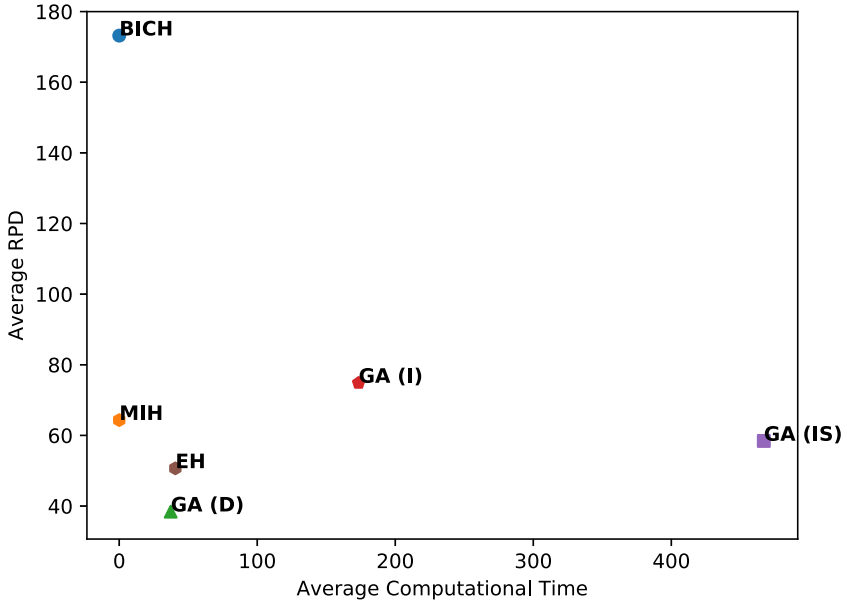

Fig. 10. ARCT x ARPD in heuristics and metaheuristics methods.

of the computational effort. Regarding the quality of the solutions, the hybrid genetic algorithm GA(D) clearly outperforms the MILP model, the two other variants of the genetic algorithm, as well as the electromagnetic heuristic proposed by Naderi et al. (2011), requiring less time than these methods. Regarding the trade-off between the quality of the solutions obtained by each method and its computational requirements, the three algorithms proposed in this paper (the two constructive heuristics BICH and MIH, and the GA) constitute efficient methods for the problem under consideration.

As extensions of this work, matheuristics could be used to improve the solutions generated by the presented methods. Future studies could also investigate the behavior of the proposed algorithms considering other objective functions, such as makespan minimization or total tardiness minimization. Finally, it is worth studying multi-objective variants of the open shop scheduling problems with sequence-dependent setup times due to the fact that, in many real industrial settings, the scheduling problems have a multi-objective nature.

Acknowledgments

The authors wish to thank the referees for their insightful comments on the earlier versions of the manuscript. The support of the National Council for Scientific and Technological Development (CNPq) grants 404232/2016-7 and 303594/2018-7 and the Spanish Ministry of Science and Innovation under the project “PROMISE” with reference DPI2016-80750-P are acknowledged and appreciated.

References

Allahverdi, A., 2015. The third comprehensive survey on scheduling problems with setup times/costs. Eur. J. Oper. Res. 246 (2), 345–378.

Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y., 2008. A survey of scheduling problems with setup times or costs. Eur. J. Oper. Res. 187 (3), 985–1032. doi:10.1016/j.ejor.2006.06.060.

Anand, E., Panneerselvam, R., 2016. Literature review of open shop scheduling problems. Intell. Inf. Manag. 7 (1), 32–52.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D.S., Smith, K., 2011. Cython: the best of both worlds. Comput. Sci. Eng. 13 (2), 31–39.

Ben-Arieh, David, Dror, Moshe, 1991. Intelligent heuristic for FMS scheduling using grouping. Journal of Intelligent Manufacturing 2 (6), 387–395.

Bruckner, P., Hurink, J., Jurish, B., Wostmann, B., 1997. A branch and bound algorithm for the open-shop problem. Discret. Appl. Math. 76 (1), 43–59.

Cankaya, B., Wari, E., Eren Tokgoz, B., 2019. Practical approaches to chemical tanker scheduling in ports: a case study on the port of houston. Marit. Econ. Logist. doi:10.1057/s41278-019-00122-w.

Fernandez-Viagas, V., Framinan, J.M., 2015. Efficient non-population-based algorithms for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness. Comput. Oper. Res. 64, 86–96.

Framinan, J., Leisten, R., Ruiz, R., 2014. Manufacturing Scheduling Systems: An Integrated View of Models, Methods, and Tools. Springer.

Framinan, J.M., Perez-Gonzalez, P., 2017. New approximate algorithms for the customer order scheduling problem with total completion time objective. Comput. Oper. Res. 78, 181–192. doi:10.1016/j.cor.2016.09.010.

Garey, M., Johnson, D., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, Norwell.

Gendreau, M., Potvin, J.-Y., 2010. Handbook of Metaheuristics, 2. Springer.

Gonzalez, T., Sahni, S., 1976. Open-shop scheduling to minimize finish time. J. Assoc. Comput. Mach. 23 (4), 665–679.

Guéret, C., Prins, C., 1998. Classical and new heuristics for the open-shop problem. Eur. J. Oper. Res. 107 (2), 306–314.

Guéret, C., Prins, C., 1999. A new lower bound for the open shop problem. Ann. Oper. Res. 92 (0), 165–183.

Kusiak, Andrew, 1985. Flexible manufacturing systems: methods and studies 12.

Khuri, S., Miryala, S., 1999. Genetic algorithms for solving open shop scheduling problems. Prog. Artif. Intell. 849.

Lin, H.-T., Lee, H.-T., Pan, W.-J., 2008. Heuristics for scheduling in a no-wait open shop with movable dedicated machines. Int. J. Prod. Econ. 111 (2), 368–377. doi:10.1016/j.ijpe.2007.01.005. Special Section on Sustainable Supply Chain.

Liu, J., Reeves, C., 2001. Constructive and composite heuristic solutions to the $P||\Sigma c_i$ scheduling problem. Eur. J. Oper. Res. 132, 439–452.

Low, C., Yeh, J.-Y., Low, F.-W., 2003. Solution models construction for open shop scheduling problem with setup, processing, and removal times separated. J. Chin. Inst. Ind.Eng. 20 (6), 565–574. doi:10.1080/10170660309509261.

Low, C., Yeh, Y., 2009. Genetic algorithm-based heuristics for an open shop scheduling problem with setup, processing, and removal times separated. Robot. Comput.-Integr. Manuf. 25 (2), 314–322.

Montgomery, D.C., 2017. Design and Analysis of Experiments. John wiley & sons.

Naderi, B., Ghomi, S.F., Aminnayeri, M., Zandieh, M., 2010. A contribution and new heuristics for open shop scheduling. Comput. Oper. Res. 37 (1), 213–221.

Naderi, B., Ghomi, S.M.T.F., Aminnayeri, M., Zandieh, M., 2011. Modeling and scheduling open shops with sequence-dependent setup times to minimize total completion time. Int. J. Adv. Manuf.Technol. 53 (5), 751–760. doi:10.1007/s00170-010-2853-6.

Naderi, B., Najafi, E., Yazdani, M., 2012. An electromagnetism-like metaheuristic for open-shop problems with no buffer. J. Ind. Eng. Int. 8 (1), 29.

Noori-Darvish, S., Mahdavi, I., Mahdavi-Amiri, N., 2012. A bi-objective possibilistic programming model for open shop scheduling problems with sequence-dependent setup times, fuzzy processing times, and fuzzy due dates. Appl. Soft Comput. 12 (4), 1399–1416.

Phadke, M.S., 1995. Quality Engineering Using Robust Design. Prentice Hall PTR.

Prata, B.A., 2015. A hybrid genetic algorithm for the vehicle and crew scheduling in mass transit systems. IEEE Latin Am. Trans. 13 (9), 3020–3025.

Rajendran, C., Ziegler, H., 1997. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. Eur. J. Oper. Res. 103, 129–138.

Roshanaei, V., Esfehani, M.S., Zandieh, M., 2010. Integrating non-preemptive open shops scheduling with sequence-dependent setup times using advanced metaheuristics. Expert Syst. Appl. 37 (1), 259–266.

Strusevich, V.A., 1993. Two machine open shop scheduling problem with setup, processing and removal times separated. Comput. Oper. Res. 20 (6), 597–611.

Taillard, E., 1993. Benchmarks for basic scheduling problems. Eur. J. Oper. Res. 64 (2), 278–285.

Yin, Yong, Stecke, Kathryn E., Li, Dongni, 2018. The evolution of production systems from Industry 2.0 through Industry 4.0. International Journal of Production Research 56 (1-2), 848–861.

Zhang, J., Wang, L., Xing, L., 2019. Large-scale medical examination scheduling technology based on intelligent optimization. J. Combinatorial Optimization 37 (1), 385–404. doi:10.1007/s10878-017-0246-6.