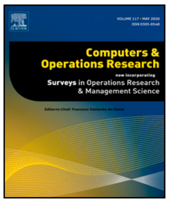
Computers & Operations Research 156 (2023) 106247

Contents lists available at ScienceDirect

Computers and Operations Research

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



A genetic algorithm for scheduling open shops with conflict graphs to minimize the makespan

Nour ElHouda Tellache [*], Laoucine Kerbache

Division of Engineering Management and Decision Sciences, College of Science and Engineering, Hamad Bin Khalifa University, Qatar Foundation, Doha, Qatar

ARTICLE INFO

Dataset link: https://nourelhoudatellache.wordpress.com/test-instances/

Keywords:
Scheduling
Open shop
Conflict graphs
Makespan
MILP
Lower bounds
Genetic algorithms

ABSTRACT

The open shop problem with conflict graph consists of scheduling jobs on an open shop system subject to conflict constraints given by a simple undirected graph G, called the conflict graph. In this graph, each vertex represents a job, and jobs that are adjacent in G are in conflict, i.e. they cannot be processed at the same time on different machines. The problem of finding a feasible schedule that minimizes the maximum completion time is known to be NP-hard even on two machines. In this paper, we present mixed integer linear programming models, lower bounds, and genetic algorithms for this problem. Extensive computational experiments are conducted on a large set of instances derived from well-known benchmarks of the basic open shop problem. The results show the effectiveness of the genetic algorithm that solves to optimality at least 93.490% of the instances and the average deviation from the lower bounds is within 0.475%. Furthermore, the algorithm improves the upper bounds obtained by the mathematical formulations and outperforms the existing heuristics.

1. Introduction

The Open Shop scheduling problem with Conflict graph (OSC) can be stated as follows: a finite set $\{J_j, j = 1,\dots,n\}$ of n jobs that has to be processed on a given set $\{M_i, i = 1,\dots,m\}$ of m dedicated machines, and a simple undirected graph $G = (V,E)$ over the jobs called the conflict graph. Each job J_j consists of m operations J_{ij} $(i = 1,\dots,m)$, where J_{ij} has to be processed on machine M_i for $p_{ij} \in \mathbb{N}$ time units without preemption. There are no restrictions on the order in which the jobs are processed on the machines. At any time, each machine can process at most one operation, and each job is processed on at most one machine. The conflict graph G models conflicts between the jobs, where each vertex represents a job and two jobs that are adjacent in G are in conflict, i.e. they cannot be processed at the same time on different machines. The objective is to find a feasible schedule that minimizes the maximum completion time also called the makespan and denoted C_{max}. Fig. 1 shows an example of an OSC problem with three jobs and three machines. In the context of our problem, we define the conflicts between operations as follows: two operations are in conflict if they are of the same job, of the same machine or of two conflicting jobs. According to the three field classification $\alpha/\beta/\gamma$ of Graham et al. (1979), we denote our scheduling problem by $Om|ConfG = (V,E)|C_{max}$, where $ConfG = (V,E)$ indicates the presence of a conflict graph $G = (V,E)$ over the jobs. The complement of the conflict graph $G = (V,E)$ is called the agreement graph $\overline{G} = (V,\overline{E})$, and it is clear that the problem of scheduling with conflict graph and the problem of scheduling with agreement graph are polynomially equivalent.

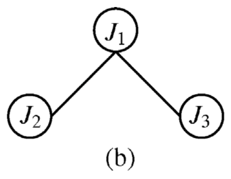
The open shop problem has found a wide range of applications in the production systems (Abreu et al., 2021), medical services (Zhang et al., 2019), timetabling (Kubiak, 2021), vehicle maintenance (Gonzalez and Sahni, 1976), telecommunications (Prins, 1994), and plastic injections (Naderi et al., 2012). Conflicts between jobs arise naturally in these applications when the jobs require the same limited resources for their processing. These resources may represent manpower, specific rooms for examination, or specific materials. In the class-teacher timetabling (Kubiak, 2021) for instance, we have n teachers giving lectures to m classes, where the students of each class follow the same program. The teachers can be seen as jobs and the classes as machines. An operation J_{ij} represents the lecture given by teacher J_j to the class M_i. If a group of teachers involve specific equipped rooms, then the lectures that can be run at the same time are constrained by the number of available rooms, and thus may be in conflict. In the context of machine scheduling, the OSC problem is related to the Open Shop problem under Resource Constraints (OSRC), where the jobs may require, besides the machines, some additional resources for their processing (see Blazewicz et al. (1986)). This may generate conflicts if the requirements of the resources exceed their capacities. In the OSRC, the operations of a job may have different requirements of the

[*] Corresponding author.
E-mail addresses: nour.tellache@gmail.com (N.E. Tellache), lakerbache@hbku.edu.qa (L. Kerbache).

https://doi.org/10.1016/j.cor.2023.106247
Received 14 August 2022; Received in revised form 7 March 2023; Accepted 10 April 2023
Available online 17 April 2023

	M_1	M_2	M_3
J_1	3	2	2
J_2	3	2	1
J_3	1	1	2

(a)



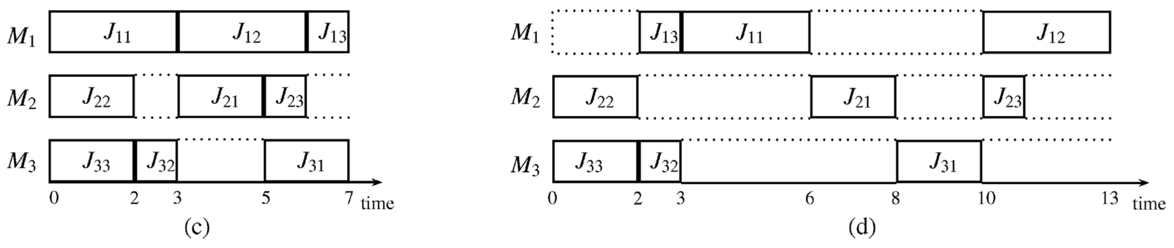


Fig. 1. Example of an OSC problem with three jobs and three machines: (a) processing times. (b) conflict graph. (c) infeasible schedule. (d) feasible schedule.

resources which generate different conflicts for the operations of the same job. However, the conflicts can be modeled between jobs if the operations of each job have the same requirements of each resource. These conflicts can be modeled in general by a conflict hypergraph over the jobs, but in some particular cases a conflict graph suffices, e.g. when the number of machines equals two or when the resources are available with one unit. Tellache and Boudhar (2017) studied the relation between the OSC and OSRC problems and showed that the OSC problem is polynomially equivalent to $O|res^t.11|C_{max}$, where $res^t\lambda\sigma\delta$ indicates the presence of additional resources such that λ (respectively σ, and δ) represents the number of resource types (respectively resource availabilities, and resource requirements) and t is for the case in which the operations of each job require the same amount of each resource.

The OSC problem is NP-hard in the strong sense for $m \geq 2$ (Tellache and Boudhar, 2017). Several complexity results have been presented in the literature. Tellache and Boudhar (2017) showed that the two-machine OSC problem is NP-hard in the strong sense for $p_{ij} \in \{1,2,3\}$ and G being a complement of a bipartite graph, but when restricted to $p_{ij} \in \{0,1,2\}$, the problem can be solved in polynomial time for arbitrary conflict graphs. Furthermore, allowing preemption makes the two-machine OSC problem easy to solve for arbitrary processing times and conflict graphs. For the case of $p_{ij} = 1$, the problem becomes NP-hard when the number of machines increases to three, but can be solved in polynomial time when restricted to complements of triangle-free graphs. The authors also found that the OSC problem is polynomially equivalent to $O|res^t.11|C_{max}$ from which new complexity results of the latter problem were derived. The same authors proved in Tellache et al. (2019) that the proportionate two-machine OSC problem is NP-hard in the strong sense for two distinct values of processing times $p_{ij} \in \{a, 2a+b\}$, ($a \geq 1$ and ($b \geq 1$ or $-a < b < 0$)) and more general conflict graphs. This result closes the complexity status of the two-machine OSC problem with two values of processing times. Recently, Tellache (2021) showed that when restricted to complements of trees, the problem is NP-hard, but when restricted to complements of caterpillars and cycles, the problem becomes polynomial. The author also studied the OSC problem with release times and proved that the two-machine OSC problem with two distinct release times in $\{0, r\}$ (r arbitrary) and $p_{ij} \in \{1,2\}$ is NP-hard in the strong sense for general conflict graphs, but when $p_{ij} = 1$, the problem is polynomially solvable for bipartite graphs.

Regarding the solution methods, a two-phase heuristic approach for the general m-machine OSC problem has been proposed in Tellache and Boudhar (2017). In the first phase, the set of all operations is partitioned into subsets, called schedule slices, such that the operations of each slice can be processed simultaneously. The objective of this phase is to build a minimum number of schedule slices as balanced as possible. This is done by finding matchings in weighted bipartite graphs constructed from the conflict graph. In the second phase, an insertion technique combined with beam search is used to make complete schedules from the schedule slices. As far as we know, this is the only solution method in the literature for the general m-machine OSC problem. We focus in the following on the basic open shop problem and some of its variants along with the different metaheuristics, mainly Genetic Algorithms (GAs), that have been proposed to solve them.

The basic Open Shop problem (OS) with the makespan minimization can be solved in polynomial time for $m = 2$, but becomes NP-hard when the number of machines increases to three (Gonzalez and Sahni, 1976). Different approximation algorithms, including metaheuristics, have been employed to solve the OS problem. An interesting class of metaheuristics is the class of evolutionary algorithms. Fang et al. (1994, 1993) were the first to propose a GA for the OS problem. They used a chromosome representation that is a list of uncompleted jobs. To each chromosome is associated a schedule that is built by reading the uncompleted jobs of the list in a circular way. For each job, an untackled operation is selected, following a priority rule among eight rules, and placed at the earliest possible time of the current schedule. The authors presented another version in which they extended the chromosome representation to include the priority rule to be applied for each job. This latter version shows the best results on most of Taillard's instances (Taillard, 1993). Louis and Xu (1996) used a different chromosome representation that is a list of operations such that each operation J_{ij} is coded with two digits: the number of J_{ij} within j, and j. Their main contribution is for the OS re-scheduling problem; they saved old solutions obtained from a GA applied to an OS problem (P_{old}), then they injected these solutions into the initial population of a new OS problem derived from P_{old} by slightly modifying it. Khuri and Miryala (1999) studied three chromosome representations. The first one consists of a permutation of the non-null operations where each operation is given a unique number. The associated schedule is obtained by scheduling each operation as early as possible following the order of the permutation. The second representation is close to that of Fang et al. (1994, 1993). The difference is in the length of the uncompleted jobs list that is equal to the number of non-null operations, and one rule (the largest processing time first rule) is used for the selection of the operation to be scheduled. The third one is similar to that of Louis and Xu (1996). Their results on Taillard's instances (Taillard, 1993) showed that the second representation yields the best results on the large instances. Prins (2000) used a chromosome representation that is a permutation of the non-null operations. The corresponding schedule is recovered by applying three schedule builders producing non-delay and active schedules (see Section 4.2 for the definitions of non-delay and active schedules). Some heuristic chromosomes have been inserted into the initial population in which each chromosome has a distinct makespan, and several crossover and mutation operators have been tested. Computational experiments have been performed on the three known benchmark sets of Taillard (1993), Brucker et al. (1997), and Guéret and Prins (1999) to derive the best combination of components. Liaw (2000) developed a hybrid GA in which a local improvement procedure, based on tabu search, is applied to each new chromosome to replace it with a local optimum one. The algorithm has been tested on Taillard (1993) and some of Brucker et al. (1997) instances, and on randomly generated instances. On Taillard's instances (Taillard, 1993), the hybrid GA outperformed the constructive heuristic of Bräsel et al. (1993), the tabu search of Liaw (1999a), and the simulated annealing of Liaw (1999b) on most of the instances. Puente et al. (2004) took the basic GA of Liaw (2000) (without the tabu search) and seeded the initial population with some heuristic chromosomes. These chromosomes correspond to non-delay and active schedules derived from probabilistic dispatching rules. Zobolas et al. (2009) combined a GA with a variable neighborhood search that applies a random change of the neighborhood of a solution (shaking), taken from a selected subset of the population, then starts a local search from the new point to find a local optimum. The algorithm was tested on the three known benchmark sets and on randomly generated instances. The results showed that this algorithm outperforms the GAs of Prins (2000) and Liaw (2000), and is competitive with the hybrid beam search-ant colony optimization of Blum (2005) and the particle swarm optimization of Sha and Hsu (2008). Ahmadizar and Hosseinabadi Farahani (2012) combined a GA with a local optimization heuristic based on randomized active schedules, and proposed a special crossover operator that preserves the relative order of jobs on machines. Computational experiments have been carried out on the three known benchmark sets. The results showed that the proposed algorithm outperforms the GAs of Prins (2000) and Liaw (2000), and is competitive with the hybrid beam search-ant colony optimization of Blum (2005) and the particle swarm optimization of Sha and Hsu (2008). Rahmani Hosseinabadi et al. (2019) studied the impact of four crossover and two mutation operators on the performance of a GA, and proposed an algorithm that combines the one-point and the linear order crossover operators and the `swap` and `move` mutations. Other metaheuristic approaches for the OS problem to minimize the makespan include ant colony optimization (Blum, 2005; Kurdi, 2022), particle swarm optimization (Sha and Hsu, 2008; Pongchairerks and Kachitvichyanukul, 2016), firefly algorithm (Lal et al., 2019), and bee colony optimization (Huang and Lin, 2011).

GAs have been also successfully applied to many variants of the OS problem. Abreu et al. (2020) developed a GA for the open shop problem with sequence-dependent setup times to minimize the sum of the completion times. The authors seeded the initial population with solutions obtained from a constructive heuristic, and presented three schedule builders to evaluate the chromosomes that are permutations of the non-null operations. Several selection, crossover, mutation and replacement operators have been tested, and a Taguchi experimental design (Phadke, 1995) has been used to evaluate the influence of each parameter and operator. Matta (2009) studied the proportionate multiprocessor open shop (also known as the hybrid open shop (de Araújo et al., 2021)) with the objective of minimizing the makespan. They introduced two mixed integer programming models; one that uses time-based decision variables while the second uses sequence-based decision variables. They also proposed a GA in which the chromosomes are permutations of the non-null operations such that each operation is indexed by its job and stage. Abreu et al. (2021) studied the integration of a production environment that consists of an open shop problem with a distribution system that is ensured by a capacitated single vehicle. The objective is to minimize the makespan of the system's production and distribution. The authors developed a mixed integer linear program and a biased random key genetic algorithm combined with an iterated greedy algorithm as a local search. Each individual of the population is a vector of random keys and a decoding scheme is proposed to convert individuals into operations sequences and job routes. For a more detailed literature on the OS problem and its variants, the reader is referred to the surveys of Strusevich (2022), Ahmadian et al. (2021) and Anand and Panneerselvam (2015).

In this paper, we develop and validate a GA for the m-machine OSC problem to enhance the state of the art results on this problem. The choice of the algorithm was motivated by the success of the evolutionary algorithms developed for the open shop problem. We also present mixed integer linear programming formulations that differ in the way the conflicting operations are modeled, and three different groups of lower bounds from which ten lower bounds are derived. Extensive computational experiments are conducted on a large set of instances derived from the benchmark instances given by Taillard (1993), Guéret and Prins (1999), and Brucker et al. (1997) for the basic open shop problem. The lower bounds are first used to evaluate the solutions of the mathematical formulations that are obtained within a fixed time limit. This step allows us to define lower and upper bounds on the optimal makespans and prove the optimality of many solutions, thereby providing a more robust evaluation of the GA's performance. Several experiments are also conduced to investigate the impact of different components and parameters on the performance of the GA. From these experiments, some problem-specific knowledge about the solution space are deduced and then incorporated in the GA to direct the search towards regions with better solutions. We further apply a variable neighborhood search to the final population of the GA to search for better solutions in the neighborhoods of the final population and escape from local optimum. The results of the selected variant of the GA demonstrate its effectiveness, where the majority of the instances are solved to optimality and the average deviations from the best lower bounds are very small. Furthermore, this variant improves the upper bounds obtained by the mathematical formulations and outperforms the two-phase heuristic of Tellache and Boudhar (2017). We provide, for future research in this area, the instances on which the experiments have been performed along with the best lower and upper bounds (or optimal makespans if proven).

The rest of this paper is organized as follows. Section 2 gives mixed integer linear programming formulations for the OSC problem. Section 3 presents lower bounds. A detailed description of the genetic algorithm's components is given in Section 4. The computational experiments are provided in Section 5.

2. Mathematical formulations

A solution to the m-machine OSC problem is characterized by three sets of lists: operations belonging to the same job, operations of the same machine, and operations of two conflicting jobs. Let J_{ij} and $J_{i'k}$ be two operations of one of the previous lists. In a feasible solution, we have either J_{ij} precedes $J_{i'k}$ or (exclusive) follows it, thus at any time at most one of these two operations is processed. These disjunctive constraints define the order of processing the operations of each list. In the following, we present three Mixed Integer Linear Programming (MILP) models that differ in the way the disjunctive constraints are formulated.

The first formulation is based on dichotomous constraints to assure that only one of each pair of constraints can hold. This idea was used in numerous scheduling problems to model disjunctive constraints (see Manne (1960) for the job shop problem). The parameter a_{jk} equals 1 if J_j and J_k are in conflict, 0 otherwise. M is defined as a very large constant. The decision variables are:

- C_{max}: the maximum completion time determined by the completion time of the last operation.

$$
(P_1)\quad
\begin{aligned}
\min\quad & C_{max} && && (1a)\\
s.t.\quad & C_{max} \geq C_{ij}, && i=1,\dots,m, j=1,\dots,n && (1b)\\
& C_{ij} - M(1-x_{ijk}) \leq C_{ik} - p_{ik} && i=1,\dots,m, 1\leq k<j\leq n && (1c)\\
& C_{ij} - C_{ik} - p_{ij} \geq -Mx_{ijk} && i=1,\dots,m, 1\leq k<j\leq n && (1d)\\
& C_{ij} - M(1-y_{ii'j}) \leq C_{i'j} - p_{i'j} && 1\leq i'<i\leq m, j=1,\dots,n && (1e)\\
& C_{ij} - C_{i'j} - p_{ij} \geq -My_{ii'j} && 1\leq i'<i\leq m, j=1,\dots,n && (1f)\\
& C_{ij} - M(1-r_{iji'k}) \leq C_{i'k} - p_{i'k} && \text{if } a_{jk}=1, 1\leq i'\neq i\leq m, 1\leq k<j\leq n && (1g)\\
& C_{ij} - C_{i'k} - p_{ij} \geq -Mr_{iji'k} && \text{if } a_{jk}=1, 1\leq i'\neq i\leq m, 1\leq k<j\leq n && (1h)\\
& x_{ijk}, y_{ii'j} \in \{0,1\} && 1\leq i'<i\leq m, 1\leq k<j\leq n && (1i)\\
& r_{iji'k} \in \{0,1\} && 1\leq i'\neq i\leq m, 1\leq k<j\leq n && (1j)\\
& C_{ij} \geq p_{ij} && i=1,\dots,m, j=1,\dots,n && (1k)
\end{aligned}
$$

Box I.

- C_{ij}: completion time of job J_j on machine M_i.
- $x_{ijk} = \begin{cases} 1 & \text{if job } J_j \text{ is scheduled any time before } J_k \text{ on machine } M_i,\\ 0 & \text{otherwise.}\end{cases}$
- $y_{ii'j} = \begin{cases} 1 & \text{if job } J_j \text{ is scheduled on } M_i \text{ then on } M_{i'},\\ 0 & \text{otherwise.}\end{cases}$
- $r_{iji'k} = \begin{cases} 1 & \text{if } a_{jk}=1 \text{ and the operation } J_{ij} \text{ is scheduled any time before } J_{i'k},\\ 0 & \text{otherwise.}\end{cases}$

The first MILP is summarized in (P_1) (see Box I).

Constraint (1b) equates the makespan to the maximum of the completion times of all operations. Constraints (1c) and (1d) are the paired disjunctive constraints between two operations of the same machine which insure that J_{ik} either precedes J_{ij} or (exclusive) follows it on M_i. Constraints (1e) and (1f) are the paired disjunctive constraints between two operations of the same job insuring that these operations cannot be processed at the same time on different machines. Constraints (1g) and (1h) are the paired disjunctive constraints between two operations of two conflicting jobs ensuring that these operations cannot be processed simultaneously on different machines.

In the second formulation, we replace each pair of inequality dichotomous constraints from (P_1) by an equality constraint that we set equal to a surplus variable, and a boundary constraint on the surplus variables to ensure feasibility as follows. For the pair {(1c), (1d)}, we define an additional continuous variable X_{ijk}(a surplus variable) that we set equal to $C_{ij} - C_{ik} - p_{ij} + Mx_{ijk}$ (constraint (2a)). From (1d), we have $X_{ijk} \geq 0$. Then, we replace $C_{ij} - C_{ik} - p_{ij} + Mx_{ijk}$ by X_{ijk} in (1c), which gives (2b). The next dichotomous constraints are replaced similarly. Therefore, the second MILP (P_2) is obtained by replacing the pair {(1c),(1d)} (respectively {(1e), (1f)}, and {(1g), (1h)}) of (P_1) by {(2a), (2b)} (respectively {(2c), (2d)}, and {(2e), (2f)}). This way of modeling the disjunctive constraints follows that of Liao and You (1992) used for the job shop problem.

$$
\begin{aligned}
& C_{ij} - C_{ik} + Mx_{ijk} - p_{ij} = X_{ijk} && i=1,\dots,m, 1\leq k<j\leq n && (2a)\\
& M - p_{ik} - p_{ij} \geq X_{ijk} && i=1,\dots,m, 1\leq k<j\leq n && (2b)\\
& C_{ij} - C_{i'j} + My_{ii'j} - p_{ij} = Y_{ii'j} && 1\leq i'<i\leq m, j=1,\dots,n && (2c)\\
& M - p_{i'j} - p_{ij} \geq Y_{ii'j} && 1\leq i'<i\leq m, j=1,\dots,n && (2d)\\
& C_{ij} - C_{i'k} + Mr_{iji'k} - p_{ij} = R_{iji'k} && \text{if } a_{jk}=1, 1\leq i'\neq i\leq m, 1\leq k<j\leq n && (2e)\\
& M - p_{i'k} - p_{ij} \geq R_{iji'k} && \text{if } a_{jk}=1, 1\leq i'\neq i\leq m, 1\leq k<j\leq n && (2f)\\
& X_{ijk}, Y_{ii'j} \geq 0 && 1\leq i'<i\leq m, 1\leq k<j\leq n && (2g)\\
& R_{iji'k} \geq 0 && 1\leq i'\neq i\leq m, 1\leq k<j\leq n && (2h)
\end{aligned}
$$

In the third formulation, we keep the first inequality of each pair of dichotomous constraints from (P_1) and we replace the second one by the sum of the two symmetric binary variables that we set equal to 1. For the pair {(1c), (1d)}, constraint (1c) insures that if job J_j precedes job J_k on M_i (i.e. $x_{ijk} = 1$), then the completion time of J_{ij} is less than or equal to the starting time of J_{ik}. Now, if $x_{ijk} = 0$, then J_k should precede job J_j on M_i that is $x_{ikj} = 1$, which is ensured by constraint (3d). The next dichotomous constraints are replaced similarly. Therefore, the MILP (P_3) is obtained by replacing the pair {(1c), (1d)} (respectively {(1e), (1f)} and {(1g), (1h)}) of (P_1) by {(3c), (3d)} (respectively {(3e), (3f)} and {(3g), (3h)}). The symmetries in the decision variables have to be considered in this model, so we replace (1i) and (1j) from (P_1) by (3i) in (P_3). Hassan et al. (2017) used this type of constraints to model the disjunctive constraints of the OS problem.

$$
\begin{aligned}
& C_{ij} - M(1-x_{ijk}) \leq C_{ik} - p_{ik} && i=1,\dots,m, 1\leq k\neq j\leq n && (3c)\\
& x_{ijk} + x_{ikj} = 1 && i=1,\dots,m, 1\leq k\neq j\leq n && (3d)\\
& C_{ij} - M(1-y_{ii'j}) \leq C_{i'j} - p_{i'j} && 1\leq i'\neq i\leq m, j=1,\dots,n && (3e)\\
& y_{ii'j} + y_{i'ij} = 1 && 1\leq i'\neq i\leq m, j=1,\dots,n && (3f)\\
& C_{ij} - M(1-r_{iji'k}) \leq C_{i'k} - p_{i'k} && \text{if } a_{jk}=1, 1\leq i'\neq i\leq m, 1\leq k\neq j\leq n && (3g)\\
& r_{iji'k} + r_{i'kij} = 1 && \text{if } a_{jk}=1, 1\leq i'\neq i\leq m, 1\leq k\neq j\leq n && (3h)\\
& x_{ijk}, y_{ii'j}, r_{iji'k} \in \{0,1\} && 1\leq i'\neq i\leq m, 1\leq k\neq j\leq n && (3i)
\end{aligned}
$$

The big M can be set in the experiments to be equal to the sum of the processing times of all operations.

3. Lower bounds

We present in this section three groups of lower bounds on the makespan. These bounds are used in the stopping criteria of the genetic algorithms and the variable neighborhood search and for benchmarking the solutions of the genetic algorithms, the two-phase heuristic approach of Tellache and Boudhar (2017), and the mathematical formulations of the previous section. The bounds LB_1 of Section 3.1, and LB_2 and LB_3 of Section 3.2 have been already presented in Tellache and Boudhar (2017). For the sake completeness, we give a description of these bounds in the respective sections.

3.1. Conflict constraints relaxation-based lower bound

This lower bound is derived by relaxing the conflict constraints between the jobs. The m-machine OSC problem is then reduced to the basic m-machine open shop problem. In this case the maximum of job lengths and of machine loads given by

$$LB_1 = \max\{\{\sum_{i=1}^{m} p_{ij} | j = 1, \dots, n\} \cup \{\sum_{j=1}^{n} p_{ij} | i = 1, \dots, m\}\}$$

is a valid lower bound on the makespan.

3.2. Weighted agreement graph-based lower bound

The idea of this lower bound is to find a set of conflicting jobs or operations with maximum total processing time. This latter value is a lower bound on the makespan since jobs or operations that are in conflict should be processed in disjoint time intervals.

Let W be a n-dimensional vector of the processing times of the jobs of V. If we regard these processing times as weights in $\overline{G} = (V, \overline{E})$, we obtain the weighted graph $\overline{G}_w = (V, \overline{E}, W)$. Any set of conflicting jobs corresponds to an independent set in $\overline{G}_w$. However, finding an independent set of maximum weight is a NP-hard problem (Garey and Johnson, 1979), thus three greedy algorithms, namely `GWMIN`, `GWMIN2`, and `GWMAX`, from Sakai et al. (2003) have been applied to $\overline{G}_w$. The resulting lower bounds are denoted LB_2, LB_3, and LB_4 respectively.

Other lower bounds can be obtained by considering the conflicts at the level of the operations. Let $\overline{G'} = (V', \overline{E'})$ be a graph constructed from the agreement graph $\overline{G} = (V, \overline{E})$ as follows. $V' = \{J_{ij} | p_{ij} \neq 0, i = 1, \dots, m, j = 1, \dots, n\}$ represents the operations of the jobs of V with non-null processing times, and two vertices are adjacent in $\overline{E'}$ if and only if the corresponding operations are not in conflict. Observe that $\overline{G'}$ is the agreement graph over the operations. Let W' be the vector of the processing times of the operations of V'. Again, if we regard these processing times as weights in $\overline{G'} = (V', \overline{E'})$, we obtain the weighted graph $\overline{G'}_w = (V', \overline{E'}, W')$. Any independent set in $\overline{G'}_w$ corresponds to a set of conflicting operations that need to be processed in disjoint time intervals. By applying the greedy algorithms `GWMIN`, `GWMIN2`, and `GWMAX` of Sakai et al. (2003) to $\overline{G'}_w$, we obtain three other bounds LB_5, LB_6, and LB_7 respectively.

Let us illustrate the computation of LB_2 and LB_3 on the example of Fig. 1. LB_2 selects a vertex v of maximum $w(v)/(d_{G_i}(v) + 1)$ ($w(v)$ is the weight of v and $d_{G_i}(v)$ is the degree of v in the graph G_i of the ith iteration), then deletes v and its neighbors from the graph. This process is repeated until no vertex remains, and the selected vertices form the independent set to be returned. The value associated with J_1 (respectively J_2, and J_3) is 2.33 (respectively 3, and 2). J_2 is then selected first and deleted with J_1 from the graph. We are left with J_3 that forms with J_2 an independent set of weight 10. LB_3 follows the same procedure, the difference is in the selection rule that maximizes $w(v)/(\sum_{u \in N^+_{G_i}(v)} w(u))$, where $N^+_{G_i}(v)$ is the neighborhood of v including v. The value associated with J_1 (respectively J_2, and J_3) is 0.41 (respectively 0.46, and 0.36). So, J_2 is selected first and deleted with J_1 from the graph, the resulting independent set is $\{J_2, J_3\}$ with weight 10.

3.3. Binary variables relaxation-based lower bound

This lower bound is obtained by running the MILP models of Section 2 on a solver for a fixed time limit and retrieving the best lower bound found so far. Note that this bound is at least as good as the relaxation of the MILP model (relaxing the binary variables). The resulting lower bounds are denoted LB_8, LB_9, and LB_{10} corresponding to (P_1), (P_2), and (P_3) respectively.

4. Genetic algorithm

GAs are metaheuristics that model the principal of natural evolution. Since their introduction by Holland (1975), they have been successfully applied to many NP-hard optimization problems including scheduling problems (see Davis (1985)). GAs are population-based algorithms, which start from an initial population of solutions and evolve through genetic operators (selection, crossover, mutation and replacement) to generate new populations. Each solution is coded as a chromosome, and each chromosome is made of genes that characterize the solution. This latter is evaluated using a problem-specific function, often related to the objective function, called the fitness. The process of GAs can be summarized as follows. Given an initial population, some solutions (parents) are selected by a selection operator to undergo recombination (crossover, mutation) and generate new solutions (children). Then, a replacement operator is called to replace old solutions by some of the children. This process is repeated on the new populations until a stopping criteria is met. The selection and replacement operators are related to the fitness function and defined to ensure the intrinsic parallelism of GAs (Prins, 2000).

Our algorithm follows the general structure described above. Its components are detailed in the following.

4.1. Chromosome representation

Mapping the set of solutions onto a set of chromosomes is usually done in a way that allows reaching all existing solutions and that maintains the feasibility of the generated chromosomes during the evolution process. As far as open shop problems are concerned, a permutation-based representation is often used, and the corresponding schedule is retrieved by means of heuristics (Prins, 2000; Liaw, 2000). The advantage of this representation is that it is adaptable to any genetic operators. For the OSC problem, we follow this representation and we define a chromosome as an ordered permutation π of the non-zero operations, so each gene stands for an operation.

4.2. Chromosome evaluation

In order to evaluate a chromosome, we decode the corresponding permutation π to a schedule using some schedule builders. The resulting makespan is the fitness value of the chromosome, and the fittest chromosome is the one with the smallest fitness value.

A schedule can be built by processing the operations on the machines according to their order in the permutation. The obtained schedule belongs to the class of semi-active schedules, in which no operation can finish earlier without changing the order of processing on any of the machines. There is another class of schedules imposing a more restrictive assumption called the class of active schedules. In this class, no operation can be started earlier in the schedule without violating feasibility. The class of active schedules is a sub-class of the class of semi-active schedules. An even smaller class is the sub-class of non-delay (or dense) schedules. In a non-delay schedule, no machine is kept idle while an operation that can be processed on that machine is waiting. A non-delay schedule is also active but the reverse is not necessarily true, thus the class of non-delay schedules is a subclass of the class of active schedules (Pinedo, 2008). In practice, restricting the search to the class of non-delay schedules tends to produce better makespans in average, however there is no guarantee that it can contain an optimal schedule.

In this section, we present three schedule builders: Algorithms 1 and 2 producing schedules within the class of active schedules and Algorithm 3 generating schedules within the class of non-delay schedules.

In Algorithm 1, the operations are inserted one by one following the order of π. At each insertion, the operation is started as early as possible subject to the feasibility constraints of the OSC problem. One can check that the obtained schedule is active. In the implementation, we associate to each machine M_i and job J_j a list of gaps denoted $G(M_i)$ and $G(J_j)$ respectively (similar to Prins (2000)). $G(M_i)$ keeps track of the idle times of M_i, and $G(J_j)$ tracks the gaps on which an operation of J_j can be processed without being in parallel with an operation of the same job or with an operation of a conflicting job. Therefore, an operation J_{ij} can be scheduled in an interval $[s,t]$ if, and only if, $t-s \geq p_{ij}$ and there exists $[a,b] \in G(M_i)$ and $[c,d] \in G(J_j)$ such that $[s,t] \subseteq [a,b] \cap [c,d]$. After each insertion of an operation J_{ij}, the lists $G(M_i)$ and $G(J_j)$ as well as the list $G(J_k)$ of every job J_k that is in conflict with J_j are updated as follows. If J_{ij} is scheduled within $[e, e+p_{ij}]$, then for every gap $[a,b]$ of these lists, we check if $[a,b]$ intersects with $[e, e+p_{ij}]$. If that is the case, $[a,b]$ is deleted if $a = e$ and $b = e+p_{ij}$, reduced if $a = e$ or (exclusive) $b = e+p_{ij}$, and split into two intervals if $e < a$ and $b < e + p_{ij}$.

Algorithm 1 Build active schedules

```
Inputs: permutation $\pi$, processing times $(p_{ij})_{1\leq i\leq m, 1\leq j\leq n}$, adjacency matrix $A$ of $G$
1: Initialize $G(M_i)$ and $G(J_j)$ to $\{[0,+\infty[\}$ for each $M_i$ and $J_j$;
2: for $o = 1,\dots,|\pi|$ do
3:     Find interval $[s,t] = [a,b] \cap [c,d]$ for $J_{ij} = \pi(o)$ such that $[a,b] \in G(M_i)$, $[c,d] \in G(J_j)$, $s$ is minimal, and $t-s \geq p_{ij}$;
4:     $c_{ij} = s + p_{ij}$;
5:     Update $G(M_i)$, $G(J_j)$, and $G(J_k)$ for every $k$ such that $A[j][k] = 1$;
Outputs: completion times $(c_{ij})_{1\leq i\leq m, 1\leq j\leq n}$ and the makespan.
```

There are at most $\lceil \frac{U}{2} \rceil$ gaps in the lists $G(M_i)$ and $G(J_j)$, where U is an upper bound on the makespan (U can be set to $\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}$). Therefore, for each operation, at most $O(U^2)$ iterations are required to find the interval $[s,t]$. The updates in Step 5 are done in at most $O(nU)$ iterations. It follows that the overall running time of Algorithm 1 is bounded by $O(|\pi| \max\{U^2, nU\})$ ($O(nmU^2)$ when $p_{ij} \neq 0$ for all $i = 1,\dots,m$ and $j = 1,\dots,n$).

Algorithm 2 gives another way of building active schedules following the mechanism of the well-known algorithm of Giffler and Thompson (1960).

Algorithm 2 Build active schedules by Giffler and Thompson mechanism (Giffler and Thompson, 1960)

```
Inputs: permutation $\pi$, processing times $(p_{ij})_{1\leq i\leq m, 1\leq j\leq n}$, adjacency matrix $A$ of $G$
1: Initialize the earliest starting times $(s_{ij})_{1\leq i\leq m, 1\leq j\leq n}$ of all the operations to zero;
2: while $\pi \neq \emptyset$ do
3:     Select an operation $J_{i'j'}$ of $\pi$ with the minimum earliest completion time $s_{i'j'} + p_{i'j'}$;
4:     Select the first operation $J_{ij}$ of $\pi$ that is in conflict with $J_{i'j'}$ (including $J_{i'j'}$) such that $s_{ij} < s_{i'j'} + p_{i'j'}$;
5:     $c_{ij} = s_{ij} + p_{ij}$;
6:     $\pi = \pi \setminus \{J_{ij}\}$;
7:     for $J_{i''j''} \in \pi$ that are in conflict with $J_{ij}$ do
8:         if $s_{i''j''} < s_{ij} + p_{ij}$ then
9:             $s_{i''j''} = s_{ij} + p_{ij}$;
Outputs: completion times $(c_{ij})_{1\leq i\leq m, 1\leq j\leq n}$ and the makespan.
```

Lemma 1. *Algorithm 2 produces feasible schedules that are active.*

Proof. The **for** loop of Step 7 enforces the earliest starting time of any of the remaining operations of π to be greater than or equal to the completion time of all the operations that have been scheduled and that are in conflict with that operation, and this ensures the feasibility of the schedule.

From Steps 3 and 4, we have $s_{i'j'} + p_{i'j'} \in [s_{ij}, s_{ij} + p_{ij}]$. The earliest starting times that have been increased after selecting J_{ij} are those of the operations that are in conflict with J_{ij} and that have an earliest starting time less than $s_{ij} + p_{ij}$. Suppose that we start one of these operations earlier than $s_{ij}+p_{ij}$. From Step 3, its completion time will be greater than or equal to $s_{i'j'} + p_{i'j'}$. One can check that the processing of this operation will intersect with J_{ij}, which violates the constraints of conflict between these two operations. Thus, the schedule produced by Algorithm 2 is active. □

There are $|\pi|$ steps needed to schedule all non-null operations in Algorithm 2. In each step, the number of iterations required to select the operation to be inserted and to update the remaining operations that are in conflict with the inserted operation is bounded by $O(|\pi|)$. Therefore, the final schedule is obtained within the time bound $O(|\pi|^2)$ ($O(n^2m^2)$ when $p_{ij} \neq 0$ for all $i = 1,\dots,m$ and $j = 1,\dots,n$).

The third schedule builder generates non-delay schedules. It is summarized in Algorithm 3.

Algorithm 3 Build non-delay schedules

```
Inputs: permutation $\pi$, processing times $(p_{ij})_{1\leq i\leq m, 1\leq j\leq n}$, adjacency matrix $A$ of $G$
1: Initialize the earliest starting times $(s_{ij})_{1\leq i\leq m, 1\leq j\leq n}$ of all the operations to zero;
2: while $\pi \neq \emptyset$ do
3:     Select the first operation $J_{ij}$ of $\pi$ with the minimum earliest starting time $s_{ij}$;
4:     $c_{ij} = s_{ij} + p_{ij}$;
5:     $\pi = \pi \setminus \{J_{ij}\}$;
6:     for $J_{i'j'} \in \pi$ that are in conflict with $J_{ij}$ do
7:         if $s_{i'j'} < s_{ij} + p_{ij}$ then
8:             $s_{i'j'} = s_{ij} + p_{ij}$;
Outputs: completion times $(c_{ij})_{1\leq i\leq m, 1\leq j\leq n}$ and the makespan.
```

Lemma 2. *Algorithm 3 produces feasible schedules that are non-delay.*

Proof. The proof of feasibility is similar to that of Lemma 1. At Step 3, we are scheduling an operation J_{ij} with the earliest starting time. Therefore, any other operation $J_{i'j'}$ has $s_{i'j'} \geq s_{ij}$. Now, if $J_{i'j'}$ is in conflict with J_{ij} and $s_{i'j'} < s_{ij} + p_{ij}$, then scheduling this operation earlier than $s_{ij} + p_{ij}$ intersects with J_{ij}. Thus, the **for** loop of Step 6 ensures feasibility without keeping any machine idle while it can process some operations. The lemma follows. □

We have $|\pi|$ steps to schedule all non-null operations in Algorithm 3. In each step, the number of iterations required to select the operation with the minimum earliest starting time and to update the remaining operations that are in conflict with the selected operation is bounded by $O(|\pi|)$. Therefore, the overall running time is bounded by $O(|\pi|^2)$ ($O(n^2m^2)$ when $p_{ij} \neq 0$ for all $i = 1,\dots,m$ and $j = 1,\dots,n$).

4.3. Initial population

We maintain a population P of size PS in which the chromosomes have distinct makespans. Each chromosome is evaluated with one of the schedule builders of the previous section. For the initial population, we consider two scenarios: generating all the chromosomes randomly, or seeding the initial population with some chromosomes obtained by sorting the operations according to eight priority rules. These priority rules are: decreasing order of p_{ij}, increasing order of p_{ij}, decreasing

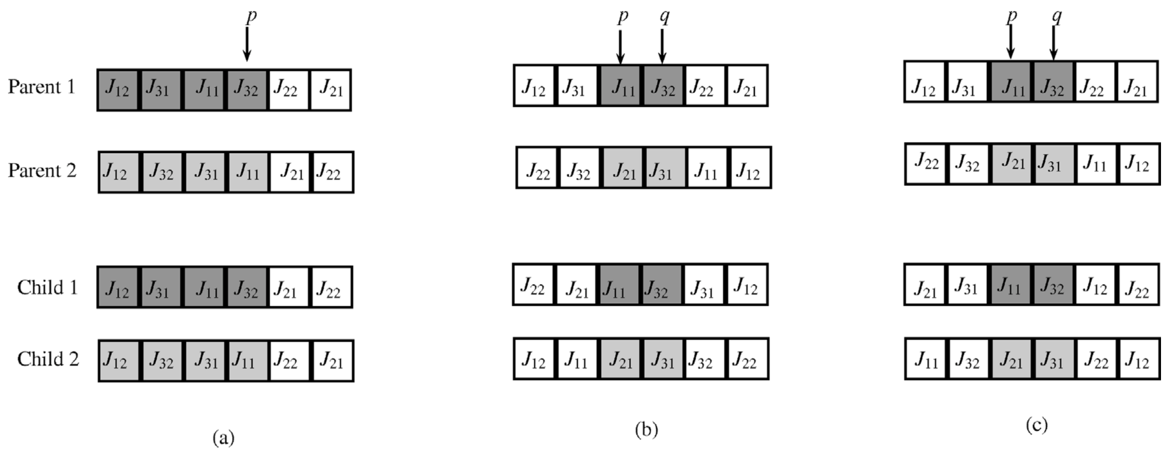

p
q
Parent 1
Parent 2
Child 1
Child 2
J_{12} J_{31} J_{11} J_{32} J_{22} J_{21}
J_{12} J_{32} J_{31} J_{11} J_{21} J_{22}
J_{12} J_{31} J_{11} J_{32} J_{21} J_{22}
J_{12} J_{32} J_{31} J_{11} J_{22} J_{21}
(a)
J_{22} J_{32} J_{21} J_{31} J_{11} J_{12}
J_{22} J_{21} J_{11} J_{32} J_{31} J_{12}
J_{12} J_{11} J_{21} J_{31} J_{32} J_{22}
(b)
J_{21} J_{31} J_{11} J_{32} J_{12} J_{22}
J_{11} J_{32} J_{21} J_{31} J_{22} J_{12}
(c)

Fig. 2. Crossover operators: (a) one-point crossover (X1). (b) linear order crossover (LOX). (c) order crossover (OX).

order of the conflict degrees f_{ij}, increasing order of the conflict degrees f_{ij}, decreasing order of f_{ij}/p_{ij}, increasing order of f_{ij}/p_{ij}, decreasing order of a_{ij}/p_{ij} (a_{ij} is the agreement degree of J_{ij}), and increasing order of a_{ij}/p_{ij}. The conflict degree f_{ij} of an operation J_{ij} is the number of operations that cannot be processed in parallel with J_{ij} (excluding the operations of the same machine). The agreement degree a_{ij} of an operations J_{ij} is the number of operations that can be processed in parallel with J_{ij}.

To generate a population with distinct makespans, we check at each iteration whether the makespan of the generated chromosome is redundant or not. In case of non-redundancy, the chromosome is added to the population, otherwise, it is dropped and we try up to max_tries times to obtain new makespan. If, at a given iteration, we fail after max_tries tries, the population size PS is set to the size of the generated population.

4.4. Crossover and mutation

Once the initial population is generated, the GA explores the search space by generating new chromosomes through crossover and mutation operators.

The crossover operates on two parent chromosomes to produce children by information exchange between the parents. This operator attempts to combine good characteristics of both parents and generate new chromosomes that may have even better characteristics. Several crossover operators have been designed for permutation-based representations. In our algorithm, we use one of the three following operators: one-point crossover (X1) (proposed by Holland (1975) and adapted to a chromosome permutation of operations by Reeves (1995)), order crossover (OX) (Goldberg, 1989), and linear order crossover (LOX) (Falkenauer and Bouffouix, 1991). In X1, the parents are cut at a random position, then a part from each other is passed to the two children. In LOX, a subsequence of operations from one parent is passed to a child while preserving the same positions. Then, the remaining operations are inserted in the empty positions of the child starting from the beginning of the chromosome and following their order in the second parent that is scanned from left to right. OX is similar to LOX, the only difference is that the scan of the second parent and the insertion of the operations in the child start after the subsequence, where the chromosomes are considered cycles. These three operators are illustrated in Fig. 2 for an instance with 3 jobs, 2 machines, and all $p_{ij} \geq 0$.

The mutation operates on one chromosome and consists of a random perturbation in its genes. This operator helps to keep a reasonable level of population diversity and to prevent a premature convergence by allowing the chromosomes to evolve independently. Many mutation operators have been presented in the literature for chromosomes that are permutations. In this work, we apply the swap (Reeves, 1995) and the move (Reeves, 1995) mutations. In the two operators, we select two positions at random and then we either swap the corresponding operations (for the swap operator) or insert the operation of one of the two positions into the other position (for the move mutation). The two operators are illustrated in Fig. 3 for an instance with 3 jobs, 2 machines, and all $p_{ij} \geq 0$.

4.5. Selection and replacement procedures

Chromosomes of the current population are selected and replaced based on their fitness value. We adopt the ranking mechanism of Reeves (1995). The population is sorted in decreasing order of the makespan, so the chromosome at position k in the sorted population has rank k. The two parents that will undergo crossover are selected as follows. A chromosome at rank k is selected as first parent with probability $2k/(PS(PS+1))$, whereas the second parent is selected with probability $1/PS$. One of the two resulting children is selected at random and will be mutated with probability p_m. The chromosome that will be replaced is chosen randomly from those with rank below the median $\lfloor PS/2 \rfloor$.

4.6. Termination

The algorithm is stopped when we find a chromosome with makespan equals the best lower bound, i.e. an optimal solution is reached. We also consider a maximum number of iterations max_iter, this number depends on the population size and the size of the instance and will be specified in the experiments.

4.7. The overall algorithm

The general framework of our genetic algorithm can now be summarized. The pseudocode is given in Algorithm 4 in which we follow the method used in Prins (2000). $C_{max}(T)$ is a makespan of the chromosome T that is obtained by one of the schedule builders of Section 4.2 (same for $C_{max}(C)$). Rand is a function generating uniformly distributed random numbers in $[0, 1]$.

The GA starts with the generation of an initial population as described in Section 4.3. After that, it iterates until one of the stopping criteria given in Section 4.6 is met. In each iteration, two parent chromosomes are selected from the population using the selection procedure of Section 4.5. One of the crossover operators of Section 4.4 is applied to the two parents to generate two children chromosomes. One child is selected randomly and undergoes mutation with probability p_m using one of the mutation operators of Section 4.4. If the mutated child has new makespan (or the child if this is not the case), the replacement procedure of Section 4.5 is called to replace a chromosome of the

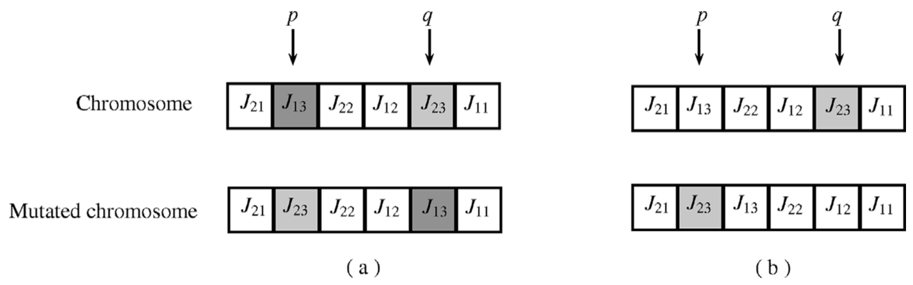

p
q
Chromosome
Mutated chromosome
J_{21} J_{13} J_{22} J_{12} J_{23} J_{11}
J_{21} J_{23} J_{22} J_{12} J_{13} J_{11}
(a)
J_{21} J_{13} J_{22} J_{12} J_{23} J_{11}
J_{21} J_{23} J_{13} J_{22} J_{12} J_{11}
(b)

Fig. 3. Mutation operators: (a) `swap` the genes in positions p and q. (b) `move` the gene in position q to position p.

current population by the mutated child (or the child if the mutated child has not a new makespan but the child has). Once all the iterations of the GA are completed, the best schedule from the last population is returned as the output.

We first run in Section 5.3 a variant of this GA that uses one schedule builder for the evaluation of the chromosomes. The knowledge gained from these experiments about the solutions of the search space is utilized in Section 5.4 to test another variant, in which we call randomly one of the schedule builders each time a chromosome requires evaluation. This combination of the schedule builders allows the algorithm to reach different regions of the search space that might contain better solutions.

Algorithm 4 General framework of the GA.

```
 1: Generate an initial population;
 2: while none of the stopping criterion is met do
 3:     Select two parents $P_1$ and $P_2$;
 4:     Apply a crossover operator to $P_1$ and $P_2$;
 5:     Randomly select one child $C$;
 6:     if Rand< $p_m$ then
 7:         $T := C$,
 8:         Mutate $T$ using a mutation operator;
 9:         if $C_{max}(T)$ is not redundant then
10:             $C := T$;
11:     if $C_{max}(C)$ is not redundant then
12:         Select a chromosome $R$ to be replaced;
13:         Replace $R$ by $C$;
    Outputs: the best makespan with the corresponding schedule.
```

4.8. Variable neighborhood search

To further improve the results of the GA, we apply to the best chromosomes generated by the GA a Variable Neighborhood Search (VNS) algorithm. VNS has been proposed by Hansen and Mladenović (2001) and consists of combining a change of neighborhood structures with a local search approach. It starts by performing a random change of neighborhood for the current solution x (shaking), then applies a local search to the new solution x' to improve it. Let x'' be the resulting local optimum. If x'' is better than x, then move there, otherwise investigate another neighborhood structure of x. We implement the VNS algorithm presented in Zobolas et al. (2009) with a modification to the stopping criteria, in which we check, at each iteration, whether the makespan of the selected solution equals the best lower bound. Furthermore, when a chromosome needs evaluation, we call the three schedule builders, Algorithms 1, 2 and 3, and choose the best makespan. The neighborhood operators used during the shaking phase are the `move` and `swap`. In the local search phase, two additional operators are used: `Or-Opt` (Or, 1976) (similar to `move` but it inserts two adjacent operations instead of one) and `2-Opt` (Croes, 1958) (similar to `swap` but it also reverses the order of the operations between the two swapped ones).

5. Computational experiments

The above mathematical models, lower bounds, genetic algorithms, and the two-phase heuristic approach of Tellache and Boudhar (2017) have been coded in C++ and tested on a machine with 62.5 GiB of RAM and 10 cores at 2.80 GHz. The mathematical models were run by calling Gurobi 9.12.

The experiments have been carried out on three sets of instances derived from the benchmark instances given by Taillard (1993), Guéret and Prins (1999), and Brucker et al. (1997) for the open shop problem. For each of these instances, we generate conflict graphs using the $G(n,p)$ Erdős and Rényi method (Erdős and Rényi, 1959). Given n vertices, the $G(n,p)$ method generates a simple undirected graph where each element of the C_2^n possible edges is present with probability p. When the value of p increases, the graph density increases too. All the instances have $n = m$.

Taillard's set (Taillard, 1993) provides 60 instances, with 10 instances for each size: 4 × 4, 5 × 5, 7 × 7, 10 × 10, 15 × 15 and 20 × 20. Guéret and Prins (1999) generated other instances having a large deviation between the lower bound they proposed in Guéret and Prins (1999) and the trivial lower bound LB_1. The sizes range from 3 × 3 to 10 × 10, with 10 instances for each size, so the total is 80 instances. All rows and columns sum up to 1000. Brucker et al. (1997) suggested 73 instances ranging from 3 × 3 to 9 × 9 with $LB_1 = 1000$. To make these instances hard, they measured the difference between LB_1 and the $MinP = \min\{\{\sum_{i=1}^{m} p_{ij} | j = 1,\dots,n\} \cup \{\sum_{j=1}^{n} p_{ij} || i = 1,\dots,m\}\}$. $MinP$ is either 1000, 900, 800, or 700 giving deviations from LB_1 of 0, 10, 20, or 30%. For each deviation, three instances are provided for each instance size except 3 × 3, 8 × 8, and 9 × 9. For 3 × 3 and 8 × 8, we have two instances with 0%, and for 9 × 9, we have only three instances with 30%.

For each of the previous open shop instances, we generated 5 conflict graphs for each value of p, $p \in \{0.2, 0.5, 0.8\}$, resulting in 900, 1200, and 1095 instances derived from the instances of Taillard (1993), Guéret and Prins (1999), and Brucker et al. (1997) respectively.

The rest of this section is organized as follows. In Section 5.1, we analyze the performance of the lower bounds of Section 3. The MILP models of Section 2 are addressed in Section 5.2. In Section 5.3, we fine-tune the parameters of the GA and report the impact of each component on its performance. Finally, in Section 5.4, we evaluate combining the calls to different schedule builders and using VNS, then we compare the resulting GA variants with the two-phase heuristic approach of Tellache and Boudhar (2017).

5.1. Performance of the lower bounds

The results of the lower bounds on the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.) are reported in Table 1 (respectively Table 2, and Table 3). Fig. 4 compares the lower bounds in terms of the number of times a given lower bound yields the best bound. We set a time limit of 30 min for the MILP models on all the instances, except the 20 × 20 instances derived from Taillard, in which the time limit is set to one hour for $p = 0.2$ and

Table 1
Comparison between the lower bounds on the instances derived from Taillard instances.

		LB_1		LB_2		LB_3		LB_4		LB_5		LB_6		LB_7		LB_8		LB_9		LB_{10}	
		a	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*
$p = 0.2$	4 × 4	0	30.14	60	1.78	60	1.78	62	1.25	54	3.33	62	1.58	26	8.51	**98**	**0.09**	**98**	**0.09**	**98**	**0.09**
	5 × 5	0	44.65	84	0.48	88	**0.36**	**90**	0.54	60	3.49	78	1.39	14	5.09	80	5.84	72	11.28	84	4.65
	7 × 7	2	48.64	84	1.26	**90**	**0.34**	84	1.51	44	5.78	60	3.36	8	5.54	0	52.94	0	65.01	0	52.93
	10 × 10	0	55.00	**86**	**1.60**	**86**	1.92	70	2.15	54	3.93	62	2.63	6	8.15	0	73.55	0	80.88	0	73.28
	15 × 15	0	61.49	**70**	**2.14**	**70**	2.72	64	5.13	42	5.12	60	3.83	0	13.13	0	86.94	0	90.23	0	87.11
	20 × 20	0	63.59	**68**	**3.28**	66	3.33	52	4.94	32	5.88	50	4.28	0	16.75	0	89.87	0	93.65	0	89.93
	All	0.33	50.58	64	1.21	**65.67**	**1.18**	61.67	1.76	42.33	3.61	53.67	2.13	9	6.74	29.67	36.56	28.33	41.25	30.33	36.34
$p = 0.5$	4 × 4	0	51.89	98	0.09	**100**	**0**	94	0.27	56	2.37	74	0.92	34	4.05	74	4.74	78	4.49	72	5.21
	5 × 5	0	57.47	88	0.42	92	**0.34**	**94**	0.46	54	2.43	84	0.64	24	3.59	36	26.80	24	37.86	38	27.30
	7 × 7	0	64.44	90	0.60	**94**	**0.50**	84	1.94	60	3.53	74	1.17	14	3.75	0	71.52	0	76.68	0	71.36
	10 × 10	0	69.52	72	1.68	**80**	**0.73**	64	2.34	46	4.20	64	1.64	2	6.18	0	82.67	0	87.01	0	82.37
	15 × 15	0	73.88	**70**	**1.76**	68	2.49	56	4.57	42	3.66	50	3.28	2	7.89	0	90.44	0	93.10	0	90.37
	20 × 20	0	78.14	**68**	**3.03**	64	3.71	52	4.39	40	4.73	48	4.20	0	10.23	0	93.68	0	96.13	0	93.57
	All	0	65.89	69.67	0.76	**72.33**	**0.68**	65.33	1.60	43	2.70	57.67	1.27	12.67	4.24	18.33	46.03	17	49.86	18.33	46.10
$p = 0.8$	4 × 4	0	63.24	**100**	**0**	**100**	**0**	96	0.12	76	1.62	86	0.34	58	2.61	42	17.24	36	21.41	34	20.13
	5 × 5	0	69.69	98	0.05	**100**	**0**	94	0.43	72	0.94	78	0.52	44	1.98	0	56.04	0	63.57	0	55.15
	7 × 7	0	76.08	96	0.10	**100**	**0**	94	0.86	68	2.06	92	0.11	20	2.33	0	80.94	0	84.22	0	81.10
	10 × 10	0	80.67	**90**	**0.17**	**90**	0.25	86	1.71	58	1.77	70	1.25	0	3.25	0	89.56	0	91.66	0	89.53
	15 × 15	0	84.79	76	0.77	74	**0.70**	**78**	1.17	36	1.71	62	0.91	0	3.06	0	94.73	0	96.55	0	94.76
	20 × 20	0	86.71	**62**	**0.81**	60	1.24	54	3.31	34	2.44	44	1.98	0	5.66	0	96.37	0	97.63	0	96.37
	All	0	76.86	76.67	0.18	**77.33**	**0.16**	74.67	0.71	51.67	1.35	64.67	0.52	20.33	2.21	7	56.42	6	59.57	5.67	56.78

a(%): Number of times in percentage a given lower bound yields the best bound.
b(%): Average percentage deviation from the best lower bound.

Table 2
Comparison between the lower bounds on the instances derived from Guéret and Prins instances.

		LB_1		LB_2		LB_3		LB_4		LB_5		LB_6		LB_7		LB_8		LB_9		LB_{10}	
		a	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	*b*
$p = 0.2$	3 × 3	0	30.74	46	7.74	46	7.74	46	7.74	44	7.74	44	7.74	22	8.08	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	0	43.83	76	5.16	76	5.16	76	5.16	76	5.16	74	5.16	0	7.52	**98**	0.03	**98**	0.03	**98**	**0.01**
	5 × 5	0	48.19	80	3.65	80	3.65	80	3.65	74	6.65	72	3.66	0	8.16	**82**	0.59	78	0.53	**82**	**0.42**
	6 × 6	0	53.84	**96**	**0.01**	**96**	**0.01**	**96**	**0.01**	86	5.01	82	4.04	0	4.85	16	1.73	4	2.15	10	1.84
	7 × 7	0	53.12	**84**	**1.19**	**84**	**1.19**	**84**	**1.19**	78	3.86	64	3.12	0	7.23	14	3.00	4	3.76	10	2.84
	8 × 8	0	56.89	**88**	**1.12**	**88**	**1.12**	**88**	**1.12**	82	3.12	84	1.42	0	8.77	4	11.08	6	14.66	4	10.99
	9 × 9	0	58.61	**94**	**0.89**	**94**	**0.89**	**94**	**0.89**	86	3.89	80	4.09	0	10.00	2	18.65	2	23.09	4	17.37
	10 × 10	0	58.00	**98**	**0.67**	**98**	**0.67**	88	4.00	82	7.00	74	6.90	0	14.01	0	25.12	0	33.98	0	22.58
	All	0	50.40	**82.75**	**2.55**	**82.75**	**2.55**	81.5	2.97	76	5.3	71.75	4.52	2.75	8.58	39.5	7.53	36.5	9.78	38.5	7.01
$p = 0.5$	3 × 3	0	48.35	86	2.02	86	2.02	86	2.02	80	2.04	78	2.04	46	2.39	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	0	54.51	**90**	1.72	**90**	1.72	**90**	1.72	**90**	1.72	76	1.92	4	3.34	74	1.03	74	1.14	74	**1.02**
	5 × 5	0	60.27	**88**	**0.87**	**88**	**0.87**	**88**	**0.87**	**88**	**0.87**	74	0.91	0	3.52	40	1.19	38	1.43	42	1.18
	6 × 6	0	67.33	98	0.67	98	0.67	96	1.33	**100**	**0**	90	0.73	0	3.88	0	5.80	0	6.69	0	6.28
	7 × 7	0	68.77	92	1.17	92	1.17	86	3.01	**96**	**0.01**	86	0.15	0	5.76	2	13.10	2	16.27	4	12.83
	8 × 8	0	70.37	**96**	**1.17**	**96**	**1.17**	94	2	94	1.83	94	1.19	0	7.06	0	29.34	0	35.25	0	29.61
	9 × 9	0	71.53	**96**	1.17	**96**	1.17	**96**	**1**	94	1.67	92	1.17	0	7.90	0	40.66	0	49.29	0	39.92
	10 × 10	0	73.60	**92**	**2.00**	**92**	**2.00**	82	4.30	80	5.07	86	3.50	0	8.55	0	50.56	0	61.11	0	49.65
	All	0	64.34	**92.25**	**1.35**	**92.25**	**1.35**	89.75	2.03	90.25	1.65	84.5	1.45	6.25	5.30	27	17.71	26.75	21.40	27.5	17.56
$p = 0.8$	3 × 3	0	57.67	**100**	**0**	**100**	**0**	**100**	**0**	88	0.04	82	0.06	78	0.22	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	0	66.50	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	88	0.10	26	0.77	46	5.32	46	4.62	46	4.47
	5 × 5	0	73.33	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	86	0.08	10	1.18	0	10.09	0	12.44	0	11.47
	6 × 6	0	75.50	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	92	0.10	2	2.08	0	17.29	0	20.43	0	16.24
	7 × 7	0	78.18	**100**	**0**	**100**	**0**	98	0.50	**100**	**0**	88	0.07	2	3.64	0	34.56	0	38.81	0	34.61
	8 × 8	0	80.41	**100**	**0**	**100**	**0**	98	0.40	**100**	**0**	82	0.39	0	3.29	0	52.73	0	60.60	0	49.63
	9 × 9	0	81.85	**98**	**0.40**	**98**	**0.40**	94	1.13	**98**	**0.40**	94	0.46	0	3.68	0	59.58	0	70.06	0	59.29
	10 × 10	0	82.91	**100**	**0**	**100**	**0**	96	0.57	**100**	**0**	98	0.05	0	4.13	0	66.67	0	76.56	0	66.37
	All	0	74.54	**99.75**	**0.05**	**99.75**	**0.05**	98.25	0.32	98.25	0.05	88.75	0.16	14.75	2.37	18.25	30.78	18.25	35.44	18.25	30.26

a(%): Number of times in percentage a given lower bound yields the best bound.
b(%): Average percentage deviation from the best lower bound.

to three hours for $p = 0.5$ and $p = 0.8$. We increased the time limit because the solver was not able to find a feasible solution during some runs, more details will be given in the next section since this concerns the upper bounds. The deviation from the best lower bound is given by $((BestLB - LB_i)/BestLB) \times 100$, where $BestLB = \max\{LB_i | i = 1, \dots, 10\}$ is the best lower bound. The last row of each table reports the results of all the instances, and the bold numbers of each row indicate the best results of the corresponding instances.

As it can be observed from Tables 1–3 and Fig. 4, the trivial lower bound LB_1 is outperformed by all the other bounds on all the instances, and this shows the importance of taking into account the constraints of conflicts. The lower bounds derived from the MILP models are more efficient than the other bounds on small size instances, and this performance decreases while increasing the density of the conflict graph because of the increase in the size of these models. If we compare between these bounds, it seems that LB_8 and LB_{10} are competitive and

Table 3
Comparison between the lower bounds on the instances derived from Brucker et al. instances.

		LB_1		LB_2		LB_3		LB_4		LB_5		LB_6		LB_7		LB_8		LB_9		LB_{10}	
		a	*b*	*a*	*b*	*a*	*b*	*a*	*b*	*a*	b	*a*	*b*	*a*	b	*a*	*b*	*a*	*b*	*a*	*b*
$p = 0.2$	3 × 3	20	24.97	49.09	3.94	49.09	3.94	49.09	3.94	49.09	5.20	56.36	2.89	41.82	5.74	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	10	33.26	68.33	1.53	68.33	1.53	68.33	1.53	66.67	3.49	68.33	3.07	21.67	6.93	**100**	**0**	**100**	**0**	**100**	**0**
	5 × 5	3.33	42.83	83.33	**0.71**	83.33	**0.71**	80	1.22	56.67	6.18	60	5.79	13.33	5.92	**85**	3.00	80	3.69	80	3.98
	6 × 6	0	47.36	86.67	0.51	88.33	0.48	**91.67**	**0.24**	68.33	2.86	76.67	2.19	3.33	7.80	1.67	14.81	0	17.98	3.33	16.11
	7 × 7	0	52.25	83.33	1.29	90	0.93	**91.67**	**0.13**	58.33	5.01	58.33	4.94	1.67	8.35	0	35.19	0	39.27	0	35.32
	8 × 8	0	52.76	78.18	2.32	80	2.24	**89.09**	**0.39**	61.82	5.28	63.64	4.66	0	11.10	0	41.63	0	46.84	0	43.25
	9 × 9	0	50.21	**86.67**	1.46	**86.67**	1.46	**86.67**	**0.49**	33.33	10.07	40	7.72	0	13.74	0	50.40	0	55.79	0	52.41
	All	5.20	42.66	75.62	1.67	77.26	1.59	**78.90**	**1.19**	59.18	4.88	63.01	4.09	12.88	7.87	45.75	17.06	44.66	19.37	45.20	17.78
$p = 0.5$	3 × 3	1.82	43.64	90.91	0.78	90.91	0.78	90.91	0.78	74.54	1.43	76.36	0.86	54.54	2.66	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	1.67	49.08	95	0.07	**96.67**	**0.03**	90	0.33	66.67	1.87	71.67	1.88	25	5.07	81.67	2.25	83.33	2.52	81.67	2.60
	5 × 5	0	58.34	96.67	0.09	**98.33**	**0.03**	93.33	0.67	75	1.40	81.67	1.04	16.67	3.70	33.33	21.21	33.33	22.60	36.67	20.50
	6 × 6	0	63.17	86.67	1.59	85	1.85	**88.33**	**0.55**	66.67	2.39	68.33	1.87	5	5.01	0	41.76	0	43.32	0	41.36
	7 × 7	0	66.11	90	0.06	**93.33**	**0.06**	81.67	1.69	50	5.29	61.67	4.06	1.67	6.24	0	56.33	0	59.41	0	56.30
	8 × 8	0	69.76	80	0.35	80	**0.33**	**81.82**	2.42	54.54	2.59	60	2.28	0	5.71	0	63.62	0	68.77	0	62.97
	9 × 9	0	67.54	**86.67**	**1.42**	80	1.85	53.33	5.46	40	3.76	53.33	2.70	0	10.97	0	64.83	0	74.41	0	66.42
	All	0.55	58.77	89.86	**0.53**	**90.41**	0.57	86.30	1.24	63.56	2.56	69.31	2.04	16.16	5.00	33.97	32.23	34.25	34.44	34.52	32.07
$p = 0.8$	3 × 3	0	53.13	94.54	0.22	94.54	0.22	94.54	0.22	85.45	0.56	83.64	0.35	70.91	1.68	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	0	63.29	95	0.10	**96.67**	**0.08**	95	0.11	80	0.74	76.67	0.60	41.67	1.89	30	10.72	35	12.62	30	11.90
	5 × 5	0	70.76	**100**	**0**	**100**	**0**	**100**	**0**	73.33	0.48	73.33	0.45	28.33	2.03	0	45.88	0	49.63	0	44.86
	6 × 6	0	73.53	**100**	**0**	**100**	**0**	98.33	0.01	61.67	0.62	63.33	0.53	8.33	2.25	0	58.66	0	59.96	0	57.75
	7 × 7	0	77.19	**100**	**0**	98.33	0.04	95	0.09	75	0.88	73.33	0.57	5	2.83	0	70.21	0	72.61	0	69.85
	8 × 8	0	79.11	**96.36**	**0.08**	**96.36**	**0.08**	90.91	0.34	69.09	0.57	65.45	0.70	3.64	2.67	0	73.73	0	79.79	0	73.06
	9 × 9	0	79.00	**80**	**1.12**	**80**	**1.12**	73.33	1.60	60	1.88	73.33	0.49	6.67	4.18	0	76.54	0	83.48	0	76.71
	All	0	69.99	**96.99**	**0.11**	**96.99**	0.11	94.79	0.18	73.42	0.69	72.60	0.53	25.20	2.31	20	44.74	20.82	47.48	20	44.47

a(%): Number of times in percentage a given lower bound yields the best bound.
b(%): Average percentage deviation from the best lower bound.

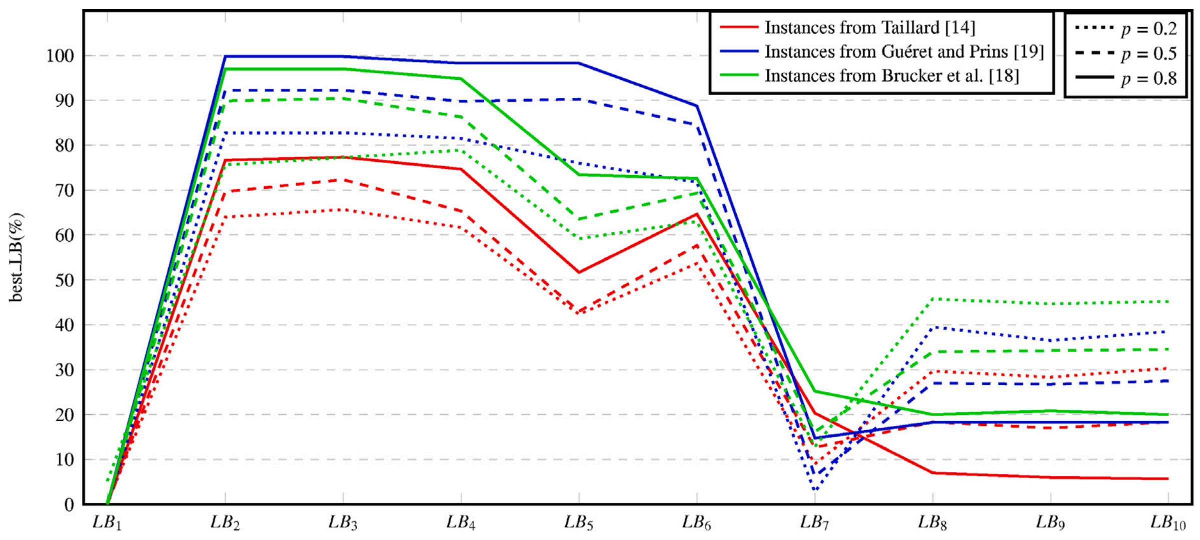


Fig. 4. Performance of the lower bounds in terms of the number of times a given lower bound yields the best bound (best_LB(%)).

they yield in general better results than LB_9. Regarding the agreement graph-based lower bounds, we can observe that the bounds based on $\overline{G}_w$ (agreement graph over jobs) are more efficient than those that are based on $\overline{G'}_w$ (agreement graph over operations), except on the instances derived from Guéret and Prins in which LB_5 produces the best bounds on some instances with $p = 0.5$ and $p = 0.8$ (see Fig. 4). If we consider the greedy algorithms of Sakai et al. (2003) that are used in the computation of these bounds, `GWMIN` and `GWMIN2` seem to yield better bounds in comparison to `GWMAX`. Recall that, `GWMIN` and `GWMIN2` are based on the same mechanism that is different from the one of `GWMAX`. In `GWMIN` and `GWMIN2`, we select a vertex following a specific rule, then we remove it and its neighbors from the graph. The independent set to be returned is formed of the selected vertices. Whereas in `GWMAX`, we delete at each iteration a vertex, according to a specific rule, until no edge remains. In this case, the independent set to be returned is formed of the remaining vertices.

In order to check whether the previous differences between the ten lower bounds in terms of *a* and *b* of Tables 1–3 are statistically significant, we conducted a Friedman test. The *p*-values obtained are less than 0.00001, indicating that there is indeed significant difference between the lower bounds in terms of the average percentage deviation from the best lower bound and the number of times a given lower bound yields the best bound.

Overall, the bounds derived from the MILP models are yielding the best results on all the instances of size 3 × 3, on the instances with $p = 0.2$ and size up to 4 × 4, and on the instances derived from Guéret and Prins with size up to 5 × 5 and $p = 0.2$. On most of the remaining instances, LB_2 and LB_3 perform better than the other bounds, where

Table 4
Comparison between the upper bounds on the instances derived from Taillard instances.

		(P_1)				(P_2)				(P_3)			
		a	b	c	d	a	b	c	d	a	b	c	d
$p = 0.2$	4 × 4	**100**	**0**	**98**	**0.087**	**100**	**0**	**98**	**0.087**	**100**	**0**	**98**	**0.087**
	5 × 5	**100**	**0**	80	5.842	**100**	**0**	72	11.282	**100**	**0**	**84**	**4.655**
	7×7	**100**	**0**	0	52.951	84	0.126	0	65.058	**100**	**0**	0	**52.934**
	10 × 10	90	**0.049**	0	74.076	26	5.301	0	82.261	**92**	0.078	0	73.815
	15 × 15	**58**	**1.002**	0	**88.314**	0	23.460	0	92.788	44	1.382	0	88.520
	20 × 20	**68**	**1.309**	0	**92.182**	0	65.618	0	96.809	32	2.804	0	92.341
	All	**86**	**0.393**	29.667	52.242	51.667	15.751	28.333	58.047	78	0.711	**30.333**	**52.059**
$p = 0.5$	4 × 4	**100**	**0**	74	4.741	**100**	**0**	**78**	**4.486**	**100**	**0**	72	5.206
	5 × 5	**100**	**0**	36	**26.799**	**100**	**0**	24	37.857	**100**	**0**	**38**	27.301
	7×7	**100**	**0**	0	71.604	62	0.601	0	76.888	92	0.041	0	**71.464**
	10 × 10	**60**	**0.528**	0	83.341	4	13.741	0	88.984	58	0.588	0	**83.051**
	15 × 15	28	3.110	0	92.547	0	39.224	0	95.899	**72**	**1.015**	0	**92.319**
	20 × 20	**70**	**1.587**	0	**95.821**	0	50.327	0	98.238	30	4.030	0	95.846
	All	**76.333**	**0.871**	**18.333**	**62.475**	44.333	17.315	17	67.059	75.333	0.946	**18.333**	62.531
$p = 0.8$	4 × 4	**100**	**0**	**42**	**17.239**	**100**	**0**	36	21.412	**100**	**0**	34	20.128
	5 × 5	**100**	**0**	0	56.038	98	0.012	0	63.581	**100**	**0**	0	**55.147**
	7×7	**86**	**0.043**	0	**80.998**	58	0.508	0	84.333	74	0.086	0	81.162
	10 × 10	44	**0.734**	0	89.973	2	9.103	0	92.593	**54**	0.841	0	**89.952**
	15 × 15	**58**	**1.641**	0	**95.550**	2	34.213	0	97.783	40	2.010	0	95.593
	20 × 20	**64**	**1.628**	0	97.670	8	–	0	–	28	3.869	0	97.721
	All	**75.333**	**0.674**	**7**	**72.911**	44.667	–	6	–	66	1.134	5.667	73.284

a(%): Number of times in percentage a given upper bound yields the best bound.
b(%): Average percentage deviation from the best upper bound.
c(%): Number of times in percentage Gurobi completed with status "optimal solution found".
d(%): Average gap reported by Gurobi in percentage.

LB_2 (respectively LB_3) yields the best bounds on 84.131% (respectively 84.851%) of the instances, and the average deviation from the best lower bounds equals 0.96% (respectively 0.944%). Note that the difference between LB_2 and LB_3 is in the selection rule of the vertices. In LB_2, the selected vertex maximizes the function $w(v)/(d_{G_i}(v)+1)$, where $w(v)$ is the weight of vertex v and $d_{G_i}(v)$ is the degree of v in the graph G_i of the ith iteration. Whereas in LB_3, the selected vertex maximizes the function $w(v)/(\sum_{u\in N^+_{G_i}(v)} w(u))$, where $N^+_{G_i}(v)$ is the neighborhood of v including v. These two bounds are competitive (the p-values, from the Friedman test, on the criteria a and b are equal to 0.378 and 0.450 respectively, indicating that there are no statistically significant differences between LB_2 and LB_3) and they give exactly the same results on the instances derived from Guéret and Prins as shown in Table 2. We think that this latter is due to the fact that all the jobs of this group of instances have the same processing time. On the other hand, LB_4 and LB_5 are showing few good results, in particular on the instances derived from Guéret and Prins and from Brucker et al. for LB_4, and on the instances derived from Guéret and Prins for LB_5 (see Fig. 4).

5.2. Performance of the upper bounds

In this section, we report the results of the runs of the MILP models. As mentioned in the previous section, we set a time limit of 30 min. However, during some runs on the 20 × 20 instances, the solver cannot find any feasible solution. We increased the time limit to one hour for $p = 0.2$ and to three hours for $p = 0.5$ and $p = 0.8$. This solved the problem for all the models, except for (P_2) which is still not able to find a feasible solution for 10 instances with $p = 0.8$.

The results on the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.) are shown in Table 4 (respectively Table 5, and Table 6). Fig. 5 depicts the performance of the three models in terms of the number of times a given model yields the best upper bound. The deviation from the best upper bound is given by $((UB_i - BestUB)/BestUB) \times 100$, where UB_i is the makespan found by (P_i) and $BestUB = \min\{UB_i | i = 1,\dots,3\}$ is the best upper bound. The gap returned by Gurobi is defined as $(|UB - LB|/|UB|) \times 100$, where UB is the incumbent objective value (that is UB_i of (P_i)), and LB is the lower bound. Note that we reported this lower bound in the previous section in LB_8, LB_9, and LB_{10}. The last row of each table reports the results of all the instances, and the bold numbers of each row indicate the best results of the corresponding instances. We also report in the fourth and fifth columns of Tables 10–12 two information that are important in this discussion: the number of times the best lower bound coincides with the best upper bound in percentage, and the average gap between the best lower and upper bounds in percentage. This gap is given by $((BestUB - BestLB)/BestUB) \times 100$.

In Table 4, we do not report the criteria b and d for (P_2) on the 20 × 20 instances with $p = 0.8$, because of the instances on which (P_2) cannot find a feasible solution within the time limit. However, the 8% of the best upper bounds found by (P_2) on this group of instances are used in the performance criteria requiring $BestUB$.

From Tables 4–6, we can observe that the three models have similar results on small size instances. The performance differences become apparent when increasing the density of the conflict graphs and the size of the instances. The clear difference is in the performance of (P_2) that becomes weaker in comparison to (P_1) and (P_3) (see Fig. 5). However, (P_1) and (P_3) are closely competitive and their results are slightly different. The Friedman test on the three models resulted in p-values that are less than 0.00001 for both a and b of Tables 4–6. However, by considering the results of (P_1) and (P_3), the p-values are equal to 0.900 and 0.374 for a and b respectively, which confirms the previous observations. If we consider the results of each model, we can see that increasing the density and the size of the instances decreases the number of instances solved to optimality and increases the gap reported by Gurobi. This was expected because of the increase in the number of decision variables. What was not expected is that the instances derived from Taillard are the hardest to solve followed by those derived from Brucker et al. and Guéret and Prins, even when comparing instances of the same size. This is not the case for the basic open shop problem, where the Brucker et al. instances were generated in order to be more difficult than the existing instances at that time including those of Taillard.

Given the time limitation, the model (P_1) (respectively (P_3)) solved to optimality 18.333% (respectively 18.111%) of the instances derived from Taillard, 27.75% (respectively 27.667%) of the instances derived

Table 5
Comparison between the upper bounds on the instances derived from Guéret and Prins instances.

		(P_1)				(P_2)				(P_3)			
		a	*b*	*c*	*d*	*a*	*b*	*c*	*d*	*a*	*b*	*c*	*d*
$p = 0.2$	3 × 3	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	**100**	**0**	**98**	0.029	**100**	**0**	**98**	0.029	**100**	**0**	**98**	**0.008**
	5 × 5	**100**	**0**	**82**	0.593	**100**	**0**	78	0.534	**100**	**0**	**82**	**0.422**
	6 × 6	**100**	**0**	**16**	**1.730**	**100**	**0**	4	2.147	**100**	**0**	10	1.841
	7 × 7	**100**	**0**	**8**	3.077	**100**	**0**	4	3.831	**100**	**0**	**8**	**2.909**
	8 × 8	**100**	**0**	**2**	12.337	96	0.002	**2**	15.893	**100**	**0**	**2**	**12.248**
	9 × 9	78	0.038	0	19.293	52	0.061	0	23.725	**94**	**0.004**	0	**18.050**
	10 × 10	72	0.101	0	26.565	28	2.086	0	36.565	**78**	**0.035**	0	**24.011**
	All	93.75	0.017	**38.25**	7.953	84.5	0.269	35.75	10.340	**96.5**	**0.005**	37.5	**7.436**
$p = 0.5$	3 × 3	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	**100**	**0**	**74**	1.029	**100**	**0**	**74**	1.138	**100**	**0**	**74**	**1.021**
	5 × 5	**100**	**0**	38	1.193	**100**	**0**	38	1.429	**100**	**0**	**40**	**1.180**
	6 × 6	**100**	**0**	0	**5.801**	**100**	**0**	0	6.695	**100**	**0**	0	6.278
	7 × 7	90	0.003	2	13.609	90	0.005	2	16.794	**94**	**0.002**	**4**	**13.354**
	8 × 8	76	0.082	0	**29.871**	54	0.299	0	35.862	**84**	**0.048**	0	30.138
	9 × 9	**68**	**0.247**	0	42.419	24	3.846	0	52.364	60	0.496	0	**41.853**
	10 × 10	**66**	0.480	0	52.929	8	4.677	0	64.378	58	**0.321**	0	**51.945**
	All	**87.5**	**0.101**	26.75	18.356	72	1.103	26.75	22.332	87	0.108	**27.25**	**18.221**
$p = 0.8$	3 × 3	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	**100**	**0**	**46**	5.324	**100**	**0**	**46**	4.622	**100**	**0**	**46**	**4.471**
	5 × 5	**100**	**0**	0	**10.095**	**100**	**0**	0	12.441	**100**	**0**	0	11.468
	6 × 6	**100**	**0**	0	**17.291**	98	0.0005	0	20.429	**100**	**0**	0	16.244
	7 × 7	**78**	**0.022**	0	**35.132**	64	0.057	0	39.371	62	0.066	0	35.209
	8 × 8	**62**	**0.101**	0	53.204	30	1.097	0	61.400	56	0.146	0	**50.200**
	9 × 9	50	**0.488**	0	60.570	18	2.799	0	71.440	**52**	0.520	0	**60.302**
	10 × 10	44	0.988	0	68.878	4	6.176	0	79.108	**62**	**0.504**	0	**68.430**
	All	**79.25**	0.200	**18.25**	31.312	64.25	1.266	**18.25**	36.101	79	**0.154**	**18.25**	**30.790**

a(%): Number of times in percentage a given upper bound yields the best bound.
b(%): Average percentage deviation from the best upper bound.
c(%): Number of times in percentage Gurobi completed with status "optimal solution found".
d(%): Average gap reported by Gurobi in percentage.

Table 6
Comparison between the upper bounds on the instances derived from Brucker et al. instances.

		(P_1)				(P_2)				(P_3)			
		a	*b*	*c*	*d*	*a*	*b*	*c*	*d*	*a*	*b*	*c*	*d*
$p = 0.2$	3 × 3	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	5 × 5	**100**	**0**	**85**	**3.002**	**100**	**0**	80	3.694	**100**	**0**	80	3.976
	6 × 6	**100**	**0**	1.667	**14.809**	**100**	**0**	0	17.986	**100**	**0**	**3.333**	16.115
	7 × 7	98.333	0.0008	0	**35.261**	96.667	0.079	0	39.357	**100**	**0**	0	35.386
	8 × 8	**98.182**	**0.003**	0	**41.639**	74.545	0.230	0	46.967	94.545	0.014	0	43.258
	9 × 9	93.333	0.014	0	**50.413**	60	2.092	0	56.671	**100**	**0**	0	52.410
	All	**99.178**	**0.001**	**45.753**	**17.070**	93.973	0.134	44.657	19.440	**99.178**	0.002	45.205	17.792
$p = 0.5$	3 × 3	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	**100**	**0**	81.667	**2.249**	**100**	**0**	**83.333**	2.523	**100**	**0**	81.667	2.603
	5 × 5	**100**	**0**	33.333	21.221	**100**	**0**	33.333	22.612	**100**	**0**	**36.667**	**20.509**
	6 × 6	98.333	**0.002**	0	41.815	98.333	0.007	0	43.373	**100**	0	0	**41.405**
	7 × 7	**90**	**0.028**	0	56.693	65	0.600	0	59.963	88.333	0.031	0	**56.677**
	8 × 8	69.091	0.207	0	63.969	34.545	2.920	0	69.894	**76.364**	**0.166**	0	**63.322**
	9 × 9	**66.667**	**0.184**	0	**65.673**	26.667	6.393	0	76.618	**66.667**	0.206	0	67.315
	All	92.055	0.044	33.973	32.389	81.096	0.802	34.246	34.799	**93.151**	**0.039**	**34.521**	**32.230**
$p = 0.8$	3 × 3	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	4 × 4	**100**	**0**	30	**10.720**	**100**	**0**	**35**	12.623	**100**	**0**	30	11.898
	5 × 5	**100**	**0**	0	45.884	**100**	**0**	0	49.628	**100**	**0**	0	**44.859**
	6 × 6	**95**	**0.004**	0	58.675	81.667	0.068	0	60.002	90	0.031	0	**57.784**
	7 × 7	66.667	0.198	0	70.310	48.333	0.846	0	72.918	**73.333**	**0.160**	0	**69.940**
	8 × 8	**65.454**	**0.355**	0	74.246	29.091	1.929	0	80.530	58.182	0.417	0	**73.610**
	9 × 9	46.667	**0.754**	0	**77.341**	26.667	4.341	0	84.663	**53.333**	0.837	0	77.543
	All	**86.301**	**0.118**	20	44.874	74.794	0.619	**20.822**	47.697	85.753	0.129	20	**44.604**

a(%): Number of times in percentage a given upper bound yields the best bound.
b(%): Average percentage deviation from the best upper bound.
c(%): Number of times in percentage Gurobi completed with status "optimal solution found".
s *d*(%): Average gap reported by Gurobi in percentage.

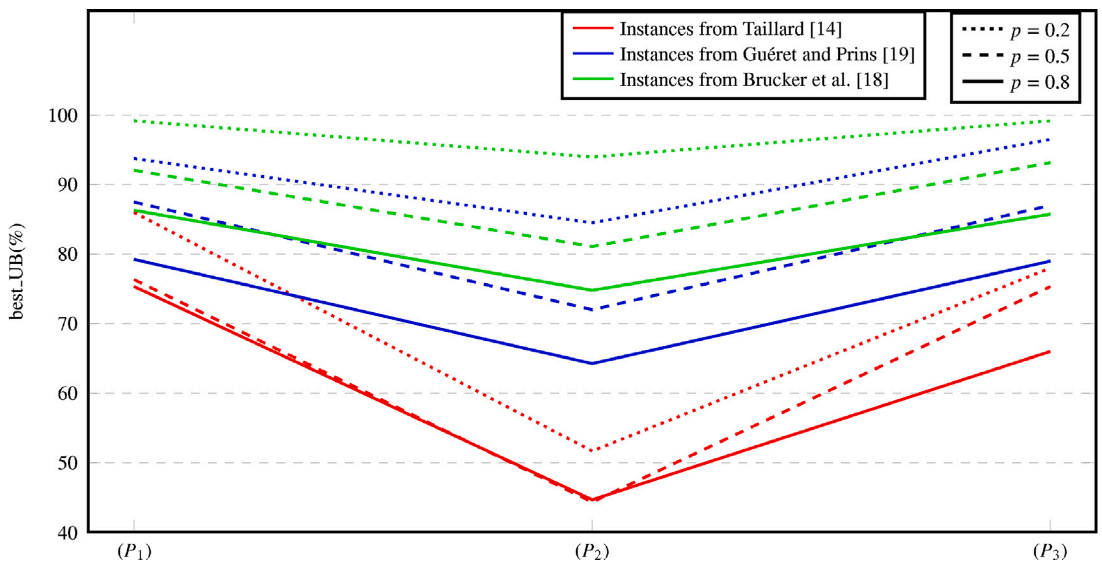


Fig. 5. Performance of the upper bounds in terms of the number of times a given upper bound yields the best bound (best_UB(%)).

from Guéret and Prins, and 33.242% (respectively 33.242%) of the instances derived from Brucker et al. The average gap reported by Gurobi equals 62.543% (respectively 62.625%) on the instances derived from Taillard, 19.207% (respectively 18.816%) on the instances derived from Guéret and Prins, and 31.444% (respectively 31.542%) on the instances derived from Brucker et al. Recall, from the previous section, that LB_2 and LB_3 outperform the lower bounds derived from the MILP models on most of the instances. Therefore, we wanted to see how far are the best upper bounds, that we obtained here, from the best lower bounds of the previous section. The third and fourth columns of Tables 10–12 show that the best lower and upper bounds coincide in 54.667% (respectively 77.167%, and 89.771%) of the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.), and the average gap between these two bounds equals 7.656% (respectively 0.822%, and 0.217%). Clearly, this significantly increases the number of instances with proven optimal makespans by a factor of 2.785 and reduces the gap between the bounds of the remaining instances.

5.3. Setting of the GA

We first investigate the impact of different components on the performance of the genetic algorithm. Recall that we have four main components. (a) The algorithm used in the chromosome evaluation: Algorithm 1 for active schedules, Algorithm 2 for active schedules following the mechanism of Giffler and Thompson (1960), and Algorithm 3 for non-delay schedules. (b) The type of the initial population: generated randomly or seeded with some specific permutations of operations. (c) The crossover type: `X1`, `OX`, and `LOX`. (d) The mutation type: `swap`, and `move`. We tested every combination of these components (36 different variants of the GA) on the three groups of instances. Each variant has been run 20 times for each instance, and the best and the average makespans are recorded for the computation of the following performance criteria (we denote these makespans by Z in this paragraph). For the instances with $BestUB = BestLB$, i.e. the optimal makespan opt is proven, (respectively instances with $BestUb \neq BestLB$), we compute the number of times the makespan Z coincides with opt (respectively with $BestLB$), and the deviation of Z from opt (respectively from $BestLB$) that is given by $((Z - opt)/opt) \times 100$ (respectively by $((Z - BestLB)/BestLB) \times 100$). The results are given in columns d and e (respectively f and g) of Tables 10–15. We also calculate, for all the instances, the deviation of Z from $BestUB$ that is given by $((BestUB - Z)/BestUB) \times 100$ (column h in Tables 10–15).

We observed from the implementation that Algorithms 1 and 2 perform well on small size instances and on instances with low density of the conflict graph (4×4 all densities, and 5×5 with $p = 0.2$ for the instances derived from Taillard, 3×3 and 4×4 all densities, and 5×5 with $p = 0.2$ for those derived from Guéret and Prins, and 3×3 all densities, and 4×4 and 5×5 with $p = 0.2$ for those derived from Brucker et al.). However, their performances decrease in comparison to Algorithm 3 when increasing the size and the density of the instances. The schedules produced by Algorithm 3 are on average short. This results in initial populations and number of generations (before a stopping criteria is met) that are smaller than the ones obtained when calling Algorithms 1 and 2. If we compare the two latter algorithms which are producing active schedules, Algorithm 2 outperforms in general Algorithm 1, except on 4×4 all densities derived from Brucker et al. on which Algorithm 1 outperforms Algorithm 2, and on 4×4 and 5×5 with $p = 0.2$ derived from Taillard, 4×4 with $p = 0.2$ and 0.5 and 5×5 with $p = 0.2$ derived from Guéret and Prins, and 5×5 with $p = 0.2$ and 0.5 derived from Brucker et al. on which Algorithm 2 and Algorithm 1 are competitive. The gap between the results of these two latter algorithms increases when increasing the density and the size of the instances. For the generation of the initial population, we observed better results when seeding the population with some specific permutations mainly for large size instances with high densities. Regarding the crossover operators, `LOX` and `OX` work better than `X1` on most of the instances, while `X1` yields the best results on few large size instances. `LOX` and `OX` are close, but `LOX` is better than `OX` on the majority of the cases. The mutation operators `move` and `swap` are closely competitive and they produce similar results especially on the instances derived from Taillard and Brucker et al.

Fig. 6 (respectively 7) shows the effects of the variation of the previous four components on the average percentage deviation from $BestLB$ (respectively the average number of times in percentage the makespan coincides with $BestLB$) including when $BestUB = BestLB$ for the three values of p ("random" is when the initial population is generated randomly and "hybrid" is when it is seeded with some specific permutations). The two figures reinforce the observations of the previous paragraph and shows that seeding the population with some specific permutations, using `LOX` and Algorithm 3 result in an improvement of the quality of the solutions, except for $p = 0.2$ in which Algorithm 2 yields the best average results. It should be noted that this latter exception is observed only on the small size instances given in the beginning of the previous paragraph. We further conducted a

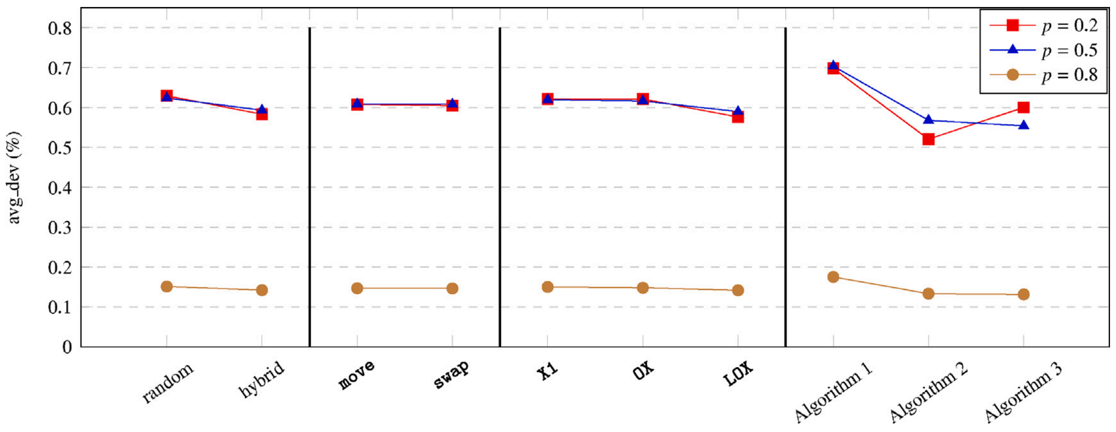


Fig. 6. Effects of the variation of the four GA's components on the average percentage deviation from the best lower bound (avg_dev (%)).

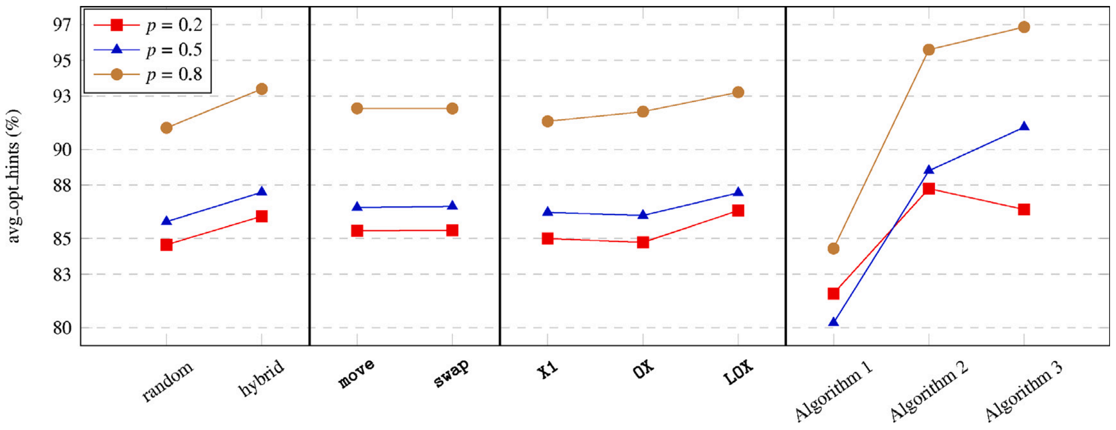


Fig. 7. Effects of the variation of the four GA's components on the number of times a given GA variant coincides with the best lower bound (avg_opt_hints (%)).

Table 7
Analysis of variance of the GA for the average percentage deviation from the best lower bound ($\alpha = 0.05$).

Source	DF	$p = 0.2$				$p = 0.5$				$p = 0.8$			
		Adj. SS	Adj. MS	F-value	p-value	Adj. SS	Adj. MS	F-value	p-value	Adj. SS	Adj. MS	F-value	p-value
Initial population	1	0.019311	0.019311	18.26	0.000	0.008440	0.008440	13.96	0.001	0.000729	0.000729	11.37	0.002
Mutation	1	0.000086	0.000086	0.08	0.778	0.000004	0.000004	0.01	0.936	0.000003	0.000003	0.04	0.834
Crossover	2	0.016357	0.008178	7.73	0.002	0.006413	0.003207	5.30	0.011	0.000462	0.000231	3.60	0.040
Schedule builder	2	0.189456	0.094728	89.57	0.000	0.164375	0.082188	135.96	0.000	0.014832	0.007416	115.65	0.000
Error	29	0.030670	0.001058			0.017530	0.000604			0.001860	0.000064		
Total	35	0.255879				0.196763				0.017886			

Table 8
Analysis of variance of the GA for the number of times in percentage the makespan coincides with the best lower bound ($\alpha = 0.05$).

Source	DF	$p = 0.2$				$p = 0.5$				$p = 0.8$			
		Adj. SS	Adj. MS	F-value	p-value	Adj. SS	Adj. MS	F-value	p-value	Adj. SS	Adj. MS	F-value	p-value
Initial population	1	22.937	22.937	36.35	0.000	24.551	24.551	49.91	0.000	42.82	42.816	50.25	0.000
Mutation	1	0.009	0.009	0.01	0.907	0.040	0.040	0.08	0.776	0.00	0.001	0.00	0.980
Crossover	2	22.738	11.369	18.02	0.000	11.447	5.724	11.64	0.000	16.54	8.271	9.71	0.001
Schedule builder	2	232.844	116.422	184.50	0.000	797.297	398.648	810.46	0.000	1121.42	560.709	658.10	0.000
Error	29	18.299	0.631			14.264	0.492			24.71	0.852		
Total	35	296.827				847.600				1205.48			

multi-factor analysis of variance (ANOVA) (Tables 7, and 8) to see if the impact of each of the four factors is significant on the performance of the GA. Tables 7, and 8 show that the type of the initial population, the crossover operator, and the schedule builders are statistically significant, and that when increasing the density of the conflict graph, the significance of the type of the initial population and the crossover operator decreases especially for the average percentage deviation from the best lower bound.

Regarding the parameters of the GA, we set max_tries to 1000 and max_iter to $100 \times PS \times \max\{n, m\}$. We tried four values of the population size PS (100, 200, 300, and 400), and two values of the mutation rate p_m (0.2, and 1) on two groups of instances from each of the three sets (the 10×10 and 15×15 derived from Taillard, the 8×8 and 10×10 derived from Guéret and Prins, and the 5×5 and 7×7 derived from Brucker et al.). The eight combinations of the values of PS and p_m have been tested on a GA that uses Algorithm 3 for the computation

Table 9
Operators and parameters of GA-ND.

Factors	value
Initial population	hybrid
Mutation	`move`
Crossover	`LOX`
Schedule builder	Algorithm 3
p_m	1
PS	300
max_tries	1000
max_iter	$100 \times PS \times \max\{n, m\}$

of the makespan, an initial population with distinct makespans seeded with some specific permutations, the `LOX` crossover, and the `move` mutation. We observed no significant differences between the solutions of the eight versions on the instances of the set of Brucker et al. On the remaining instances, systematic mutation slightly outperforms $p_m = 0.2$. On the other hand, increasing the population size requires longer CPU time (which was expected), but this is not yielding necessarily better results and the differences are not significant in many cases, so we decided to set PS to 300. A multi-factor analysis of variance (ANOVA) on PS and p_m showed no significant impact on the performance of the GA with p-values that are equal to 0.690 and 0.974 respectively.

We present in Tables 10–12 the results of the variant that uses a population of 300 chromosomes with distinct makespans seeded with some specific permutations, Algorithm 3 for the computation of the makespan, the `LOX` crossover, and a systematic `move` mutation (Table 9). We denote this variant by `GA-ND`. Dashes "–" in these tables mean that we either do not have instances with proven optimal makespans (for d and e) or all the instances have proven optimal makespans (for f and g). Similarly to the previous tables, the row "All" reports the results of all the instances corresponding to a given density. We can observe from these results that the performance of `GA-ND` increases when increasing the density of the conflict graph. This can be explained by the sets of feasible schedules that are smaller and the lower bounds that are tighter. In terms of CPU time, the instances with $p = 0.2$ (respectively $p = 0.5$, and $p = 0.8$) require on average 9.500 (respectively 14.117, and 7.480) seconds, while the average CPU time over all the instances is 10.365 seconds. This indicates that instances with medium density of the conflict graph are the hardest to solve. On the set derived from Taillard (respectively Guéret and Prins, and Brucker et al.), `GA-ND` hits 95.935% (respectively 99.136%, and 96.541%) of the optimal makespans found in the previous section, and the average deviation from these optimal makespans is 0.071% (respectively 0.006%, and 0.094%). On the other hand, `GA-ND` coincides with the best lower bounds on 65.441% (respectively 79.562%, and 84.821%) of the remaining instances (i.e. on instances with $BestUb \neq BestLB$), and the average deviation from these lower bounds equals 2.172% (respectively 1.877%, and 1.019%). In summary, `GA-ND` solved to optimality 82.111% (respectively 94.666%, and 95.342%) of the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.), and the average deviation from the best lower bounds (including the proven optimal makespans) is 1.0236% (respectively 0.433%, and 0.188%). The comparison of these latter results with those of the previous section show that `GA-ND` improves on average the best upper bounds obtained by the MILP models in terms of both optimal makespans found and deviation from the best lower bounds, and this is in a short CPU time compared to the time limit we set for the MILP models. The deviation of `GA-ND` from these best upper bounds is slightly negative on some small size instances with low densities, but it increases when increasing the size and the density of the instances, where the average deviation over all the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.) equals 6.895% (respectively 0.459%, and 0.042%).

Table 10
Results of `GA-ND` on the instances derived from Taillard instances.

	Ins.	a	b	c	Best					Average							
					d	e	f	g	h	d	e	f	g	h	i	j	k
$p = 0.2$	4 × 4	50	100	0	68	0.584	–	–	−0.584	68	0.584	–	–	−0.584	0.112	17.049	2675.520
	5 × 5	50	100	0	92	0.114	–	–	−0.114	92	0.114	–	–	−0.114	0.156	75.911	3003.080
	7 × 7	50	100	0	100	0	–	–	0	99.700	0.001	–	–	−0.001	0.106	154.545	476.309
	10 × 10	50	76	1.687	100	0	33.333	7.383	0.161	100	0	33.333	7.383	0.161	3.439	175.997	40194.600
	15 × 15	50	8	9.016	100	0	73.913	1.234	8.099	100	0	72.174	1.248	8.088	25.589	272.575	117645.000
	20 × 20	50	0	21.501	–	–	50	2.840	19.470	–	–	47.900	2.919	19.415	150.482	280.676	319056.000
	All	300	64	5.367	89.583	0.182	58.333	2.661	4.505	89.505	0.182	56.620	2.703	4.494	29.981	162.792	80508.418
$p = 0.5$	4 × 4	50	100	0	100	0	–	–	0	100	0	–	–	0	0.022	43.612	0
	5 × 5	50	100	0	100	0	–	–	0	100	0	–	–	0	0.049	108.574	0
	7 × 7	50	94	0.234	100	0	33.333	4.117	0.004	100	0	33.333	4.119	0.004	0.451	175.276	6823.640
	10 × 10	50	18	3.119	100	0	63.415	1.954	1.658	100	0	63.415	1.960	1.653	8.443	239.844	88800.400
	15 × 15	50	0	18.686	–	–	56	3.298	16.214	–	–	55.800	3.324	16.195	49.957	294.600	202925.000
	20 × 20	50	0	32.206	–	–	46	4.157	29.686	–	–	42	4.470	29.488	224.149	300.000	355284.000
	All	300	52	9.041	100	0	54.167	3.231	7.927	100	0	52.708	3.350	7.890	47.178	193.651	108972.173
$p = 0.8$	4 × 4	50	100	0	100	0	–	–	0	100	0	–	–	0	0.012	17.571	0
	5 × 5	50	100	0	100	0	–	–	0	100	0	–	–	0	0.023	42.554	0
	7 × 7	50	80	0.172	100	0	90	0.593	0.060	100	0	90	0.593	0.060	0.198	108.964	2627.370
	10 × 10	50	8	2.983	100	0	91.304	0.282	2.744	100	0	91.304	0.282	2.744	2.157	197.175	24005.200
	15 × 15	50	0	13.735	–	–	92	0.255	13.527	–	–	91.500	0.256	13.526	9.637	288.587	42839.200
	20 × 20	50	0	34.463	–	–	58	1.555	33.499	–	–	56.6	1.605	33.467	140.169	300	271432.000
	All	300	48	8.559	100	0	80.769	0.701	8.305	100	0	80.160	0.718	8.299	25.366	159.142	56817.295

a: Number of instances.
b(%): Number of times in percentage the best lower bound coincides with the best upper bound.
c(%): Average gap between the best lower and upper bounds in percentage.
d(%): Number of times in percentage the makespan coincides with the proven optimal makespan.
e(%): Average percentage deviation from the proven optimal makespan.
f(%): Number of times in percentage the makespan coincides with the best lower bound.
g(%): Average percentage deviation from the best lower bound.
h(%): Average percentage deviation from the best upper bound.
i(s): CPU time in seconds.
j: Average population size.
k: Average number of generations.

Table 11
Results of GA-ND on the instances derived from Guéret and Prins instances.

	Ins.	*a*	*b*	*c*	Best					Average							
					d	*e*	*f*	*g*	*h*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*
$p = 0.2$	3 × 3	50	100	0	100	0	–	–	0	100	0	–	–	0	0.007	2.240	0
	4 × 4	50	100	0	96	0.009	–	–	−0.009	96	0.009	–	–	−0.009	0.027	40.411	248.040
	5 × 5	50	100	0	96	0.044	–	–	−0.044	96	0.044	–	–	−0.044	0.071	75.215	989.856
	6 × 6	50	100	0	100	0	–	–	0	97.7	0.001	–	–	−0.001	0.327	180.623	5781.480
	7 × 7	50	94	0.074	100	0	0	1.279	0	95	0.003	0	1.279	−0.003	1.796	260.846	34717.600
	8 × 8	50	88	1.305	97.727	0.016	0	13.101	−0.014	86.591	0.029	0	13.157	−0.032	3.615	263.977	63606.200
	9 × 9	50	76	0.697	100	0	33.333	3.476	0.006	86.184	0.008	12.917	3.587	−0.027	6.150	275.146	94568.100
	10 × 10	50	48	1.615	100	0	73.077	3.610	0.089	91.875	0.005	33.461	3.691	0.044	9.489	295.534	128076.000
	All	400	88.25	0.461	98.583	0.009	48.936	4.639	0.003	94.164	0.013	21.808	4.719	−0.009	2.685	174.249	40998.409
$p = 0.5$	3 × 3	50	100	0	100	0	–	–	0	100	0	–	–	0	0.007	1.8	0
	4 × 4	50	100	0	98	0.008	–	–	−0.008	98	0.008	–	–	−0.008	0.032	66.482	128.02
	5 × 5	50	98	0.003	100	0	0	0.160	0	100	0	0	0.160	0	0.176	184.070	3015.80
	6 × 6	50	100	0	100	0	–	–	0	99.7	0.0001	–	–	−0.0001	0.165	234.562	2009.43
	7 × 7	50	92	0.575	95.652	0.002	25	8.346	0.0008	92.391	0.004	25	8.352	−0.001	1.708	254.078	32481.700
	8 × 8	50	66	0.664	100	0	70.588	2.006	0.077	96.212	0.002	61.176	2.026	0.069	2.780	274.747	46017.700
	9 × 9	50	32	2.429	100	0	76.471	1.976	1.311	100	0	67.353	1.989	1.302	4.782	293.327	69499.200
	10 × 10	50	32	3.959	100	0	82.353	2.265	2.704	100	0	76.029	2.290	2.690	4.665	300	55821.100
	All	400	77.5	0.954	99.032	0.002	74.445	2.354	0.511	98.097	0.002	66.833	2.372	0.506	1.789	201.133	26121.619
$p = 0.8$	3 × 3	50	100	0	100	0	–	–	0	100	0	–	–	0	0.007	1.96	0
	4 × 4	50	100	0	100	0	–	–	0	100	0	–	–	0	0.013	20.829	0
	5 × 5	50	100	0	100	0	–	–	0	100	0	–	–	0	0.017	49.120	0.185
	6 × 6	50	98	0.003	100	0	0	0.133	0	100	0	0	0.133	0	0.195	134.416	3331.140
	7 × 7	50	74	0.597	100	0	84.615	2.564	0.025	100	0	84.231	2.564	0.025	0.564	202.903	10030.700
	8 × 8	50	30	0.814	100	0	94.286	0.714	0.372	100	0	92	0.715	0.371	0.906	241.188	14450.100
	9 × 9	50	18	1.785	100	0	97.561	0.0005	1.784	100	0	92.073	0.003	1.782	1.493	267.112	22416.800
	10 × 10	50	6	5.213	100	0	93.617	0.428	4.869	100	0	91.064	0.432	4.865	2.366	268.380	30858.300
	All	400	65.75	1.051	100	0	93.431	0.573	0.881	100	0	90.292	0.576	0.880	0.695	148.238	10135.903

a: Number of instances.
b(%): Number of times in percentage the best lower bound coincides with the best upper bound.
c(%): Average gap between the best lower and upper bounds in percentage.
d(%): Number of times in percentage the makespan coincides with the proven optimal makespan.
e(%): Average percentage deviation from the proven optimal makespan.
f(%): Number of times in percentage the makespan coincides with the best lower bound.
g(%): Average percentage deviation from the best lower bound.
h(%): Average percentage deviation from the best upper bound.
i(s): CPU time in seconds.
j: Average population size.
k: Average number of generations.

5.4. Improvement of the GA and comparison with the two-phase heuristic of Tellache and Boudhar (2017)

We observed from the experiments of the previous section that, for some instances, the optimal schedules are not non-delay (e.g. some 4 × 4 and 5 × 5 instances with $p = 0.2$ derived from Taillard, 4 × 4 with $p = 0.2$ and 0.5, and 5 × 5 with $p = 0.2$ instances derived from Guéret and Prins, and 3 × 3 with $p = 0.2$ and 0.5, and 4 × 4 and 5 × 5 with $p = 0.2$ instances derived from Brucker et al.). To improve the performance of our GA on these instances, we allow the algorithm to explore some regions of active schedules that are non-delay. This is achieved by combining the calls to the algorithms producing non-delay and active schedules. In the implementation, we randomly call, at each step requiring the computation of the makespan, Algorithm 2 with probability p_{active}, otherwise we apply Algorithm 3 (i.e. randomly applied with probability $1 - p_{active}$). We chose Algorithm 2 for building active schedules because it outperforms in general Algorithm 1 as seen in the previous section. We ran this variant for five different values of p_{active} (0.1, 0.2, 0.5, 0.8, and 0.9) on all the instances, and we kept the other parameters and components similar to those of GA-ND. We observed that 0.1 and 0.2 give the best results in most of the cases, and this was expected since Algorithm 3 produces the shortest makespans for the majority of the instances especially those with large sizes and high densities. Therefore, we fix in the following p_{active} to 0.1 and we denote the resulting variant by GA-ND-GT.

The detailed results of GA-ND-GT are given in Tables 13–15. It can be seen from these tables that GA-ND-GT improves the results of GA-ND especially on the instances mentioned in the beginning of this section without decreasing the performance of the algorithm on the other sets on which GA-ND performs well. If we consider all the instances of each group, GA-ND-GT solved to optimality 83.889% (respectively 95.333%, and 98.265%) of the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.), and the average deviation from the best lower bounds is 1.002% (respectively 0.43%, and 0.112%). In terms of computational times, the average CPU time has slightly increased in comparison to GA-ND: instances with $p = 0.2$ (respectively $p = 0.5$ and $p = 0.8$) require on average 10.269 (respectively 14.782 and 7.773) seconds, while the average CPU time over all instances is 10.941 seconds.

To further enhance the performance of the GA, we applied the VNS algorithm of Section 4.8 to the final population of GA-ND-GT. We evaluated this variant on a subset of the final population (best, half, and all chromosomes), and varied the maximum number of iterations for the local search (100, 200, 300, and 500). We tested the 12 combinations on the 5 × 5 and 10 × 10 derived from Taillard, the 5 × 5, 8 × 8, 9 × 9, and 10 × 10 derived from Guéret and Prins, and the 4 × 4, 5 × 5, 7 × 7, and 9 × 9 derived from Brucker et al.. We observed from the experiments that applying VNS to all the chromosomes of the final population with 200 iterations of the local search yielded the best results on average. We denote the variant using this combination of parameters as HGA-ND-GT.

Table 12
Results of GA-ND on the instances derived from Brucker et al. instances.

	Instances	*a*	*b*	*c*	Best					Average							
					d	*e*	*f*	*g*	*h*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*
$p = 0.2$	3 × 3	55	100	0	69.091	0.934	–	–	−0.934	69.091	0.934	–	–	−0.934	0.026	3.127	507.582
	4 × 4	60	100	0	81.667	0.465	–	–	−0.465	81.667	0.465	–	–	−0.465	0.072	22.307	1506.850
	5 × 5	60	100	0	93.333	0.124	–	–	−0.124	93.333	0.124	–	–	−0.124	0.195	143.555	3787.160
	6 × 6	60	100	0	100	0	–	–	0	100	0	–	–	0	0.051	236.879	100.590
	7 × 7	60	98.333	0.092	100	0	0	5.850	0	100	0	0	5.850	0	0.287	221.012	3715.710
	8 × 8	55	98.182	0.0009	100	0	100	0	0.0009	100	0	100	0	0.0009	0.159	227.694	1554.620
	9 × 9	15	100	0	100	0	–	–	0	100	0	–	–	0	0.116	208.160	222.627
	All	365	99.452	0.015	91.185	0.239	49.992	2.925	−0.237	91.185	0.239	49.992	2.925	−0.237	0.132	145.870	1817.477
$p = 0.5$	3 × 3	55	100	0	96.364	0.100	–	–	−0.100	96.364	0.100	–	–	−0.100	0.009	1.872	60.036
	4 × 4	60	100	0	100	0	–	–	0	100	0	–	–	0	0.028	100.748	0
	5 × 5	60	98.333	0.008	100	0	0	0.484	0	100	0	0	0.484	0	0.153	191.865	2503.860
	6 × 6	60	98.333	0.050	100	0	0	3.097	0	100	0	0	3.097	0	0.207	195.297	3127.190
	7 × 7	60	80	0.591	100	0	75	3.331	0.038	100	0	75	3.331	0.038	0.604	249.613	10675.800
	8 × 8	55	60	0.787	100	0	81.818	0.874	0.481	100	0	81.591	0.874	0.481	1.186	248.773	18881.700
	9 × 9	15	53.333	1.779	100	0	71.429	3.571	0.306	100	0	71.429	3.571	0.305	2.687	299.257	36000.300
	All	365	88.219	0.298	99.379	0.017	74.418	2.041	0.076	99.379	0.017	74.302	2.041	0.076	0.454	171.303	7014.277
$p = 0.8$	3 × 3	55	100	0	100	0	–	–	0	100	0	–	–	0	0.006	1.745	0
	4 × 4	60	100	0	100	0	–	–	0	100	0	–	–	0	0.010	14.961	0
	5 × 5	60	100	0	100	0	–	–	0	100	0	–	–	0	0.029	64.757	0
	6 × 6	60	90	0.033	100	0	83.333	0.209	0.012	100	0	83.333	0.209	0.012	0.179	148.562	3075.100
	7 × 7	60	73.333	0.129	100	0	100	0	0.129	100	0	100	0	0.129	0.044	155.533	122.934
	8 × 8	55	38.182	1.415	100	0	88.235	0.565	1.097	100	0	88.235	0.565	1.097	1.105	179.320	17547.900
	9 × 9	15	26.667	2.382	100	0	100	0	2.382	100	0	100	0	2.382	0.071	204.017	377.753
	All	365	81.644	0.338	100	0	92.537	0.305	0.286	100	0	92.537	0.305	0.286	0.213	98.761	3185.432

a: Number of instances.
b(%): Number of times in percentage the best lower bound coincides with the best upper bound.
c(%): Average gap between the best lower and upper bounds in percentage.
d(%): Number of times in percentage the makespan coincides with the proven optimal makespan.
e(%): Average percentage deviation from the proven optimal makespan.
f(%): Number of times in percentage the makespan coincides with the best lower bound.
g(%): Average percentage deviation from the best lower bound.
h(%): Average percentage deviation from the best upper bound.
i(s): CPU time in seconds.
j: Average population size.
k: Average number of generations.

We compare the variants of the GA and the two-phase heuristic approach of Tellache and Boudhar (2017) in Tables 13–15 and Figs. 8 and 9. 24 heuristic variants can be derived from this latter approach by varying the beam search technique used (Beam1, Beam2, and Beam3) and the priority rule of selecting the schedule slices (PR1, PR2, PR3, PR4, PR5, PR6, PR7, and PR8). We ran these variants on all the instances. The combination of Beam2 and PR5 produces the best results in most of the cases, which is in line with the conclusions of Tellache and Boudhar (2017). We report the results of this variant in Tables 13–15. The meanings of dashes "–", rows "All" and the bold numbers of these tables are similar to those of the previous sections.

It can be seen from Tables 13–15 that HGA-ND-GT improves the results of GA-ND-GT (more apparent on the instances derived from Guéret and Prins), especially in terms of the averages over the 20 runs of the GA, but at the expense of an increase in the CPU time. Overall, HGA-ND-GT yields excellent results; it coincides with 99.797% (respectively 99.784%, and 99.898%) of the proven optimal makespans of the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.), and the average deviation from these makespans equals 0.010% (respectively 0.0009%, and 0.005%). On the other instances, HGA-ND-GT hits 65.441% (respectively 83.212%, and 84.821%) of the best lower bounds, and the average deviation from these bounds is 2.169% (respectively 1.857%, and 1.019%). If we consider all the instances of each group, HGA-ND-GT solved to optimality 84.222% (respectively 96.000%, and 98.356%) of the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.), and the average deviation from the best lower bounds is 0.989% (respectively 0.425%, and 0.109%). These results and column *h* of Tables 13–15 show that HGA-ND-GT improves in most of the cases the best upper bounds obtained by the MILP models in terms of both optimal makespans found and deviation from the best lower bounds (see the fifth paragraph of Section 5.2 for a summary of the MILP models results), and this is in a short CPU time in comparison to the time limit we set for the MILP models. The average deviation from these best upper bounds equals 6.929% (respectively 0.467%, and 0.121%) on the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.).

Regarding the two-phase heuristic, this latter solves to optimality 61.333% (respectively 56%, and 73.607%) of the instances derived from Taillard (respectively Guéret and Prins, and Brucker et al.), and the average deviation from the best lower bounds equals 2.101% (respectively 2.135%, and 0.857%). Clearly, HGA-ND-GT (and also GA-ND-GT and GA-ND) outperforms this heuristic on all the instances: the average deviation over all the instances is divided by 3.552 (from 1.687% to 0.475%), and the optimum makespans that have been retrieved increases from 63.537% to 93.490%. To confirm statistically these observations, we conducted a Friedman test on the results of HGA-ND-GT and the two-phase heuristic (Tellache and Boudhar, 2017). The obtained *p*-values are less than 0.00001 for both the average percentage deviation from the best lower bound and the number of times a given algorithm coincides with the best lower bound. These results show that HGA-ND-GT significantly outperforms the two-phase heuristic of Tellache and Boudhar (2017) with respect to the two criteria. However, the two-phase heuristic (Tellache and Boudhar, 2017)

Table 13
Comparison of GA-ND-GT and HGA-ND-GT with the two-phase heuristic of Tellache and Boudhar (2017) on the instances derived from Taillard instances.

p		Two-phase heuristic						GA-ND-GT											HGA-ND-GT										
								Best					Average						Best					Average					
		d	*e*	*f*	*g*	*h*	*i*	*d*	*e*	*f*	*g*	*h*	*d*	*e*	*f*	*g*	*h*	*i*	*d*	*e*	*f*	*g*	*h*	*d*	*e*	*f*	*g*	*h*	i
0.2	4 × 4	70	1.309	–	–	−1.309	**0.000**	**100**	**0**	–	–	**0**	89.5	0.171	–	–	−0.171	0.571	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.702
	5 × 5	84	0.470	–	–	−0.470	**0.000**	92	0.114	–	–	−0.114	92	0.114	–	–	−0.114	0.531	**98**	**0.028**	–	–	**−0.028**	92.5	0.1	–	–	−0.1	1.794
	7 × 7	74	0.512	–	–	−0.512	**0.008**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.145	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.151
	10 × 10	81.579	0.341	25	8.791	−0.413	**0.075**	**100**	**0**	**33.333**	**7.383**	**0.161**	**100**	**0**	**33.333**	**7.383**	**0.161**	4.551	**100**	**0**	**33.333**	**7.383**	**0.161**	**100**	**0**	**33.333**	**7.383**	**0.161**	5.127
	15 × 15	**100**	**0**	41.304	3.336	6.435	**1.651**	**100**	**0**	**73.913**	**1.234**	**8.099**	**100**	**0**	71.956	1.250	8.087	28.512	**100**	**0**	**73.913**	**1.234**	**8.099**	**100**	**0**	73.261	1.236	8.097	30.782
	20 × 20	–	–	20	7.256	16.221	**10.200**	–	–	**50**	2.832	19.476	–	–	47.8	2.937	19.402	158.639	–	–	**50**	**2.825**	**19.481**	–	–	47.8	2.933	19.405	171.619
	All	77.604	0.664	29.629	5.757	3.325	**1.989**	97.917	0.030	**58.333**	2.657	4.604	95.182	0.074	56.481	2.712	4.561	32.158	**99.479**	**0.007**	**58.333**	**2.653**	**4.619**	98.047	0.026	57.037	2.705	4.594	35.029
0.5	4 × 4	96	0.079	–	–	−0.079	**0.000**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.073	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.097
	5 × 5	88	0.226	–	–	−0.226	**0.001**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.100	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.120
	7 × 7	87.234	0.125	33.333	6.133	−0.227	**0.014**	**100**	**0**	**33.333**	**4.117**	**0.004**	**100**	**0**	**33.333**	**4.117**	**0.004**	0.627	**100**	**0**	**33.333**	**4.117**	**0.004**	**100**	**0**	**33.333**	**4.117**	**0.004**	0.749
	10 × 10	**100**	**0**	39.024	3.019	0.840	**0.248**	**100**	**0**	**63.415**	**1.953**	**1.658**	**100**	**0**	**63.415**	1.960	1.653	9.524	**100**	**0**	**63.415**	**1.953**	**1.658**	**100**	**0**	**63.415**	1.961	1.650	10.937
	15 × 15	–	–	30	5.856	14.242	**4.929**	–	–	**56**	**3.300**	**16.212**	–	–	55.9	3.325	16.194	53.411	–	–	**56**	**3.300**	**16.212**	–	–	**56**	**3.300**	**16.212**	60.494
	20 × 20	–	–	20	8.852	26.652	**40.023**	–	–	**46**	4.160	29.684	–	–	41.8	4.479	29.483	231.166	–	–	**46**	**4.156**	**29.687**	–	–	41.5	4.466	29.491	238.885
	All	91.026	0.135	29.166	6.094	6.867	**7.536**	**100**	**0**	**54.167**	3.232	7.926	**100**	**0**	52.674	3.353	7.889	49.150	**100**	**0**	**54.167**	**3.231**	**7.927**	**100**	**0**	52.604	3.341	7.893	51.881
0.8	4 × 4	90	0.121	–	–	−0.121	**0.000**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.040	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.045
	5 × 5	88	0.114	–	–	−0.114	**0.001**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.086	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.097
	7 × 7	90	0.061	40	1.115	−0.092	**0.019**	**100**	**0**	**90**	**0.593**	**0.060**	**100**	**0**	**90**	**0.593**	**0.060**	0.352	**100**	**0**	**90**	**0.593**	**0.060**	**100**	**0**	**90**	**0.593**	**0.060**	0.393
	10 × 10	**100**	**0**	67.391	0.435	2.611	**0.391**	**100**	**0**	**91.304**	**0.282**	**2.744**	**100**	**0**	**91.304**	**0.282**	**2.744**	2.462	**100**	**0**	**91.304**	**0.282**	**2.744**	**100**	**0**	**91.304**	**0.282**	**2.744**	2.653
	15 × 15	–	–	36	0.697	13.159	**12.223**	–	–	**92**	**0.255**	**13.527**	–	–	90.8	0.256	13.526	11.391	–	–	**92**	**0.255**	**13.527**	–	–	**92**	**0.255**	**13.527**	10.600
	20 × 20	–	–	10	3.247	32.421	**112.902**	–	–	**58**	**1.554**	**33.5**	–	–	55.9	1.602	33.469	142.513	–	–	**58**	**1.554**	**33.500**	–	–	56.3	1.602	33.470	151.382
	All	89.583	0.098	37.179	1.464	7.977	**20.923**	**100**	**0**	**80.769**	**0.701**	**8.305**	**100**	**0**	79.711	0.717	8.3	26.141	**100**	**0**	**80.769**	**0.701**	**8.305**	**100**	**0**	80.224	0.716	8.300	27.528

d(%): Number of times in percentage the makespan coincides with the proven optimal makespan.
e(%): Average percentage deviation from the proven optimal makespan.
f(%): Number of times in percentage the makespan coincides with the best lower bound.
g(%): Average percentage deviation from the best lower bound.
h(%): Average percentage deviation from the best upper bound.
i(s): CPU time in seconds.

Table 14
Comparison of GA-ND-GT and HGA-ND-GT with the two-phase heuristic of Tellache and Boudhar (2017) on the instances derived from Guéret and Prins instances.

p		Two-phase heuristic						GA-ND-GT											HGA-ND-GT										
								Best					Average						Best					Average					
		d	*e*	*f*	*g*	*h*	*i*	*d*	*e*	*f*	*g*	*h*	*d*	*e*	*f*	*g*	*h*	*i*	*d*	*e*	*f*	*g*	*h*	*d*	*e*	*f*	*g*	*h*	i
0.2	3 × 3	46	7.323	–	–	−7.323	**0.00**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.03	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.03
	4 × 4	70	1.033	–	–	−1.033	**0.00**	**100**	**0**	–	–	**0**	99.4	5e−4	–	–	−5e−4	0.20	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.29
	5 × 5	64	0.959	–	–	−0.959	**0.00**	**100**	**0**	–	–	**0**	98.6	0.001	–	–	−0.001	0.27	**100**	**0**	–	–	**0**	99.9	8e−5	–	–	−8e−5	0.34
	6 × 6	64	0.288	–	–	−0.288	**0.00**	**100**	**0**	–	–	**0**	98.1	9e−4	–	–	−9e−4	0.37	**100**	**0**	–	–	**0**	99.8	1e−4	–	–	−1e−4	0.50
	7 × 7	31.915	3.302	0	6.693	−3.428	**0.01**	**100**	**0**	0	**1.279**	**0**	95.851	0.002	0	**1.279**	−0.002	1.79	**100**	**0**	0	**1.279**	**0**	**100**	**0**	0	**1.279**	**0**	2.38
	8 × 8	50	2.192	0	17.300	−2.417	**0.02**	95.454	0.018	0	**13.101**	−0.015	86.364	0.029	0	13.167	−0.033	4.11	**97.727**	**0.016**	0	**13.101**	**−0.014**	96.704	0.017	0	13.108	−0.016	9.92
	9 × 9	52.632	2.363	0	11.330	−3.667	**0.04**	**100**	**0**	33.333	3.478	0.005	84.737	0.009	12.083	3.568	−0.023	6.85	**100**	**0**	**50**	**3.449**	**0.012**	99.868	7e−5	44.583	3.484	0.004	14.56
	10 × 10	62.500	0.569	0	10.479	−3.708	**0.09**	**100**	**0**	73.077	3.611	0.088	88.958	0.007	33.461	3.687	0.0462	10.10	**100**	**0**	**80.769**	**3.598**	**0.095**	**100**	**0**	67.5	3.623	0.082	22.06
	All	54.958	2.366	0	11.325	−2.853	**0.02**	99.433	**0.002**	48.936	4.640	0.010	94.802	0.006	21.595	4.713	−0.002	2.96	**99.717**	**0.002**	**57.447**	**4.625**	**0.012**	99.533	0.002	48.723	4.649	0.009	6.26
0.5	3 × 3	80	2.333	–	–	−2.333	**0.00**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.02	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.03
	4 × 4	76	0.826	–	–	−0.826	**0.00**	**100**	**0**	–	–	**0**	99	8e−4	–	–	−8e−4	0.18	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.23
	5 × 5	61.224	0.667	0	1.002	−0.671	**0.00**	**100**	**0**	0	**0.160**	**0**	**100**	**0**	0	**0.160**	**0**	0.22	**100**	**0**	0	**0.160**	**0**	**100**	**0**	0	**0.160**	**0**	0.28
	6 × 6	70	0.777	–	–	−0.777	**0.01**	**100**	**0**	–	–	**0**	99.7	1e−4	–	–	−1e−4	0.18	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.25
	7 × 7	47.826	1.688	0	13.929	−1.993	**0.02**	95.652	0.002	**25**	**8.346**	8e−4	92.935	0.003	**25**	8.350	−8e−4	1.83	**97.826**	**0.001**	**25**	**8.346**	**0.001**	97.609	**0.001**	**25**	**8.346**	**0.001**	2.25
	8 × 8	66.667	0.822	5.882	5.798	−1.746	**0.05**	**100**	**0**	**76.471**	2.006	0.077	95.909	0.001	60.294	2.024	0.070	3.16	**100**	**0**	**76.471**	**2.004**	**0.078**	**100**	**0**	73.529	2.009	0.076	6.17
	9 × 9	93.750	0.081	8.823	4.589	−0.417	**0.10**	**100**	**0**	**85.294**	1.968	1.315	**100**	**0**	68.088	1.993	1.300	5.38	**100**	**0**	**85.294**	**1.963**	**1.318**	**100**	**0**	79.118	1.973	1.312	13.58
	10 × 10	**100**	**0**	23.529	4.943	1.063	**0.19**	**100**	**0**	82.353	2.260	2.707	**100**	**0**	76.176	2.292	2.689	5.18	**100**	**0**	**85.294**	**2.255**	**2.710**	**100**	**0**	83.529	2.266	2.704	16.05
	All	70.322	1.082	13.333	5.326	−0.962	**0.05**	99.355	3e−4	78.889	2.349	0.512	98.306	7e−4	67.000	2.374	0.507	2.02	**99.677**	**2e-4**	**80.000**	**2.345**	**0.513**	99.645	2e−4	76.444	2.354	0.512	4.85
0.8	3 × 3	90	0.804	–	–	−0.804	**0.00**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.01	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.02
	4 × 4	92	0.372	–	–	−0.372	**0.00**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.06	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.08
	5 × 5	80	0.429	–	–	−0.429	**0.00**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.08	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.12
	6 × 6	73.469	0.757	0	5.2	−0.843	**0.01**	**100**	**0**	0	**0.133**	**0**	99.898	3e−5	0	**0.133**	3e−5	0.32	**100**	**0**	0	**0.133**	**0**	**100**	**0**	0	**0.133**	**0**	0.40
	7 × 7	56.757	0.617	23.077	3.732	−0.732	**0.03**	**100**	**0**	**84.615**	**2.564**	**0.025**	**100**	**0**	83.846	**2.564**	**0.025**	0.67	**100**	**0**	**84.615**	**2.564**	**0.025**	**100**	**0**	83.077	2.565	**0.025**	0.84
	8 × 8	86.667	0.068	31.429	1.859	−0.441	**0.09**	**100**	**0**	**94.286**	**0.714**	**0.372**	**100**	**0**	91.571	0.715	0.371	1.02	**100**	**0**	**94.286**	**0.714**	**0.372**	**100**	**0**	**94.286**	**0.714**	**0.372**	2.14
	9 × 9	88.889	0.019	24.390	0.881	1.093	**0.22**	**100**	**0**	97.561	5e−4	1.784	**100**	**0**	91.463	0.003	1.782	1.71	**100**	**0**	**100**	**0**	**1.785**	**100**	**0**	98.658	3e−4	1.784	2.65
	10 × 10	**100**	**0**	25.532	1.779	3.706	**0.55**	**100**	**0**	**93.617**	0.427	4.869	**100**	**0**	90.425	0.431	4.866	2.76	**100**	**0**	**93.617**	**0.426**	**4.870**	**100**	**0**	93.511	0.427	**4.870**	7.28
	All	80.608	0.537	26.277	1.741	0.147	**0.11**	**100**	**0**	93.431	**0.572**	**0.881**	99.981	6e−6	89.744	0.576	0.880	0.83	**100**	**0**	**94.161**	**0.572**	**0.881**	**100**	**0**	93.577	0.573	**0.881**	1.69

d(%): Number of times in percentage the makespan coincides with the proven optimal makespan.
e(%): Average percentage deviation from the proven optimal makespan.
f(%): Number of times in percentage the makespan coincides with the best lower bound.
g(%): Average percentage deviation from the best lower bound.
h(%): Average percentage deviation from the best upper bound.
i(*s*): CPU time in seconds.

Table 15
Comparison of GA-ND-GT and HGA-ND-GT with the two-phase heuristic of Tellache and Boudhar (2017) on the instances derived from Brucker instances.

p		Two-phase heuristic						GA-ND-GT											HGA-ND-GT										
								Best					Average						Best					Average					
		d	*e*	*f*	*g*	*h*	*i*	*d*	*e*	*f*	*g*	*h*	*d*	*e*	*f*	*g*	*h*	*i*	*d*	*e*	*f*	*g*	*h*	*d*	*e*	*f*	*g*	*h*	*i*
0.2	3 × 3	78.182	1.779	–	–	−1.779	**0.000**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.042	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.055
	4 × 4	66.667	2.217	–	–	−2.217	**0.000**	**100**	**0**	–	–	**0**	96.667	0.036	–	–	−0.036	0.576	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.699
	5 × 5	80	1.494	–	–	−1.494	**0.000**	96.667	0.041	–	–	−0.041	93.667	0.116	–	–	−0.116	0.524	**98.333**	**0.013**	–	–	**−0.013**	94.417	0.081	–	–	−0.081	1.874
	6 × 6	53.333	0.865	–	–	−0.865	**0.003**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.054	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.071
	7 × 7	72.881	0.716	0	5.850	−0.704	**0.006**	**100**	**0**	0	5.850	**0**	**100**	**0**	0	5.850	**0**	0.307	**100**	**0**	0	5.850	0	**100**	**0**	0	5.850	0	0.378
	8 × 8	61.111	1.465	0	5.800	−1.543	**0.020**	**100**	**0**	**100**	**0**	**9e-4**	**100**	**0**	**100**	**0**	**9e-4**	0.199	**100**	**0**	**100**	**0**	**9e-4**	**100**	**0**	**100**	**0**	**9e-4**	0.210
	9 × 9	73.333	1.631	–	–	−1.631	**0.039**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.192	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.199
	All	68.870	1.428	0	5.825	−1.435	**0.006**	99.449	0.007	**49.992**	**2.925**	−0.007	98.402	0.025	**49.992**	**2.925**	−0.025	0.284	**99.724**	**0.002**	**49.992**	**2.925**	**−0.002**	99.077	0.013	**49.992**	**2.925**	−0.013	0.545
0.5	3 × 3	94.545	0.425	–	–	−0.425	**0.000**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.029	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.037
	4 × 4	76.667	0.605	–	–	−0.605	**0.000**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.095	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.120
	5 × 5	81.356	0.235	0	0.484	−0.231	**0.001**	**100**	**0**	0	0.484	**0**	**100**	**0**	0	0.484	**0**	0.191	**100**	**0**	0	0.484	0	**100**	**0**	0	0.484	0	0.243
	6 × 6	79.661	0.673	0	6.140	−0.711	**0.004**	**100**	**0**	0	**3.097**	**0**	**100**	**0**	0	**3.097**	**0**	0.237	**100**	**0**	0	**3.097**	**0**	**100**	**0**	0	**3.097**	**0**	0.291
	7 × 7	72.917	0.330	16.667	4.563	−0.470	**0.015**	**100**	**0**	**75**	**3.331**	**0.038**	**100**	**0**	**75**	**3.331**	**0.038**	0.668	**100**	**0**	**75**	**3.331**	**0.038**	**100**	**0**	**75**	**3.331**	**0.038**	0.813
	8 × 8	75.758	0.202	13.636	3.986	−0.859	**0.059**	**100**	**0**	**81.818**	**0.874**	**0.481**	**100**	**0**	81.591	0.875	**0.481**	1.356	**100**	**0**	**81.818**	**0.874**	**0.481**	**100**	**0**	**81.818**	**0.874**	**0.481**	1.600
	9 × 9	87.500	0.136	42.857	4.369	−0.101	**0.110**	**100**	**0**	**71.429**	**3.571**	**0.306**	**100**	**0**	**71.429**	3.573	0.305	2.873	**100**	**0**	**71.429**	**3.571**	**0.306**	**100**	**0**	**71.429**	**3.571**	**0.306**	3.276
	All	80.745	0.425	18.604	4.178	−0.529	**0.017**	**100**	**0**	**74.418**	**2.041**	**0.091**	**100**	**0**	74.302	2.042	**0.091**	0.522	**100**	**0**	**74.418**	**2.041**	**0.091**	**100**	**0**	**74.418**	**2.041**	**0.091**	0.623
0.8	3 × 3	96.364	0.126	–	–	−0.126	**0.000**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.015	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.019
	4 × 4	91.667	0.157	–	–	−0.157	**0.000**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.073	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.092
	5 × 5	91.667	0.034	–	–	−0.034	**0.001**	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.114	**100**	**0**	–	–	**0**	**100**	**0**	–	–	**0**	0.143
	6 × 6	81.481	0.254	0	0.593	−0.254	**0.007**	**100**	**0**	**83.333**	**0.209**	**0.012**	**100**	**0**	**83.333**	**0.209**	**0.012**	0.238	**100**	**0**	**83.333**	**0.209**	**0.012**	**100**	**0**	**83.333**	**0.209**	**0.012**	0.293
	7 × 7	75.000	0.125	37.500	0.729	−0.156	**0.023**	**100**	**0**	**100**	**0**	**0.129**	**100**	**0**	**100**	**0**	**0.129**	0.116	**100**	**0**	**100**	**0**	0.129	**100**	**0**	**100**	**0**	**0.129**	0.132
	8 × 8	95.238	0.002	38.235	1.081	0.788	**0.082**	**100**	**0**	**88.235**	**0.565**	**1.097**	**100**	**0**	**88.235**	**0.565**	**1.097**	1.229	**100**	**0**	**88.235**	**0.565**	**1.097**	**100**	**0**	**88.235**	**0.565**	**1.097**	1.495
	9 × 9	100	0	45.454	0.213	2.235	**0.150**	**100**	**0**	**100**	**0**	**2.382**	**100**	**0**	**100**	**0**	**2.382**	0.208	**100**	**0**	**100**	**0**	**2.382**	**100**	**0**	**100**	**0**	**2.382**	0.235
	All	88.591	0.126	35.821	0.811	0.093	**0.024**	**100**	**0**	**92.537**	**0.305**	**0.286**	**100**	**0**	**92.537**	**0.305**	**0.286**	0.285	**100**	**0**	**92.537**	**0.305**	**0.286**	**100**	**0**	**92.537**	**0.305**	**0.286**	0.346

d(%): Number of times in percentage the makespan coincides with the proven optimal makespan.
e(%): Average percentage deviation from the proven optimal makespan.
f(%): Number of times in percentage the makespan coincides with the best lower bound.
g(%): Average percentage deviation from the best lower bound.
h(%): Average percentage deviation from the best upper bound.
i(s): CPU time in seconds.

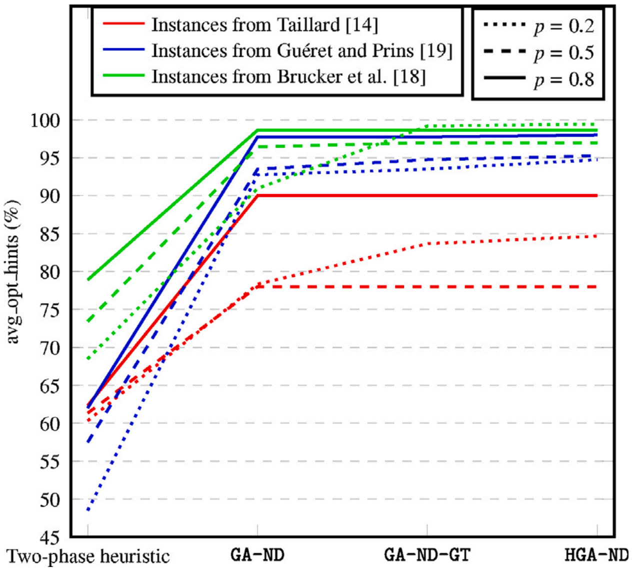


Fig. 8. Comparison between the two-phase heuristic of Tellache and Boudhar (2017), GA-ND, GA-ND-GT, and HGA-ND-GT in terms of number of times a given algorithm coincides with the best lower bound (avg_opt_hints (%)).

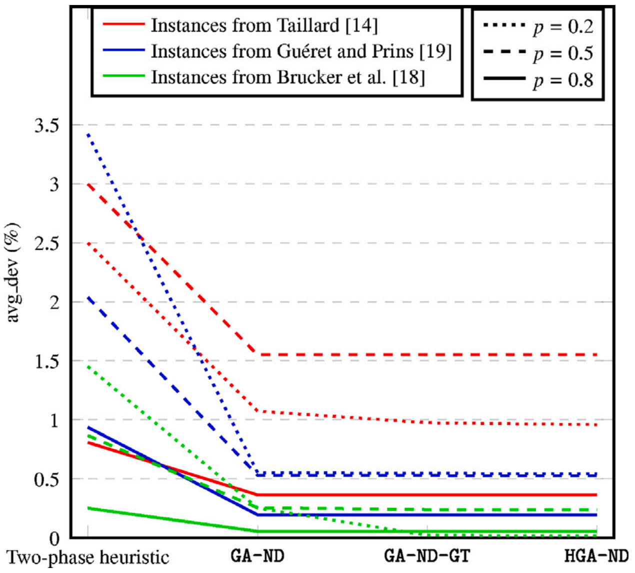


Fig. 9. Comparison between the two-phase heuristic of Tellache and Boudhar (2017), GA-ND, GA-ND-GT, and HGA-ND-GT in terms of average deviation from the best lower bound (avg_dev (%)).

outperforms HGA-ND-GT in terms of CPU time. This latter requires on average 12.406 (respectively 16.650, and 8.508) seconds on instances with $p = 0.2$ (respectively $p = 0.5$, and $p = 0.8$) (12.521 seconds over all instances), whereas the two-phase heuristic (Tellache and Boudhar, 2017) requires on average 0.570 (respectively 2.146, and 5.944) seconds (2.887 seconds over all instances).

6. Conclusions

We studied in this paper the open shop problem subject to conflict constraints given by a conflict graph G. We first presented three MILP models that differs in the modeling of the conflicts between the operations, and three groups of lower bounds from which ten lower bounds were derived. We also developed genetic algorithms and conducted several experiments to select the best variants. The MILP models can solve to optimality 26.979% of the instances within 30 min (except for the 20 × 20 instances in which the time limit is set to one hour for $p = 0.2$ and three hours for $p = 0.5$ and $p = 0.8$), and the average gap reported by Gurobi is 35.607%. The comparison of the best makespans returned by these MILP models with the best lower bounds reveals that at least 75.149% of these makespans are in fact optimal and the average deviation from the best lower bounds equals 2.54%. This means that the models are yielding in many cases optimal or near optimal solutions earlier but the solver is taking too much time to prove the optimality of the incumbent solution or to slightly improve it.

The principal of GAs that relies on a population of candidate solutions seems well adapted to the OSC problem, which is characterized by a larger solution space compared to flow shop and job shop problems due to the free job routes. The selected variant of the GA performs very well: at least 93.490% of the instances are optimally solved and the average deviation from the best lower bounds equals 0.475%. This clearly improves the previous gap between the best upper bounds produced by the MILP models and the best lower bounds, that is divided by 5.27, and increases considerably the proven optimal makespans. This is in a short CPU time that equals on average 12.521 seconds. Furthermore, the genetic algorithm outperforms the two-phase heuristic approach that has been proposed in Tellache and Boudhar (2017) for the OSC problem. This latter solves to optimality at least 63.537% of the instances and the average deviation from the best lower bounds is 1.687%, but it requires less CPU time compared to the selected GA: 2.887 seconds on average.

One additional advantage of GAs is their ability to be easily parallelized. Therefore, possible extensions of this work could explore parallel GAs for the OSC problem allowing to solve larger instances faster. Furthermore, investigating techniques for measuring and maintaining diversity of the population of solutions in ordered chromosomes, such as through the longest common subsequence, could enhance the exploration of the solution space. The modification of the proposed GA to solve the open shop problem with conflict graph with other criteria (e.g. the sum of the completion times and the total tardiness minimization), and applying other metaheuristics to improve the solutions obtained from this study are also topics for future research.

CRediT authorship contribution statement

Nour ElHouda Tellache: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Laoucine Kerbache:** Resources, Writing – review & editing, Funding acquisition.

Data availability

Instances used for the experiments, along with the corresponding best lower and upper bounds (or optimal makespans if proven) are available at: https://nourelhoudatellache.wordpress.com/test-instances/.

Acknowledgments

The authors thank Christian Prins for providing the benchmark instances of the basic open shop problem of Brucker et al. (1997). The authors wish to thank also the anonymous reviewers for their valuable comments that helped to improve the paper.

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.

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.

Ahmadian, M.M., Khatami, M., Salehipour, A., Cheng, T.C.E., 2021. Four decades of research on the open-shop scheduling problem to minimize the makespan. European J. Oper. Res. 295 (2), 399–426.
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.
Anand, E., Panneerselvam, R., 2015. Literature review of open shop scheduling problems. Intell. Inform. Manag. 7 (01), 33.
de Araújo, K.A.G., e Bonates, T.O., de Athayde Prata, B., 2021. Modeling and scheduling hybrid open shops for makespan minimization. J. Model. Manag..
Blazewicz, J., Cellary, W., Slowinski, R., Weglarz, J., 1986. Scheduling under resource constraints-deterministic models. Ann. Oper. Res. 7.
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, 95–110.
Brucker, P., Hurink, J., Jurisch, B., Wöstmann, B., 1997. A branch & bound algorithm for the open-shop problem. Discrete Appl. Math. 76 (1), 43–59.
Croes, G.A., 1958. A method for solving traveling-salesman problems. Oper. Res. 6 (6), 791–812.
Davis, L., 1985. Job shop scheduling with genetic algorithms. In: Proceedings of the 1st International Conference on Genetic Algorithms. L. Erlbaum Associates Inc., USA, pp. 136–140.
Erdős, P., Rényi, A., 1959. On random graphs I. Publ. Math. 6, 290–297.
Falkenauer, E., Bouffouix, S., 1991. A genetic algorithm for job shop. In: Proceedings. 1991 IEEE International Conference on Robotics and Automation, vol. 01. pp. 824–829.
Fang, H.L., Ross, P., Corne, D., 1993. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop problem. In: Proceedings of the 5th International Conference on Genetic Algorithms. pp. 375–382.
Fang, H.L., Ross, P., Corne, D., 1994. A promising hybrid GA/Heuristic approach for open-shop scheduling problems. In: Proceedings of the 11th European Conference on Artificial Intelligence. John Wiley & Sons Ltd, pp. 590–594.
Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide To the Theory of NP-Completeness. Freeman, New York.
Giffler, B., Thompson, G.L., 1960. Algorithms for solving production-scheduling problems. Oper. Res. 8 (4), 487–503.
Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning, first ed. Addison-Wesley Longman Publishing Co., Inc., USA, ISBN: 0201157675.
Gonzalez, T., Sahni, S., 1976. Open shop scheduling to minimize finish time. J. Assoc. Comput. Mach. 23 (4), 665–679.
Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math. 5, 287–326.
Guéret, C., Prins, C., 1999. A new lower bound for the open-shop problem. Ann. Oper. Res. 92, 165–183.
Hansen, P., Mladenović, N., 2001. Variable neighborhood search: Principles and applications. European J. Oper. Res. 130 (3), 449–467.
Hassan, M.-A., Kacem, I., Martin, S., Osman, I.M., 2017. Mathematical formulation for open shop scheduling problem. In: 4th International Conference on Control, Decision and Information Technologies. pp. 0803–0808.
Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. The University of Michigan Press.
Huang, Y.M., Lin, J.C., 2011. A new bee colony optimization algorithm with idle-time-based filtering scheme for open shop-scheduling problems. Expert Syst. Appl. 38 (5), 5438–5447.
Khuri, S., Miryala, S.R., 1999. Genetic algorithms for solving open shop scheduling problems. In: Progress in Artificial Intelligence. Springer Berlin Heidelberg, pp. 357–368.
Kubiak, W., 2021. On a conjecture for the university timetabling problem. Discrete Appl. Math. 299, 26–49.
Kurdi, M., 2022. Ant colony optimization with a new exploratory heuristic information approach for open shop scheduling problem. Knowl.-Based Syst. 242, 108323.
Lal, A.H., KR, V., Haq, A.N., 2019. Minimising the makespan in open shop scheduling problems using variants of discrete firefly algorithm. Int. J. Adv. Oper. Manag. 11 (4), 275–286.
Liao, C.-J., You, C.-T., 1992. An improved formulation for the job-shop scheduling problem. J. Oper. Res. Soc. 43 (11), 1047–1054.
Liaw, C.-F., 1999a. A tabu search algorithm for the open shop scheduling problem. Comput. Oper. Res. 26 (2), 109–126.
Liaw, C.-F., 1999b. Applying simulated annealing to the open shop scheduling problem. IIE Trans. 31, 457–465.
Liaw, C.-F., 2000. A hybrid genetic algorithm for the open shop scheduling problem. European J. Oper. Res. 124 (1), 28–42.
Louis, S.J., Xu, Z., 1996. Genetic algorithms for open shop scheduling and re-scheduling. In: Proceedings of the 11th International Conference on Computers and their Application. pp. 99–102.
Manne, A.S., 1960. On the job-shop scheduling problem. Oper. Res. 8 (2), 219–223.
Matta, M.E., 2009. A genetic algorithm for the proportionate multiprocessor open shop. Comput. Oper. Res. 36 (9), 2601–2618.
Naderi, B., Najafi, E., Yazdani, M., 2012. An electromagnetism-like metaheuristic for open-shop problems with no buffer. J. Ind. Eng. Int. 8 (1), 1–8.
Or, I., 1976. Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking (Ph.D. thesis). Northwestern University.
Phadke, M.S., 1995. Quality Engineering using Robust Design. Prentice Hall PTR.
Pinedo, M.L., 2008. Scheduling: Theory, Algorithms, and Systems, third ed. Springer Publishing Company, Incorporated, ISBN: 0387789340.
Pongchairerks, P., Kachitvichyanukul, V., 2016. A two-level particle swarm optimisation algorithm for open-shop scheduling problem. Int. J. Comput. Sci. Math. 7, 575–585.
Prins, C., 1994. An overview of scheduling problems arising in satellite communications. J. Oper. Res. Soc. 45 (6), 611–623.
Prins, C., 2000. Competitive genetic algorithms for the open-shop scheduling problem. Math. Methods Oper. Res. 52, 389–411.
Puente, J., Díez, H.R., Varela, R., Vela, C.R., Hidalgo, L.P., 2004. Heuristic rules and genetic algorithms for open shop scheduling problem. In: Current Topics in Artificial Intelligence. Springer Berlin Heidelberg, pp. 394–403.
Rahmani Hosseinabadi, A.A., Vahidi, J., Saemi, B., Sangaiah, A.K., Elhoseny, M., 2019. Extended genetic algorithm for solving open-shop scheduling problem. Soft Comput. 23 (13), 5099–5116.
Reeves, C.R., 1995. A genetic algorithm for flowshop sequencing. Comput. Oper. Res. 22 (1), 5–13.
Sakai, S., Togasaki, M., Yamazaki, K., 2003. A note on greedy algorithms for the maximum weighted independent set problem. Discrete Appl. Math. 126, 313–322.
Sha, D.Y., Hsu, C.-Y., 2008. A new particle swarm optimization for the open shop scheduling problem. Comput. Oper. Res. 35 (10), 3243–3261.
Strusevich, V.A., 2022. Complexity and approximation of open shop scheduling to minimize the makespan: A review of models and approaches. Comput. Oper. Res..
Taillard, E., 1993. Benchmarks for basic scheduling problems. European J. Oper. Res. 64 (2), 278–285, Benchmarks at http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html.
Tellache, N.E.H., 2021. New complexity results for shop scheduling problems with agreement graphs. Theoret. Comput. Sci. 889, 85–95.
Tellache, N.E.H., Boudhar, M., 2017. Open shop scheduling problems with conflict graphs. Discrete Appl. Math. 227, 103–120.
Tellache, N.E.H., Boudhar, M., Yalaoui, F., 2019. Two-machine open shop problem with agreement graph. Theoret. Comput. Sci. 796, 154–168.
Zhang, J., Wang, L., Xing, L., 2019. Large-scale medical examination scheduling technology based on intelligent optimization. J. Combinat. Optim. 37 (1), 385–404.
Zobolas, G.I., Tarantilis, C.D., Ioannou, G., 2009. SOLVING THE OPEN shop scheduling PROBLEM VIA a HYBRID GENETIC-variable neighborhood SEARCH ALGORITHM. Cybern. Syst. 40 (4), 259–285.