Solving Fuzzy Job-Shop Scheduling Problems with a Multiobjective Optimizer

Thanh-Do Tran, Ramiro Varela, Inés González-Rodríguez, El-Ghazali Talbi

Solving Fuzzy Job-Shop Scheduling Problems with a Multiobjective Optimizer

Thanh-Do Tran[1]*, Ramiro Varela[2], Inés González-Rodríguez[3], and El-Ghazali Talbi[1]

[1] DOLPHIN Team, Inria Lille – Nord Europe and LIFL, Université Lille 1, France
[2] A.I. Centre and Department of Computer Science, University of Oviedo, Spain
[3] Dept. of Mathematics, Statistics and Computing, University of Cantabria, Spain

Abstract. In real-world manufacturing environments, it is common to face a job-shop scheduling problem (JSP) with uncertainty. Among different sources of uncertainty, processing times uncertainty is the most common. In this paper, we investigate the use of a multiobjective genetic algorithm to address JSPs with uncertain durations. Uncertain durations in a JSP are expressed by means of triangular fuzzy numbers (TFNs). Instead of using expected values as in other work, we consider all vertices of the TFN representing the overall completion time. As a consequence, the proposed approach tries to obtain a schedule that optimizes the three component scheduling problems (corresponding to the lowest, most probable, and largest durations) all at the same time. In order to verify the quality of solutions found by the proposed approach, an experimental study was carried out across different benchmark instances. In all experiments, comparisons with previous approaches that are based on a single-objective genetic algorithm were also performed.

1 Introduction

Job-shop scheduling problems (JSPs) are known to be one of the hardest classes of combinatorial problems. They have formed an important body of research since the late fifties, with multiple applications in industry, finance, and science [23]. In fact, JSPs are not only NP-complete but also among the worst NP-complete class members [21]. Therefore, they have usually been solved by using heuristic techniques, rather than exact methodologies.

During the past two decades, various proposals based on genetic algorithms have been introduced to solve large JSP instances: [4], [27], [22], and [15], to name but a few. Even though most proposals deal with crisp JSPs, i.e. all relevant information is assumed to be concrete, in many real-life situations, it is often the case that the exact duration of a task is not known in advance [10]. Instead, based on previous experience, an expert may have some knowledge about this duration and it is therefore possible to estimate this processing time. In such a

* TDT was supported by the ECSC Scholarship Program of the Master in Soft Computing and Intelligent Data Analysis at the European Centre for Soft Computing and, currently, is supported by a CORDI-S Doctoral Fellowship of Inria.

situation, it is neither possible nor plausible to represent processing times with concrete numbers.

Depending on the available knowledge and the representation technique being used, the information about durations can be modeled by an interval for the possible processing times or its most typical values. When deeper knowledge of the problem is available, fuzzy intervals—which are considered as an alternative to probability distributions—can also be used; however, this technique usually requires complex computation [13]. When only little knowledge is available, we can use a confidence interval to represent the uncertain duration of a task. In this context, if some values in the interval appear to be more probable than others, it is natural to extend the representation to a fuzzy interval or fuzzy number [13].

In this work, following [10], triangular fuzzy numbers (TFNs) are employed to represent uncertain durations in a JSP. By this representation, only the *sum* and *maximum* operations are needed to calculate the *completion time of each task* in a job. Then, the *completion time of each job* is computed via a semi-active schedule builder [12]. This job completion time is also represented by a TFN. When completion times of all jobs have been determined, the *overall completion time of the JSP* (aka makespan) is taken as the maximum completion time over all the jobs. By taking the *expected value* of the overall completion time as an objective function, genetic and memetic algorithms have been successfully applied to search for a (near-) optimal schedule to the problem [11,14,24].

Being a single number, an expected value cannot fully represent the overall completion time that is expressed by a triangular fuzzy number. Consequently, using the expected value as an objective function implies an approximation of the problem to be solved. Such approximation might, to some extent, result in the loss of information; and therefore the obtained schedule might always be different from the true optimal one in some situations. To overcome this conspicuous drawback of the current techniques, we have investigated a new approach based on a multi-objective genetic algorithm.

This new approach enables us to take into account all three vertices of a TFN representing the overall completion time in the objective function. As a consequence, this approach considers at the same time three different scheduling problems corresponding to the lowest, most probable, and largest durations. With the use of a multi-objective genetic algorithm, the proposed approach tries to obtain a schedule that optimizes the three component scheduling problems all at the same time. In order to verify the quality of solutions found by the proposed approach, an experimental study has been carried out across different benchmark instances. Then a new proposal on analyzing the tolerance for the imprecision of knowledge representation is presented.

2 Job-Shop Scheduling with Uncertain Durations

2.1 Crisp Job-Shop Scheduling Problems

A general JSP [16] can be defined as scheduling a set of n jobs $\{J_1, J_2, \cdots, J_n\}$ on a set of m physical resources or machines $\{M_1, M_2, \cdots, M_m\}$, subject to a set

of constraints. For a job J_i to be completed, a series of tasks have to be done in a predefined order. Those tasks are enumerated by an index j, and, obviously, j is at most equal to m. The job J_i is then said to be composed of tasks θ_{ij}'s, where $j = 1, 2, \cdots, m$. It is noteworthy that a task denoted by θ_{ij} does not imply that it will be processed on the j-th machine. The index j is used only for task enumeration of the job J_i.

In such a general JSP, the predefined orders of tasks form *precedence constraints*; that is, m tasks $\{\theta_{i1}, \theta_{i2}, \cdots, \theta_{im}\}$ of the job J_i, where $i = 1, 2, \cdots, n$, have to be sequentially scheduled. Also, there are *capacity constraints*; that is, each task θ_{ij} requires the uninterrupted and exclusive use of one of the machines for its whole processing time. A solution to this problem is a schedule s—which is an allocation of starting times for all the tasks. Such a solution, besides being feasible (i.e. all the constraints hold), has to be optimal according to some criteria, for instance, the makespan is minimal [10].

Without loss of generality, let us now consider a JSP instance of size $n \times m$ (i.e. n jobs and m machines), let $\mathbf{p}$ be a duration (aka processing time) matrix and $\mathbf{v}$ be a machine matrix such that p_{ij} is the processing time of task θ_{ij} and v_{ij} is the machine required by θ_{ij}, where $i = 1, 2, \cdots, n$ and $j = 1, 2, \cdots, m$. Let σ be a feasible *task processing order*, i.e. a lineal ordering of tasks which is compatible with a processing order of tasks that may be carried out such that all constraints hold. A feasible schedule s may be derived from σ using a semi-active schedule builder [17,12]. Let $S_{ij}(\sigma, \mathbf{p}, \mathbf{v})$ and $C_{ij}(\sigma, \mathbf{p}, \mathbf{v})$ denote the starting and completion times of the task θ_{ij}. According to the semi-active schedule builder, the starting and completion times can be computed as follows:

$$S_{ij}(\sigma, \mathbf{p}, \mathbf{v}) \;=\; C_{i(j-1)}(\sigma, \mathbf{p}, \mathbf{v}) \;\vee\; C_{rs}(\sigma, \mathbf{p}, \mathbf{v}), \tag{1}$$

$$C_{ij}(\sigma, \mathbf{p}, \mathbf{v}) \;=\; S_{ij}(\sigma, \mathbf{p}, \mathbf{v}) \;+\; p_{ij}, \tag{2}$$

where θ_{rs} is the task preceding θ_{ij} in the machine according to the processing order σ, $C_{i0}(\sigma, \mathbf{p}, \mathbf{v})$ is assumed to be zero and, analogously, $C_{rs}(\sigma, \mathbf{p}, \mathbf{v})$ is taken to be zero if θ_{ij} is the first task to be processed in the corresponding machine. The completion time of job J_i will then be $C_i(\sigma, \mathbf{p}, \mathbf{v}) = C_{im}(\sigma, \mathbf{p}, \mathbf{v})$, and the makespan $C_{\max}(\sigma, \mathbf{p}, \mathbf{v})$ is the maximum completion time of any job under a given candidate schedule σ:

$$C_{\max}(\sigma, \mathbf{p}, \mathbf{v}) \;=\; \vee_{1 \le i \le n} \big[C_i(\sigma, \mathbf{p}, \mathbf{v})\big]. \tag{3}$$

For the sake of notation simplicity, we follow [10] to write $C_{\max}(\sigma)$ when the problem (and thus $\mathbf{p}$ and $\mathbf{v}$) is fixed or even $C_{\max}$ when no confusion is possible.

2.2 Modeling Uncertain Durations with TFNs

In real-life applications, it is often the case that the exact duration of a task, i.e. the time it takes to be processed, is not known in advance. However, based on previous experience, an expert may have some knowledge, usually uncertain, about the duration. The most straightforward representation for uncertain

durations would be a human-originated confidence interval. If some values appear to be more plausible than others, a natural extension is a fuzzy interval or fuzzy number [14]. The simplest model for this case is a triangular fuzzy number (TFN) [19], which use an interval $\left[p^1, p^3\right]$ of possible values and a modal value p^2 in between. For a TFN A, denoted by $A = (p^1, p^2, p^3)$, the membership function takes the following triangular shape:

$$\mu_A(x) = \begin{cases} 0\,, & x < p^1 \\ \frac{x-p^1}{p^2-p^1}\,, & p^1 \leq x \leq p^2 \\ \frac{x-p^3}{p^2-p^3}\,, & p^2 < x \leq p^3 \\ 0\,, & x > p^3. \end{cases} \tag{4}$$

In the JSP, we essentially need two operations on fuzzy quantities: the *sum* and the *maximum*. These operations are obtained by extending the corresponding operations on real numbers using the so-called *Extension Principle*. However, computing the resulting expression is cumbersome, if not intractable [10].

For the sake of simplicity and tractability of numerical calculations, we follow [7] to *approximate* the results of these operations by a TFN. In other words, we evaluate the sum and maximum operations on only three defining points of each TFN. The approximated sum coincides with the sum of TFNs as defined by the Extension Principle; thus, for any pair of TFNs M and N, if $S = M + N$ denotes their sum, we have:

$$S \;=\; \left(m^1 + n^1,\; m^2 + n^2,\; m^3 + n^3\right). \tag{5}$$

Unfortunately, for the maximum of TFNs, there is no such simplified expression. For any two TFNs M and N, let $F = N \vee M$ denote their [true] maximum, and $G = (m^1 \vee n^1,\; m^2 \vee n^2,\; m^3 \vee n^3)$ its *approximated value*, an illustration of the distinction between the maximum and its approximation is given in Fig. 1. It is interesting to note that such an approximated maximum can trivially be extended to the case of more than two TFNs [13].

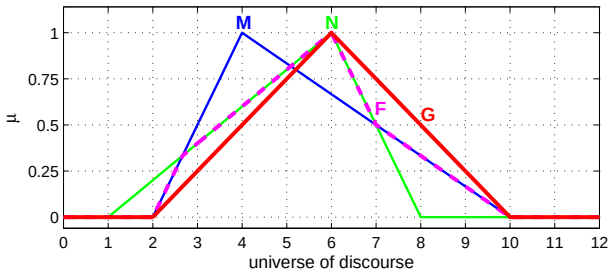

Fig. 1. Exact (magenta) and approximated (red) maximum operations.

Besides being simple, it is clear that this approximation possesses another nice property of preserving the support and modal value of the true maximum. It is however remarkable that, at all α-cuts, the lower and upper bounds of

the approximated maximum are either smaller than or equal to the lower and upper bounds of the true maximum, respectively. More formally, let $[\underline{f}_\alpha, \overline{f}_\alpha]$ and $[\underline{g}_\alpha, \overline{g}_\alpha]$ be the α-cuts of F and G, respectively, it holds that:

$$\forall \alpha \in [0,1], \;\; \underline{f}_\alpha \leq \underline{g}_\alpha \;\text{ and }\; \overline{f}_\alpha \leq \overline{g}_\alpha \,. \tag{6}$$

In possibility theory, the membership function μ_Q of a fuzzy quantity Q can be interpreted as a possibility distribution on real numbers; and this allows us to define the *expected value* of a fuzzy quantity. For a given TFN $A = (p^1, p^2, p^3)$, a typical model [20] for defining its expected value $\mathrm{E}[A]$ is given by:

$$\mathrm{E}\,[A] = \frac{1}{4}\left(p^1 + 2p^2 + p^3\right). \tag{7}$$

Importantly, the expected value coincides with the *neutral scalar substitute* of a fuzzy interval [28]. The neutral scalar substitute is among the most natural defuzzification procedures proposed in the literature [3]. The expected value can also be obtained as the center of gravity of its *mean value*, or using the *area compensation* method proposed by Fortemps and Roubens [8]. Most importantly, it induces a *total ordering* $\leq_{\mathrm{E}}$ in the set of fuzzy intervals [3,7]. For any two fuzzy intervals M and N, $M \leq_{\mathrm{E}} N$ if and only if $\mathrm{E}[M] \leq \mathrm{E}[N]$. Obviously, for any two TFNs $A = (a^1, a^2, a^3)$ and $B = (b^1, b^2, b^3)$, if $a^i \leq b^i \;\forall i$, then $A \leq_{\mathrm{E}} B$; the reverse, however, does not hold.

2.3 Fuzzy Job-Shop Scheduling Problems

The fuzzy JSP considered in this study is the JSP with uncertain processing times (durations). Since processing times of operations are fuzzy intervals, the sum and maximum operations used to propagate constraints (in Eqs. 1 and 2) are taken to be the corresponding operations on fuzzy intervals, and approximated for the particular case of TFNs as explained in Sect. 2.2. The obtained schedule will be a fuzzy schedule in the sense that the starting and completion times of all tasks as well as the makespan are all fuzzy intervals. However, the task processing ordering σ that determines the schedule s is crisp—there is no uncertainty regarding the order in which the tasks have to be processed.

In order to demonstrate the graphical representation of a fuzzy JSP and a particular schedule for one of its instances, let's consider an instance with 3 jobs and 3 machines, having the following machine allocation and *fuzzy* processing time matrices:

$$\mathbf{p} = \begin{pmatrix} (3,\,4,\,6) & (2,\,3,\,4) & (1,\,2,\,5) \\ (1,\,2,\,4) & (2,\,3,\,4) & (1,\,2,\,3) \\ (2,\,3,\,5) & (2,\,3,\,4) & (1,\,2,\,4) \end{pmatrix}, \qquad \mathbf{v} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \end{pmatrix}. \tag{8}$$

For a task processing order $\sigma = (\theta_{31},\, \theta_{11},\, \theta_{32},\, \theta_{12},\, \theta_{21},\, \theta_{33},\, \theta_{13},\, \theta_{22},\, \theta_{23})$, we have the corresponding Gantt chart as shown in Fig. 2. This Gantt chart uses each particular color for all tasks that belong to each particular job; tasks associated with different jobs are accordingly colored differently.

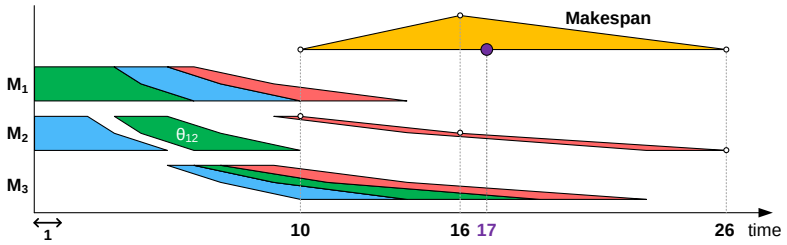

Fig. 2. A sample Gantt chart for a fuzzy JSP instance.

Since we could build a feasible schedule s from a feasible task processing order σ, we would therefore restate the goal of the fuzzy job-shop problem as finding an *optimal* task processing order σ, in the sense that the makespan for the schedule derived from that task processing order is *minimal*.

In [10], the authors employed a single-objective genetic algorithm that is enhanced by a local search to optimize the expected value of the makespan. However, the expected value does not account for the width of a makespan. For instance, the expected value cannot distinguish between two fuzzy makespans $A = (2, 6, 10)$ and $B = (4, 6, 8)$, since both have the same expected value 6, but one would think $B = (4, 6, 8)$ is better because it is less uncertain and thus it provides more accurate information on the possible values of the makespan.

On the other hand, the approximated maximum operator has identical support and modal value to its exact version. As a consequence, the induced fuzzy makespan also has identical support and modal value to the exact fuzzy makespan. Thanks to this nice property, we could try to take into consideration the three *exact* vertices of a fuzzy makespan—instead of its *approximated* expected value—in searching for an optimal schedule to the JSP. Such an idea naturally calls for the use of a multiobjective optimization algorithm. As more information from the exact fuzzy makespan is considered, the schedule obtained under this perspective is expected to be more reliable and robust than those returned when considering only the modal value or the expected value of the makespan.

3 An Evolutionary Approach to the Fuzzy JSPs

In order to apply a genetic algorithm to solving a combinatorial problem in general and a JSP in particular, we need to define a chromosome encoding strategy. Among different proposals available in the literature, the two most popular encoding schemes are the conventional permutations (CP) [2] and permutations with repetition (PR) [1]. In both cases, a chromosome expresses a total ordering of all operations of the problem [26]. For example, if we have a problem with $n = 3$ jobs and $m = 4$ machines, one possible ordering is given by the permutation (θ_{21}, θ_{11}, θ_{12}, θ_{31}, θ_{32}, θ_{22}, θ_{33}, θ_{13}, θ_{23}, θ_{24}, θ_{14}, θ_{34}), where θ_{ij} represents the task j of the job i (where $i = 1, 2, 3$; and $j = 1, 2, 3, 4$). In the CP scheme, operations are codified by the numbers $1, 2, \cdots, n \times m$, starting from the first job, so that the previous ordering would be codified by the chromo-

some (5, 1, 2, 9, 10, 6, 11, 3, 7, 8, 4, 12). Whereas in the PR encoding scheme, an operation is codified just by its job number; hence the previous order would be given by (2, 1, 1, 3, 3, 2, 3, 1, 2, 2, 1, 3) [26]. Also in [26], the authors have demonstrated that PR scheme is better than the CP. Taking that result, we will use the PR encoding scheme in this work.

After initialized randomly, each chromosome undergoes the crossover and mutation stages. New chromosome will then be created and selected to the next generation based on their fitness values—which are the expected makespans of the schedule they encode—by the well-known binary tournament selection strategy. To create new offspring, the job order crossover (JOX) [1,12] and a simple swap mutation are employed. Specifically, given any two parents, the JOX selects a random subset of jobs from the first parent and copies the associated genes to the offspring at the same positions as they appear in that parent. Then, the remaining genes are taken from the second parent such that they maintain their relative ordering. The second offspring is produced in the same manner but considering the second parent first. Following the JOX, the simple swap mutation operator randomly selects and swaps two consecutive genes that encode two different jobs (i.e. having different values). For the single-objective GA, the generational-with-elitism survivor selection is employed; that is, the whole offspring population replaces the parent population except for the best chromosome in the parent population being copied directly to the offspring population.

The feasible search space for a JSP is usually very large; we therefore need to enrich GAs with some advanced algorithm that can somehow limit the feasible space to a narrower one but still guarantee the existence of optimal schedules. The algorithm proposed by Giffler and Thompson [9]—which is commonly referred to as the G&T algorithm—is the best known algorithm for that aim. In fact, this algorithm can be regarded as a transfer function that transforms a candidate schedule to a very similar yet better one in terms of makespan. In this sense, different candidate schedules might be transformed to a unique better schedule by this algorithm. In this work, for both single- and multi-objective GAs, we use the extended G&T algorithm for the fuzzy JSP as proposed in [13].

Our main aim in this work is to optimize simultaneously the three vertices of a makespan. Thus, we need to employ a technique in evolutionary multiobjective optimization (EMO) to handle these three objectives all at the same time, and to evolve a population of candidate solutions over generations in such a way that they get gradual improvements in all objectives. With the framework presented above for a single-objective GA, it is straightforward to extend the algorithm to a multiobjective version by the application of one of the existing EMO techniques. Among various EMO algorithms, the non-dominated sorting genetic algorithm (NSGA-II) [25,5,6] has gained a lot of popularity in the last few years, and becomes a landmark against which other EMO algorithms are often compared. In this work, we will therefore employed NSGA-II as a multiobjective optimizer to address the fuzzy JSP. The job order crossover and simple swap mutation described above will replace the simulated binary crossover and polynomial mutation in the original design of NSGA-II [6].

4 Analyzing the Tolerance for Knowledge Representation

A set of experiments has been conducted to examine the ability of the proposed approach to tolerate the imprecision inherent in representing the expert knowledge about task processing times by TFNs. This imprecision is due to the fact that the expert might not be completely sure about his specification of the fuzzy numbers. In other words, the specification of a fuzzy number is fuzzy itself. The location of the modal value as well as the support of a TFN is naturally imprecise. Accordingly, for a certain task, the processing time might be specified by various fuzzy numbers that are slightly different from each other. Another context can also be the case, that is, when more-than-one experts are jointly specifying the processing times of tasks. Obviously, they may have different knowledge about the tasks, and therefore the TFNs specified by each of them might be different from those specified by the others.

In the experiments, we have selected three typical benchmark instances from a famous library of 40 instances proposed by Lawrence [18]. On the other hand, three genetic algorithms that share common components have been implemented to enable a fairer comparative analysis. These algorithms are: (1) GAcrisp is the single-objective genetic algorithm considering only the most probable processing time of tasks, which is equivalent to a crisp JSP; (2) GAfuzzy is the single-objective genetic algorithm considering the expected value of the fuzzy makespan as the objective function; and (3) NSGAII is the multi-objective genetic algorithm (i.e. NSGA-II) considering at the same time the three vertices of a fuzzy makespan as its objectives. In all algorithms, we have used the binary tournament selection, the JOX with a crossover rate of 0.85, and the simple swap mutation with a mutation rate of 0.1; a randomly initialized population with 100 chromosomes will evolve across 100 generations.

The three benchmark instances that have been used are LA04 (10×5), LA09 (15×5), and LA18 (10×10). For each benchmark, we randomly sample 10 fuzzy instances in the following manner. The modal value (p^2) for each fuzzy processing time (p_{ij}) is sampled uniformly at random in an interval having the crisp processing time as its center (p), with the lower and upper bounds being 95%p and 105%p. Then, the left (p^1) and right (p^3) extremes of that fuzzy processing time is uniformly sampled at random in the intervals [50%p, 95%p] and [105%p, 150%p], respectively. An example to illustrate this procedure is given in Fig. 3. In this illustration, the red triangle is a randomly sampled fuzzy processing time, which has the three vertices being sampled in the blue, violet, and green intervals.

For each benchmark, each of the ten sampled fuzzy instances is tested on the three algorithms, repeated 10 times with different initial populations in each repetition. With 3 algorithms tested on 3 benchmarks having 10 sampled fuzzy instances for each, and repeated 10 runs, we have done $3\times3\times10\times10 = 900$ runs.

For GAcrisp and GAfuzzy, we obtain a single best solution in each of the runs. This is however not the case for NSGAII; NSGAII returns a set of non-dominated solutions in the last generation. To facilitate the comparison with GAcrisp and GAfuzzy, a single *best* solution from the obtained Pareto-optimal set must be

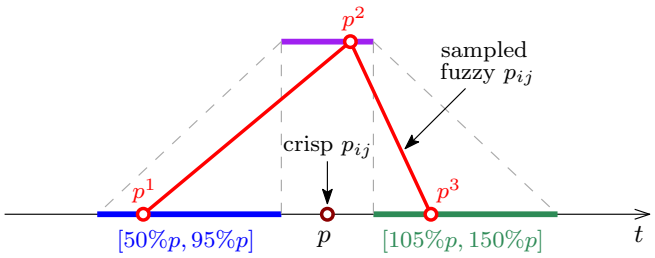

Fig. 3. Sampling fuzzy processing time.

nominated. For that purpose, we calculate the expected value of each solution in the Pareto-optimal set and select the one with the best (minimal) expected value of the makespan. In this way, we could reach a single best solution in each run of all the algorithms, and the results from the 10 runs can be summarized by box plots as presented in Fig. 4 for LA04. In this figure, each box contains 10 results from the 10 runs of a corresponding algorithm on a single sampled fuzzy instance. Also, the three algorithms referred to as `Crisp`, `Fuzz`, and `NSGAII` are the GAcrisp, GAfuzzy, and NSGAII, respectively. In addition, the ten sampled fuzzy instances are enumerated as `Fuzz Samp 1` to `Fuzz Samp 10`.

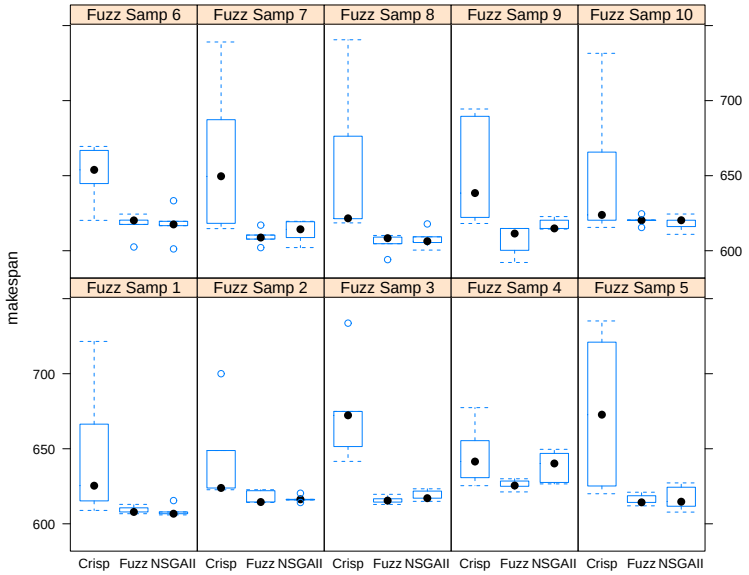

Fig. 4. Results on asymmetrically sampled fuzzy versions of LA04 (10×5).

Taking a look at Fig. 4, it is not difficult to realize that `Crisp`—which is the GA working with only the most probable task processing time in the sampled fuzzy instances—has makespans varying from instance to instance; whereas, the other two algorithms, i.e. `Fuzz` and `NSGAII`, are less sensitive to the random sampling. In addition, their expected makespans are much better (lower) than those of the `Crisp` on average. It should be noticed that, due to the space

limitation, only the boxplot for LA04 is presented here; the similar plots for LA09 and LA18 also exhibit completely the same trend.

The comparison between `Fuzz` (i.e. the GA working with the expected fuzzy makespan) and `NSGAII` (i.e. the NSGA-II working with the three vertices of the fuzzy makespan) is however not intuitive. In fact, their results are close together for LA04 and LA18; and for LA09, their results look identical. (Notice again that the results for LA09 and LA18 are not shown here). Besides, the boxes for `Crisp` is larger than those for the other algorithms, which would suggest that under the current experimental setting of the algorithms (100 chromosomes evolved over 100 generations), `Fuzz` and `NSGAII` converge better than `Crisp`. Consequently, their results from different runs do not vary so much as those of the `Crisp`. Nonetheless, we have initialized the three algorithms by the same random seeds and they also share the same common structural components of the GA. In such a context, a more plausible explanation could be that, taking into consideration the triangular fuzzy processing time of tasks instead of only the most probable one, we could always gain benefits regardless of whether the expected value or all the three vertices of a makespan is utilized as the objective function(s).

What still remains interesting to know is how these algorithms perform on average in terms of the mean and variation of makespans over all the sampled fuzzy instances. To answer this question, we first take the mean (or median, alternatively) of each box, i.e. we are averaging the 10 runs. Then we calculate the mean and standard deviation of these ten means (or ten medians, alternatively). In other words, we are averaging the results of the 10 sampled fuzzy instances. As we have run each test only 10 times, the use of median could be a better estimate of the actual performance of the algorithms. Respective results from these calculations are fully presented in Table 1.

Table 1. Mean and standard deviation (SD) of approximated makespans of all fuzzy samples. For each fuzzy sample, the approximated makespan is either the mean or median makespan over all 10 runs of an algorithm. Numbers in brackets are SD.

Instance	**Average over runs**	**Algorithm**		
		GAcrisp	GAfuzzy	NSGAII
LA04	mean	655.9 (11.14)	614.3 (6.56)	617.4 (8.34)
	median	642.3 (19.49)	614.7 (5.90)	616.8 (9.34)
LA09	mean	980.9 (11.04)	952.8 (7.51)	952.8 (7.51)
	median	975.1 (14.23)	952.8 (7.51)	952.8 (7.51)
LA18	mean	958.8 (13.30)	886.7 (12.09)	894.2 (8.16)
	median	956.3 (21.28)	884.5 (14.30)	891.6 (9.74)

The results shown in Table 1 suggest that NSGA-II has a promising ability to tolerate the imprecision in representing the expert's knowledge about the fuzzy processing time. Under different randomly sampled fuzzy processing time, the final results of NSGA-II clearly exhibit less variation than those of GAcrisp

on all the benchmarks. However, more extensive experiments are advocated in order to draw a further conclusion about whether NSGA-II is better than the single-objective GA using the expected value of the makespan as its objective function. In fact, NSGAII has the same variation as GAfuzzy on the benchmark LA09, and a better (smaller) variation on LA18, but a worse (larger) variation on LA04, in comparison with GAfuzzy.

5 Conclusions

In this work, we have investigated the application of a multi-objective genetic algorithm — the NSGA-II — to solving JSPs with uncertain durations, where uncertainty is modeled by triangular fuzzy numbers. The novelty of the investigation is that, we have considered the three vertices of a triangular fuzzy makespan all at the same time as three objectives of NSGA-II, rather than just using a representative which is the expected value as in previous work.Such a new proposal is often preferable to the existing approach in terms of offering the decision maker more options in selecting a scheduling strategy according to his preference to the earliest, most probable, or latest completion time. To validate the proposed multi-objective approach, a set of experiments has been performed. Even though the simulation results on a limited number of benchmarks do not strongly demonstrate the superiority of the proposal, they have provided some evidence for the imprecision tolerance ability of the obtained schedules with respect to the knowledge representation of the experts. The results have also suggested that a more comprehensive validation on a larger set of benchmark instances as well as a more extensive simulation would bring about a clear insight into the difference between the proposal and other available approaches.

References

1. Bierwirth, C.: A generalized permutation approach to job shop scheduling with genetic algorithms. OR Spektrum 17(2-3), 87–92 (Sep 1995)
2. Bierwirth, C., Mattfeld, D.C.: Production scheduling and rescheduling with genetic algorithms. Evolutionary Computation 7(1), 1–17 (1999)
3. Bortolan, G., Degani, R.: A review of some methods for ranking fuzzy subsets. Fuzzy Sets and Systems 15(1), 1–19 (1985)
4. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms—I. representation. Computers & Industrial Engineering 30(4), 983–997 (1996)
5. Deb, K., Agarwal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Parallel Problem Solving from Nature PPSN VI, pp. 849–858 (2000)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
7. Fortemps, P.: Jobshop scheduling with imprecise durations: a fuzzy approach. IEEE Transactions on Fuzzy Systems 5(4), 557–569 (1997)

8. Fortemps, P., Roubens, M.: Ranking and defuzzification methods based on area compensation. Fuzzy Sets and Systems 82(3), 319–330 (1996)
9. Giffler, B., Thompson, G.L.: Algorithms for solving production-scheduling problems. Operations Research 8(4), 487–503 (1960)
10. Gonzalez-Rodriguez, I., Puente, J., Vela, C., Varela, R.: Semantics of schedules for the fuzzy job-shop problem. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 38(3), 655–666 (2008)
11. Gonzalez-Rodriguez, I., Vela, C., Puente, J.: A memetic approach to fuzzy job shop based on expectation model. In: IEEE Int. Conf. on Fuzzy Systems. pp. 1–6 (2007)
12. González, M., Vela, C., Varela, R.: Scheduling with memetic algorithms over the spaces of semi-active and active schedules. Lecture Notes in Computer Science, vol. 4029, pp. 370–379. Springer Berlin / Heidelberg (2006)
13. González-Rodríguez, I., Vela, C., Puente, J.: A genetic solution based on lexicographical goal programming for a multiobjective job shop with uncertainty. Journal of Intelligent Manufacturing 21(1), 65–73 (2010)
14. González-Rodríguez, I., Vela, C.R., Puente, J., Hernández-Arauzo, A.: Improved local search for job shop scheduling with uncertain durations. In: Nineteenth Int. Conf. on Automated Planning and Scheduling (ICAPS-2009). pp. 154–161 (2009)
15. Gonçalves, J.F., de Magalhães Mendes, J.J., Resende, M.G.C.: A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167(1), 77–95 (2005)
16. Jain, A.S., Meeran, S.: Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research 113(2), 390–434 (1999)
17. Jensen, M.T.: Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. Applied Soft Computing 1(1), 35–52 (2001)
18. Lawrence, S.: Supplement to "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques". Tech. rep., GSIA, Carnegie Mellon University, Pittsburgh PA (1984)
19. Lin, F.T., Yao, J.S.: Using fuzzy numbers in knapsack problems. European Journal of Operational Research 135(1), 158–176 (2001)
20. Liu, B., Liu, Y.K.: Expected value of fuzzy variable and fuzzy expected value models. IEEE Transactions on Fuzzy Systems 10(4), 445–450 (2002)
21. Nakano, R., Yamada, T.: Conventional genetic algorithm for job shop problems. In: Proceedings of ICGA. pp. 474–479 (1991)
22. Park, B.J., Choi, H.R., Kim, H.S.: A hybrid genetic algorithm for the job shop scheduling problems. Computers & Industrial Engineering 45(4), 597–613 (2003)
23. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer, 3rd edn. (2008)
24. Puente, J., Vela, C.R., González-Rodríguez, I.: Fast local search for fuzzy job shop scheduling. In: Proceedings of ECAI 2010. pp. 739–744. IOS Press (2010)
25. Srinivas, N., Deb, K.: Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation 2(3), 221–248 (1994)
26. Varela, R., Serrano, D., Sierra, M.: New codification schemas for scheduling with genetic algorithms. In: Mira, J., Álvarez, J. (eds.) Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Lecture Notes in Computer Science, vol. 3562, pp. 15–20. Springer Berlin / Heidelberg (2005)
27. Vázquez, M., Whitley, D.: A comparison of genetic algorithms for the static job shop scheduling problem. In: Parallel Problem Solving from Nature PPSN VI, LNCS, vol. 1917, pp. 303–312. Springer Berlin / Heidelberg (2000)
28. Yager, R.R.: A procedure for ordering fuzzy subsets of the unit interval. Information Sciences 24(2), 143–161 (1981)