Tardiness Minimisation for Job Shop Scheduling with Interval Uncertainty*

Hernán Díaz[1][0000−0003−2615−0042], Juan José Palacios[1][0000−0002−0479−1490], Irene Díaz[1][0000−0002−3024−6605], Camino R. Vela[1][0000−0001−9271−2360], and Inés González-Rodríguez[2][0000−0003−3266−009X]

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

Abstract. This paper considers the interval job shop scheduling problem, a variant of the deterministic problem where task durations and due dates are uncertain and modelled as intervals. With the objective of minimising the total tardiness with respect to due dates, we propose a genetic algorithm. Experimental results are reported to assess its behaviour and compare it with the state-of-the-art algorithms, showing its competitiveness. Additional results in terms of solution robustness are given to illustrate the relevance of the interval ranking method used to compare schedules as well as the benefits of taking uncertainty into account during the search process.

Keywords: Job shop scheduling · Total tardiness · Interval uncertainty · Genetic algorithms · Robustness

1 Introduction

Scheduling consists in allocating a set of limited existing resources to perform a set of tasks with specific performance measures. It plays an important role in manufacturing systems because, if properly done, it can reduce material-handling costs and times as well as improving efficiency [16]. The job shop is one of the most relevant scheduling problems, since it has been considered to be a good model for many practical applications as well as a challenge to the research community due to its complexity. This complexity is the reason why metaheuristic search techniques are especially suited for solving the job shop problem [19].

A majority of contributions to the family of job shop scheduling problems concentrate on minimising the execution time span of the project (known as makespan). However, in recent years there has been a growing interest in due-date related criteria [5, 8, 14]. On-time fulfilment becomes especially relevant in

* 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

modern pull-oriented supply chain systems concerned with meeting customer's demand in terms of due dates; a tardy job may result in delay-compensation cost, customer dissatisfaction or loss of reputation among others. It is also relevant in complex supply chain or manufacturing systems integrating planning and scheduling; tardy jobs in one of the scheduling stages may cause serious disruptions and delays in subsequent stages, with the associated costs.

Traditionally, it has been assumed in scheduling that design variables such as task processing times or due dates are deterministic. However, this assumption may seem naïve in real industrial settings because in real-world production scheduling problems such variables are quite often characterised vaguely due to the available information being incomplete or imprecise. Fuzzy sets have been used by many researchers to model uncertain durations in scheduling problems [2]. Regarding due dates, fuzzy sets are mostly used in the literature to model flexibility; there are however some cases where fuzzy due dates model uncertainty [4]. An alternative to fuzzy sets for dealing with uncertainty are intervals. Interval uncertainty is present as soon as information is incomplete. An expert working on a project may be reluctant or unable to provide point values for the duration of each task, while estimating a minimal and a maximal duration may be felt as more realistic. Also, the end of the job may depend on several uncertain events such as changing customer orders or dependencies on other components of a larger manufacturing system. Interval scheduling provides us with the possibility of focussing on significant scheduling decisions and give quite robust solutions, with little sensitivity to uncertainties.

Interval uncertainty is not new in scheduling, although contributions in the literature are still scarce. In [10], a genetic algorithm is proposed for a job shop problem with interval processing times and interval due dates to minimise the total tardiness with respect to job due dates. A population-based neighborhood search for a interval job shop, but with the objective of the makespan is presented in [9]. A multiobjective interval job shop problem with non-resumable jobs and flexible maintenance is solved in [11] by means of a multiobjective artificial bee colony algorithm that minimises both the makespan and the total tardiness. In [12], a dual-resource constrained job shop with heterogeneous resources is considered, and a dynamical neighbourhood search is proposed for lexicographic minimisation of carbon footprint and makespan.

In the following, we consider the job shop scheduling problem with intervals modelling both uncertain durations and uncertain due dates. With the goal of minimising total tardiness, an interval in this case, we propose a genetic algorithm that provides the basis for developing more sophisticated search methods in the future as well as allowing to study the influence of the ranking method used for comparing the interval objective function. A preliminary experimental study also highlights the benefits of considering the uncertainty during the search process in terms of robustness, specially with one of the ranking methods.

2 Problem definition

The classical *job shop scheduling problem*, or *JSP* in short, consists in a set of jobs $J = \{J_1, \ldots, J_n\}$ to be scheduled 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 *capacity constraints*, whereby each task $o(j,l)$ requires the uninterrupted and exclusive use of a specific machine $\nu_{o(j,l)} \in M$ for its whole processing time $p_{o(j,l)}$. Additionally, each job J_j has a due date d_j by which it is desirable that the job be completed.

A solution to this problem is a *schedule* $\boldsymbol{s}$, i.e. an allocation of starting times $s_{o(j,l)}$ for each task $o(j,l)$, which is feasible (in the sense that all constraints hold) as well as optimal according to some criterion, in our case, minimum total tardiness with respect to due dates.

A schedule $\boldsymbol{s}$ establishes an order π among tasks requiring the same machine. Conversely, given a task processing order π, the schedule $\boldsymbol{s}(\pi)$ may be computed as follows. Let us 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 p_o to refer to its processing time. The set of all tasks is denoted $O = \{1, \ldots, N\}$. For every task $o \in O$, let $s_o(\pi)$ and $c_o(\pi)$ denote respectively the starting and completion times of o given π, let $PM_o(\pi)$ and $SM_o(\pi)$ denote the predecessor and successor tasks of o in its required machine, and let PJ_o and SJ_o denote respectively the predecessor and successor tasks of o in its job. If o is the first task to be processed in its machine or its job, $PM_o(\pi) = 0$ or $PJ_o = 0$, where 0 represents a dummy task such that $p_0 = 0$ and $s_0 = 0$. Then the starting time $s_o(\pi)$ is given by $s_o(\pi) = \max(s_{PJ_o} + p_{PJ_o}, s_{PM_o(\pi)} + p_{PM_o(\pi)})$ and the completion time is computed as $c_o(\pi) = s_o(\pi) + p_o$. Notice that the completion time of each job J_j for $\boldsymbol{s}(\pi)$ is the completion time of the last task in that job, given by $C_j(\pi) = c_{o(j,m_j)(\pi)}$. The total tardiness of the schedule is given by $T_{tot}(\pi) = \sum_{j=1}^{n} T_j(\pi)$, where $T_j(\pi)$ is the tardiness of job J_j according to π, $T_j(\pi) = \max(0, C_j(\pi) - d_j)$.

2.1 Uncertain Processing Times and Due Dates

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. In addition, due dates may not be perfectly defined, being dependent on external factors such as changing customer orders or dynamic manufacturing requirements. If only an upper and a lower bound of each duration and due date are known, uncertainty 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 total tadiness mimisation requires three arithmetic operations on $\mathbb{IR}$: addition, subtraction and maximum. These are defined by extending the corresponding

operations on real numbers [13], so given two intervals $\mathbf{a} = [\underline{a}, \overline{a}], \mathbf{b} = [\underline{b}, \overline{b}] \in \mathbb{IR}$,

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

$$\mathbf{a} - \mathbf{b} = [\underline{a} - \overline{b}, \overline{a} - \underline{b}], \tag{2}$$

$$\max(\mathbf{a}, \mathbf{b}) = [\max(\underline{a}, \underline{b}), \max(\overline{a}, \overline{b})]. \tag{3}$$

Comparisons are a key point when processing times and due dates take the form of intervals as the "best" schedule should be the one with "minimal" total tardiness (an interval). However, it is well known that 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]. Among the multiple existing rankings in $\mathbb{IR}$, here we consider the following:

$$\mathbf{a} \leq_{Lex1} \mathbf{b} \Leftrightarrow \underline{a} < \underline{b} \vee (\underline{a} = \underline{b} \wedge \overline{a} < \overline{b}) \tag{4}$$

$$\mathbf{a} \leq_{Lex2} \mathbf{b} \Leftrightarrow \overline{a} < \overline{b} \vee (\overline{a} = \overline{b} \wedge \underline{a} < \underline{b}) \tag{5}$$

$$\mathbf{a} \leq_{YX} \mathbf{b} \Leftrightarrow \underline{a} + \overline{a} < \underline{b} + \overline{b} \vee (\underline{a} + \overline{a} = \underline{b} + \overline{b} \wedge \overline{a} - \underline{a} \leq \overline{b} - \underline{b}) \tag{6}$$

$$\mathbf{a} \leq_{MP} \mathbf{b} \Leftrightarrow m(\mathbf{a}) \leq m(\mathbf{b}) \text{ with, } m(\mathbf{a}) = \frac{(\underline{a} + \overline{a})}{2} \tag{7}$$

(4), (5) and (6) actually define total order relations in $\mathbb{IR}$ [3]. Both (4) and (5) are derived from a lexicographical order of interval extreme points. The ranking proposed in expression (6) is proposed in [18], and the last one (midpoint order) is a particular case of the classical Hurwitz criterion and is equivalent to the one proposed in [9] for interval job shop.

2.2 The JSP with Interval Uncertainty

Given the above, the *Interval Job Shop Scheduling Problem* or *IJSP* for total tardiness minimisation can be formulated as follows:

$$\min_{R} \mathbf{T_{tot}} \tag{8}$$

$$\text{subject to:} \quad \mathbf{T_{tot}} = \sum_{j=1}^{n} \mathbf{T}_j \tag{9}$$

$$\underline{T}_j = \max(0, \underline{c}_{o(j,m_j)} - \overline{d_j}) \tag{10}$$

$$\overline{T}_j = \max(0, \overline{c}_{o(j,m_j)} - \underline{d_j}) \tag{11}$$

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

$$\overline{c}_o = \overline{s}_o + \overline{p}_o, \ \forall o \in O \tag{13}$$

$$\underline{s}_{o(j,l)} \geq \underline{c}_{o(j,l-1)}, \ 1 \leq l \leq m_j, 1 \leq j \leq n \tag{14}$$

$$\overline{s}_{o(j,l)} \geq \overline{c}_{o(j,l-1)}, \ 1 \leq l \leq m_j, 1 \leq j \leq n \tag{15}$$

$$\underline{s}_o \geq \underline{c}_{o'} \ \vee \ \underline{s}_{o'} \geq \underline{c}_o, \forall o \neq o' \in O : \nu_o = \nu_{o'} \tag{16}$$

$$\overline{s}_o \geq \overline{c}_{o'} \ \vee \ \overline{s}_{o'} \geq \overline{c}_o, \forall o \neq o' \in O : \nu_o = \nu_{o'} \tag{17}$$

where the minimum $\min_R \mathbf{T_{tot}}$ in (8) is the smallest interval according to a given ranking R in the set of intervals $\mathbb{IR}$. Constraint (9) defines the total tardiness as the addition of the tardiness of each job J_j. Constraints (10) and (11) define the tardiness of each job J_j as the interval difference between the completion time of the job and its due date. Constraints (12) and (13) establish the relationship between the starting and completion time of each task. Constraints (14) and (15) correspond to precedence relations between tasks within each job, and constraints (16) and (17) establish that the execution of two tasks requiring the same machine cannot overlap.

The resulting problem will be denoted $J|\underline{p}_o \leq p_o \leq \overline{p}_o, \underline{d}_j \leq d_j \leq \overline{d}_j|\mathbf{T}_{tot}$, following the three-field notation schema for scheduling problems. Clearly, this problem is NP-hard, since setting all processing times and due dates to crisp numbers yields the classical JSP, which is itself NP-hard [16].

3 Robustness on Interval Schedules

A solution to the IJSP provides an interval of possible values for the total tardiness computed from the possible values for the starting and completion times of each task and the due date for each job. As it is impossible at the time of scheduling to predict what the exact due dates, starting and completion times will be when the project is actually executed, a solution to a job shop problem with uncertainty should be understood as an a-priori or predictive solution [6]. Once the project is finished, and tasks have been executed according to the ordering π provided by the schedule, we shall know real duration of the tasks, deterministic times $p_o \in [\underline{p}_o, \overline{p}_o]$ for all tasks $o \in O$. Specific due dates are also unknown until an actual instance of the project is tackled. Only at the moment of executing the project are actual due dates $d_j \in [\underline{d}_j, \overline{d}_j]$ available for each job J_j. After execution, when processing times and due dates are exactly known and the a-posteriori solution is available, delays w.r.t. each job $T_j \in [\underline{T}_j, \overline{T}_j]$ can be computed as well as the resulting total tardiness, being desirable that the predictive schedule does not differ much from the executed one.

This coincides with the idea of robust schedule, as one that minimises the effect of executional uncertainties on its performance [1]. The formalization of this concept leads to different robustness measures [17]. In this work, the concept of ϵ-robustness, first proposed for fuzzy scheduling problems in [15], is adapted to the interval framework.

ϵ-robustness intends to measure the predictive error of the a-priori total tardiness, $\mathbf{T_{tot}}$ (an interval), compared to the real total tardiness T^{ex}_{tot} obtained after an execution (corresponding to a specific realization of task processing times $P^{ex} = \{p^{ex}_o \in [\underline{p}_o, \overline{p}_o], o \in O\}$ and job due dates $d^{ex}_j \in [\underline{d}_j, \overline{d}_j]$, $1 \leq j \leq n$). Assuming that tasks are executed without unnecessary delays at their earliest possible starting times, it is clear that $T^{ex}_{tot} \in \mathbf{T_{tot}}$. Thus, the prediction is always accurate in terms of bounds for the possible objective values after execution. On the other hand, in absence of other information it seems straightforward to estimate the total tardiness as the expected or mean value of the uniform

distribution on $\mathbf{T_{tot}}$ and then measure the error of the prediction made by the a-priori metric as the (relative) deviation of the executed objective value with respect to this expected value.

In consequence, a predictive schedule with total tardiness interval value $\mathbf{T_{tot}}$ is ϵ-*robust* if the relative error made by $E[\mathbf{T_{tot}}]$ with respect to the total tardiness T_{tot}^{ex} of the executed schedule is bounded by ϵ, that is:

$$R_{ex} = \frac{|T_{tot}^{ex} - E[\mathbf{T_{tot}}]|}{E[\mathbf{T_{tot}}]} \leq \epsilon \text{ with } E[\mathbf{T_{tot}}] = (\overline{T}_{tot} - \underline{T}_{tot})/2 \text{ and } \epsilon \geq 0. \quad (18)$$

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.

When the problem is tested on synthetic benchmark instances for job shop, real data regarding executions of the project are not available. In this case K possible scenarios are obtained using Monte-Carlo simulations. Thus, deterministic values for due dates and processing times are sampled on their respective interval using uniform probability distributions. Then, the average ϵ-robustness of the predictive schedule across the K possible configurations, denoted $\bar{\epsilon}$, can be calculated as:

$$\bar{\epsilon} = \frac{1}{K}\sum_{k=1}^{K} R_k = \frac{1}{K}\sum_{k=1}^{K} \frac{|T_{tot}^{k} - E[\mathbf{T_{tot}}]|}{E[\mathbf{T_{tot}}]}. \quad (19)$$

with T_{tot}^{k} denoting the exact total tardiness obtained after executing tasks according to the ordering provided by the predictive schedule $\boldsymbol{s}$ for each scenario $k = 1, \ldots, K$. This value provides an estimate of how robust is the schedule $\boldsymbol{s}$ across different processing times configurations. Again, the lower $\bar{\epsilon}$, the better.

4 A Genetic Algorithm for Tardiness Minimisation

Genetic algorithms, either on their own or combined with other metaheuristics such as tabu search [19], are a powerful tool for solving scheduling problems. In brief, a genetic algorithm starts by generating a pool of initial solutions, representing a population P_0 of individuals of a species. This population is evaluated and a fitness value, typically the value of the objective function, is assigned to each individual. The population is then left to evolve until a stopping criterion is met, usually for a fixed amount of generations or consecutive iterations without improvement. At each iteration i, individuals from population P_i are paired for mating following a selection procedure, and recombination operators of crossover and mutation are applied to each pair with probability p_{cross} and p_{mut} respectively, simulating natural evolution. The new population of individuals Off_i is evaluated and a replacement operator is applied to combine P_i and Off_i into a new population P_{i+1} for the next iteration, rewarding individuals with better fitness and keeping a constant population size. Once the stopping criterion is met, the best individual according to the interval ranking is selected from the last generation and returned.

In a genetic algorithm, each individual codifies a solution as a chromosome, typically an array of values. The design of encoding and decoding algorithms pose the most crucial step in designing the algorithm. To encode solutions, we use classical permutations with repetition of job's numbers. These represent linear orders of the set of tasks, where each task $o(j,l)$ is represented by its job number j. For example, a linear order ($o(3,1)$, $o(1,1)$, $o(3,2)$, $o(2,1)$, $o(2,2)$, $o(1,2)$) is encoded as (3 1 3 2 2 1). The decoding follows an insertion strategy, so we always obtain a so-called active schedule in the sense that no operation can start earlier without disrupting the starting time of at least another operation. This strategy is done by iterating through the chromosome and scheduling each task $o(j,l)$ at its earliest feasible insertion position. Let η_k be the number of tasks already 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. A feasible insertion position $q, 0 \leq q < \eta_k$ for $o(j,l)$ is a position that verifies both:

$$\max\{\underline{c}_{\sigma(q,k)}, \underline{c}_{o(j,l-1)}\} + \underline{p}_{o(j,l)} \leq \underline{s}_{\sigma(q+1,k)} \tag{20}$$

$$\max\{\overline{c}_{\sigma(q,k)}, \overline{c}_{o(j,l-1)}\} + \overline{p}_{o(j,l)} \leq \overline{s}_{\sigma(q+1,k)}, \tag{21}$$

being $q = \eta_k$ if there no feasible insertion position.

The earliest feasible insertion position q^* is that with smallest q value. Therefore, operation $o(j,l)$ is scheduled at starting time:

$$\mathbf{s}_{o(j,l)} = \max\{\mathbf{c}_{\sigma(q^*,k)}, \mathbf{c}_{o(j,l-1)}\} \tag{22}$$

5 Experimental Results

We conduct a series of experiments to study three different aspects: the behaviour of the proposed genetic algorithm, the potential advantage of managing uncertainty during the search process, and the influence of the different interval rankings. For these tests, we consider the instances proposed in [10], which are, as far as we know, the only instances available in the literature for interval JSP with uncertain due dates. The set consists on 7 instances, 4 of them of size 10×10 (instances 1-4) and the remaining 3 of size 15×10. All the experiments reported in this section have been run on a PC with Intel Xeon Gold 6132 processor at 2.6 Ghz and 128 Gb RAM with Linux (CentOS v6.9), using a C++ implementation.

First, a preliminary study is carried out to find the best setup for the genetic algorithm. An initial base setup is established and then parameters are tuned sequentially, trying different combinations of operators and probabilities. Specifically, the considered operators and values are the following:

- Crossover operator: Generalised Order Crossover (GOX), **Job-Order Crossover (JOX)** and Precedence Preservative Crossover (PPX)
- Crossover probability: 0.7, 0.8, 0.9 and **1.0**
- Mutation operator: Swap, Inversion and **Insertion**
- Mutation probability: **0.05**, 0.10, 0.15 and 0.30

- Selection operator: **Shuffle**, Roulette, Stochastic Universal Sampling (SUS) and Tournament 1/3 on the population
- Replacement operator: Generational replacement with elitism (k=1, 5%, 10%), Tournament 2/4 parents-offspring allowing repetition and **Tournament 2/4 parents-offspring without repetitions**

The best setup values obtained are highlighted in bold. In all cases the algorithm is run until 25 consecutive iterations pass without improving the best found solution. In addition, three different populations sizes are tested: 100, 250 and 500. Since this parameter has a heavy influence on the runtime of the algorithm, it requires a more careful study. When a population size of 250 is used, the runtime increases 1.8 times w.r.t. using 100, and the average quality of the obtained solutions improves 2.1%. However, a further increase to 500 multiplies the runtime by 2, but the obtained results improve only 0.7%. If runtime is of great importance, a population size of 100 might be the most adequate, but when it is not the case, a population size of 250 offers a significant improvement in quality. Further increasing the population size does not seem to improve results at the same pace, so for this study we choose to use 250 individuals.

To assess the performance of our genetic algorithm (GA in the following), we compare it with the best-known results in the literature. Different methods for the IJSP can be found in the literature minimising total tardiness (see Section 1). Among the published results, the best reported values for total tardiness are those obtained by the genetic algorithm proposed in [10]. We shall refer to this GA as GA-L to distinguish it from ours in the comparisons. The authors use a method equivalent to $\leq_{MP}$ to rank different intervals in their GA-L, so we will also adopt this ranking for the sake of a fair comparison. Table 1 shows, for each algorithm, the best solution obtained across all runs (20 runs for GA-L and 30 for GA), the average expected total tardiness across those runs and the average running time in seconds for each algorithm. A first look at the results shows that, in average values, the GA proposed in this work outperforms GA-L in 5 out of the 7 instances, with greater improvement on the large instances. On average, GA is 5.5% better on instance 5, 12.5% better on instance 6 and 21.4% on instance 7. On the other hand, for smaller instances, there is no clear winner: GA is better on instances 2 and 4 (2.0% and 1.2% respectively), but worse on instances 1 and 3 (1.9% and 3.9% respectively). Furthermore, for instances 1 and 3, the best solution found by GA does not even reach the average results of GA-L. To have a better understanding of this situation, a basic model of these instances is designed to be solved with the IBM CPLEX CP Optimizer solver. Even though CP Optimizer cannot reach the optimal solution for all instances with the current version of the model, it obtains lower bounds that are actually higher than the best solutions obtained by GA-L on instances 2, 3 and 4. This indicates that the published results for those instances are unattainable, and therefore we must be very cautious when comparing with them. Figure 1 depicts the best solution found by GA on instance 4, which is optimal according to CPLEX CP Optimizer. It is a Gantt chart adapted to intervals, so for each task, instead of a bar, there is a trapezoid where the upper side corresponds to the

Instance	GA-L Best	GA-L E[Best]	GA-L Avg.	GA-L Time	GA Best	GA E[Best]	GA Avg.	GA Time
1	[5, 321]	163.0	166.1	6.5	[3, 335]	169.0	169.3	0.4
2	[0, 493]	246.5	264.0	6.6	[0, 497]	248.5	258.6	0.5
3	[7, 459]	233.0	243.4	6.4	[8, 479]	243.5	252.9	0.6
4	[4, 451]	227.5	245.2	6.3	[1, 458]	229.5	242.3	0.5
5	[79, 1678]	878.5	943.9	21.5	[43, 1651]	847.0	891.9	1.4
6	[0, 1048]	524.0	568.8	22.0	[0, 949]	474.5	497.4	1.4
7	[69, 1524]	796.5	999.0	21.1	[68, 1376]	722.0	785.0	1.2

Table 1: Computational results and times of GA-L and GA

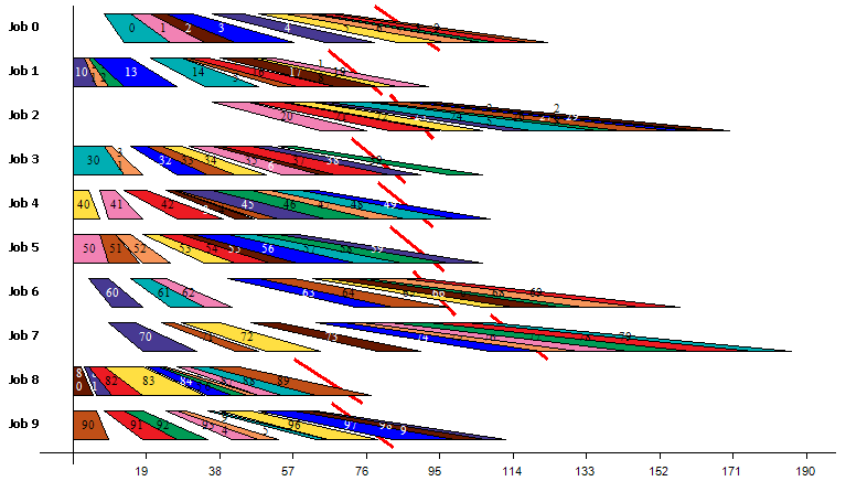
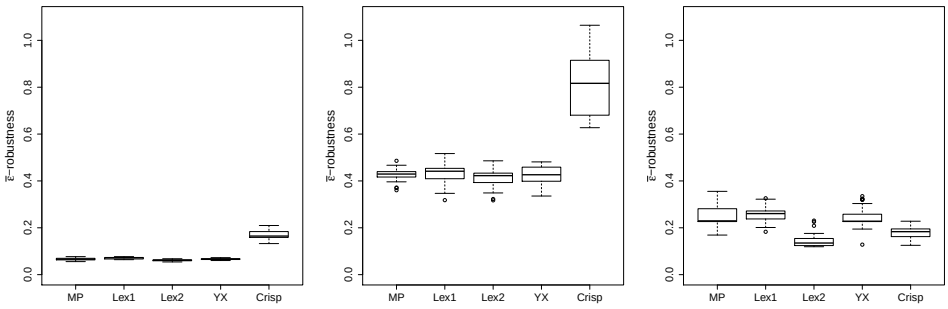


Fig. 1: Gantt diagram of the optimal solution found for Instance 4

earliest possible starting and completion times while the lower side corresponds to the latest possible starting and completion time; each row in the diagram shows the task sequence per job, colours determine the machine where each task needs to be executed, and thick red lines are the due dates. Regarding runtime, GA is 93.5% faster than GA-L. Notice however, that runtimes for GA-L are those provided by the authors using their own machine, therefore comparisons in this sense must be done with caution as well.

Using the intervals during the search process might add some extra difficulty to the problem: known concepts need to be adapted or redefined and solving methods redesigned to handle uncertainty, usually with an increased complexity. One may wonder if solving the crisp problem that results from considering only the midpoint of the interval processing times and due dates would lead to similar results with the added advantage of having all the available tools for deterministic JSP. It is also interesting to study the influence of the choice of



(a) Instance5 (b) Instance6 (c) Instance7

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

ranking methods in the interval setting. To shed some light on these questions, we carry out a new set of experiments. For each of the 7 instances, we run our algorithm 30 times considering each of the four different ranking methods and 30 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 $\bar{\epsilon}$-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 instances 5, 6 and 7, the boxplots with the $\bar{\epsilon}$ values of the 30 schedules obtained by GA in each case. Results for instances 2 and 4 are very similar those obtained for instance 6: in all of them the $\bar{\epsilon}$ values obtained from optimising the crisp instance are significantly worse than those obtained using intervals, independently of the ranking method used. Also, Mann-Whitney-U tests show that there are no significant differences between the results obtained using different ranking methods, except on instance 2, where using $\leq_{Lex1}$ yields significantly worse results than using $\leq_{MP}$ or $\leq_{YX}$. On the other hand, results for instances 1 and 3 are similar to those illustrated for instance 5. Again, the $\bar{\epsilon}$ values obtained from the solutions to the crisp instance are clearly worse than those obtained using intervals during the search process. However, in these instances $\leq_{Lex2}$ obtains solutions that are more robust than those obtained by any other ranking method. This is confirmed by the statistical tests, showing a significant difference between the $\bar{\epsilon}$ values of the schedules obtained using $\leq_{Lex2}$, and those of the schedules obtained with the other ranking methods. Results obtained on instance 7 are very different to the previous ones. This is the only instance where the solutions obtained from solving the crisp problem are not worse that those obtained from solving the interval JSP. In fact, the statistical tests show that they are significantly better than using the ranking methods $\leq_{MP}$, $\leq_{Lex1}$ and $\leq_{YX}$, but significantly worse than using $\leq_{Lex2}$. Further research is required on this particular instance for a better understanding on how the structure of the

problem may affect the choice of a ranking method. A preliminary analysis shows that even though the size is the same as on instances 5 and 6, task durations and due dates are larger in instance 7. For instance, the largest due dates are 172 (interval [140, 172]) and 215 (interval [176, 215]) for instances 5 and 6 respectively, while it is 1080 for instance 7 (interval [1033, 1080]). In addition, the width of the due-date intervals for instance 7 is also larger than for any other instance, directing us towards the idea that when uncertainty in due dates is big, a more conservative ranking method like $\leq_{Lex2}$ is the most adequate, being the others less reliable. Finally, it is worth mentioning that, according to the statistical tests, there is no significant difference between $\leq_{MP}$ and $\leq_{YX}$ on any instance. This seems natural, since $\leq_{YX}$ can be understood as a refinement of $\leq_{MP}$, but it also shows that this refinement does not necessarily translate into more robust schedules.

6 Conclusions

In this work we have tackled the job shop scheduling problem with uncertain durations and uncertain due dates modelled as intervals (IJSP). We have proposed a first approach to solving the IJSP using a genetic algorithm with an insertion decoding strategy. The algorithm has been tested on the 7 instances available in literature, outperforming the best-known results in 5 of them. In the largest instances, the proposed algorithm is always better, offering an improvement of up to 21% w.r.t the best previously-reported results. This is not the case for the smaller instances, where the results are not always better than the reported ones. However, we have checked –using IBM ILOG CP Optimizer– that these published results are infeasible, and therefore, not suitable for meaningful comparisons. A more detailed study has shown that the choice of interval ranking method plays a very important role in the final solution's performance, especially in terms of robustness. In addition, incorporating the interval uncertainty to the search process yields more robust solutions than solving the crisp problem, specifically with one of the ranking methods. Further work is needed to obtain more powerful search methods tailored for handling interval uncertainty and to thoroughly analyse the influence of different ranking methods in order to make a proper choice for the problem at hand.

References

1. 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. European Journal of Operational Research **161**, 86–110 (2005)
2. Behnamian, J.: Survey on fuzzy shop scheduling. Fuzzy Optimization and Decision Making **15**, 331–366 (2016). https://doi.org/10.1007/s10700-015-9225-5
3. Bustince, H., Fernandez, J., Kolesárová, A., Mesiar, R.: Generation of linear orders for intervals by means of aggregation functions. Fuzzy Sets and Systems **220**, 69–77 (2013)

4. Chanas, S., Kasperski, A.: On two single machine scheduling problems with fuzzy processing times and fuzzy due dates. European Journal of Operational Research **147**, 281–296 (2003)
5. González, M., Vela, C.R., Varela, R.: An efficient memetic algorithm for the flexible job shop with setup times. In: Proceedings of the 23th International Conference on Automated Planning and Scheduling (ICAPS-2013). pp. 91–99 (2013)
6. González Rodríguez, I., Puente, J., Vela, C.R., Varela, R.: Semantics of schedules for the fuzzy job shop problem. IEEE Transactions on Systems, Man and Cybernetics, Part A **38**(3), 655–666 (2008)
7. Karmakar, S., Bhunia, A.K.: A comparative study of different order relations of intervals. Reliable Computing **16**, 38–72 (2012)
8. Kuhpfahl, J., Bierwirth, C.: A study on local search neighbourhoods for the job shop scheduling problem with total weighted tardiness objective. Computers & Operations Research **261**, 44–57 (2016)
9. Lei, D.: Population-based neighborhood search for job shop scheduling with interval processing time. Computers & Industrial Engineering **61**, 1200–1208 (2011). https://doi.org/10.1016/j.cie.2011.07.010
10. Lei, D.: Interval job shop scheduling problems. International Journal of Advanced Manufacturing Technology **60**, 291–301 (2012). https://doi.org/10.1007/s00170-011-3600-3
11. Lei, D.: Multi-objective artificial bee colony for interval job shop scheduling with flexible maintenance. International Journal of Advanced Manufacturing Technology **66**, 1835–1843 (2013). https://doi.org/10.1007/s00170-012-4463-y
12. Lei, D., Guo, X.: An effective neighborhood search for scheduling in dual-resource constrained interval job shop with environmental objective. International Journal of Production Economics **159**, 296–303 (2015). https://doi.org/10.1016/j.ijpe.2014.07.026
13. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis. Society for Industrial and Applied Mathematics (2009)
14. Mou, J., Gao, L., Pan, Q., Mu, J.: Multi-objective inverse scheduling optimization of single-machine shop system with uncertain due-dates and processing times. Cluster Computing **20**(1), 371–390 (2017)
15. Palacios, J.J., González-Rodríguez, I., Vela, C.R., Puente, J.: Robust swarm optimisation for fuzzy open shop scheduling. Natural Computing **13**(2), 145–156 (2014)
16. Pinedo, M.L.: Scheduling. Theory, Algorithms, and Systems. Springer, fifth edn. (2016)
17. Roy, B.: Robustness in operational research and decision aiding: A multi-faceted issue. European Journal of Operational Research **200**, 629–638 (2010)
18. Xu, Z., Yager, R.R.: Some geometric aggregation operators based on intuitionistic fuzzy sets. International Journal of General Systems **35**(4), 417–433 (2006)
19. Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J.: Review of job shop scheduling research and its new perspectives under industry 4.0. Journal of Intelligent Manufacturing **30**(4), 1809–1830 (2019). https://doi.org/10.1007/s10845-017-1350-2