A Genetic Approach to the Job Shop Scheduling Problem with Interval Uncertainty

Hernán Díaz[1], Inés González-Rodríguez[2], Juan José Palacios[1], Irene Díaz[1(✉)], and Camino R. Vela[1]

[1] Department of Computing, University of Oviedo, Oviedo, Spain
{diazhernan,palaciosjuan,sirene,crvela}@uniovi.es
[2] Department of Maths, Stats and Computing, University of Cantabria, Santander, Spain
gonzalezri@unican.es

Abstract. In this paper we tackle a variant of the job shop scheduling problem where task durations are uncertain and only an interval of possible values for each task duration is known. We propose a genetic algorithm to minimise the schedule's makespan that takes into account the problem's uncertainty during the search process. The behaviour of the algorithm is experimentally evaluated and compared with other state-of-the-art algorithms. Further analysis in terms of solution robustness proves the advantage of taking into account interval uncertainty during the search process with respect to considering only the expected processing times and solving the problem's crisp counterpart. This robustness analysis also illustrates the relevance of the interval ranking method used to compare schedules during the search.

Keywords: Job shop scheduling · Interval processing time · Genetic algorithms · Robustness

1 Introduction

Scheduling plays an important role in most manufacturing and production systems as well as in most information processing environments, transportation and distribution settings and, more generally, in service industries [24]. One of the most relevant problems in scheduling is the job shop, both because it is considered to be a good model for many practical applications and because it poses

Supported by the Spanish Government under research grants TIN2016-79190-R and TIN2017-87600-P and by the Principality of Asturias Government under grant IDI/2018/000176.

M.-J. Lesot et al. (Eds.): IPMU 2020, CCIS 1238, pp. 663–676, 2020.
https://doi.org/10.1007/978-3-030-50143-3_52

a challenge to the research community due to its complexity. This complexity is the reason why approximate methods and, in particular, metaheuristic search techniques, are especially suited for solving the job shop [28].

Traditionally, it has been assumed that scheduling problems are deterministic. However, for many real-world problems design variables such as processing times may be subject to perturbations or changes, dependent on human factors, etc. The most common approach to handling uncertainty is that of stochastic scheduling, modelling the duration of tasks by probability distributions [24]. However, not only does this present some tractability issues, but also, probability distributions are better suited to model variability of repetitive tasks instead of uncertainty due to lack of information [8]. Alternatively, fuzzy scheduling models uncertain durations as fuzzy numbers or fuzzy intervals, that is, possibility distributions representing more or less plausible values, in an approach that is computationally more appealing and presupposes less knowledge [9]. A third and simpler way of representing uncertainty for activity durations are intervals. Interval uncertainty is present as soon as information is incomplete and it does not assume any further knowledge. Also, it represents a first step towards solving problems in the other uncertain frameworks. Indeed, an interval can be seen as a uniform probability distribution or the support of an unknown probability distribution [1]. Also, an interval not only is a particular case of a fuzzy interval, but also the α-cuts of fuzzy intervals are intervals, so a fuzzy scheduling problem can be decomposed in multiple interval scheduling problems.

Despite its interest, research on the job shop scheduling problem with interval activity durations is still scarce. In [15] a job shop scheduling problem with interval processing times is considered and a population-based neighborhood search (PNS) is presented to optimize the makespan. A genetic algorithm is proposed in [16], but with the objective of minimising the total tardiness with respect to job due dates. Different variants of multiobjective interval job shop problems are considered in [17] and [19]. The former incorporates non-resumable jobs and flexible maintenance to the problem and proposes a multi-objective artificial bee colony algorithm to minimise both the makespan and the total tardiness. The latter considers a dual-resource constrained job shop with heterogeneous resources, and a dynamical neighbourhood search is proposed for lexicographic minimisation of carbon footprint and makespan. Finally, we find a flexible job shop problem with interval processing times in [18], where a shuffled frog-leaping algorithm is adopted to minimise the makespan. Uncertain activity durations are also modelled as intervals in scheduling problems other than the job shop and its variants in [1,11,20,26]. At a more theoretical level, several attempts have been made to study how to compute earliest and latest starting times of all activities and, therefore, critical paths, over all duration scenarios in an activity-on-node network where the duration of every activity is an interval. This is essential to devise successful local search methods, as shown in deterministic job shop. A summary of the main results together with a thorough literature review can be found in [3].

This paper constitutes a starting point for a systematic study of solving methods for the interval job shop scheduling problem with makespan minimi-

sation. We propose a solution for Interval Job Shop Scheduling Problem using genetic algorithms. The paper is organised as follows. Section 2 briefly describes the interval job shop scheduling problem. In Sect. 3 the concept of ϵ-robustness is introduced to measure the error of the prediction made by the a-priori makespan compared to the executed makespan with respect to its expected value used in this work is introduced. Section 4 details the basic schema of a genetic algorithm and the coding approach. Finally, in Sect. 5 an experimental study is developed to check the performance of the genetic algorithm in solving this problem. In addition, we also test if modelling uncertainty as intervals for processing times during the search process is worth the while and carry out some preliminary analysis of the influence of the different interval rankings. Some conclusions are drawn in Sect. 6.

2 The Job Shop Problem with Interval Durations

The classical *job shop scheduling problem*, or *JSP* in short, consists in scheduling a set of jobs $J = \{J_1, \ldots, J_n\}$ on a set of physical resources or machines $M = \{M_1, \ldots, M_m\}$, subject to a set of constraints. There are *precedence constraints*, so each job J_j, $j = 1, \ldots, n$, consists of $m_j \leq m$ tasks $(o(j,1), \ldots, o(j,m_j))$ to be sequentially scheduled. There are also *resource constraints*, whereby each task $o(j,l)$ requires the uninterrupted and exclusive use of a machine $\nu_{o(j,l)} \in M$ for its whole processing time $p_{o(j,l)}$. We assume w.l.o.g. that tasks are indexed from 1 to $N = \sum_{j=1}^{n} m_j$, so we can refer to a task $o(j,l)$ by its index $o = \sum_{i=1}^{j-1} m_i + l$ and simply write ν_o, p_o to refer respectively to its machine and processing time. The set of all tasks is denoted $O = \{1, \ldots, N\}$.

A solution to this problem is a *schedule* $\boldsymbol{s}$, i.e. an allocation of starting times for each task, which, besides being *feasible* (in the sense that all precedence and resource constraints hold), is *optimal* according to some criterion, most commonly minimising the makespan C_{max}, that is, the completion time of the last operation (and therefore, of the whole project).

2.1 Interval Durations

In real-life applications, it is often the case that the time it takes to process a task is not exactly known in advance; instead, only some uncertain knowledge about the duration is available. If only an upper and a lower bound of each duration are known, an uncertain processing time can be represented as a closed interval of possible values denoted $\mathbf{a} = [\underline{a}, \overline{a}] = \{x \in \mathbb{R} : \underline{a} \leq x \leq \overline{a}\}$.

Let $\mathbb{IR}$ denote the set of closed intervals. The job shop problem with makespan mimisation essentially requires two arithmetic operations on $\mathbb{IR}$: addition and maximum. These are defined by extending the corresponding operations on real numbers [21], so given two intervals $\mathbf{a} = [\underline{a}, \overline{a}], \mathbf{b} = [\underline{b}, \overline{b}] \in \mathbb{IR}$,

$$\mathbf{a} + \mathbf{b} = [\underline{a} + \underline{b}, \overline{a} + \overline{b}], \tag{1}$$

$$\max(\mathbf{a}, \mathbf{b}) = [\max(\underline{a}, \underline{b}), \max(\overline{a}, \overline{b})]. \tag{2}$$

Another issue to be taken into account when processing times take the form of intervals is that of comparisons. Indeed, if several schedules are available, the "best" one would be the one with "minimal" value of the makespan (an interval). However, there is no natural total order in the set of intervals, so an interval ranking method needs to be considered among those proposed in the literature [7,14].

In [7], the authors highlight the following three total orders in $\mathbb{IR}$ with certain nice behaviour (called admissibility in that work):

$$\mathbf{a} \leq_{Lex1} \mathbf{b} \Leftrightarrow \underline{a} < \underline{b} \vee (\underline{a} = \underline{b} \wedge \overline{a} < \overline{b}) \tag{3}$$

$$\mathbf{a} \leq_{Lex2} \mathbf{b} \Leftrightarrow \overline{a} < \overline{b} \vee (\overline{a} = \overline{b} \wedge \underline{a} < \underline{b}) \tag{4}$$

$$\mathbf{a} \leq_{YX} \mathbf{b} \Leftrightarrow \underline{a} + \overline{a} < \underline{b} + \overline{b} \vee (\underline{a} + \overline{a} = \underline{b} + \overline{b} \wedge \overline{a} - \underline{a} \leq \overline{b} - \underline{b}) \tag{5}$$

Both (3) and (4) are derived from a lexicographical order of interval extreme points while the last one is proposed in [27]. Obviously, all three linear orders can be used to rank intervals. In [15], a different ranking method is used for the interval job shop:

$$\mathbf{a} \leq_{pd} \mathbf{b} \Leftrightarrow P(\mathbf{b} \geq \mathbf{a}) \geq 0.5 \vee P(\mathbf{a} \geq \mathbf{b}) \leq 0.5 \tag{6}$$

where $P(\mathbf{a} \geq \mathbf{b})$ is the possibility degree that $\mathbf{a}$ is greater or equal than $\mathbf{b}$ as introduced in [13]:

$$P(\mathbf{a} \geq \mathbf{b}) = \begin{cases} 0 & \underline{a} \geq \overline{b} \\ 0.5 \cdot \frac{\overline{b}-\underline{a}}{\overline{a}-\underline{a}} \cdot \frac{\overline{b}-\underline{a}}{\overline{b}-\underline{b}} & \underline{b} \leq \underline{a} < \overline{b} \leq \overline{a} \\ \frac{\underline{b}-\underline{a}}{\overline{a}-\underline{a}} + 0.5 \cdot \frac{\overline{b}-\underline{b}}{\overline{a}-\underline{a}} & \underline{a} < \underline{b} < \overline{b} \leq \overline{a} \\ \frac{\underline{b}-\underline{a}}{\overline{a}-\underline{a}} + \frac{\overline{a}-\underline{b}}{\overline{a}-\underline{a}} \cdot \frac{\overline{b}-\overline{a}}{\overline{b}-\underline{b}} + 0.5 \frac{\overline{a}-\underline{b}}{\overline{a}-\underline{a}} \cdot \frac{\overline{a}-\underline{b}}{\overline{b}-\underline{b}} & \underline{a} < \underline{b} \leq \overline{a} < \overline{b} \\ \frac{\overline{b}-\overline{a}}{\overline{b}-\underline{b}} + 0.5 \cdot \frac{\overline{a}-\underline{a}}{\overline{b}-\underline{b}} & \underline{b} \leq \underline{a} < \overline{a} < \overline{b} \\ 1 & \overline{a} \leq \underline{b} \end{cases} \tag{7}$$

It can be easily shown that this ranking is equivalent to the one induced by the interval midpoint:

$$\mathbf{a} \leq_{MP} \mathbf{b} \Leftrightarrow m(\mathbf{a}) \leq m(\mathbf{b}) \tag{8}$$

where $\forall \mathbf{a} \in \mathbb{IR}$, $m(\mathbf{a}) = \frac{(\underline{a}+\overline{a})}{2}$. It coincides with the classical Hurwicz criterion for interval comparison with $\alpha = 1/2$ [12], used for interval scheduling in [1]. Also, since the interval's midpoint is the expected value of the uniform probability distribution in that interval, using the midpoint for comparing interval-valued objective functions is also closely related to the stochastic dominance based on expectation used in stochastic scheduling [24].

2.2 Interval Schedules

A schedule $\boldsymbol{s}$ establishes an order π among tasks requiring the same machine. Conversely, given a task processing order π, the schedule $\boldsymbol{s}$ (starting times of all

tasks) may be computed as follows. For every task $o \in O$ with processing time $\mathbf{p_o}$, let $\mathbf{s_o}(\pi)$ and $\mathbf{c_o}(\pi)$ denote respectively the starting and completion times of o, let $PM_o(\pi)$ and $SM_o(\pi)$ denote the predecessor and successor tasks of o in the machine ν_o according to π, and let PJ_o and SJ_o denote respectively the predecessor and successor tasks of o in its job ($PM_o(\pi) = 0$ or $PJ_o = 0$ if o is the first task to be processed in its machine or its job). Then the starting time $\mathbf{s_o}(\pi)$ of o is an interval given by $\mathbf{s_o}(\pi) = \max(\mathbf{s_{PJ_o}} + \mathbf{p_{PJ_o}}, \mathbf{s_{PM_o(\pi)}} + \mathbf{p_{PM_o(\pi)}})$. Clearly, $\mathbf{c_o}(\pi) = \mathbf{s_o}(\pi) + \mathbf{p_o}(\pi)$. If there is no possible confusion regarding the processing order, we may simplify notation by writing $\mathbf{s_o}$ and $\mathbf{c_o}$. The completion time of the last task to be processed according to π thus calculated will be the makespan, denoted $\mathbf{C_{max}}(\pi)$ or simply $\mathbf{C_{max}}$. We obtain an *interval-valued schedule* in the sense that the starting and completion times of all tasks and the makespan are intervals, interpreted as the possible values that the times may take. However, notice that the task processing ordering π that determines the schedule is crisp; there is no uncertainty regarding the order in which tasks are to be processed.

2.3 Problem Formulation

We are now in a position to formulate the *Interval Job Shop Scheduling Problem* or *IJSP* in short, as follows:

$$\min_R \mathbf{C_{max}} \tag{9}$$

$$\text{subject to:} \quad \mathbf{C_{max}} = \max_{1 \le j \le n} \{\mathbf{c_{o(j,m_j)}}\} \tag{10}$$

$$\underline{c}_o = \underline{s}_o + \underline{p}_o, \ \forall o \in O \tag{11}$$

$$\overline{c}_o = \overline{s}_o + \overline{p}_o, \ \forall o \in O \tag{12}$$

$$\underline{s}_{o(j,l)} \ge \underline{c}_{o(j,l-1)}, \ 1 \le l \le m_j, 1 \le j \le n \tag{13}$$

$$\overline{s}_{o(j,l)} \ge \overline{c}_{o(j,l-1)}, \ 1 \le l \le m_j, 1 \le j \le n \tag{14}$$

$$\underline{s}_o \ge \underline{c}_{o'} \ \vee \ \underline{s}_{o'} \ge \underline{c}_o, \forall o \ne o' \in O : \nu_o = \nu_{o'} \tag{15}$$

$$\overline{s}_o \ge \overline{c}_{o'} \ \vee \ \overline{s}_{o'} \ge \overline{c}_o, \forall o \ne o' \in O : \nu_o = \nu_{o'} \tag{16}$$

where the minimum $\min_R \mathbf{C_{max}}$ in (9) is the smallest interval according to a given ranking R in the set of intervals $\mathbb{IR}$. Constraint (10) defines the makespan as the maximum completion time of the last task of each job. Constraints (11) and (12) establish the relationship between the starting and completion time of each task. Constraints (13) and (14) correspond to precedence relations between tasks within each job and constraints (15) and (16) establish that the execution of two tasks requiring the same machine cannot overlap. Notice that the completion time of each job J_j in the resulting schedule $\boldsymbol{s}$ is the completion time of the last task in that job, given by $\mathbf{C_j} = \mathbf{c_{o(j,m_j)}}$.

The resulting problem will be denoted $J|\underline{p}_o \le p_o \le \overline{p}_o|\mathbf{C}_{max}$, following the three-field notation schema for scheduling problems. Clearly, the IJSP is NP-hard, since setting all processing times to crisp numbers yields the classical JSP, which is itself NP-hard [24].

3 Robust Schedules

A solution to the IJSP provides an interval of possible values for the starting time of each task and, hence, an interval of possible values for the makespan. In fact, it is impossible to predict what the exact starting and completion times will be until the project is actually executed. This idea is the basis for a semantics for fuzzy schedules from [10] by which solutions to a job shop problem with uncertainty should be understood as a-priori solutions. Only when tasks are executed according to the ordering π provided by the schedule we shall know their real duration and, hence, obtain an a-posteriori solution with deterministic times $p_o \in [\underline{p}_o, \overline{p}_o]$ for all tasks $o \in O$.

It would be expected that the predictive schedule does not differ much from the actual executed one. This is strongly related to the idea of robust schedule as one that minimises the effect of executional uncertainties on its performance [4]. This high-level definition is subject to many different interpretations when it comes to specifying robustness measures [25]. Here, we adapt the concept of ϵ-robustness first proposed for fuzzy scheduling problems in [22] inspired by the work on stochastic scheduling from [5].

The rationale behind this concept is to measure the predictive error of the a-priori makespan, the interval $\mathbf{C_{max}}$, compared to the actual makespan C^{ex}_{max} obtained after execution. Notice that C^{ex}_{max} is a real number that corresponds to a specific realisation of task processing times $P^{ex} = \{p^{ex}_o \in [\underline{p}_o, \overline{p}_o], o \in O\}$, usually called a *configuration* in the literature. Assuming that tasks are executed without unnecessary delays at their earliest possible starting times (as explained in Sect. 2.2), it is clear that $C^{ex}_{max} \in \mathbf{C_{max}}$. Thus, the prediction is always accurate in terms of bounds for the possible makespan values after execution. Now, if we are to give a single value as predicted makespan based on the interval $\mathbf{C_{max}}$, in the absence of further information it seems natural to consider the expected or mean value of the uniform distribution on that interval, $E[\mathbf{C_{max}}] = (\overline{C}_{max} - \underline{C}_{max})/2$. We can then measure the error of the prediction made by the a-priori makespan as the (relative) deviation of the executed makespan with respect to this expected value. In consequence, for a given $\epsilon \geq 0$, a predictive schedule with makespan interval value $\mathbf{C_{max}}$ will be considered to be ϵ-*robust* if the relative error made by $E[\mathbf{C_{max}}]$ with respect to the makespan C^{ex}_{max} of the executed schedule is bounded by ϵ, that is:

$$\frac{|C^{ex}_{max} - E[\mathbf{C_{max}}]|}{E[\mathbf{C_{max}}]} \leq \epsilon. \tag{17}$$

Clearly, the smaller the bound ϵ, the more accurate the a-priori prediction is or, in other words, the more robust the interval schedule is.

Although the expression for the expected value $E[\mathbf{C_{max}}]$ is the same as the interval's midpoint used in the ranking criterion $\leq_{MP}$, this is just a mere coincidence. In general, a robustness measure must be independent of the ranking method used to compare schedules. In particular, $E[\mathbf{C_{max}}]$ represents a prediction based on $\mathbf{C_{max}}$ in the absence of further knowledge on how values are

distributed in that interval, whereas the midpoint $m(\mathbf{C_{max}})$ is a weighted average of the optimistic $\overline{C}_{max}$ and pessimistic $\underline{C}_{max}$ makespan values, representing a decision maker's equilibrium between those two extreme attitudes.

Finally, this measure of robustness is dependent on a specific configuration P^{ex} of task processing times obtained upon execution of the predictive schedule $\boldsymbol{s}$. In the absence of real data regarding executions of the project, as is the case with the usual synthetic benchmark instances for job shop, we may resort to Monte-Carlo simulations. The idea is to simulate K possible configurations $P^k = \{p_o^k \in [\underline{p}_o, \overline{p}_o], o \in O\}$ for task processing times, using uniform probability distributions to sample possible durations for every task. For each configuration $k = 1, \ldots, K$, let C_{max}^k denote the exact makespan obtained after executing tasks according to the ordering provided by $\boldsymbol{s}$. Then, the average ϵ-robustness of the predictive schedule across the K possible configurations, denoted $\overline{\epsilon}$, can be calculated as:

$$\overline{\epsilon} = \frac{1}{K} \sum_{k=1}^{K} \frac{|C_{max}^k - E[\mathbf{C_{max}}]|}{E[\mathbf{C_{max}}]}. \tag{18}$$

This value provides an estimate of how robust is the predictive schedule $\boldsymbol{s}$ across different processing times configurations. Again, the lower $\overline{\epsilon}$, the better.

4 A Genetic Algorithm for the IJSP

Genetic algorithms have proved to be a very useful tool for solving job shop problems, either on their own or combined with other metaheuristics [28]. Roughly speaking, a genetic algorithm starts by building a set of initial solutions or initial population P_0. This population is then evaluated and the algorithm begins an iterative process until a stopping criterion is met, typically a fixed number of iterations or consecutive iterations without improvement. At each step i, individuals from the population P_i are selected and paired for mating and, recombination operators of crossover and mutation are applied to each pair with probability p_{cross} and p_{mut} respectively, creating a new population of offspring solutions Off_i. The new population is evaluated and a replacement operator is applied to merge P_i and Off_i into the new population P_{i+1} for the next iteration. Once the stopping criterion is met, the best solution according to the interval ranking is selected and returned from the last population. Algorithm 1 summarises these steps.

In this work, several well-known selection, recombination and replacement operators for Job Shop Scheduling problems are tried in order to find the best setup for the genetic algorithm. The set of operators and their impact on solving this problem are detailed in Sect. 5. A crucial part in designing algorithms is how to encode and decode solutions. Following [6], we encode a solution as a permutation with repetition. This is a permutation of the set of tasks, where each task $o(j, l)$ is represented by its job number j. For example, a topological order $(o(2,1),\ o(1,1),\ o(2,2),\ o(3,1),\ o(3,2),\ o(1,2))$ is encoded as $(2\,1\,2\,3\,3\,1)$.

Require: An IJSP instance
Ensure: A schedule
  Generate a pool P_0 of random solutions.
  Evaluate P_0
  $i \leftarrow 0$;
  while stop condition not satisfied **do**
    $Off_i \leftarrow$ pairs of individuals selected from P_i;
    for each pair of individuals in Off_i **do**
      Apply crossover operator with probability p_{cross};
      Apply mutation operator with probability p_{mut};
    Evaluate Off_i
    $P_{i+1} \leftarrow$ Apply replacement operator in (P_i, Off_i);
    $i \leftarrow i + 1$;
  $Best \leftarrow$ Best solution in P_i based on the order on intervals;
  return $Best$

Algorithm 1: Main steps of the genetic algorithm

The decoding is done using an insertion strategy: we iterate along the chromosome and for each task $o(j,l)$ we schedule it at its earliest feasible insertion position as follows. Let η_k be the number of tasks scheduled on machine $k = \nu_{o(j,l)}$ and let $\sigma_k = (0, \sigma(1,k), ..., \sigma(\eta_k, k))$ denote the partial processing order of tasks already scheduled in machine k. Then a feasible insertion position $q, 0 \leq q < \eta_k$ for $o(j,l)$ is a position such that $\max\{\underline{c}_{\sigma(q,k)}, \underline{c}_{o(j,l-1)}\} + \underline{p}_{o(j,l)} \leq \underline{s}_{\sigma(q+1,k)}$ and $\max\{\overline{c}_{\sigma(q,k)}, \overline{c}_{o(j,l-1)}\} + \overline{p}_{o(j,l)} \leq \overline{s}_{\sigma(q+1,k)}$, so the earliest feasible insertion position is the smallest value q^* verifying these inequalities. We set $\mathbf{s}_{o(j,l)} = \max\{\mathbf{c}_{\sigma(q^*,k)}, \mathbf{c}_{o(j,l-1)}\}$ if q^* exists, and $\mathbf{s}_{o(j,l)} = \max\{\mathbf{c}_{\sigma(\eta_k,k)}, \mathbf{c}_{o(j,l-1)}\}$ otherwise.

5 Experimental Study

The purpose of the experimental study is threefold: assess the proposed genetic algorithm, see if considering the uncertainty in processing times during the search process is worth the while and carry out a preliminary analysis of the influence of the different interval rankings.

To test the algorithm, we consider 12 very well-known instances for the job shop problem: classical instances FT10 (size 10×10) and FT20 (20×5), and instances La21, La24, La25 (15×10), La27, La29 (20×10), La38, La40 (15×15), and ABZ7, ABZ8, ABZ9 (20×15) that form the set of 10 problems identified in [2] as hard to solve for classical JSP. The processing times are modified to be intervals as follows: given the original crisp processing time of an operation p_o, the interval time is generated as $\mathbf{p_o} = [p_o - \delta, p_o + \delta]$, where δ is a random value in $[0, 0.15p_o]$. The resulting IJSP instances are available online[1]. All the experiments reported in this section have been run on a PC with Intel Xeon

[1] Repository section at http://di002.edv.uniovi.es/iscop.

Gold 6132 processor at 2.6 Ghz and 128 Gb RAM with Linux (CentOS v6.9), using a C++ implementation.

Regarding the algorithm's parameter configuration, we have run several tests to find the best setup. A first batch of experiments are conducted to test different recombination operations and probabilities as well as selection strategies. The considered operators are given in Table 1, with the best setup values in bold. In all cases the stopping criterion is set to 500 iterations.

Table 1. Parameter tuning with the best configuration in bold

Parameter	Tested values
Crossover operator	Generalised Order Crossover (GOX)
	Job-Order Crossover (JOX)
	Precedence Preservative Crossover (PPX)
Crossover probability	0.5, 0.75, **1**
Mutation operator	**Insertion**, Inversion, Swap
Mutation probability	0, **0.15**, 0.25
Selection operator	Roulette
	Tournament (t = 3)
	Shuffle
	Stochastic Universal Sampling (SUS)
Replacement	Generational replacement with elitism (k = 1, 12, 25)
	Tournament 2/4 parents-offspring (allowing repetition)
	Tournament 2/4 parents-offspring (no repetition)

A second test based on convergence demonstrates that the best population size is 250. Figure 1 shows the average evolution of the expected value of makespan across 30 runs of the algorithm on instance FT10. The dotted line corresponds to the expected makespan of the best solution in the population and the continuous line to the average of the whole population. It is clear that within 500 iterations, the algorithm reaches a convergence point. The behaviour on the remaining instances is similar, so we adopt this number of iterations as stopping criterion for the algorithm.

To asses the performance of the genetic algorithm (GA in the following), we compare it with the PNS algorithm proposed in [15], which to our knowledge constitutes the state-of-the art in the IJSP with makespan minimisation. The authors use $\leq_{pd}$ to rank different intervals in PNS. Since $\leq_{pd}$ is equivalent to $\leq_{MP}$ and for the sake of a fair comparison, we also adopt the same ranking. GA is run on the same set of 17 instances as PNS, which are adapted versions of the well-known crisp instances ORB1–5, LA16–25 and ABZ5–6, and the stopping criterion is set to 25 consecutive iterations without improvement. Table 2 shows for each algorithm, the average expected makespan across all the runs (20 runs

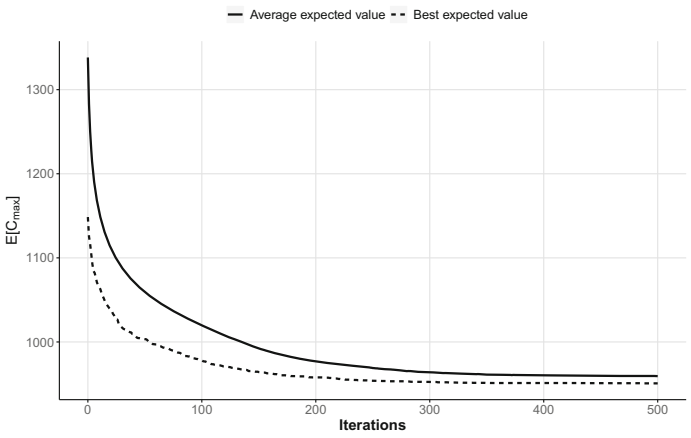


Fig. 1. Evolution of the best and average individual for FT10 instance.

for PNS and 30 for GA) together with runtimes, as well as a column with the relative difference between the performances of GA and PNS. We can see that, despite not having a neighbourhood search component, GA outperforms PNS in 14 out of the 17 instances, and is marginally worse in the remaining 3 instances (0.4% worse in ORB2, 0.5% in La16 and 0.01% in La17). Overall, GA obtains an average improvement of 1.2% compared to PNS. Additionally, a t-test for paired samples is run to compare the results of GA and PNS (after both samples pass a Kolmogorov-Smirnov normality test), confirming that there are indeed significant differences between both algorithms for a significance level of 0.05. Regarding runtime, GA is 93.8% faster than PNS. Notice however, that runtimes of PNS are those provided by the authors using their own machine and therefore comparisons in this sense must be done with caution.

We have used the set of 17 instances considered in [15] in Table 2 to compare GA with the state-of-the-art. However, in the deterministic case, the original crisp instances have already been solved to optimality and their fuzzy counterparts offer little room for improvement, as shown in [23]. For this reason we will now switch to the set of more challenging instances introduced at the beginning of this section for the remaining experimental results.

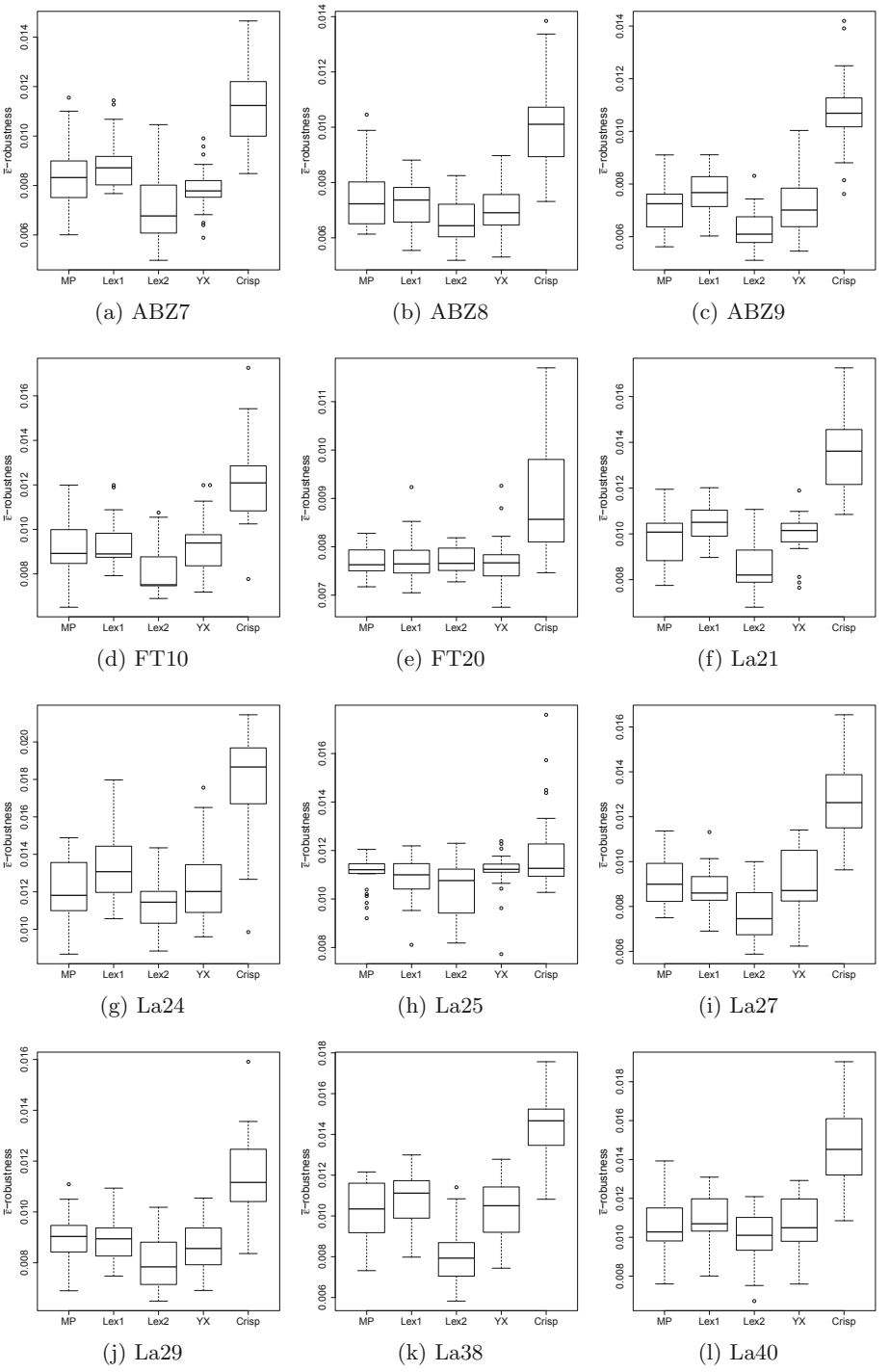
One may wonder if solving the crisp problem that results from considering only the midpoint of the interval processing times yields similar results to using intervals with the added advantage of having all the available tools for deterministic JSP. Including uncertainty in the search process adds some difficulty to the problem: different concepts need to be adapted or redefined and solving methods tailored to handle the uncertainty need to be proposed, usually with an increased complexity. It is also natural to see if the choice of a ranking method in the interval setting has any influence on the outcome. To try to answer these questions, we carry out a new set of experiments. For every IJSP instance we run GA 30 times considering each of the four different ranking methods and 30

Table 2. Computational results and times of PNS and GA

Instance	PNS		GA		Relative diff.
	Avg. $E[C_{max}]$	Runtime (s)	Avg. $E[C_{max}]$	Runtime (s)	
ORB1	1187.00	6.7	1171.12	0.5	−1.3%
ORB2	968.25	6.8	971.93	0.4	0.4%
ORB3	1145.23	3.5	1117.23	0.7	−2.4%
ORB4	1110.85	6.8	1087.13	0.6	−2.1%
ORB5	974.60	6.9	955.02	0.4	−2.0%
ABZ5	1308.45	5.3	1296.58	0.3	−0.9%
ABZ6	1012.40	6.3	998.95	0.3	−1.3%
La16	1019.40	6.2	1024.52	0.4	0.5%
La17	834.45	6.5	834.50	0.3	0.0%
La18	912.95	6.7	900.75	0.3	−1.3%
La19	919.65	6.0	904.95	0.5	−1.6%
La20	966.50	6.4	952.87	0.3	−1.4%
La21	1173.45	16.9	1150.73	1.0	−1.9%
La22	1036.05	16.8	1019.98	1.1	−1.6%
La23	1105.45	16.8	1083.27	0.9	−2.0%
La24	1047.55	16.9	1038.40	0.9	−0.9%
La25	1089.15	17.1	1077.05	1.0	−1.1%

times on the instance's crisp counterpart. Notice that the objective function is an interval in the first four cases and a crisp value in the last one, so they are not directly comparable. Instead, we measure the $\overline{\epsilon}$-robustness of the 30 solutions obtained by GA in each case using $K = 1000$ possible realisations, to compare the resulting solutions in terms of their quality as predictive schedules.

Figure 2 depicts for each instance the boxplots with the $\overline{\epsilon}$ values with the schedules that result from the 30 runs of GA in each case. We can see that, regardless of the ranking considered, solutions are more robust when intervals are taken into account during the search process. This is confirmed by several t-tests, showing that the $\overline{\epsilon}$-robustness of the interval schedules, regardless of the ranking, is significantly better than the one of the crisp schedule for a significance level of 0.05 on all instances expect La25. Regarding the choice of ranking method, according to the t-tests there is no significant difference between $\leq_{MP}$ and $\leq_{YX}$ on any instance. This is actually not surprising, since $\leq_{YX}$ can be understood as a refinement of $\leq_{MP}$, but it shows that this refinement does not necessarily translate into more robust schedules. Also, there are no significant differences between $\leq_{MP}$ and $\leq_{Lex1}$ on any instance except ABZ9, La21 and La24. More interestingly, the ranking $\leq_{Lex2}$ yields solutions significantly more robust than those obtained using any other ranking on all instances except FT20, La24 and La40, where it is not significantly better than $\leq_{MP}$ and $\leq_{YX}$. We may conclude



(a) ABZ7 (b) ABZ8 (c) ABZ9

(d) FT10 (e) FT20 (f) La21

(g) La24 (h) La25 (i) La27

(j) La29 (k) La38 (l) La40

Fig. 2. $\bar{\epsilon}$-robustness of solutions obtained with four different rankings and solving the crisp counterpart

that solving the interval JSP results in more robust schedules than solving a simpler deterministic counterpart and that the choice of interval ranking method does have an influence on the outcome.

6 Conclusions

In this work we have developed an approach to solving the IJSP using a GA. Results show that GA is competitive with the existing methods from the literature. In addition, incorporating the interval uncertainty in the search process yields more robust solutions than solving an alternative crisp problem. On the other hand, the choice of interval ranking method plays an important role in the final solution's performance. Further work needs to be done to obtain more powerful search methods specifically designed for handling interval uncertainty and to thoroughly analyse the influence of different ranking methods in order to make a proper choice.

References

1. Allahverdi, A., Aydilek, H., Aydilek, A.: Single machine scheduling problem with interval processing times to minimize mean weighted completion time. Comput. Oper. Res. **51**, 200–207 (2014). https://doi.org/10.1016/j.cor.2014.06.003
2. Applegate, D., Cook, W.: A computational study of the job-shop scheduling problem. ORSA J. Comput. **3**, 149–156 (1991)
3. Artigues, C., Briand, C., Garaix, T.: Temporal analysis of projects under interval uncertainty. In: Schwindt, C., Zimmermann, J. (eds.) Handbook on Project Management and Scheduling. IHIS, vol. 2, pp. 911–927. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-05915-0_11
4. Aytung, H., Lawley, M.A., McKay, K., Shantha, M., Uzsoy, R.: Executing production schedules in the face of uncertainties: a review and some future directions. Eur. J. Oper. Res. **161**, 86–110 (2005)
5. Bidot, J., Vidal, T., Laboire, P.: A theoretic and practical framework for scheduling in stochastic environment. J. Sched. **12**, 315–344 (2009)
6. Bierwirth, C.: A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectr. **17**, 87–92 (1995)
7. Bustince, H., Fernandez, J., Kolesárová, A., Mesiar, R.: Generation of linear orders for intervals by means of aggregation functions. Fuzzy Sets Syst. **220**, 69–77 (2013)
8. Dubois, D., Prade, H., Smets, P.: Representing partial ignorance. IEEE Trans. Syst. Man Cybern. Part A **26**(3), 361–377 (1996)
9. Dubois, D., Prade, H., Sandri, S.: On possibility/probability transformations. In: Fuzzy Logic, Theory and Decision Library, vol. 12, pp. 103–112. Kluwer Academic (1993)
10. González Rodríguez, I., Puente, J., Vela, C.R., Varela, R.: Semantics of schedules for the fuzzy job shop problem. IEEE Trans. Syst. Man Cybern. Part A **38**(3), 655–666 (2008)
11. Han, Y., Gong, D., Yaochu, J., Pan, Q.K.: Evolutionary multi-objective blocking lot-streaming flow shop scheduling with interval processing time. Appl. Soft Comput. **42**, 229–245 (2016)

12. Hurwicz, L.: A class of criteria for decision-making under ignorance. Cowles Commission Discussion Paper. Statistics **370** (1951)
13. Jiang, C., Han, X., Liu, G., Liu, G.: A nonlinear interval number programming method for uncertain optimization problems. Eur. J. Oper. Res. **188**(1), 1–13 (2008)
14. Karmakar, S., Bhunia, A.K.: A comparative study of different order relations of intervals. Reliable Comput. **16**, 38–72 (2012)
15. Lei, D.: Population-based neighborhood search for job shop scheduling with interval processing time. Comput. Ind. Eng. **61**, 1200–1208 (2011). https://doi.org/10.1016/j.cie.2011.07.010
16. Lei, D.: Interval job shop scheduling problems. Int. J. Adv. Manuf. Technol. **60**, 291–301 (2012). https://doi.org/10.1007/s00170-011-3600-3
17. Lei, D.: Multi-objective artificial bee colony for interval job shop scheduling with flexible maintenance. Int. J. Adv. Manuf. Technol. **66**, 1835–1843 (2013). https://doi.org/10.1007/s00170-012-4463-y
18. Lei, D., Cao, S.: A novel shuffled frog-leaping algorithm for flexible job shop scheduling with interval processing time. In: Proceedings of the 36th Chinese Control Conference, pp. 2708–2713 (2017)
19. Lei, D., Guo, X.: An effective neighborhood search for scheduling in dual-resource constrained interval job shop with environmental objective. Int. J. Prod. Econ. **159**, 296–303 (2015). https://doi.org/10.1016/j.ijpe.2014.07.026
20. Matsveichuk, N., Sotskov, Y., Egorova, N., Lai, T.C.: Schedule execution for two-machine flow-shop with interval processing times. Math. Comput. Modell. **49**(5), 991–1011 (2009). https://doi.org/10.1016/j.mcm.2008.02.004
21. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis. Society for Industrial and Applied Mathematics (2009)
22. Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Robust swarm optimisation for fuzzy open shop scheduling. Nat. Comput. **13**(2), 145–156 (2014). https://doi.org/10.1007/s11047-014-9413-1
23. Palacios, J.J., Puente, J., Vela, C.R., González-Rodríguez, I.: Benchmarks for fuzzy job shop problems. Inf. Sci. **329**, 736–752 (2016). https://doi.org/10.1016/j.ins.2015.09.042
24. Pinedo, M.L.: Scheduling. Theory, Algorithms, and Systems, 5th edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-26580-3
25. Roy, B.: Robustness in operational research and decision aiding: a multi-faceted issue. Eur. J. Oper. Res. **200**, 629–638 (2010)
26. Sotskov, Y.N., Egorova, N.G.: Single machine scheduling problem with interval processing times and total completion time objective. Algorithms **11**(5), 66 (2018)
27. Xu, Z., Yager, R.R.: Some geometric aggregation operators based on intuitionistic fuzzy sets. Int. J. Gen. Syst. **35**(4), 417–433 (2006)
28. Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J.: Review of job shop scheduling research and its new perspectives under Industry 4.0. J. Intell. Manuf. **30**(4), 1809–1830 (2017). https://doi.org/10.1007/s10845-017-1350-2