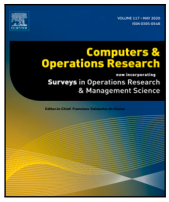
Computers & Operations Research 142 (2022) 105744

Contents lists available at ScienceDirect

Computers and Operations Research

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



New efficient heuristics for scheduling open shops with makespan minimization

Levi R. Abreu [a,*], Bruno A. Prata [b], Jose M. Framinan [c], Marcelo S. Nagano [a]

[a] Department of Production Engineering, University of São Paulo, São Paulo, Brazil
[b] Department of Industrial Engineering, Federal University of Ceará, Ceará, Brazil
[c] Department of Industrial Organization and Business Management I, University of Seville, Seville, Spain

ARTICLE INFO

Keywords:
Production scheduling
Combinatorial optimization
Heuristics
Open shop

ABSTRACT

This paper deals with the so-called Open Shop Scheduling Problem (OSSP) with makespan objective, which consists of scheduling a set of jobs that must visit a set of machines in no established order so the maximum completion time among the jobs is minimized. This problem is known to be NP-hard, and the absence of specific routes for the processing of the jobs makes its solution space extremely large. In this work we propose several efficient constructive heuristics that exploit some specific properties of the OSSP. We carry out an extensive computational experience using problem instances taken from the related literature to assess the performance of the proposed algorithms as compared to existing ones with respect to the quality of the solutions and the CPU time required. The extensive computational tests show the excellent performance of the heuristics proposed, resulting in the best-so-far heuristics for the problem.

1. Introduction

Shop scheduling problems are widely studied optimization problems because of their many industrial applications. In the last decades, variants of the flow shop scheduling problem and job shop scheduling problem have received a lot of attention by researchers (see e.g. Fernandez-Viagas et al., 2017; Fan et al., 2018; Zhang et al., 2019 for recent reviews on permutation flowshop, hybrid flowshop, and job shop scheduling, respectively). However, this has not been the same for the Open Shop Scheduling Problem (OSSP), which has received much less attention (Anand and Panneerselvam, 2016; Adak et al., 2020; Ahmadian et al., 2021). This problem is first described by Gonzalez and Sahni (1976), and consists of scheduling a set of jobs on a set of machines, in which each job operation has an associated processing time. However, unlike flow shop and job shop scheduling problems, in the OSSP there are no predefined routes for the jobs in the machines. The OSSP has several industrial applications such as plastic molding, chemical processes, oil industry, and food production, while in the service sector, it is used to model medical care services, vehicle maintenance, telecommunications, and museum visit schedules (Gonzalez and Sahni, 1976; Lin et al., 2008; Naderi et al., 2010, 2012; Vincent et al., 2010; de Abreu et al., 2021; Abreu et al., 2021).

When the objective considered is the minimization of the maximum completion times of the jobs (makespan) and there is only one machine, the OSSP can be reduced to a single machine problem and every schedule is optimal. For the case of two machines, there are polynomial algorithms with optimally proof (Gonzalez and Sahni, 1976; Pinedo, 2016). However, for problems with three or more machines, the OSSP with makespan objective is NP-Complete (Garey and Johnson, 2012). Therefore, although some branch-and-bound algorithms have been proposed for this problem (Bruckner et al., 1997; Guéret and Prins, 1999; Guéret et al., 2000), exact methods are quite limited for solving realistic-size problem instances.

In view of the aforementioned hardness of the OSSP with makespan objective, different approximate algorithms have been proposed. These can be broadly classified as either constructive heuristics, or local search/metaheuristic approaches. Regarding constructive heuristics, several contributions have been presented: Pinedo (2016) proposed two dispatching rules: Longest Alternate Processing Times (LAPT) and Longest Total Remaining Processing Times on Other Machines first (LTRPOM). LAPT schedules first the jobs with the longest processing time in other machine and the LTRPOM allocates a job first with the greater sum of processing times in other machine. Liaw (1998) presented a dispatching rule called Dense Schedule/Longest Total Remaining Processing (DS/LTRP), which is an improvement of the LTRPOM applying the well-known label correction algorithm (Skriver and Andersen, 2000).

Ramudhin and Marier (1996) adapted to the OSSP the shifting bottleneck procedure heuristic, originally used to solve the job shop

* Corresponding author.
E-mail address: leviribeiro@alu.ufc.br (L.R. Abreu).

https://doi.org/10.1016/j.cor.2022.105744
Received 28 June 2021; Received in revised form 12 January 2022; Accepted 1 February 2022
Available online 19 February 2022

scheduling problem. The heuristic iteratively attempts to select the bottleneck job or machine to re-optimize the jobs' processing sequence. Strusevich (1998) proposed a greedy heuristic for the open shop, considering job priorities. The results for the three-machine case showed that the method obtains solutions with a maximum deviation of $\frac{3}{2}$ from the optimal solution. Guéret and Prins (1998) presented two constructive heuristics, the first based on dispatching rules and the second based on the construction of matchings in a bipartite graph. Bai and Tang (2011) proposed a modified rotation scheduling heuristic for the problem, with relevant theoretical contributions, such as proof of optimally when the number of jobs tends to infinity.

Regarding the application of local search methods for the OSSP, Colak and Agarwal (2005) proposed a neural network algorithm that uses ten heuristic rules and local search procedures, González-Rodríguez et al. (2010) proposed a heuristic local search with neighborhood procedure based on graph theory to solve OSSP with triangular fuzzy processing times. Finally, Naderi et al. (2010) presented new efficient constructive algorithms with a local search that outperforms other existing algorithms such as LAPT.

For the OSSP it is clear that, since the space of solutions is extremely large due to the absence of a predefined routing of the jobs, the solution encoding scheme plays a key role (Ahmadian et al., 2021). Three different encoding schemes have been used in the literature, i.e.: the disjunctive graph representation, the rank matrix, and the permutation list. The disjunctive graph representation can be used exclusively for the makespan objective, and it was introduced by Liaw in a series of papers (see Liaw (1999b,a) and Liaw (2000)). The rank matrix encoding was first proposed by Bräsel et al. (1993), and it consists of a matrix in a data structure in which each row represents the sequence of operations for a given job on the machines, and each column represents the sequence of jobs on each machine. The permutation list encoding consists of a sequence of the operations (i.e. each tuple job, machine is given a number so a solution is represented by a sequence). Clearly, the permutation encoding scheme is much simpler, however its main disadvantage is its redundancy, as different sequences may indeed represent the same schedule. Naderi et al. (2010) presented four theorems to drastically reduce the redundancies in the permutation list encoding. They also propose four local search algorithms (IRH1, IRH2, IRH3 and IRH4) using these properties.

With respect to the application of metaheuristics for the OSSP, the aforementioned references by Liaw presented a tabu search (Liaw, 1999b), a simulated annealing (Liaw, 1999a), and a genetic algorithm (Liaw, 2000). Prins (2000) proposed a Genetic Algorithm (GA) using the permutation list encoding with two special features: a population with individuals with different makespan values and a procedure for re-ordering the generated chromosomes. This algorithm outperformed the then-existing heuristics and metaheuristics. Blum (2005) proposed hybridized beam-search algorithm with ACO (Ant Colony Optimization) using the permutation list encoding. Sha and Hsu (2008) presented a new Particle Swarm Optimization (PSO) algorithm using the permutation list scheme with an innovative encoding for the particles and a particle movement based on an insertion operator. Their computational results include several new best known solutions for the unsolved problems, and it is shown to outperform the algorithms by Liaw (1999b), Prins (2000), and Blum (2005).

Also using the permutation list encoding, Ahmadizar and Hosseinabadi Farahani (2012) proposed a hybrid genetic algorithm with a local search optimization procedure which outperforms the previously reported metaheuristics for the OSSP. Ghosn et al. (2016) proposed a parallel genetic algorithm using deterministic and random moves. Pongchairerks and Kachitvichyanukul (2016) proposed a two level PSO that performs very well on the benchmark instances. Finally, an extended genetic algorithm was proposed by Rahmani Hosseinabadi et al. (2018) to solve OSSP.

As it can be seen from the above review, there are several methods to provide approximate solutions for the OSSP with makespan objective. However, we think that there is room for improving the state-of-the-art of the problem by proposing new efficient constructive heuristics which incorporate some knowledge of the problem domain. More specifically, in this paper we first suggest using a look-ahead mechanism to estimate the contribution to the makespan of a partial solution in order to discard less-promising solutions, as well as some reasoning about the machines idle-time between operations. We believe that the development of efficient constructive heuristics for the OSSP is important for (at least) two reasons: (1) efficient constructive heuristics may provide high quality solutions in reduced computation times, which is required in many manufacturing environments and that allows to tackle problems of realistic size – particularly for the OSSP as the space of solutions grows very quickly with the problem size –, and (2) most metaheuristics and local search techniques use some constructive heuristic(s) as a starting solution, so designing more efficient constructive heuristics also boosts the performance of these procedures. These two aspects will be checked when developing our proposals. Once these fast constructive heuristics are developed, we combine them with a beam search algorithm and a cheapest insertion procedure. Finally, these are embedded in a local search (LS) procedure which uses the permutation list encoding but takes into consideration the four redundancy theorems proposed by Naderi et al. (2010) to overcome the disadvantages of the chosen encoding. All the algorithms proposed in the paper are compared with the existing constructive heuristics and local search approaches (i.e. LAPT, LTRPOM, DS/LTRP, EGA, IRH1, IRH2, IRH3 and IRH4) in an exhaustive computational experience.

The remainder of this paper is organized as follows: in Section 2, the scheduling problem treated in this paper is formally stated and a Mixed-Integer Linear Programming (MILP) model that will be used to obtain the optimal solutions for small-sized instances is presented. In Section 3, the proposed algorithms are described; in Section 4, we discuss some results of the computational experiments and statistical tests. Finally, in Section 5 we draw some conclusions and suggestions for future works.

2. Problem statement and MILP model

The problem considers n jobs that must be processed in m machines. Each job has a processing time on each machine and can visit the machines in any order. Furthermore, the usual hypotheses in scheduling apply: The processing of operations on the machines occurs at different times, i.e., a particular job cannot be processed at the same time on more than one machine. In addition, we deal with the non-preemptive case of the OSSP, hence the processing of the jobs cannot be interrupted, i.e., the job once started on a machine, it must be processed until the end of the task. The objective of the decision problem is to minimize the maximum completion time among the jobs (makespan).

If we use the permutation list as an encoding scheme (see e.g. Khuri and Miryala (1999)), a solution of the problem is given by a sequence s containing all the operations to be performed in the shop. The schedule corresponding to solution s consist in scheduling operation k corresponding to job j in machine i in order to start as earliest as possible but not before any previous job in the schedule. A pseudo-code of this active scheduler decoding scheme is given in Fig. 2.

As an example, we present the classic instance GP03-01 of Guéret and Prins (1999) in Table 1. The instance has three jobs and three machines. Using the permutation list encoding, the operations to be performed in an instance with 3 jobs and 3 machines are presented in Table 2, Taking into account also the processing times in Table 1, it can be seen that the sequence $s = (9, 3, 5, 6, 4, 8, 7, 2, 1)$ returns a solution with a makespan of 2064 time units, as shown in Fig. 1.

There are several ways to model the OSSP problem using mathematical programming, being different regarding to the way in which the decision variables are defined. More specifically, three types of notations can be used, i.e. positional notation, sequential notation and

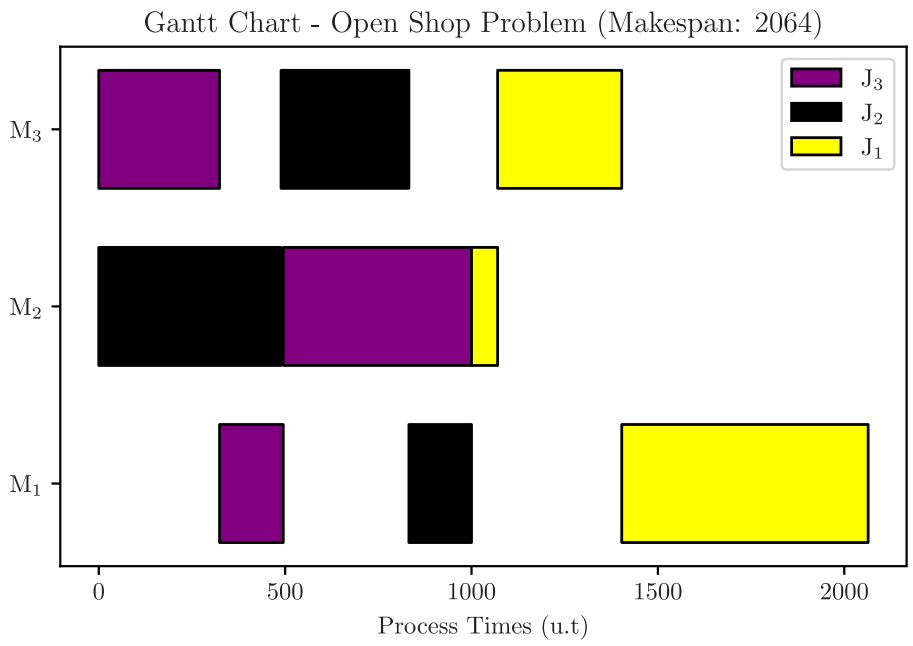


Fig. 1. Gantt chart for the presented solution.

Table 1
Processing times for open shop example.

$i\backslash j$	1	2	3
1	661	168	171
2	70	489	505
3	333	343	324

Table 2
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

Data: A solution with sequence Π
Result: The maximum completion time ($makespan$)
$U \leftarrow \Pi$;
$M \leftarrow$ list with time acumulated in each machine;
$J \leftarrow$ list with time acumulated in each job;
while $\|U\| > 0$ **do**
  $\pi_{zd} \leftarrow$ operation in the first position $\in U$;
  $U \leftarrow U - \{\pi_{zd}\}$;
  update J and M with time of π_{zd} operation;
end
$makespan \leftarrow \max_{i \in \{1,\ldots,m\}} M_i$

Fig. 2. Active schedule decoding scheme procedure.

time-indexed notation. In the study by Naderi et al. (2011a) it is illustrated that models with sequential notation perform better in OSSP problems due to their smaller number of variables and constraints, as compared to positional and time-indexed notation. Therefore, we have used the sequential notation in the MILP model developed.

Although MILP models are not efficient to solve medium and large size instances of many production scheduling problems due to their NP-hard nature, we find useful to present a MILP model for the problem with the aim of assessing in Section 4 the quality of the constructive heuristics proposed for small instances where the optima can be found.

To do so, we adapt the formulation proposed by Naderi et al. (2011b) with sequential notation for the open shop with sequence-dependent setup times and total completion time minimization. We consider a dummy job 0 preceding the first job on each machine. Hereafter, the notation used for the problem is presented.

Indices and sets
j: index for jobs $\{1,2,\ldots,n\}$.
k: index for jobs (including the dummy job 0) $\{0,1,2,\ldots,n\}$.
i, l: indices for machines $\{1,2,\ldots,m\}$.

Parameters
p_{ji}: processing time of job j in machine i (operation O_{ji}).
M: a large and positive number.

Decision variables
C_{ji}: completion time of job j in machine i.
C_{max}: makespan

$$Y_{jik} = \begin{cases} 1, & \text{if operation } O_{ji} \text{ is processed immediately after } O_{ki} \\ 0, & \text{otherwise} \end{cases}$$

$$X_{jil} = \begin{cases} 1, & \text{if operation } O_{ji} \text{ is processed after } O_{jl} \\ 0, & \text{otherwise} \end{cases}$$

The proposed MILP model is as follows.

$$\min C_{max} \tag{1}$$

subject to

$$\sum_{k=0,k\neq j}^{n} Y_{jik} = 1, \qquad \forall j,i \tag{2}$$

$$\sum_{j=1,j\neq k}^{n} Y_{jik} \leq 1, \qquad \forall i,k>0 \tag{3}$$

$$\sum_{j=1,j\neq k}^{n} Y_{ji0} = 1, \qquad \forall i \tag{4}$$

$$Y_{jik} + Y_{kij} \leq 1, \qquad \forall i,j<n,k>j \tag{5}$$

$$C_{ji} \geq C_{ki} + p_{ji} - (1 - Y_{jik}) \times M, \qquad \forall j,i,k,k\neq j \tag{6}$$

$$C_{ji} \geq C_{jl} + p_{ji} - (1 - X_{jil}) \times M, \qquad \forall j,i<m,l>i \tag{7}$$

$$C_{jl} \geq C_{ji} + p_{jl} - X_{jil} \times M, \qquad \forall j,i<m,l>i \tag{8}$$

$$C_{max} \geq C_{ji}, \quad \forall j, i \tag{9}$$

$$C_{0i} = 0, \quad \forall i \tag{10}$$

$$C_{max}, C_{ji} \geq 0, \quad \forall j, i \tag{11}$$

$$Y_{jik} \in \{0, 1\}, \quad \forall j, i, k \neq j \tag{12}$$

$$X_{jil} \in \{0, 1\}, \quad \forall j, i < m, l > i \tag{13}$$

The objective function (1) is the minimization of the makespan. Set of constraints (2) ensures that all the jobs are scheduled only once on each machine. Set of constraints (3) enforces that each job present at most one successor on each machine. Set of constraints (4) guarantees that the dummy job is preceding any other job on each machine. Set of constraints (5) avoids that a given job is simultaneously the predecessor and successor of another job. Constraints sets (6), (7), and (8) guarantee that the jobs are processed in the machines according to the previously defined processing times. Set of constraints (9) determines the makespan (maximum among all completion times). Constraint set (10) determines the completion time of dummy job. Finally, constraint sets (11), (12), and (13) determine the domain of the decision variables. The proposed model includes n^2m binary decision variables, $n \cdot m + 1$ continuous decision variables and $n \cdot m(\frac{1}{2} + \frac{3}{2}n + m) + n$ constraints.

3. Proposed algorithms

This section is devoted to present the new algorithms proposed for the problem under consideration. More specifically, in Section 3.1 we present three constructive heuristics for the problem using a beam search and cheapest insertion procedure, while in Section 3.2 we present an efficient local search procedure with reduction of the search space which can be initialized with any of the aforementioned constructive heuristics. The constructive heuristics start from the initial solutions obtained by an adaptation to our problem of the methods by Abreu et al. (2020) for the problem with setups, and are combined with an efficient beam search strategy and cheapest insertion. Beam search algorithms have been successfully applied to other scheduling environments (Ruiz and Stützle, 2008; Dong et al., 2008; Kizilay et al., 2019). However, to the best of our knowledge, beam search algorithms have not been tested in the open shop environment.

3.1. Constructive heuristics

In this section we present six constructive algorithms to solve the OSSP for makespan minimization. These heuristics adapt the algorithms proposed by Abreu et al. (2020) that presented high-quality results for the OSSP with sequence-dependent setup times to minimize the total completion time. In Section 3.1.1 we propose the Bounded Insertion Constructive Heuristic + Beam Search (BICH-BS) algorithm, which uses a projection of the makespan of the complete sequence for each step of the construction procedure to select the most promising partial sequence. The rationale of this heuristics is to reduce the solution search space by discarding sequences that would increase the lower bound and, consequently, the makespan of the solution in the short term. In Section 3.1.2 we present Minimal Idleness Heuristic + Beam Search (MIH-BS), a new constructive procedure that takes into consideration the minimization of the idleness of the machines in the production environment under study. The rationale of this heuristic is the following: the insertion of operations with less (local) idleness provides an increase in the utilization of the machines and consequently, it can potentially reduce the makespan at the end of the solution. In Section 3.1.3 we propose a method combining the two above-mentioned strategies. All these constructive methods presented are embedded in a beam-search procedure to explore the potential of using the constructive heuristics as a starting point of a local search procedure. Finally, in Section 3.1.4 we propose a hybridization of three proposed constructive heuristic by Abreu et al. (2020) with adaptation of cheapest insertion as improvement heuristic for OSSP.

3.1.1. Bounded Insertion Constructive Heuristic + Beam Search (BICH-BS)

The BICH-BS algorithm that we proposed for the OSSP is the result of the hybridization of two general approximate procedures (i.e. BICH and BS) adapted for the problem under consideration with a new procedure for the search space reduction based on the machine released earlier. We first give a brief description of these procedures and how they have been hybridized, and secondly we provide the pseudo-code with a detailed explanation.

The BICH procedure was first proposed by Fernandez-Viagas and Framinan (2015) for the permutation flowshop scheduling problem with makespan minimization (PFSP) subject to a maximum tardiness, and it can be considered a state-of-the-art algorithm for this problem. The BICH algorithm starts with an empty solution, and then, for each unscheduled operation, an estimation of the lower bound if the unscheduled operation is inserted is obtained. The operation with the best expected lower bound is selected and inserted in the current solution, and the algorithm continues constructing the solution until all operations have been inserted.

As it can be seen, the main idea of the BICH heuristic is to employ a mechanism to limit the number of solutions to be explored in the solution space, hence it seems particularly well-suited for combinatorial optimization problems with an extremely large number of feasible solutions, such as the problem under consideration. An adaptation of the BICH for the open-shop scheduling problem with sequence-dependent setups has been proposed by Abreu et al. (2020). Therefore, for our problem we propose a procedure that uses this heuristic (where setup times are considered to be zero) as initial solution and then the so-obtained solution is improved using a beam search (BS) strategy. The BS is a search algorithm based on nodes search, very similar to Branch and Bound, but only the best β nodes are selected for expansion, thus consuming less computational time as only a subset of nodes from the set of all possible solutions to the problem are explored (Birgin et al., 2020).

The main elements of both BICH and BS are adapted to our problem. For the BICH, the operation returning the lowest expected lower bound is selected. For the BS, for each iteration, several operations are considered to be inserted in the solution. The domain knowledge of the problem is the insertion of operations based on the expected lower bound of the problem. This hedge prevents placing operations in positions that contribute negatively to the expected lower bound.

The lower bound for the open shop required by BICH is calculated using the well-known Eq. (14) (Pinedo, 2016):

$$LB = \max\left\{\max_{j\in\{1,\ldots,n\}}\sum_{i=1}^{m} p_{ij}, \max_{i\in\{1,\ldots,m\}}\sum_{j=1}^{n} p_{ij}\right\} \tag{14}$$

One can observe that this lower bound can be computed with low computational effort. In addition, the computation of the expected lower bound required for BS can be also performed in a fast manner. First, we must calculate the expected contribution to the makespan (EMC) with the addition of the operation of job j in machine k (π_{kj}) in a partial solution Π, using a matrix of processing times P where the processing times of the operations previously inserted in the partial solution Π are equal to zero, and also that of the operation π_{kj} for the EMC calculation ($P_{kj} = 0$). Eq. (15) presents EMC calculation.

$$EMC(\pi_{kj}, P) = \max\left\{\max_{l\in\{1,\ldots,n\}}\sum_{i=1}^{m} P_{il}, \max_{i\in\{1,\ldots,m\}}\sum_{l=1}^{n} P_{il}\right\}, \text{with} P_{kj} = 0 \tag{15}$$

Therefore, using ECM, the expected lower bound can be calculated with the makespan of the operations presented in the partial solution Π with the operation π_{kj}, adding in the value of the makespan, the ECM considering the insertion of the operation π_{kj} in the partial solution. Finally, we present the expected lower bound of insertion of operation π_{kj} in the partial solution Π in Eq. (16).

$$makespan(\Pi \cup \{\pi_{kj}\}, p) + ECM(\pi_{kj}, P) \tag{16}$$

where Π is a (partial) sequence of operations, π_{kj} is the operation of job j in machine k, p is a default processing time matrix, and P is a processing time matrix with the processing time of the operation π_{kj} and all other allocated in Π equal to zero. The main feature regarding the hybridization of BS and BICH is the fact that the best β operations are tested concerning the expected lower bound in each iteration. However, in the classic BICH, a single solution is constructed for each iteration through the selection of the best operation; thereby, this algorithm does not evaluate solutions with the insertions of different operations. In contrast, in BICH-BS, the search tree adds the new nodes, and in the next iterations, the insertion of β operations in each of the generated nodes is tested. At the end of the algorithm, the so-built node with the lowest makespan is returned.

Note that, since forcing the BICH-BS algorithm to select the best β nodes from all the existing set may demand a high computational cost, we propose a new local search mechanism (LS) that performs an initial filter. More specifically, the filter chooses only operations that contain the machine that is released earlier. We considered the operation π_{kj} to calculate the expected lower bound, where k is the index of the machine released earlier. In this manner, the search is improved by reducing the number of operations whose expected lower bound has to be computed.

The complete pseudo code of the proposed algorithm BICH-BS is shown in Fig. 3. In the pseudocode, EMC is a function that calculates the expected makespan contribution, with a processing times matrix P and considering the time of candidate operation π_{kj} equal to zero so the time of this operation in the partial lower bound is not considered. If BICH selects this operation, it is inserted in the solution, and its processing time in the P matrix will be equal to zero. Therefore, this operation will not be considered in the next iterations. Finally, p is an example of an instance as presented in Table 1. The algorithm returns a sequence of operations $\Pi := \{\pi_{11}, \pi_{13}, \ldots, \pi_{mn}\}$ as a solution for the OSSP.

In lines 1–8 of the pseudo-code, the main parameters are initialized, including the $\mathcal{N}$ tree with the starting node with the empty parameters. $\mathcal{N}$ is a list of tuples where each tuple represents a node of a partial solution constructed in each iteration with the insertion of an operation into the sequence Π. N_i denotes a node of a partial solution with an index of i, z is a counter for the number of nodes created in each iteration of BS, and N_z is the node of the last partial solution created. Line 9 corresponds to the main loop of the algorithm, while all operations are not allocated to the last created node, the algorithm's steps must be executed. Lines 15–16 select the operations to be tested for the expected lower bound, consider only the possible operations to be programmed on the machine released earlier, with jobs still available for programming in this machine.

Lines 17–19 creates the new nodes for each candidate operation to be inserted into the solution in the current node of the for loop in line 11. The best node operation is inserted in the current node on lines 30–33. In lines 35–40, the best new nodes found are selected to be added to the $\mathcal{N}$ tree. Line 42 selects the best solution found from all the nodes that have been created by BICH-BS.

3.1.2. Minimal Idleness Heuristic + Beam Search (MIH-BS)

In this section we propose the MIH-BS algorithm, which is the results of hybridizing the MIH heuristic (Abreu et al., 2020) for the open-shop problem with setups with the BS strategy. First we describe the MIH heuristic and then we describe its adaptation and hybridization with the BS.

MIH is a heuristic procedure which relies in the idea that classical dispatching rules for the OSSP are largely based on LPT algorithms to sort operations in descending order of their processing times. In view of the similarities of the open-shop with the parallel machine environment, the allocation of jobs with the longest processing times using LPT might be an interesting strategy. However, this may cause a high idleness time when applied to the open shop: While in the parallel machine environment this idleness is zero, in the open shop it can increase the waiting time of a given solution. The idea of the MIH is that partial solutions with low values of cumulative processing times would usually present a low makespan in final solution. Therefore, an indicator for idleness can be calculated by the accumulated times for jobs and machines in the production system over the execution of operations in the scheduling sequence. If the cumulative time for a given job in the system is greater than the accumulated 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 cumulative time of this job is lower than the cumulative time of a given machine, this job was already processed in another machine and its processing in the current machine will not result in idleness (Abreu et al., 2020). Φ_{ij} represents the idleness generated by the allocation of job j to machine i. M and J store The cumulative processing times for each machine and job are stored in M_i and J_j respectively, and both are updated every time a new operation is inserted into the sequence. Eq. (17) presents the procedure to calculate idleness.

$$\Phi_{ij} = \begin{cases} J_j - M_i, & \text{if } J_j > M_i \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

The MIH algorithm starts with an empty solution, and all unscheduled operations are inserted and their idleness is computed. The operation that results in the lowest idleness is inserted, and the algorithm continues constructing the solution until all the operations have been inserted. In the case of a tie, the decision is arbitrary. When operations have the same idleness, for tie-breaking, the operation with the lowest index is selected.

For our problem, the MIH originally proposed for the open-shop with setups is adapted. The main difference with the BS hybridization presented before is that the best β operations are tested with respect to their idleness in each iteration. The new nodes are added to the search tree, and in the next iterations, the insertion of β operations in each of the generated nodes is tested. At the end of the algorithm, the node with the lowest makespan is returned.

Furthermore, for the MIH-BS algorithm selecting the best β nodes from all the existing sets may demand a high computational cost. To overcome this problem we suggest the same mechanisms of the BICH-BS algorithm in Section 3.1.1, i.e. the operations selected to calculate the expected idleness are only the operations present in the machine released earlier in order to optimize the search by reducing the number of operations in which the expected idleness has to be computed.

Fig. 4 shows the complete pseudo code for MIH-BS. In the algorithm, Φ_{ij} the idleness of operation π_{ij}, and p is a example of instance like in Table 1. As it can be seen, the structure of the algorithm is similar to that of BICH-BS. The main differences refer to lines 15–16 (where the operations to be tested are selected based on the expected idleness, considering only the possible operations programmed in the machine released earlier), and line 42 where the best solution found among all nodes that have been created by MIH-BS is selected.

3.1.3. A combined approach + Beam Search (BICH-MIH-BS)

Framinan and Perez-Gonzalez (2017) present a constructive heuristic in which there is a look-ahead procedure for measure the potential contribution of the candidate operations to the objective function and an estimation of the contribution to the objective function of the non-scheduled operations in the sequence solution. This look-ahead mechanism is the main feature in our combined approach, taking into account the makespan lower bound, the machine's idleness, as well as the BS scheme.

On the basis of such reasoning, we develop a constructive heuristic that adapts MIH and BICH taking into consideration the contribution of an operation for the makespan objective as well as the idleness indicator with a beam-search procedure. We adopt a weight aggregation function for combining the two objectives, i.e. idleness minimization and makespan minimization.

Data: EMC(.), p, β
Result: A sequence $\Pi_{best} := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$
$\Pi \leftarrow \{\}$;
$P \leftarrow$ 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 alocated in machine k, $\forall k \in \{1, ..., m\}$;
$\mathcal{N} \leftarrow$ set of nodes for solution, each node is a tuple; // $\mathcal{N}_i \leftarrow (\Pi, M, J, \Omega, P)$.
$z \leftarrow 1$; // number of nodes created in search.
$\mathcal{N} \leftarrow \mathcal{N} \cup (\Pi, M, J, \Omega, P)$
while $\|\mathcal{N}_z.\Pi\| < n \times m$ **do**
  new_nodes $\leftarrow \{\}$; // set of new nodes created.
  foreach *node* $i \in \mathcal{N}$ **do**
    if $\|\mathcal{N}_i.\Pi\| < n \times m$ **then**
      continue search for next node, this node i was completed;
    end
    machine $k \leftarrow \underset{r \in \{1,...,m\}, \|\mathcal{N}_i.\Omega_r\| < n}{\text{argmin}} M_r$; // index from machine released more early.
    $\mathcal{J} \leftarrow$ list of jobs j sort by makespan$(\mathcal{N}_i.\Pi \cup \{\pi_{kj}\}, p) + EMC(\pi_{kj}, P)$ with $j \notin \mathcal{N}_i.\Omega_k$;
    foreach $j \in \mathcal{J}$ **do**
      if j *is the first job in the list* $\mathcal{J}$ **then**
        w $\leftarrow j$; // The node $\mathcal{N}_i$ continues the insertion with the best job j by BICH criteria.
      else
        $\Pi^{'} \leftarrow \mathcal{N}_i.\Pi \cup \{\pi_{kj}\}$;
        $\Omega^{'}_k \leftarrow \mathcal{N}_i.\Omega_k \cup \{j\}$;
        $P^{'} \leftarrow copy(\mathcal{N}_i.P)$;
        $P^{'}_{kj} \leftarrow 0$;
        $J^{'}, M^{'} \leftarrow \mathcal{N}_i.J, \mathcal{N}_i.M$;
        update $J^{'}$ and $M^{'}$ with time of π_{kj} operation;
        new_nodes $\leftarrow$ new_nodes $\cup$ $(\Pi^{'}, M^{'}, J^{'}, \Omega^{'}, P^{'})$;
      end
    end
    $\mathcal{N}_i.\Pi \leftarrow \mathcal{N}_i.\Pi \cup \{\pi_{kw}\}$;
    $\mathcal{N}_i.\Omega_k \leftarrow \mathcal{N}_i.\Omega_k \cup \{w\}$;
    $\mathcal{N}_i.P_{kw} \leftarrow 0$;
    update $\mathcal{N}_i.J$ and $\mathcal{N}_i.M$ with time of π_{kw} operation;
  end
  best_nodes $\leftarrow$ the β best nodes $\in$ new_nodes by makespan;
  foreach *node* $i \in$ *best_nodes* **do**
    $z \leftarrow z + 1$; // A new node is create.
    extract Π, M, J, Ω, P from node i;
    $\mathcal{N} \leftarrow \mathcal{N} \cup (\Pi, M, J, \Omega, P)$;
  end
end
$\Pi_{best} \leftarrow$ the best solution found $\in \mathcal{N}_i.\Pi \ \forall i \in \{1, ..., z\}$;

Fig. 3. Pseudocode of the BICH-BS heuristic.

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, Φ_{ij} the expected contribution for the idleness minimization, p is a default processing time matrix, and P is a processing time matrix with the processing time of the operation π_{ij} and all other allocated in Π equal to zero. The performance indicator Ψ_{ij} can be computed as follows:

$$\Psi_{ij} = (1 - \alpha) \times (makespan(\Pi \cup \{\pi_{ij}\}, p) + EMC(\pi_{ij}, P)) + \alpha \times \Phi_{ij} \quad (18)$$

If $\alpha = 1$ the combined approach is equal to MIH and if $\alpha = 0$ the combined approach is equal to BICH. This heuristic need finding the best empirical α to solve the proposed problem.

As it can be seen, the algorithm is similar to the ones presented in Sections 3.1.1 and 3.1.2, just changing the selection criteria to Eq. (18) in order to select the operation to be inserted in the solution.

Therefore, BICH-MIH starts with an empty solution, and all unscheduled operations are considered by calculating their hybrid indicator. The operation resulting in the smallest Ψ is inserted, and the algorithm

Data: Φ, p, β
Result: A sequence $\Pi_{best} := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$
$\Pi \leftarrow \{\}$;
$P \leftarrow \text{copy}(p)$;
$M \leftarrow$ list with time cumulative in each machine;
$J \leftarrow$ list with time cumulative in each job;
$\Omega_k \leftarrow$ list with the jobs alocated in machine k, $\forall k \in \{1, ..., m\}$;
$\mathcal{N} \leftarrow$ set of nodes for solution, each node is a tuple; // $\mathcal{N}_i \leftarrow (\Pi, M, J, \Omega, P)$.
$z \leftarrow 1$; // number of nodes created in search.
$\mathcal{N} \leftarrow \mathcal{N} \cup (\Pi, M, J, \Omega, P)$
while $\|\mathcal{N}_z.\Pi\| < n \times m$ **do**
  new_nodes $\leftarrow \{\}$; // set of new nodes created.
  foreach *node* $i \in \mathcal{N}$ **do**
    if $\|\mathcal{N}_i.\Pi\| < n \times m$ **then**
      continue search for next node, this node i was completed;
    end
    machine $k \leftarrow \underset{r \in \{1,...,m\}, \|\mathcal{N}_i.\Omega_r\| < n}{\text{argmin}} M_r$; // index from machine released more early.
    $\mathcal{J} \leftarrow$ list of jobs j sort by Φ_{kj} with $j \notin \mathcal{N}_i.\Omega_k$;
    foreach $j \in \mathcal{J}$ **do**
      if j *is the first job in the list* $\mathcal{J}$ **then**
        w $\leftarrow j$; // The node $\mathcal{N}_i$ continues the insertion with the best job j by MIH criteria.
      else
        $\Pi^{'} \leftarrow \mathcal{N}_i.\Pi \cup \{\pi_{kj}\}$;
        $\Omega^{'}_k \leftarrow \mathcal{N}_i.\Omega_k \cup \{j\}$;
        $P^{'} \leftarrow copy(\mathcal{N}_i.P)$;
        $P^{'}_{kj} \leftarrow 0$;
        $J^{'}, M^{'} \leftarrow \mathcal{N}_i.J, \mathcal{N}_i.M$;
        update $J^{'}$ and $M^{'}$ with time of π_{kj} operation;
        new_nodes $\leftarrow$ new_nodes $\cup$ $(\Pi^{'}, M^{'}, J^{'}, \Omega^{'}, P^{'})$;
      end
    end
    $\mathcal{N}_i.\Pi \leftarrow \mathcal{N}_i.\Pi \cup \{\pi_{kw}\}$;
    $\mathcal{N}_i.\Omega_k \leftarrow \mathcal{N}_i.\Omega_k \cup \{w\}$;
    $\mathcal{N}_i.P_{kw} \leftarrow 0$;
    update $\mathcal{N}_i.J$ and $\mathcal{N}_i.M$ with time of π_{kw} operation;
  end
  best_nodes $\leftarrow$ the β best nodes $\in$ new_nodes by makespan;
  foreach *node* $i \in$ *best_nodes* **do**
    $z \leftarrow z + 1$; // A new node is create.
    extract Π, M, J, Ω, P from node i;
    $\mathcal{N} \leftarrow \mathcal{N} \cup (\Pi, M, J, \Omega, P)$;
  end
end
$\Pi_{best} \leftarrow$ the best solution found $\in \mathcal{N}_i.\Pi$ $\forall i \in \{1, ..., z\}$;

Fig. 4. Pseudocode of the MIH-BS heuristic.

continues the construction of the solution until all the operations have been inserted.

The main difference with the BS hybridization is that, in each iteration, the best β operations are tested with respect to the hybrid criterion in Eq. (18). The new nodes are added to the search tree and, in the next iterations, the insertion of β operations in each of the generated nodes is tested. At the end of the algorithm, the node with the lowest makespan is returned.

For the MIH-BS algorithm selecting the best β nodes from the entire existing set may require a high computational cost. To overcome this problem we use the same mechanisms of the BICH-BS and MIH-BS algorithms proposed in Sections 3.1.1 and 3.1.2, respectively. The operations selected to calculate the hybrid indicator Ψ are only operations present in the machine released earlier to optimize the search by reducing the number of operations to be computed.

Fig. 5 shows the complete pseudo code for BICH-MIH-BS. In the pseudo code, As discussed previously, $EMC(.)$ calculates the expected makespan contribution of instance, Ψ_{ij} is the indicator combining the expected makespan and the idleness of inserting operation π_{ij}, and p is a example of instance like in Table 1.

Data: $EMC(.)$, Φ, Ψ, p, β, α
Result: A sequence $\Pi_{best} := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$
1 $\Pi \leftarrow \{\}$;
2 $P \leftarrow$ copy(p);
3 $M \leftarrow$ list with time cumulative in each machine;
4 $J \leftarrow$ list with time cumulative in each job;
5 $\Omega_k \leftarrow$ list with the jobs alocated in machine k, $\forall k \in \{1, ..., m\}$;
6 $\mathcal{N} \leftarrow$ set of nodes for solution, each node is a tuple; // $\mathcal{N}_i \leftarrow (\Pi, M, J, \Omega, P)$.
7 $z \leftarrow 1$; // number of nodes created in search.
8 $\mathcal{N} \leftarrow \mathcal{N} \cup (\Pi, M, J, \Omega, P)$
9 **while** $\|\mathcal{N}_z.\Pi\| < n \times m$ **do**
10 new_nodes $\leftarrow \{\}$; // set of new nodes created.
11 **foreach** *node* $i \in \mathcal{N}$ **do**
12 **if** $\|\mathcal{N}_i.\Pi\| < n \times m$ **then**
13 continue search for next node, this node i was completed;
14 **end**
15 machine $k \leftarrow \underset{r \in \{1,...,m\}, \|\mathcal{N}_i.\Omega_r\| < n}{\text{argmin}} M_r$; // index from machine released more early.
16 $\mathcal{J} \leftarrow$ list of jobs j sort by Ψ_{kj} with $j \notin \mathcal{N}_i.\Omega_k$;
17 **foreach** $j \in \mathcal{J}$ **do**
18 **if** j *is the first job in the list* $\mathcal{J}$ **then**
19 w $\leftarrow j$; // The node $\mathcal{N}_i$ continues the insertion with the best job j by BICH-MIH criteria.
20 **else**
21 $\Pi^{'} \leftarrow \mathcal{N}_i.\Pi \cup \{\pi_{kj}\}$;
22 $\Omega_k^{'} \leftarrow \mathcal{N}_i.\Omega_k \cup \{j\}$;
23 $P^{'} \leftarrow copy(\mathcal{N}_i.P)$;
24 $P_{kj}^{'} \leftarrow 0$;
25 $J^{'}, M^{'} \leftarrow \mathcal{N}_i.J, \mathcal{N}_i.M$;
26 update $J^{'}$ and $M^{'}$ with time of π_{kj} operation;
27 new_nodes $\leftarrow$ new_nodes $\cup$ $(\Pi^{'}, M^{'}, J^{'}, \Omega^{'}, P^{'})$;
28 **end**
29 **end**
30 $\mathcal{N}_i.\Pi \leftarrow \mathcal{N}_i.\Pi \cup \{\pi_{kw}\}$;
31 $\mathcal{N}_i.\Omega_k \leftarrow \mathcal{N}_i.\Omega_k \cup \{w\}$;
32 $\mathcal{N}_i.P_{kw} \leftarrow 0$;
33 update $\mathcal{N}_i.J$ and $\mathcal{N}_i.M$ with time of π_{kw} operation;
34 **end**
35 best_nodes $\leftarrow$ the β best nodes $\in$ new_nodes by makespan;
36 **foreach** *node* $i \in$ *best_nodes* **do**
37 $z \leftarrow z + 1$; // A new node is create.
38 extract Π, M, J, Ω, P from node i;
39 $\mathcal{N} \leftarrow \mathcal{N} \cup (\Pi, M, J, \Omega, P)$;
40 **end**
41 **end**
42 $\Pi_{best} \leftarrow$ the best solution found $\in \mathcal{N}_i.\Pi$ $\forall i \in \{1, ..., z\}$;

Fig. 5. Pseudocode of the BICH-MIH-BS heuristic.

The algorithm is similar to BICH-BS and MIH-BS. The main differences are in lines 15–16, where the operations to be tested is selected based on the combined indicator of makespan and the expected idleness, considering only the possible operations to be programmed on the machine released earlier, with jobs still available for programming in this machine. Line 42 selects the best solution found among all the nodes that have been created by BICH-MIH-BS.

3.1.4. Constructive heuristics with Cheapest Insertion(IST)

The cheapest insertion heuristics is a constructive heuristic used in many production scheduling problems (see e.g. Wu and Che (2020), or Rossi and Nagano (2020)). It starts with a pre-established sequence of operations and it constructs a solution by inserting the unscheduled operations one by one in an iterative manner. Each operation is inserted in the position where it obtains the best value of the objective function (cheapest insertion). IST is a greedy constructive heuristic, being of simple implementation in the most diverse scheduling problems.

```
   Data: W, p
   Result: A sequence Π_best := {π_11, π_12, ..., π_mn}
1  Π ← {};
2  W ← a solution ordered by a constructive heuristic as BICH, MIH or BICH-MIH;
3  best_make ← best makespan generated in each interation;
4  best_pos ← best position found by insertion πij in solution;
5  while ||W|| > 0 do
6      π_ij ← first operation ∈ W;
7      best_make ← ∞
8      foreach position pos ∈ Π do
9          Π' ← a solution with insertion of π_ij in position pos in Π;
10         if makespan(Π', p) < best_make then
11             best_make ← makespan(Π', p);
12             best_pos ← pos;
13         end
14     end
15     Π ← solution with insertion of of π_ij in position best_pos;
16     W ← W/{π_ij};
17 end
```

Fig. 6. Pseudocode of the cheapest insertion heuristic.

Here we propose hybridizing the BICH, MIH, and BICH-MIH described above with the improvement of the solutions through the IST heuristics, with the calculation of the makespan using a decoding scheme, explained in Fig. 8.

More specifically, the algorithm starts with a list of operations sorted according to one of the three criteria: BICH, MIH, or BICH-MIH. Then, the operation not yet allocated in the solution, present in the list of ordered operations, is tested in each possible positions in the solution. The chosen position is that where the best makespan is obtained. The algorithm finishes when all the operations have been inserted into the solution.

The complete pseudo-code of the cheapest insertion heuristics adapted for the OSSP is shown in Fig. 6 where W is the list of operations sorted by some criteria and p is a sample instance as in Table 1. The Algorithm returns a solution $\Pi_{best} := \{\pi_{11}, \pi_{12}, \ldots, \pi_{mn}\}$.

Lines 1–4 set the parameters for the execution of the algorithm. Line 5 corresponds to the algorithm's main while loop: while the list of ordered operations is not empty, the algorithm runs. Lines 6–7 select the operation to be tested in the current iteration. In lines 8–14, the operation is tested in all possible positions in the Π solution. In lines 15–16, the operation is inserted in the best position found.

3.2. Local search

As already mentioned, since the search space of the OSSP is very large, neighborhood search algorithms play a key role for finding high-quality solutions, mainly for large-sized instances. The local search applied is the well-known 2-opt algorithm, with a best improvement strategy. This local search procedure includes the search space reduction mechanisms based on the theorems proposed by Naderi et al. (2010) aiming to reduce the movements that generate redundant solutions.

The search consists in the pairwise exchange (swap) of a given operation and the rest, respecting the redundancy constraints. When all feasible exchanges are performed, there are two possibilities: (a) if there is no improvement, the next operation in the sequence is selected; (b) otherwise, the swap that generates the best makespan is selected. Thereafter, the search is restarted so the procedure stops when all the feasible swaps are evaluated.

In Fig. 7 the proposed local search is presented. In this figure, makespan refers to the objective function, redundancy is a function that returns whether a swap between two operations is redundant or not based in Naderi et al. theorems (Naderi et al., 2010), Π is a solution that will receive a local search and p is a example of instance such as illustrated in Table 1.

The solution decoding used for the algorithms with local search procedures is different from the previous one used in constructive algorithms. The main change is the consideration of a non-delay schedule in makespan calculation, which consists in the minimization of the idle time of machines. With this type of decoding, a given machine is not kept idle if there are still jobs to be processed, thus no machine is kept idle at a time when it could start processing other operations (Sha and Hsu, 2008). The decoded solutions with non-delay have an equal or better makespan than solutions decoded without non-delay. With non-delay decoding, multiple permutations get the same makespan value, which can reduce the search space of the solutions, improving the efficiency of the local search algorithms (Naderi et al., 2010). Therefore, this decoding scheme prioritizes the processing of the operations with the earliest starting time in each iteration of the makespan calculation. This decoding is not used in the iterations of constructive heuristics due to its high computational requirements.

This decoding always prioritizes scheduling first the operations with lower start time s_{ij}. Where there is more than one operation with the same start time, the operation π_{zd} with the earliest relative position is prioritized. Fig. 8 presents the decoding scheme used in the proposed local search.

4. Computational results

The proposed constructive heuristics are evaluated using the literature test problems proposed by Taillard (1993), Guéret and Prins (1999) and Bruckner et al. (1997), which are the usual testbeds employed in OSSP. In the test problems by Guéret and Prins a fixed interval for the processing times is considered, with random values uniformly distributed between 1 and 1000, and a constant value for the lower bound equal to 1000. Different problem sizes are considered with $n, m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$. For each class we have randomly generated 10 test instances, totaling 80 instances. Brucker et al. test problems are generated with random values between 1 and 500 uniformly distributed and 6 sets of problem sizes $n, m \in \{3, 4, 5, 6, 7, 8\}$, totaling 60 instances. Taillard test problems were generated with random values uniformly distributed between 1 and 100, without a lower bound constraint. Problem classes were considered according to the combination of the 6

Data: redundancy(.), Π, p
Result: A new sequence W
$W \leftarrow$ an sequence Π generated by construtive heuristics;
$BestMake \leftarrow makespan(W)$;
$r \leftarrow m \times n$;
improvemment $\leftarrow$ True;
while $r > 0$ **do**
  if $improvemment = False$ **then**
    r^{--}
  end
  improvemment $\leftarrow$ False;
  $\pi_{ij} \leftarrow$ operation in r position in W;
  for $r2 = 1$ ***to*** $m \times n$ **do**
    $\pi_{zd} \leftarrow$ operation in $r2$ position in W;
    if $redundancy(\pi_{ij}, \pi_{zd}) = False$ **then**
      $WW \leftarrow$ solution with swap between π_{ij} and π_{zd} operations;
      if $makespan(WW, p) < BestMake$ **then**
        improvemment $\leftarrow$ True;
        $BestMake \leftarrow makespan(WW, p)$;
        $r3 \leftarrow r2$;
      end
    end
    if $improvemment = True$ **then**
      $\pi_{zd} \leftarrow$ operation in $r3$ position in W;
      $W \leftarrow$ solution with swap between π_{ij} and π_{zd} operations; `// the best swap founded in search`
      $r \leftarrow m \times n$; `// the search restarts`
    end
  end
end

Fig. 7. Local search with reduction of search space.

Data: A solution with sequence Π
Result: A sequence $S := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$ with encoding scheme
$S \leftarrow \{\}$;
$U \leftarrow \Pi$;
$M \leftarrow$ list with time acumulated in each machine;
$J \leftarrow$ list with time acumulated in each job;
$s_{ij} \leftarrow$ start time for processing of operation π_{ij}
while $\|U\| > 0$ **do**
  $y \leftarrow \min \left\{ \max_{i \in \{1,...,m\}} M_i, \max_{k \in \{1,...,n\}} J_k \right\}$;
  $R = \{\pi_{ij} | s_{ij} = y, \pi_{ij} \in U\}$;
  $\pi_{zd} \leftarrow$ 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

Fig. 8. Non-delay schedule decoding scheme procedure.

sets of problem size $n, m \in \{4, 5, 7, 10, 15, 20\}$. For each size, 10 instances are randomly generated, totaling 60 instances. The complete instances set has 192 instances.

The above mentioned algorithms were implemented in the Intel® Distribution for Python* integrated development environment https://software.intel.com/content/www/us/en/develop/tools/distribution-for-python.html and were run in C with Cython library http://cython.org/ (Behnel et al., 2011). The computational experience was performed on a PC with Intel Core i7-4771 CPU 3.50 GHz and 12 GB memory. The results of all computational tests and statistical analyzes are available in https://bit.ly/34OUhVg.

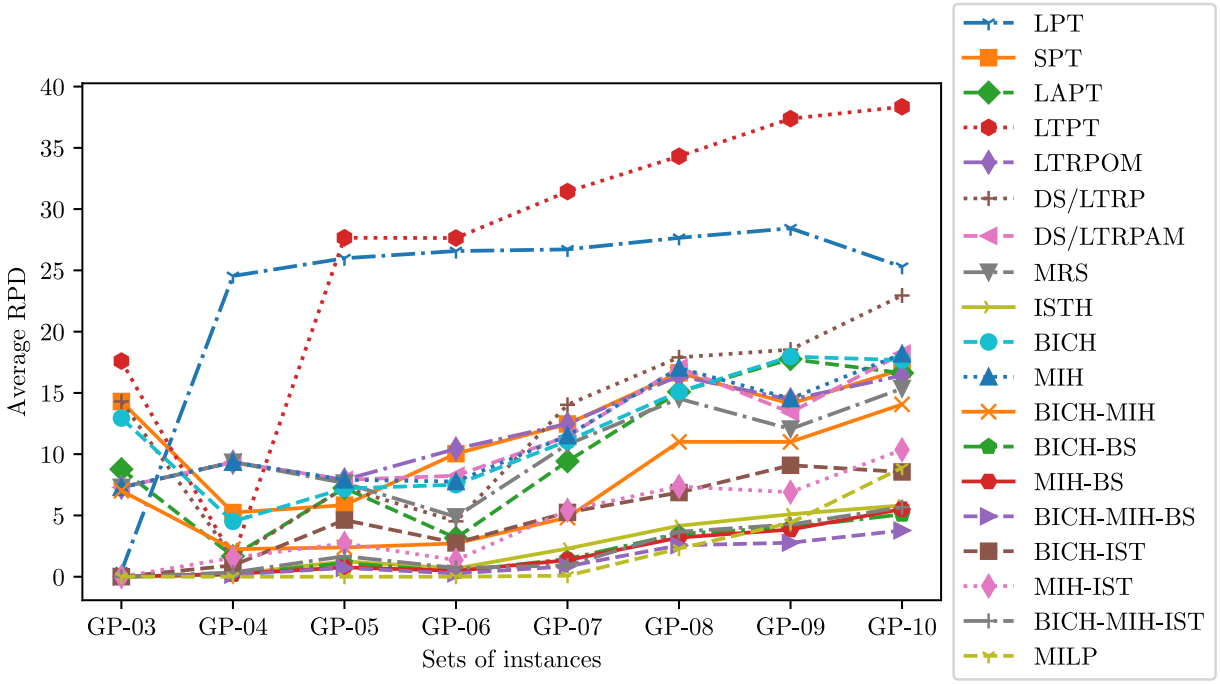


Fig. 9. Benchmark of constructive heuristics for each set of instances proposed by Guéret and Prins (1999).

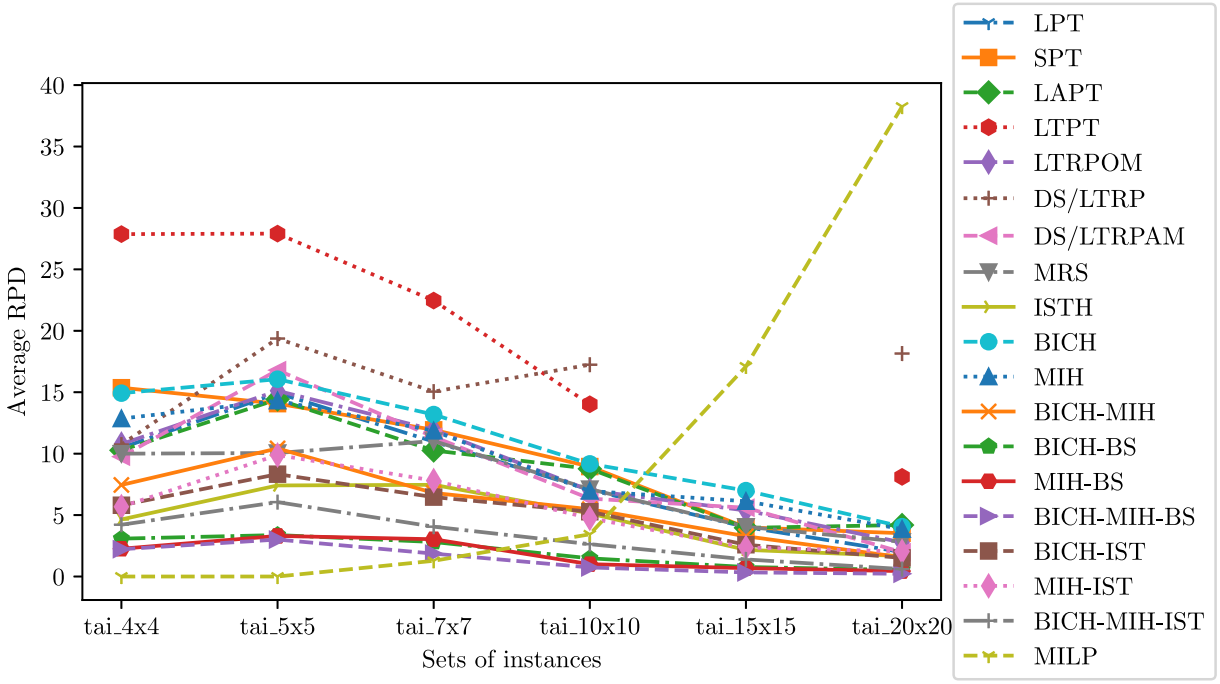


Fig. 10. Benchmark of constructive heuristics for each set of instances proposed by Taillard (1993).

The parameter α of the proposed algorithm was tuned after several simulations. For each instance size, a value of α contained in the set $A = \{0.00,0.11,0.22,0.33,0.44,0.56,0.67,0.78,0.89,1.00\}$ was tested. The best α values for BICH-MIH are presented in the Table 3 for each problem size. In the small-sized instances, the α value is close to 1 and the combined constructive heuristic allocates operations prevailing the reduction of idleness. In the large-sized instances, the α value is close to 0 and the constructive heuristic allocates operations prevailing bounded insertion. The parameter β is set as $\beta = 4$ after preliminary calibration experiments with $\beta = \{1, 2, \ldots 10\}$, since this value was found to be a good trade-off between solution quality and computational times in all problem sizes tested.

Table 3
The α values used for constructive heuristics BICH-MIH in each problem size.

Problem size	3	4	5	6	7	8	9	10	15	20
α	0.89	0.89	0.56	0.11	0.22	0.22	0.67	0.67	0.89	0.22

The statistic used in the analysis of the computational experiments is the gap between the evaluated method (sol_{ik}) and best known solution

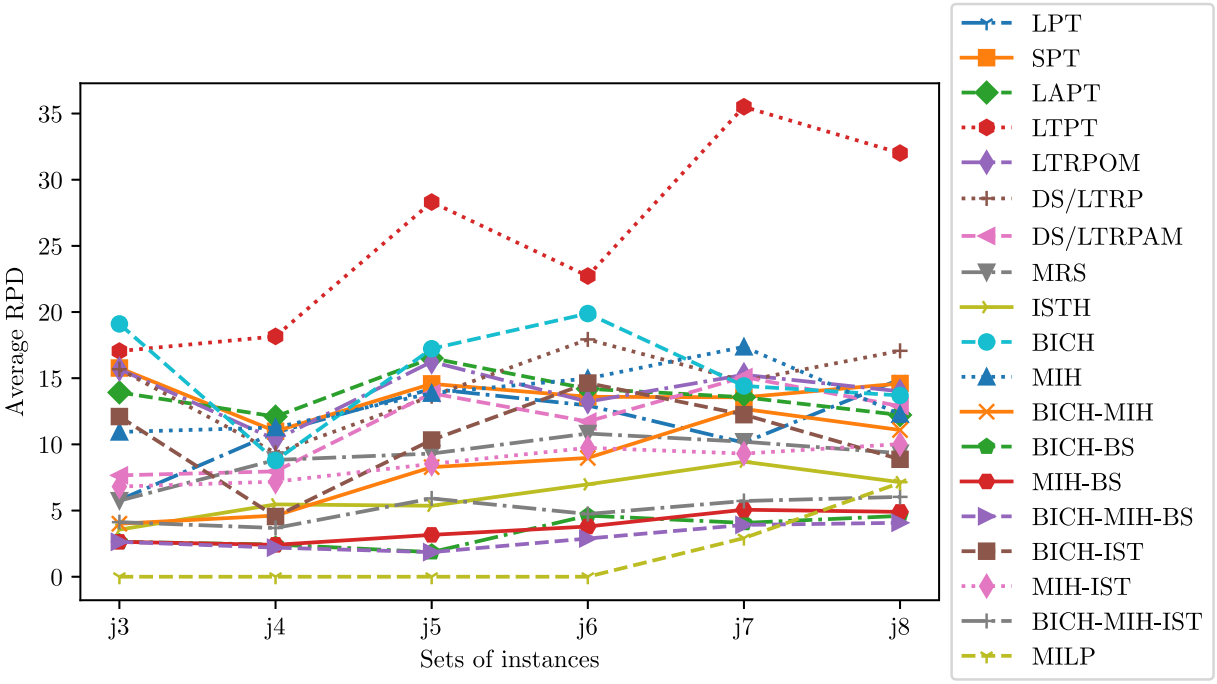


Fig. 11. Benchmark of constructive heuristics for each set of instances proposed by Bruckner et al. (1997).

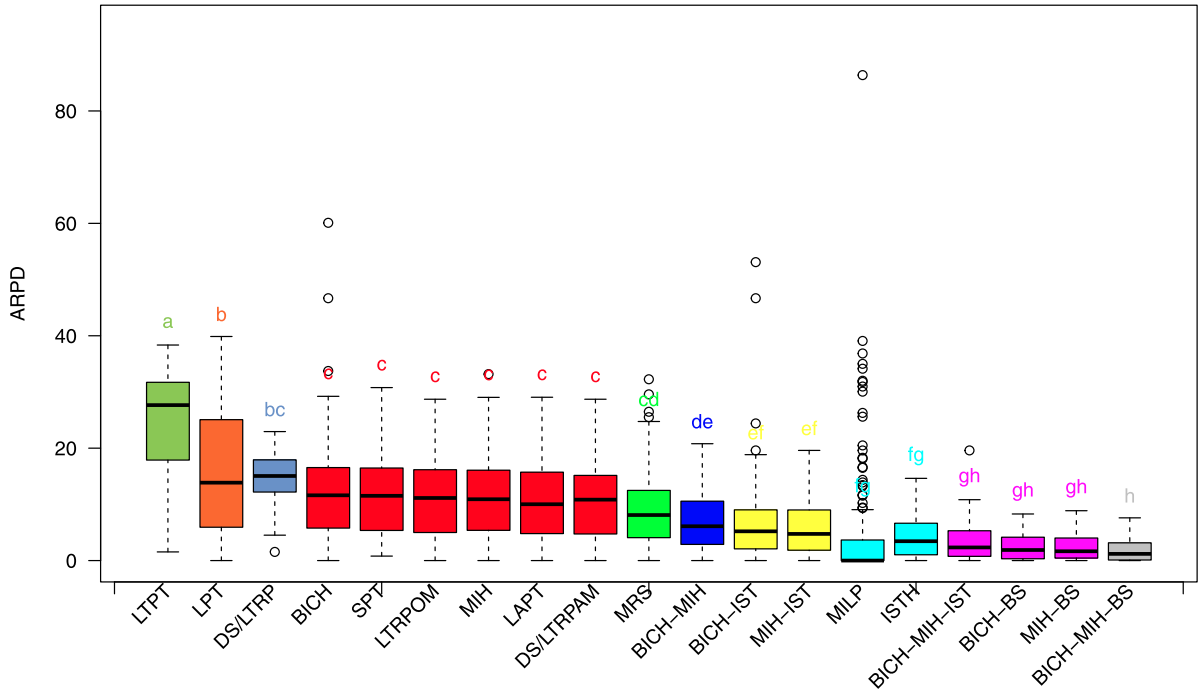


Fig. 12. Boxplot and Tukey HSD groups at the 95% confidence level for the constructive heuristics in all sets of instances.

(BKS_i), as presented in Eq. (19). The value sol_{ik} meaning the solution obtained by method k run on instance i, and BKS_i denotes the best known solution for instance i.

$$RPD_{ik} = \frac{sol_{ik} - BKS_i}{BKS_i} \cdot 100 \tag{19}$$

4.1. Computational results for constructive heuristics

Initially, we consider in our analysis constructive algorithms for the OSSP. The considered algorithms are listed below. All constructive algorithms used the decoding scheme procedure for the computation of the makespan of the final solution.

- Longest Processing Time (LPT): sort operations in decreasing order of their processing times
- Shortest Processing Time (SPT): sort operations in non-decreasing order of their processing times.
- Longest Alternate Processing Times (LAPT): The priority rule developed by Pinedo (2016) for the OSSP.

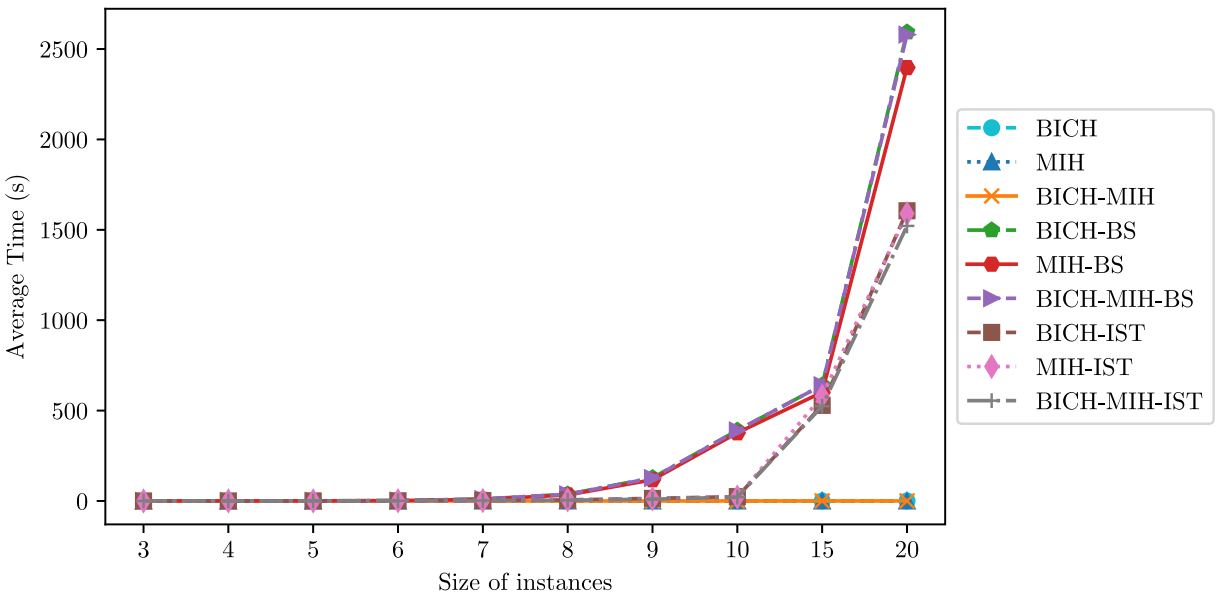


Fig. 13. Dependence between solution time and problem size for the constructive heuristics.

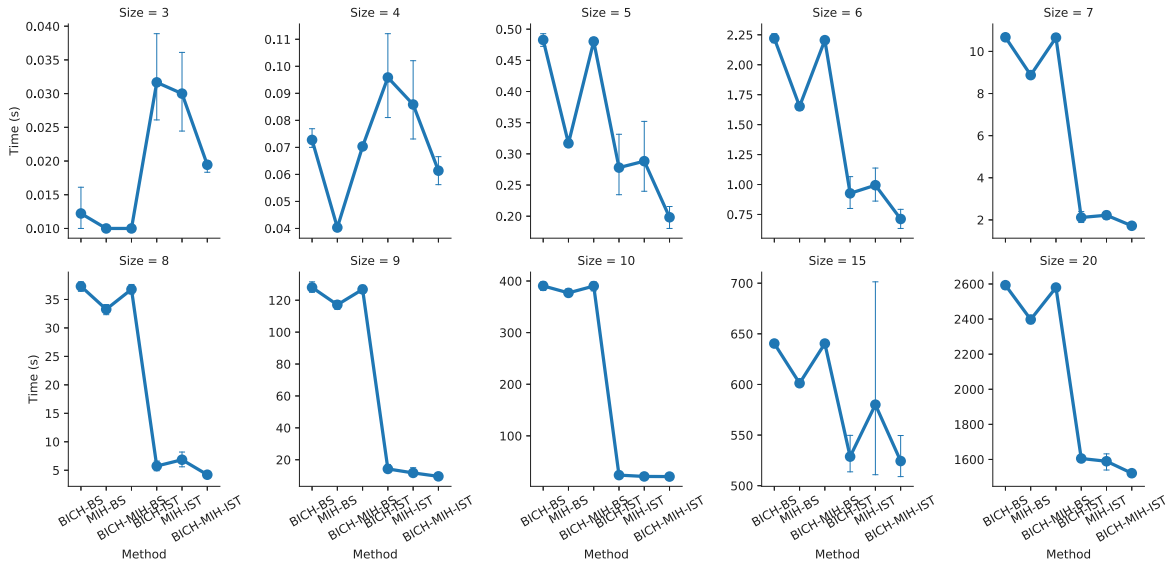


Fig. 14. Average computation times for the constructive heuristics.

- Longest total processing time (LTPT): a variant of the LAPT rule proposed by Naderi et al. (2010).
- Longest Total Remaining Processing Times on Other Machines first (LTRPOM): a priority rule developed by Pinedo (2016). It is a more general rule than LAPT.
- Dense Scheduling/Longest Total Remaining Processing (DS/LTRP) developed by Liaw (1998).
- Dense Scheduling/Longest Total Remaining Processing Time (DS/LTRPAM) developed by Colak and Agarwal (2005).
- Modified Rotation Scheduling (MRS) a constructive heuristic developed by Bai and Tang (2011).
- Cheap Insertion Heuristic with LPT initial solution (ISTH) adapted for OSSP.
- Bounded Insertion Constructive Heuristic (BICH) developed by Abreu et al. (2020).
- Minimal Idleness Heuristic (MIH) developed by Abreu et al. (2020).
- Combined algorithm approach (BICH-MIH) developed by Abreu et al. (2020).
- Bounded Insertion Constructive Heuristic with Beam Search procedure (BICH-BS).
- Minimal Idleness Heuristic with Beam Search procedure (MIH-BS).
- Combined algorithm with Beam Search procedure (BICH-MIH-BS).
- Bounded Insertion Constructive Heuristic with Cheap Insertion procedure (BICH-IST).
- Minimal Idleness Heuristic with Cheap Insertion procedure (MIH-IST).
- Combined algorithm with Cheap Insertion procedure (BICH-MIH-IST).

For comparison purposes, we are also considering the results of the MILP model expressed by Eqs. (1)–(13), which was modeled and run on IBM ILOG CPLEX version 12.7, with 3600 s of time limit.

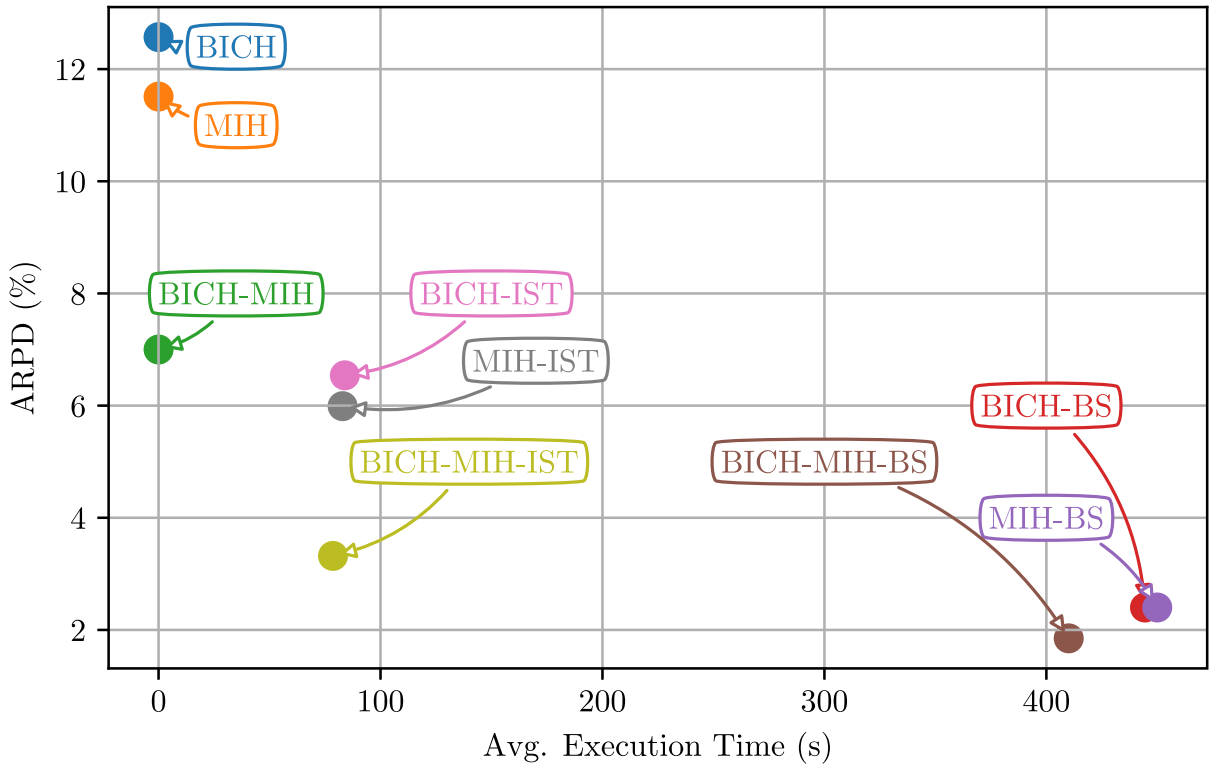


Fig. 15. Pareto chart for average computational times and ARPD of proposed constructive heuristics.

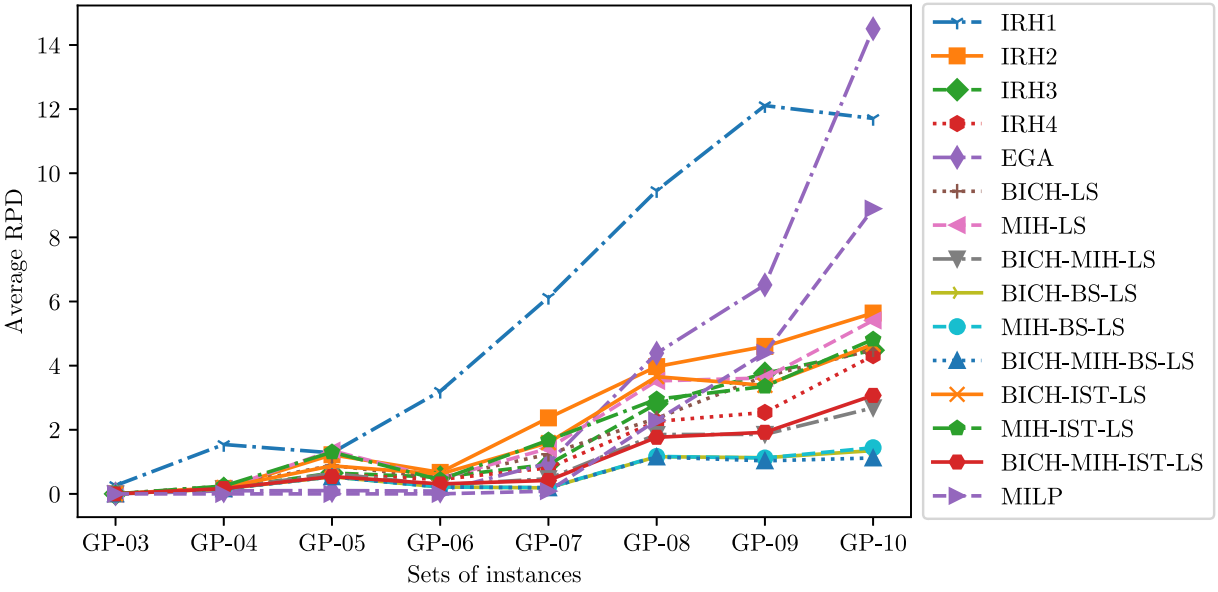


Fig. 16. Benchmark of constructive heuristics with local search for each set of instances proposed by Guéret and Prins (1999).

A summary of the computational results is presented in Table 4. It can be highlighted that the computational times for the constructive heuristics (without the BS or IST procedure) are negligible (less than 1 s). The results of Average RPD (ARPD) in each set of instance of Guéret and Prins, Taillard and Brucker are presented in Figs. 9–11 respectively.

In order to validate the results, it is important to verify whether the previous differences in the RPD values are statistically significant. We apply an analysis of variance (ANOVA) (Montgomery, 2017). The p-value is very close to zero. We can see in Fig. 12 the ARPD boxplot for all constructive heuristics tested with HSD Tukey group ($\alpha = 0.05$) of the similar mean result. We can see that there are statistically significant differences between the ARPD values among the constructive heuristics proposed. The combined approach with IST and the constructive heuristics with BS gets the best results.

According to the results for the 80 Guéret and Prins test problems, the following comments can be highlighted. The MILP model returns the best solutions for small and medium sizes classes of instances and BICH-MIH-BS returns the best solutions for large instances sizes with 9 and 10 problems size. The LPT and SPT rules are the worst algorithms for the instances analyzed. ISTH outperforms BICH-IST and MIH-IST, showing that the LPT initial sequence gives better results, but

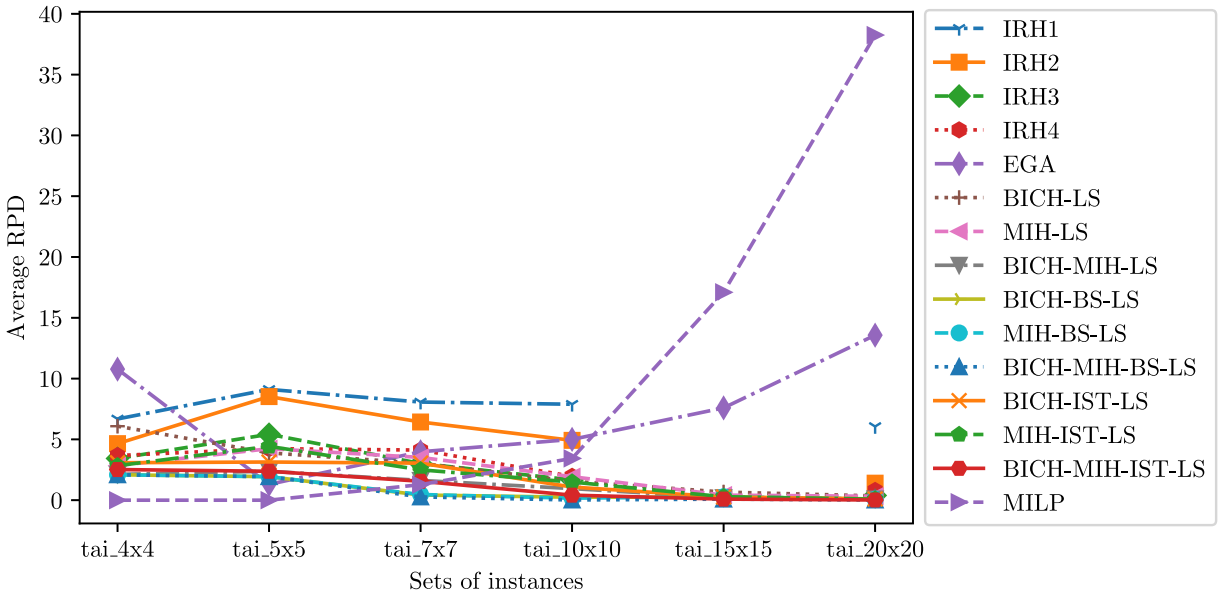


Fig. 17. Benchmark of constructive heuristics with local search for each set of instances proposed by Taillard (1993).

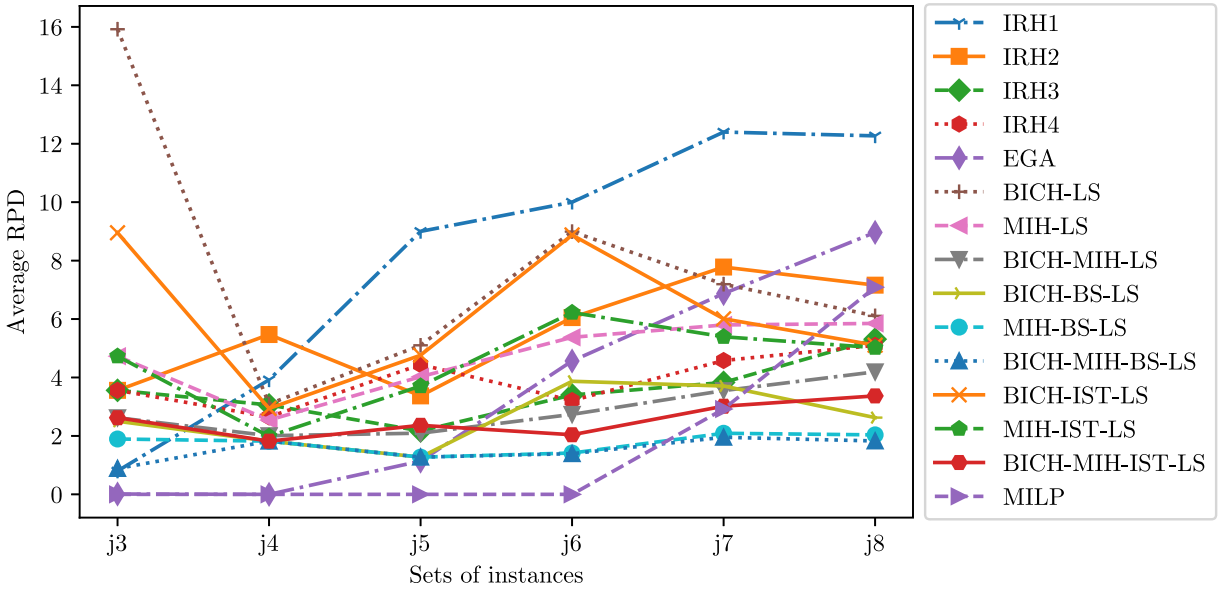


Fig. 18. Benchmark of constructive heuristics with local search for each set of instances proposed by Bruckner et al. (1997).

BICH-MIH-IST gets better results than ISTH. Therefore, the combined approach as the initial sequence indicates an improvement in the construction procedure of IST. The BS algorithms give the best results with ARPD less than 2%. Considering only the constructive heuristics, the proposed BICH-MIH-BS algorithm presented the best average results for all the eight classes of instances.

With regard the results for the 60 Taillard test problems, the following comments can be made. The MILP model returns the best average results only for the 4, 5 and 7 problems sizes. In general, the LTPT, BICH and SPT rules are the worst algorithms for this set of problems. For large problems size, the MILP model returns the worst results. The behavior of all other constructive heuristics is similar to the results for the Guéret and Prins test problems. The combined BICH-MIH-BS presents the best results for the largest instances (tai_10 × 10, tai_15 × 15 and tai_20 × 20).

With respect to the results for the 80 Brucker et al. test problems, the following comments can be done. The MILP model finds the optimal solution within the time limit for the test instances with 4, 5, and 7 jobs. However, for instances with size 8 the MILP model returns solutions of average quality within the allowed time limit. The LTPT presents the worst results compared to all others methods. Considering only the constructive heuristics, the proposed BICH-MIH-BS algorithm presents the best average results for all the eight classes of instances. For largest instances, (size 8) the BICH-MIH-BS presents the best results.

Taking into consideration the three sets of instances, the proposed approach BICH-MIH-BS presents the lower ARPD for all the analyzed constructive heuristics as well as lower than the MILP model within the time limit. With respect to the ARPD, the difference among BICH-MIH-BS and MILP methods is significant because they are clustered in different groups (with h and fg letters, respectively). Also, they are represented in different color groups that indicate groups with different ARPD. Therefore, the BICH-MIH-BS outperforms MILP (within the given time limit) in terms of ARPD.

Regarding computational times, Table 5 shows the average computational times of constructive heuristics in each set of instances.

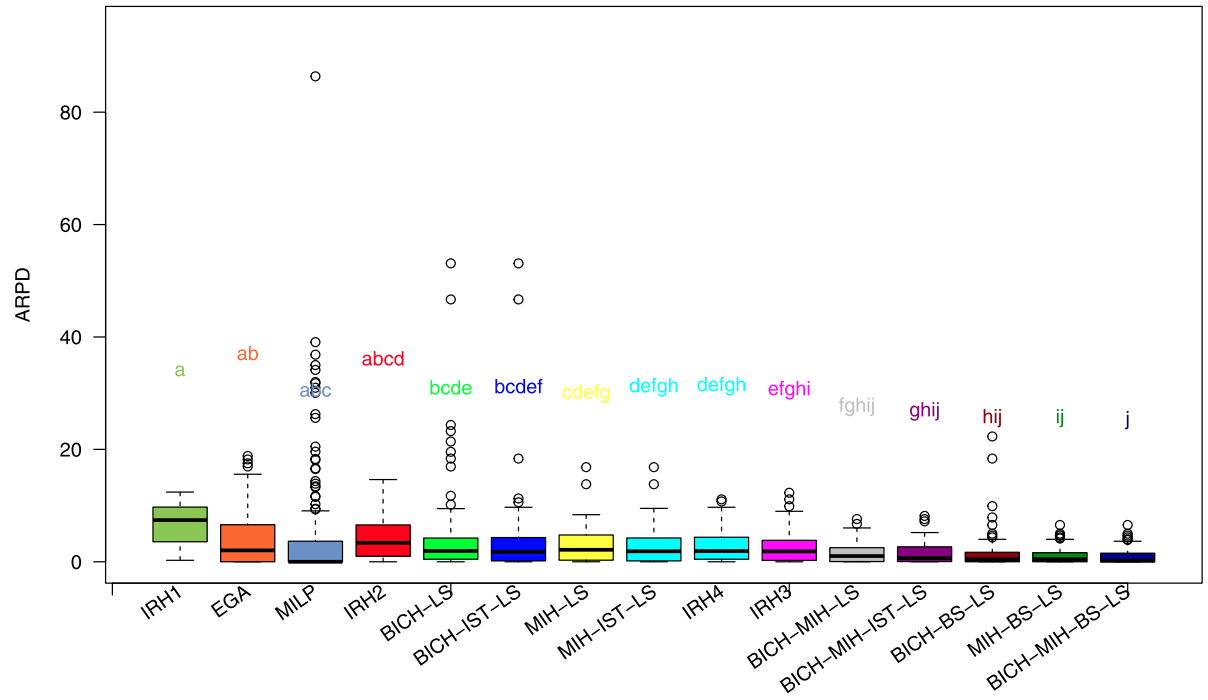


Fig. 19. Boxplot and Tukey HSD groups at the 95% confidence level for the constructive heuristics with local search in all sets of instances.

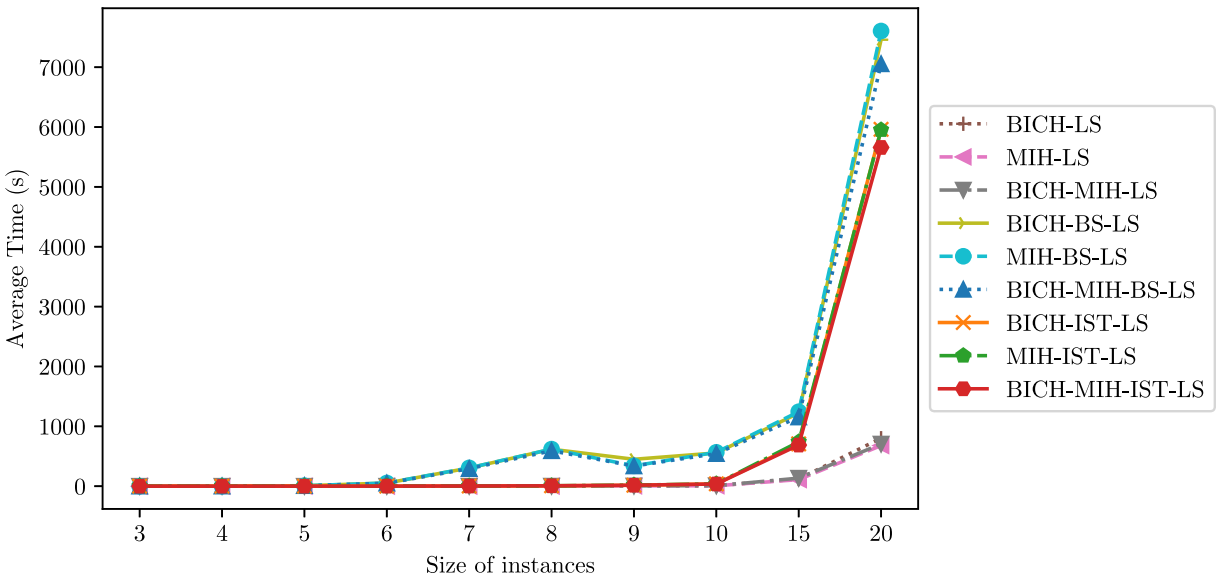


Fig. 20. Dependence between solution time and problem size for the constructive heuristics with local search.

Fig. 13 illustrates the dependence between solution time and problem size for the constructive heuristics. This figure presents the average computational for each evaluated solution procedure for each size of instance. In the smaller test instances we have 3 machines and 3 jobs (9 operations), and in the larger test instances we have 20 machines and 20 jobs (400 operations).

We can observe that the computation time increases exponentially when the number of machines and jobs is greater than 10. This trend is even more evident for the solution procedures based on IST and BS. For instance sizes with 15 or more machines and jobs, the IST-based algorithms present smaller computation times than the BS-based algorithms. Thus, the IST-based algorithms still are competitive with computation times less than 1700 s. The increase of the problem size does not imply a substantial augment in the computation times for the IST and BS algorithms.

Fig. 14 illustrates the average computation times with a 95% confidence interval for each constructive heuristic. For the test instances with less than 5 machines and jobs, the BS-based algorithms have lower computation times than the IST-based algorithms. Concerning larger instance sizes, IST-based algorithms present lower computation times than the BS-based algorithms. We can observe that this trend becomes more evident with the increase of instance sizes. In summary, we can conclude that the constructive heuristics based on IST and BS algorithms can treat problems presenting less than 15 machines and jobs with adequate computation times (approximately 10 min).

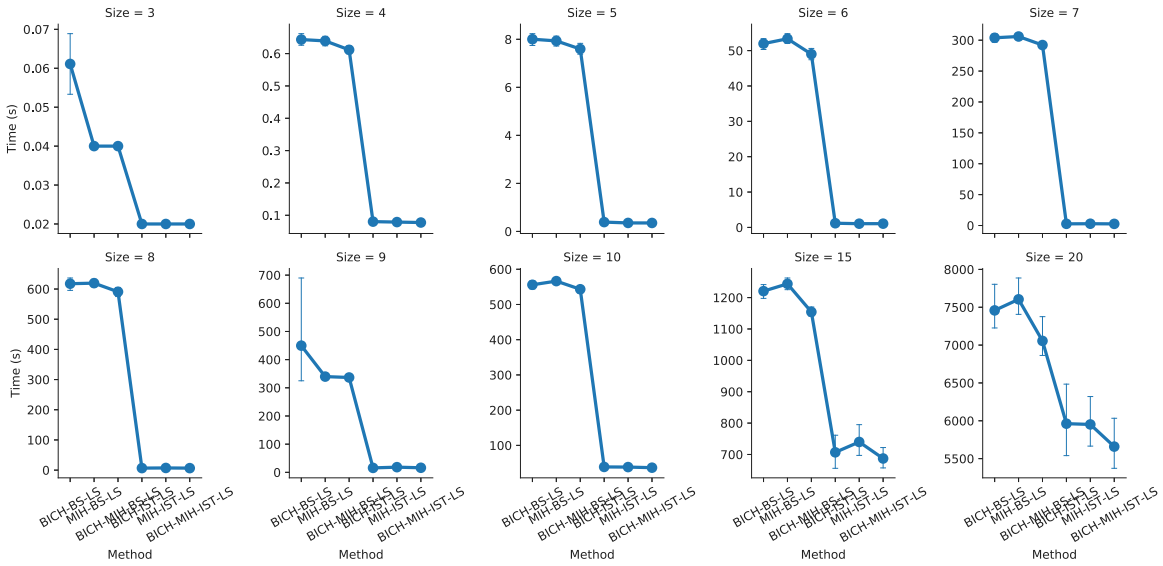


Fig. 21. Average computation times for the constructive heuristics with local search.

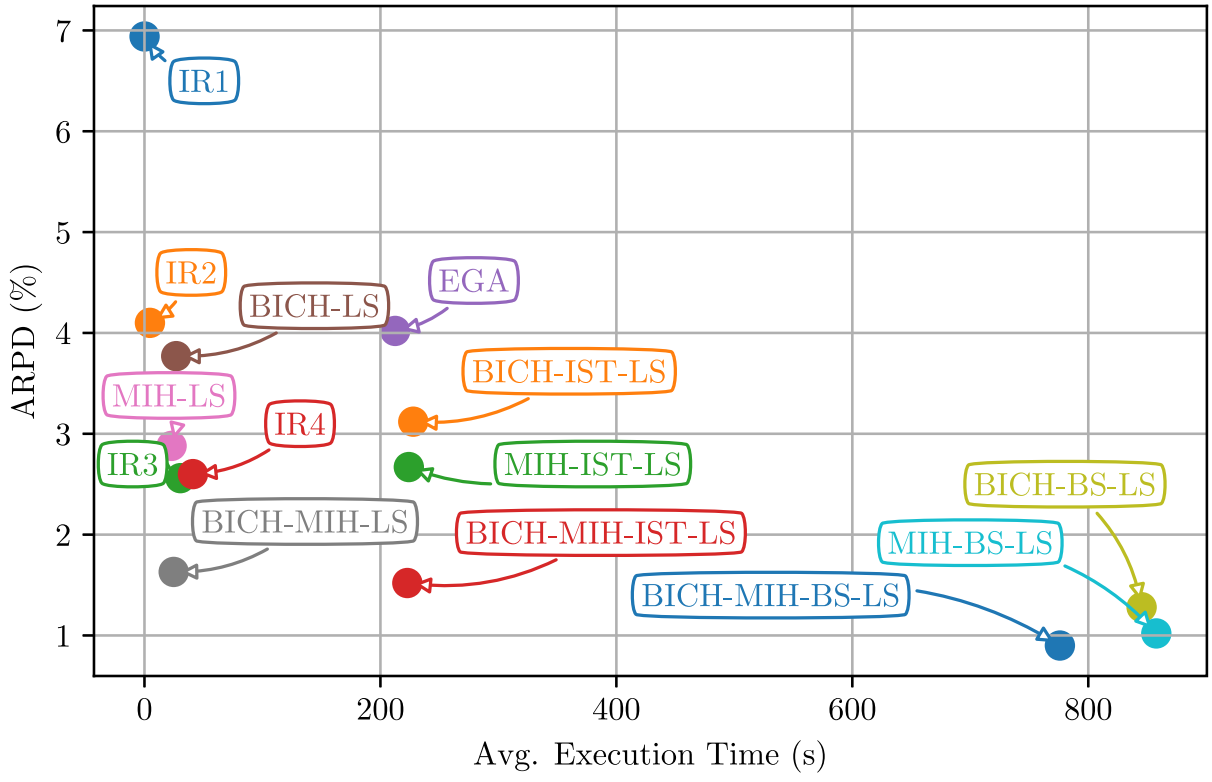


Fig. 22. Pareto chart for average computational times and ARPD of proposed constructive heuristics with local search.

The Pareto Chart of average computational times and ARPD of our proposed methods is presented in Fig. 15. As it can be seen, the proposed IST approach presents a better combination of solution quality and computational efficiency. The constructive heuristics proposed by Abreu et al. (2020) present the best computation times, while the hybridization of BICH-MIH with BS proposed in this paper gives the best ARPD results.

The hybridization of the heuristics with BS and IST procedure outperforms the BICH and MIH algorithms, being the differences statistically significant as they are in different groups in Fig. 12. Therefore, the BICH-MIH-BS turns out to be the best constructive heuristic for the OSSP with a good trade-off between solution quality and computational times.

4.2. Computational results for local search heuristics

In this section, we evaluate the performance of the following local search/metaheuristic algorithms (in addition, the MILP model is considered for comparison purposes):

- Insertion and Reinsertion Heuristic 1–4 (IRH1 to IRH4): local search algorithms proposed by Naderi et al. (2010).

Table 4
Results of constructive heuristics and mixed integer programming for each set of instances.

Benchmark	MILP	LPT	SPT	LAPT	LTPT	LTRPOM	DS/LTRP	DS/LTRPAM	MRS	ISTH	BICH	MIH	BICH-MIH	BICH-BS	MIH-BS	BICH-MIH-BS	BICH-IST	MIH-IST	BICH-MIH-IST
Guéret and Prins																			
GP-03	**0.00**	0.45	14.30	8.77	17.60	7.29	14.30	7.29	7.29	**0.00**	12.95	7.29	7.02	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
GP-04	**0.00**	24.54	5.24	1.67	1.54	9.35	1.54	9.35	9.35	0.20	4.51	9.35	2.25	0.20	0.23	0.17	0.93	1.56	0.35
GP-05	**0.00**	25.99	5.84	7.24	27.66	7.91	7.27	7.91	7.65	1.26	7.20	7.91	2.39	1.12	0.76	0.71	4.63	2.73	1.68
GP-06	**0.00**	26.58	10.04	3.16	27.64	10.44	4.52	8.24	4.89	0.67	7.51	7.77	2.72	0.43	0.48	0.27	2.76	1.36	0.71
GP-07	**0.09**	26.71	12.46	9.42	31.43	12.48	14.03	11.45	10.80	2.30	11.09	11.51	4.83	1.46	1.35	0.85	5.25	5.43	0.92
GP-08	**2.29**	27.65	16.64	15.09	34.31	16.39	17.90	17.06	14.53	4.16	15.09	17.04	11.01	3.50	3.19	2.56	6.87	7.36	3.66
GP-09	4.40	28.43	14.12	17.76	37.38	14.44	18.53	13.41	12.05	5.10	17.96	14.54	11.00	4.03	3.85	**2.77**	9.09	6.90	4.26
GP-10	8.90	25.30	16.92	16.62	38.35	16.43	22.95	18.22	15.36	5.82	17.68	18.19	14.06	5.09	5.52	**3.77**	8.55	10.36	5.68
Taillard																			
tai_4 × 4	**0.00**	10.47	15.37	10.27	27.87	10.78	10.74	9.76	10.00	4.65	14.92	12.86	7.45	3.08	2.26	2.22	5.80	5.68	4.22
tai_5 × 5	**0.00**	14.93	14.07	14.42	27.91	15.12	19.37	16.79	10.05	7.41	16.06	14.31	10.42	3.38	3.30	3.01	8.33	9.89	6.06
tai_7 × 7	**1.28**	10.98	11.94	10.26	22.46	11.94	15.05	11.37	11.03	7.46	13.18	11.86	6.80	2.81	3.03	1.86	6.46	7.81	4.06
tai_10 × 10	3.44	7.06	8.95	8.77	14.02	7.01	17.24	6.34	7.12	5.14	9.17	6.96	5.48	1.48	1.01	**0.74**	5.27	4.74	2.63
tai_15 × 15	17.10	4.03	4.00	3.96	–	5.37	–	5.60	4.05	2.17	6.99	6.13	3.28	0.77	0.68	**0.33**	2.59	2.36	1.39
tai_20 × 20	38.25	1.99	3.52	4.19	8.11	2.77	18.15	1.98	2.83	1.67	4.06	3.86	1.57	0.53	0.44	**0.22**	1.51	2.05	0.61
Brucker																			
j3	**0.00**	5.80	15.77	13.93	17.05	15.68	15.68	7.66	5.75	3.56	19.10	10.94	4.01	2.63	2.63	2.63	12.10	6.81	4.12
j4	**0.00**	10.89	10.99	12.10	18.16	10.36	9.36	7.96	8.82	5.47	8.78	11.28	4.62	2.43	2.41	2.20	4.55	7.18	3.68
j5	**0.00**	14.19	14.57	16.52	28.31	16.24	13.64	13.86	9.31	5.36	17.23	13.87	8.28	1.86	3.16	1.86	10.33	8.53	5.91
j6	**0.00**	12.94	13.60	14.20	22.72	13.23	17.93	11.73	10.82	6.98	19.89	14.98	8.98	4.62	3.80	2.89	14.63	9.74	4.74
j7	**2.92**	10.13	13.56	13.56	35.51	15.26	14.79	15.09	10.20	8.69	14.42	17.37	12.68	4.07	5.06	3.91	12.25	9.31	5.72
j8	7.09	14.92	14.60	12.22	32.01	14.01	17.07	12.89	9.34	7.14	13.70	12.28	11.09	4.59	4.90	**4.07**	8.88	10.02	6.03
Min	**0.00**	0.45	3.52	1.67	1.54	2.77	1.54	1.98	2.83	**0.00**	4.06	3.86	1.57	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Average	4.29	15.20	11.83	10.71	24.74	11.62	14.21	10.70	9.06	4.26	12.57	11.51	7.00	2.40	2.40	**1.85**	6.54	5.99	3.32
Max	38.25	28.43	16.92	17.76	38.35	16.43	22.95	18.22	15.36	8.69	19.89	18.19	14.06	5.09	5.52	**4.07**	14.63	10.36	6.06

Table 5
Computational times of constructive heuristics for each set of instances.

Benchmark	LPT	SPT	LAPT	LTPT	LTRPOM	DS/LTRP	DS/LTRPAM	MRS	ISTH	BICH	MIH	BICH-MIH	BICH-BS	MIH-BS	BICH-MIH-BS	BICH-IST	MIH-IST	BICH-MIH-IST
Guéret and Prins																		
GP-03	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.07	0.04	0.04	0.02	0.02	0.02
GP-04	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.64	0.64	0.58	0.07	0.06	0.05
GP-05	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	7.85	7.81	6.95	0.16	0.15	0.14
GP-06	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	52.39	52.75	46.96	0.45	0.44	0.44
GP-07	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	318.57	314.98	290.10	1.27	1.27	1.26
GP-08	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	598.94	597.44	528.89	3.67	3.44	3.25
GP-09	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	868.45	871.00	829.78	7.88	7.71	7.12
GP-10	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1284.96	1324.42	1127.94	17.33	17.39	16.28
Taillard																		
tai_4 × 4	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.62	0.61	0.57	0.04	0.04	0.04
tai_5 × 5	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	7.85	7.81	7.18	0.16	0.16	0.15
tai_7 × 7	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	294.73	300.38	273.18	1.35	1.27	1.25
tai_10 × 10	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1111.83	1132.63	1002.33	18.24	17.97	16.59
tai_15 × 15	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1504.02	1511.32	1422.52	388.13	380.79	349.85
tai_20 × 20	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1933.77	1951.65	1852.64	1234.93	1222.78	1170.42
Brucker																		
j3	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.05	0.04	0.04	0.01	0.01	0.01
j4	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.67	0.67	0.60	0.04	0.04	0.04
j5	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	8.35	8.21	7.71	0.14	0.14	0.14
j6	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	51.52	54.03	46.94	0.44	0.44	0.44
j7	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	297.71	302.14	273.28	1.26	1.26	1.25
j8	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	545.49	560.42	483.02	3.27	3.27	3.25
Min	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.05	0.04	0.04	0.01	0.01	0.01
Average	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	444.42	449.95	410.06	83.94	82.93	78.60
Max	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1933.77	1951.65	1852.64	1234.93	1222.78	1170.42

- Bounded Insertion Constructive Heuristic followed by local search (BICH-LS).
- Minimal Insertion Heuristic followed by local search (MIH-LS).
- Combined algorithm followed by local search (BICH-MIH-LS).
- Bounded Insertion Constructive Heuristic with Beam Search procedure followed by local search (BICH-BS-LS).
- Minimal Idleness Heuristic with Beam Search procedure followed by local search (MIH-BS-LS).
- Combined algorithm with Beam Search procedure followed by local search (BICH-MIH-BS-LS).
- Bounded Insertion Constructive Heuristic with Cheap Insertion procedure followed by local search (BICH-IST-LS).
- Minimal Idleness Heuristic with Cheap Insertion procedure followed by local search (MIH-IST-LS).
- Combined algorithm with Cheap Insertion procedure followed by local search (BICH-MIH-IST-LS).
- The genetic algorithm EGA proposed by Rahmani Hosseinabadi et al. (2018).

A summary of the computational results is presented in Table 6. The results for the test instances of Guéret and Prins, Taillard and Brucker are presented in Figs. 16–18 respectively.

In order to validate the results, as in the previous experiments, an ANOVA is applied in order to verify if the observed differences in the results of the local search algorithms are statistically significant. The p-value is very close to zero. We can see in Fig. 19 the ARPD boxplot for all constructive heuristics with local search tested with HSD Tukey group ($\alpha = 0.05$) of the similar mean result. We can see that there are statistically significant differences between the ARPD values among the local search algorithms tested. The combined approach with IST and the constructive heuristics with BS gets the best results with medians very close to zero.

According to the results obtained for the eighty Guéret and Prins test problems, the following comments can be highlighted: The MILP returns the best solutions for small and medium sizes classes of instances and BICH-MIH-BS-LS returns the best solutions for large instances sizes with 8, 9 and 10 problems size. The IR1 method is the worst algorithm for the analyzed instances. The methods with beam search procedure outperforms BICH-LS, MIH-LS and BICH-MIH-LS, showing that the new approach to construct solutions gives better results. The BS algorithms give the best results with ARPD less than 1%. Considering only the constructive heuristics with local search, the proposed BICH-MIH-BS-LS algorithm presented the best average results for all the eight classes of instances.

With respect to the results for the sixty Taillard test problems, the following comments can be made. The MILP model returns the best average results only for the 4 and 5 problems sizes. In general, the IR1, IR2 and EGA are the worst algorithms for this set of problems. For large problems size, the MILP model returns the worst results. The behavior of all other constructive heuristics with local search is similar to the results for the Guéret and Prins test problems. The combined BICH-MIH-BS-LS presented the best results for the largest instances (tai_7 × 7, tai_10 × 10, tai_15 × 15 and tai_20 × 20).

Table 6
Results of constructive heuristics with local search, meta heuristics and mixed integer programming for each set of instances.

Benchmark	MILP	IRH1	IRH2	IRH 3	IRH 4	EGA	BICH-LS	MIH-LS	BICH-MIH-LS	BICH-BS-LS	MIH-BS-LS	BICH-MIH-BS-LS	BICH-IST-LS	MIH-IST-LS	BICH-MIH-IST-LS
Guéret and Prins															
GP-03	**0.00**	0.26	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
GP-04	**0.00**	1.54	0.17	0.17	0.17	0.10	0.24	0.16	0.16	0.16	0.16	0.16	0.20	0.24	0.17
GP-05	**0.00**	1.28	1.22	0.65	0.60	0.10	0.89	1.35	0.63	0.53	0.53	0.53	0.87	1.30	0.54
GP-06	**0.00**	3.20	0.67	0.52	0.45	0.09	0.50	0.47	0.28	0.21	0.21	0.21	0.62	0.44	0.31
GP-07	**0.09**	6.13	2.37	0.92	0.81	0.96	1.24	1.42	0.47	0.19	0.20	0.19	1.60	1.67	0.42
GP-08	2.29	9.46	3.97	2.81	2.26	4.39	2.36	3.52	1.85	1.16	1.17	**1.15**	3.66	2.95	1.76
GP-09	4.40	12.11	4.60	3.76	2.54	6.52	3.67	3.61	1.86	1.13	1.12	**1.03**	3.38	3.35	1.92
GP-10	8.90	11.71	5.64	4.48	4.31	14.51	4.50	5.41	2.68	1.35	1.44	**1.12**	4.66	4.82	3.07
Taillard															
tai_4 × 4	**0.00**	6.68	4.65	3.43	3.67	0.70	6.08	2.84	2.12	2.09	2.09	2.09	3.08	2.84	2.52
tai_5 × 5	**0.00**	9.11	8.52	5.42	4.24	1.30	3.89	4.24	2.38	1.94	1.94	1.94	3.14	4.43	2.38
tai_7 × 7	1.28	8.07	6.43	3.02	4.12	3.98	3.04	3.52	1.63	0.43	0.47	**0.27**	3.05	2.50	1.56
tai_10 × 10	3.44	7.89	4.92	1.67	1.96	5.00	1.56	1.93	0.95	0.25	0.16	**0.02**	1.08	1.49	0.41
tai_15 × 15	17.10	–	–	–	–	7.59	0.67	0.42	0.21	0.25	0.28	**0.09**	0.26	0.28	**0.09**
tai_20 × 20	38.25	6.06	1.40	0.39	0.80	13.57	0.35	0.32	0.10	0.10	0.08	**0.02**	0.10	0.08	**0.02**
Brucker															
j3	**0.00**	0.80	3.56	3.56	3.56	0.01	15.92	4.73	2.62	2.50	1.89	0.88	8.95	4.73	2.62
j4	**0.00**	3.92	5.47	3.06	2.65	**0.00**	3.09	2.54	2.01	1.82	1.82	1.82	2.94	1.99	1.82
j5	**0.00**	9.00	3.37	2.17	4.44	1.14	5.10	4.03	2.09	1.27	1.27	1.27	4.76	3.71	2.36
j6	**0.00**	10.00	6.05	3.40	3.21	4.56	8.99	5.38	2.74	3.87	1.41	1.39	8.88	6.22	2.04
j7	2.92	12.40	7.78	3.83	4.58	6.87	7.19	5.80	3.55	3.71	2.09	**1.95**	6.01	5.39	3.02
j8	7.09	12.27	7.16	5.31	5.10	8.97	6.11	5.85	4.19	2.63	2.04	**1.83**	5.11	5.02	3.37
Min	**0.00**	0.26	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Average	4.29	6.94	4.10	2.56	2.60	4.02	3.77	2.88	1.63	1.28	1.02	**0.90**	3.12	2.67	1.52
Max	38.25	12.40	8.52	5.42	5.10	14.51	15.92	5.85	4.19	3.87	**2.09**	**2.09**	8.95	6.22	3.37

Table 7
Computational times of constructive heuristics with local search and meta heuristics for each set of instances.

Benchmark	IRH1	IRH2	IRH 3	IRH 4	EGA	BICH-LS	MIH-LS	BICH-MIH-LS	BICH-BS-LS	MIH-BS-LS	BICH-MIH-BS-LS	BICH-IST-LS	MIH-IST-LS	BICH-MIH-IST-LS
Guéret and Prins														
GP-03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	0.08	0.07	0.02	0.02	1.64
GP-04	0.00	0.00	0.01	0.01	36.76	0.01	0.01	0.01	1.21	1.22	1.11	0.09	0.09	1.47
GP-05	0.00	0.01	0.03	0.05	108.36	0.04	0.03	0.04	15.06	14.87	13.07	0.55	0.49	1.27
GP-06	0.00	0.02	0.11	0.15	134.51	0.15	0.12	0.16	99.52	99.49	89.42	1.06	1.14	1.73
GP-07	0.01	0.06	0.37	0.47	196.41	0.43	0.38	0.46	607.42	599.72	556.41	3.15	3.16	3.86
GP-08	0.02	0.15	1.14	1.20	300.01	1.17	1.02	1.24	1140.86	1139.38	1005.18	6.20	6.37	6.85
GP-09	0.03	0.35	2.77	2.72	300.01	2.41	2.92	2.72	1655.42	1669.84	1560.10	15.75	18.35	17.42
GP-10	0.05	0.66	4.19	5.43	300.01	4.61	4.82	5.54	2439.32	2507.13	2139.55	39.68	39.62	40.19
Taillard														
tai_4 × 4	0.00	0.00	0.01	0.01	192.29	0.01	0.01	0.01	1.18	1.16	1.08	0.07	0.06	1.40
tai_5 × 5	0.00	0.01	0.04	0.05	254.16	0.04	0.04	0.04	14.99	14.73	13.81	0.26	0.23	1.51
tai_7 × 7	0.01	0.06	0.39	0.49	300.01	0.32	0.38	0.37	563.31	572.50	517.97	2.16	2.56	3.09
tai_10 × 10	0.05	0.68	4.47	5.54	300.01	4.59	3.90	5.26	2140.05	2146.40	1885.65	37.98	37.17	38.15
tai_15 × 15	–	–	–	–	300.03	73.27	66.62	81.98	2849.63	2880.84	2702.72	706.82	739.87	702.76
tai_20 × 20	1.18	79.46	517.91	707.79	300.06	449.55	378.03	395.84	3660.44	3755.26	3506.56	3731.02	3619.92	3616.29
Brucker														
j3	0.00	0.00	0.00	0.00	37.50	0.00	0.00	0.00	0.10	0.08	0.07	0.02	0.02	1.39
j4	0.00	0.00	0.01	0.01	67.73	0.01	0.01	0.01	1.27	1.26	1.15	0.09	0.08	1.54
j5	0.00	0.01	0.04	0.05	222.09	0.04	0.03	0.04	15.85	15.73	14.60	0.35	0.34	1.58
j6	0.01	0.02	0.13	0.16	300.00	0.16	0.13	0.18	97.82	101.85	89.28	1.31	0.96	1.97
j7	0.01	0.06	0.37	0.46	300.01	0.37	0.33	0.47	569.31	571.65	516.13	2.85	2.95	3.66
j8	0.02	0.15	0.94	1.10	300.01	0.85	0.82	1.06	1031.35	1060.00	904.93	7.31	8.19	8.52
Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.08	0.07	0.02	0.02	1.27
Average	0.08	4.63	30.46	41.08	212.50	26.90	22.98	24.77	845.21	857.66	775.94	227.84	224.08	222.81
Max	1.18	79.46	517.91	707.79	300.06	449.55	378.03	395.84	3660.44	3755.26	3506.56	3731.02	3619.92	3616.29

For the Guéret and Prins instances, as well as for the Taillard instances, the proposed algorithms with beam search and cheapest insertion procedure present better results than IRx and EGA. In general, the MILP method obtains the better results for the small-sized instances and the BICH-MIH-BS-LS present the best results for the large-sized instances.

Finally, for the Brucker instances, the MILP method outperforms the other evaluated methods, with the exception of the j7 and j8 sets of instances, in which the BICH-MIH-BS-LS algorithm returns the best results.

On average, the MILP method yields poor results, because the model returned low-quality results for the large-sized test problems in the allotted CPU time. The proposed BICH-MIH-BS-LS algorithm outperforms all the other evaluated methods, showing that the combined approach with a weighted aggregation function is more efficient. Overall, the proposed approach BICH-MIH-BS-LS presented the lower ARPD for all the analyzed constructive heuristics as well as lower than the MILP model. Furthermore, the differences among BICH-MIH-BS-LS, EGA, MILP and methods proposed by Naderi et al. (2010) and Abreu et al. (2020) are significant because they are in different groups with different letters. The methods with beam search procedure and BICH-MIH-IST-LS present the lowest mean values for ARPD.

Regarding computational times, Table 7 shows the average computational times of constructive heuristics and metaheuristics in each set of instances.

With respect to computational times for the constructive heuristics with local search, Fig. 20 illustrates the dependence between solution time and problem size for the constructive heuristics with local search. We can observe that the BS-based algorithms have started to increase their computation times from the instances with 8 or more machines and jobs. For problem sizes with 15 or more machines and jobs, we can observe that all the constructive heuristics with local search increased their computation times.

Fig. 21 illustrates the average computation times with a 95% confidence interval for each constructive heuristic with local search. In contrast with the constructive heuristics without local search, in this situation, the IST-based heuristics present computation times substantially smaller than the BS-based heuristics. The two classes of algorithms can deal with problems with less than 15 jobs and machines (around 2000 s).

As a conclusion of the computational time analysis, IST- and BS-based constructive heuristics can handle 20 × 20 instances in less than one hour while their local search counterpart would require more than double of the time. Since the computation times around this instance size grow rapidly, we believe that 20 × 20 (which represents a total of 400 operations in the shop) represents the limit in the problem size that, for many decision scenarios, can be realistically addressed in a standard computer.

The Pareto Chart of average computational times and ARPD is presented in Fig. 22. On the basis of the above, the proposed IST approach presents a better combination of solution quality and computational

efficiency. The constructive heuristics proposed by Abreu et al. (2020) with local search and IRx algorithms presents best computation times and BS approach presents the best ARPD results.

The constructive heuristics with BS and IST procedure outperforms the classic approach of BICH and MIH algorithms proposed by Abreu et al. (2020), IRx algorithms proposed by Naderi et al. (2010) and EGA proposed by Rahmani Hosseinabadi et al. (2018). The BICH and MIH with BS procedure gives the best results than BICH and MIH with IST procedure, but the algorithms with IST procedure gets good computational times, becoming a good choice for industrial applications with operational level of scheduling problems. The BICH-MIH-BS-LS becoming the best constructive heuristic with local search for OSSP.

As a summary of the performance of the BS-based algorithms, note that this family of algorithms obtains very low ARPD values at the costs of requiring more CPU time. As illustrated in Fig. 15, the inclusion of beam search procedures resulted in a substantial improvement in the quality of the solutions found when compared to the algorithms considering cheap insertion procedures, even if it is to note that this increases the computation times. In Fig. 15, we can also observe that the BICH-MIH-BS dominated the MIH-BS and BICH-BS algorithms.

Concerning the local search versions of the proposed algorithms, it can be seen in Fig. 22 that, when compared to IR3 and IR4 – the best versions of IRx algorithms – the BICH-MIH-LS presented lower ARPD values with a similar average computational time. With respect to the ARPD values, the BICH-MIH-BS-LS algorithm presented the best results, dominating the MIH-BS-LS and BICH-BS-LS algorithms.

5. Concluding remarks

In this paper, we focus on the classical variant of the OSSP. The objective function is to minimize the total time to complete the schedule (makespan). We develop new beam search and cheapest insertion procedures hybridized with constructive heuristics adapted from the problem with setup considerations. Finally, an efficient local search algorithm (LS) that leads to excellent results within an admissible computational effort is proposed.

A number of computational experiments were carried out in order to evaluate the performance of the proposed algorithms. The results of the proposed approaches are presented considering the literature benchmark instances proposed by Guéret and Prins (1999), Taillard (1993) and Bruckner et al. (1997). We used the relative percentage deviation statistics as performance measure. Taking into consideration the above mentioned literature benchmark instances, the proposed constructive algorithm BICH-MIH-BS outperforms the existing constructive heuristics, the BICH-MIH-BS-LS algorithm outperforms the four local search algorithms proposed by Naderi et al. (2010) and the genetic algorithm proposed by Rahmani Hosseinabadi et al. (2018).

As extensions of this work, we suggest the consideration of explicit travel times and resource utilization in the OSSP to address more realistic environments. In addition, future studies could also investigate the behavior of the proposed approaches considering different objective functions, such as total tardiness minimization with due dates of jobs or total completion time minimization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was financed in part by the Coordination for the Improvement of Higher Education Personnel (CAPES) - through Grant no. 88882.379108/2019-01, the National Council for Scientific and Technological Development (CNPq) - through Grant nos. [404232/2016-7, 306075/2017-2, 430137/2018-4, 303594/2018-7, 312585/2021-7 and 407151/2021-4], the São Paulo Research Foundation (FAPESP) - through Grant no. #2020/16341-5, the Spanish Ministry of Science and Education (ASSORT project — Grant no. PID2019-108756RB-I00), and the Andalusian Government (projects DEMAND — Grant no. P18-FR-1149 and EFECTOS - Grant no. US-1264511).

References

Abreu, L.R., Cunha, J.O., Prata, B.A., Framinan, J.M., 2020. A genetic algorithm for scheduling open shops with sequence-dependent setup times. Comput. Oper. Res. 113, 104793.

Abreu, L.R., Tavares-Neto, R.F., Nagano, M.S., 2021. A new efficient biased random key genetic algorithm for open shop scheduling with routing by capacitated single vehicle and makespan minimization. Eng. Appl. Artif. Intell. 104, 104373.

Adak, Z., Arıoğlu Akan, M.O., Bulkan, S., 2020. Multiprocessor open shop problem: literature review and future directions. J. Combin. Optim. 40, 547–569.

Ahmadian, M.M., Khatami, M., Salehipour, A., Cheng, T., 2021. Four decades of research on the open-shop scheduling problem to minimize the makespan. European J. Oper. Res..

Ahmadizar, F., Hosseinabadi Farahani, M., 2012. A novel hybrid genetic algorithm for the open shop scheduling problem. Int. J. Adv. Manuf. Technol. 62 (5), 775–787. http://dx.doi.org/10.1007/s00170-011-3825-1.

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

Bai, D., Tang, L., 2011. Performance analysis of rotation schedule and improved strategy for open shop problem to minimise makespan. Internat. J. Systems Sci. 42 (7), 1143–1153.

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.

Birgin, E.G., Ferreira, J.E., Ronconi, D.P., 2020. A filtered beam search method for the m-machine permutation flowshop scheduling problem minimizing the earliness and tardiness penalties and the waiting time of the jobs. Comput. Oper. Res. 114, 104824.

Blum, C., 2005. Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput. Oper. Res. 32 (6), 1565–1591.

Bräsel, H., Tautenhahn, T., Werner, F., 1993. Constructive heuristic algorithms for the open shop problem. Computing 51 (2), 95–110. http://dx.doi.org/10.1007/BF02243845.

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

Colak, S., Agarwal, A., 2005. Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem. Nav. Res. Logist. 52 (7), 631–644.

de Abreu, L.R., Araújo, K.A.G., de Athayde Prata, B., Nagano, M.S., Moccellin, J.V., 2021. A new variable neighbourhood search with a constraint programming search strategy for the open shop scheduling problem with operation repetitions. Eng. Optim. 1–20.

Dong, X., Huang, H., Chen, P., 2008. An improved NEH-based heuristic for the permutation flowshop problem. Comput. Oper. Res. 35 (12), 3962–3968.

Fan, K., Zhai, Y., Li, X., Wang, M., 2018. Review and classification of hybrid shop scheduling. Prod. Eng. 12 (5), 597–609. http://dx.doi.org/10.1007/s11740-018-0832-1.

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.

Fernandez-Viagas, V., Ruiz, R., Framinan, J.M., 2017. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. European J. Oper. Res. 257 (3), 707–721. http://dx.doi.org/10.1016/j.ejor.2016.09.055, URL http://www.sciencedirect.com/science/article/pii/S0377221716308074.

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.

Garey, M., Johnson, D., 2012. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, Norwell.

Ghosn, S.B., Drouby, F., Harmanani, H.M., 2016. A parallel genetic algorithm for the open-shop scheduling problem using deterministic and random moves. Int. J. Artif. Intell. 14 (1), 130–144.

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

González-Rodríguez, I., Palacios, J.J., Vela, C.R., Puente, J., 2010. Heuristic local search for fuzzy open shop scheduling. In: International Conference on Fuzzy Systems. IEEE, pp. 1–8.

Guéret, C., Jussien, N., Prins, C., 2000. Using intelligent backtracking to improve branch-and-bound methods: An application to open-shop problems. European J. Oper. Res. 127 (2), 165–183.

Guéret, C., Prins, C., 1998. Classical and new heuristics for the open-shop problem. European 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, 165–183.

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

Kizilay, D., Tasgetiren, M.F., Pan, Q.-K., Gao, L., 2019. A variable block insertion heuristic for solving permutation flow shop scheduling problem with makespan criterion. Algorithms 12 (5), 100.

Liaw, C.-F., 1998. An iterative improvement approach for the nonpreemptive open shop scheduling problem. European J. Oper. Res. 111 (3), 509–517.

Liaw, C.-F., 1999a. Applying simulated annealing to the open shop scheduling problem. IIE Trans. 31 (5), 457–465.

Liaw, C., 1999b. A tabu search algorithm for the open shop scheduling problem. Comput. Oper. Res. 52 (2), 109–126.

Liaw, C.-F., 2000. A hybrid genetic algorithm for the open shop scheduling problem. European J. Oper. Res. 124 (1), 28–42.

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. http://dx.doi.org/10.1016/j.ijpe.2007.01.005, URL http://www.sciencedirect.com/science/article/pii/S0925527307000515 Special Section on Sustainable Supply Chain.

Montgomery, D.C., 2017. Design and Analysis of Experiments. John Wiley & Sons.

Naderi, B., Fatemi Ghomi, S., Aminnayeri, M., Zandieh, M., 2011a. A study on open shop scheduling to minimise total tardiness. Int. J. Prod. Res. 49 (15), 4657–4678.

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., 2011b. Modeling and scheduling open shops with sequence-dependent setup times to minimize total completion time. Int. J. Adv. Manuf. Technol. (ISSN: 1433-3015) 53 (5), 751–760.

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.

Pinedo, M., 2016. Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, New York.

Pongchairerks, P., Kachitvichyanukul, V., 2016. A two-level particle swarm optimisation algorithm for open-shop scheduling problem. Int. J. Comput. Sci. Math. 7 (6), 575–585.

Prins, C., 2000. Competitive genetic algorithms for the open-shop scheduling problem. Math. Methods Oper. Res. 52 (3), 389–411.

Rahmani Hosseinabadi, A.A., Vahidi, J., Saemi, B., Sangaiah, A.K., Elhoseny, M., 2018. Extended genetic algorithm for solving open-shop scheduling problem. Soft Comput. http://dx.doi.org/10.1007/s00500-018-3177-y.

Ramudhin, A., Marier, P., 1996. The generalized shifting bottleneck procedure. European J. Oper. Res. 93 (1), 34–48.

Rossi, F.L., Nagano, M.S., 2020. Heuristics and metaheuristics for the mixed no-idle flowshop with sequence-dependent setup times and total tardiness minimisation. Swarm Evol. Comput. 100689.

Ruiz, R., Stützle, T., 2008. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European J. Oper. Res. 187 (3), 1143–1159.

Sha, D., Hsu, C., 2008. A new particle swarm optimization for the open shop scheduling problem. Comput. Oper. Res. 35 (10), 3243–3261.

Skriver, A.J., Andersen, K.A., 2000. A label correcting approach for solving bicriterion shortest-path problems. Comput. Oper. Res. 27 (6), 507–524.

Strusevich, V.A., 1998. A greedy open shop heuristic with job priorities. Ann. Oper. Res. 83, 253–270.

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

Vincent, F.Y., Lin, S.-W., Chou, S.-Y., 2010. The museum visitor routing problem. Appl. Math. Comput. 216 (3), 719–729.

Wu, X., Che, A., 2020. Energy-efficient no-wait permutation flow shop scheduling by adaptive multi-objective variable neighborhood search. Omega 94, 102117.

Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J., 2019. Review of job shop scheduling research and its new perspectives under industry 4.0. J. Intell. Manuf. 30 (4), 1809–1830. http://dx.doi.org/10.1007/s10845-017-1350-2.