A memetic algorithm for due-date satisfaction in fuzzy job shop scheduling

Juan José Palacios[1], Inés González-Rodríguez[2], Camino R. Vela[1], and Jorge Puente[1]

[1] Department of Computer Science,
University of Oviedo, (Spain) {palaciosjuan,puente,crvela}@uniovi.es,
http://di002.edv.uniovi.es/iscop
[2] Department of Mathematics, Statistics and Computing,
University of Cantabria, (Spain) ines.gonzalez@unican.es

Abstract. We consider the job shop scheduling problem with with fuzzy sets modelling uncertain durations and flexible due dates. With the goal of maximising due-date satisfaction, we propose a memetic algorithm that combines intensification and diversification by integrating local search in a genetic algorithm. Experimental results illustrate the synergy between both components of the algorithm as well as its potential to provide good solutions.

1 Introduction

Traditionally, it has been assumed that scheduling problems are static and certain: all activities and their durations are precisely known in advance and do not change as the solution is being executed. However, for many real-world scheduling problems design variables are subject to perturbations or changes, causing optimal solutions to the original problem to be of little or no use in practice. It is also common to handle all constraints as sharp, while in some cases there is certain flexibility and some constraints are better expressed in terms of preference, so it is possible to satisfy them to a certain degree.

A source of changes in scheduling problems is uncertainty in activity durations. Within the great diversity of approaches dealing with this kind of uncertainty, fuzzy sets and possibility theory provide an interesting framework, with a tradeoff between the expressive power of probability and its associated computational complexity and knowledge demands. Additionally, fuzzy sets can be used to model flexibility or gradeness in certain management constraints such as due dates [4].

The variant of job shop scheduling problem with fuzzy durations and, optionally, fuzzy due dates, is called fuzzy job shop [1]. Most contributions in the literature concentrate on minimising the project's makespan, but some authors have tackled the problem of maximising due-date satisfaction, either on its own or in a multiobjective setting, combined with makespan.

In this paper, we intend to advance in the study of the fuzzy job shop scheduling problem, and in particular, in a metaheuristic method to maximise due-date satisfaction when uncertain task durations and flexible due dates are fuzzy sets.

2 The Fuzzy Job Shop Problem

The classical *job shop scheduling problem*, also denoted *JSP*, consists in scheduling a set of jobs $\{J_1, \ldots, J_n\}$ on a set $\{M_1, \ldots, M_m\}$ of physical resources or machines, subject to a set of constraints. There are *precedence constraints*, so each job J_i, $i = 1, \ldots, n$, consists of m tasks $\{\theta_{i1}, \ldots, \theta_{im}\}$ to be sequentially scheduled. Also, there are *capacity constraints*, whereby each task θ_{ij} requires the uninterrupted and exclusive use of one of the machines for its whole processing time. Additionally, each job J_i may have a due date d_i by which it is desirable that the job be completed. A solution to this problem is a schedule, 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, in our case, maximal due-date satisfaction.

2.1 Fuzzy Durations and Flexible Due Dates

In real-life applications, it is difficult, if not impossible, to foresee in advance the exact time it will take to process a task. It is reasonable however to have some knowledge (albeit uncertain) about the duration, possibly based on previous experience. The crudest representation of such uncertain knowledge would be a human-originated confidence interval and, if some values appear to be more plausible than others, then a natural extension is a fuzzy interval or fuzzy number. The simplest model is a *triangular fuzzy number* or *TFN*, denoted $\widehat{a} = (a^1, a^2, a^3)$, given by an interval $[a^1, a^3]$ of possible values and a modal value $a^2 \in [a^1, a^3]$, so its membership function takes the following triangular shape:

$$\mu_{\widehat{a}}(x) = \begin{cases} \frac{x-a^1}{a^2-a^1} & : a^1 \leq x \leq a^2 \\ \frac{x-a^3}{a^2-a^3} & : a^2 < x \leq a^3 \\ 0 & : x < a^1 \text{ or } a^3 < x \end{cases} \tag{1}$$

Triangular fuzzy numbers (or, more generally, fuzzy intervals) are widely used in scheduling as a model for uncertain processing times [1, 4, 12].

In the job shop, we essentially need two operations on fuzzy numbers, the sum and the maximum. These are usually defined by extending the corresponding operations on real numbers. The resulting addition is pretty straightforward, so for any pair of TFNs $\widehat{a}$ and $\widehat{b}$ we have $\widehat{a} + \widehat{b} = (a^1 + b^1, a^2 + b^2, a^3 + b^3)$. Unfortunately, computing the extended maximum is not that simple and the set of TFNs is not even closed under this operation. Hence, it is common in the fuzzy scheduling literature to approximate the maximum of two TFNs as $\max(\widehat{a}, \widehat{b}) \approx (\max\{a^1, b^1\}, \max\{a^2, b^2\}, \max\{a^3, b^3\})$. Besides its extended use, several arguments can be given in favour of this approximation (cf. [12]).

Fuzzy sets can also be used to model flexible due dates. Consider the case where there is a preferred delivery date d^1, but some delay may be allowed until a later date d^2. Satisfying the due date constraint thus becomes a matter of degree, our degree of satisfaction that the job is finished on a certain date. A

fuzzy set $\widetilde{d} = (d^1, d^2)$ can be used to model such gradual satisfaction level with a decreasing membership function:

$$\mu_{\widetilde{d}}(x) = \begin{cases} 1 & : x \leq d^1 \\ \frac{x-d^2}{d^1-d^2} & : d^1 < x \leq d^2 \\ 0 & : d^2 < x \end{cases} \tag{2}$$

This expresses a flexible threshold "less than", representing the satisfaction level $sat(t) = \mu_{\widetilde{d}}(t)$ for the ending date t of the job [4].

When the job's completion time is no longer a real number t but a TFN $\widehat{c}$, the degree to which $\widehat{c}$ satisfies the due-date constraint $\widetilde{d}$ may be measured using the *agreement index* [15]:

$$AI(\widehat{c}, \widetilde{d}) = \frac{area(\widetilde{d} \cap \widehat{c})}{area(\widehat{c})} \tag{3}$$

where $area(\widetilde{d} \cap \widehat{c})$ and $area(\widehat{c})$ denote the areas under the membership functions of $(\widetilde{d} \cap \widehat{c})$ and $\widehat{c}$ respectively. This essentially measures the degree to which $\widehat{c}$ is contained in $\widetilde{d}$ following the standard definition of degree of subsethood. $AI(\widehat{c}, \widetilde{d})$ ranges between 0, when the due date is not satisfied at all, and 1, when the due date is fully satisfied.

2.2 Fuzzy Schedules

To determine a solution for a fuzzy JSP, it is necessary to establish partial task processing orders on all machines. These can be represented by a linear processing order π. A schedule (starting and completion times of all tasks) may be easily computed based on π. For every task x with processing time $\widehat{p}_x$, let $PM_x(\pi)$ and $SM_x(\pi)$ denote the tasks preceding and succeeding x in the machine sequence provided by π, and let PJ_x and SJ_x denote respectively the predecessor and successor tasks of x in the job sequence. Then the starting time $\widehat{s}_x(\pi)$ and completion time $\widehat{c}_x(\pi)$ of x according to π are two TFNs given by:

$$\widehat{s}_x(\pi) = \max(\widehat{s}_{PJ_x} + \widehat{p}_{PJ_x}, \widehat{s}_{PM_x(\pi)} + \widehat{p}_{PM_x(\pi)}), \tag{4}$$

$$\widehat{c}_x(\pi) = \widehat{s}_x(\pi) + \widehat{p}_x(\pi). \tag{5}$$

The completion time of each job J_i, denoted $\widehat{c}_i(\pi)$, is the completion time of the last task in that job. If there is no possible confusion regarding the processing order, we may simplify notation by writing $\widehat{s}_x$, $\widehat{c}_x$ and $\widehat{c}_i$.

The resulting schedule is fuzzy in the sense that the starting and completion times of all tasks are fuzzy intervals, interpreted as possibility distributions on the values that the times may take. However, notice that the task processing ordering π that determines the schedule is deterministic; there is no uncertainty regarding the order in which tasks are to be processed.

Having built a schedule from π, we can now evaluate the degree of satisfaction of due dates. Indeed, the agreement index $AI_i(\widehat{c}_i(\pi), \widetilde{d}_i)$ as defined in

(3), denoted AI_i for short, measures to what degree is each job's flexible due date $\widetilde{d}_i$ satisfied in this schedule, $i = 1, \ldots, n$. The overall value of due-date satisfaction for the schedule is then obtained by aggregating the individual AI_i values for $i = 1, \ldots, n$. Two main approaches for aggregation can be found in the literature: the minimum agreement index $AI_{min} = \min_{i=1,\ldots,n} AI_i$, and the average agreement index $AI_{avg} = \frac{1}{n} \sum_{i=1,\ldots,n} AI_i$. The minimum corresponds to the classical approach of fuzzy decision making, while the average provides an alternative for which the compensation property holds. Both aggregated indices need to be maximised.

The resulting job shop problem, with fuzzy processing times and fuzzy due dates, and where the objective is to maximise the aggregated agreement index AI_{agg} (where AI_{agg} can be AI_{avg} or AI_{min}) can be denoted $J|\widehat{p}_i, \widetilde{d}_i|AI_{agg}$ according to the three-field notation from [7].

3 A Memetic Algorithm to Maximise AI_{agg}

Hybrid algorithms, combining genetic algorithms with local search methods, have proved to be very powerful in different optimisation problems. The reason is their ability to integrate the intensification provided by the local search with the diversification provided by the population-based algorithm. In particular, some state-of-the-art methods for different variants of fuzzy job shop are hybrids of this kind [11, 12]. This motivates our proposal of a memetic algorithm, combining a genetic component with local search.

3.1 Genetic Component

For the genetic component of our algorithm, solutions are codified into chromosomes as permutations with repetitions [2]. Each permutation represents a feasible task processing order π by identifying each operation θ_{ij} with j-th occurrence of index i in the permutation. For example, in a problem with three jobs and three machines, sequence (1,3,2,2,3,1,1,3,2) represents the task ordering $\pi = (\theta_{11}, \theta_{31}, \theta_{21}, \theta_{22}, \theta_{32}, \theta_{12}, \theta_{13}, \theta_{33}, \theta_{23})$. For fitness evaluation, chromosomes are decoded into schedules using an insertion schedule generation scheme as proposed in [13] and the resulting AI_{agg} is taken as fitness value.

The algorithm starts from a random population. It then iterates until $maxIter$ consecutive iterations pass without any improvement in the best solution found so far. At each iteration a new generation is built from the previous one by applying the genetic operators of selection, recombination and replacement. In the selection phase all chromosomes are randomly paired, and then each pair is mated to obtain two offspring by applying crossover and mutation with a certain probability. Two individuals are then selected using tournament from each pair of parents and their two offspring to pass onto the next generation. In order to keep diversity, when possible the replacement strategy selects two individuals with different fitness values. For recombination, two classical operators are used: JOX crossover [9] and insertion mutation.

3.2 Local Search Component

This component follows a typical local search schema: starting from a given solution, at each step it selects a promising element from a neighbourhood structure to replace the current solution, until a stopping criterion is met. In our case, we use a simple hill climbing, where the local search moves to the first neighbour improving the objective value of the current solution. The search stops when it reaches a solution without improving neighbours. This strategy is very fast compared to other local search strategies, making it appealing for large neighbourhoods.

For the deterministic JSP several local search methods have been proposed where neighbours are generated by selecting (according to some criterion) two tasks that are sequentially scheduled in a machine and changing their relative order. This is equivalent to reversing an arc in a graph G representing a solution. In this graph, nodes correspond to tasks and directed arcs, weighted with the processing time of the task in the origin, represent immediate precedence between the two tasks either in the job or the machine. Another node, representing the start of the project is added and connected with zero weight to the first task in each job. Also, depending on the objective function, there is a single end node to which the last task of each job is (e.g. for the makespan minimisation) or there is an end node per job (this is the case of some objective functions considering due dates) [3].

The same approach is extended for the fuzzy JSP with makespan minimisation in [6]. To select arcs to be reversed, the solution graph G (with fuzzy arc weights) is decomposed in three parallel graphs G^j, $j = 1, 2, 3$, with identical topology but such that arcs are weighted with the j-th component of the processing time of the task corresponding to the source node. This allows to define critical paths in G as those paths from the start to the end that are critical (in the usual deterministic sense) in any of the parallel graphs G^j. Arcs to be reversed are then chosen from the set of arcs in a critical path that correspond to machine precedence. Neighbours thus generated are shown to be feasible solutions. It is also shown that reversing any non-critical arc cannot possibly lead to a solution with shorter critical paths and, hence, better makespan.

Based on these ideas, we propose two different neighbourhood structures, one for each aggregation of agreement indices. In the case of AI_{avg}, for its value to increase in a neighbouring solution it must be the case that at least one of the agreement indices AI_i improves. This implies reducing the completion time $\widehat{c}_i(\pi)$ of that job or, equivalently, reducing the length of the longest path from the start node to the end node of job J_i in the solution graph. Therefore, we consider that a path is critical for job J_i if and only if it is a longest path from the start node to the last node of job J_i in any of the parallel graphs G^j. Notice that there might be more than one critical path for each job. Let CP_i denote the set of critical paths for job J_i, $i = 1, \ldots, n$. An improvement in $\widehat{c}_i(\pi)$ (and hence AI_{avg}) can only be obtained by reversing machine arcs belonging to one of the paths in CP_i, $i = 1, \ldots, n$. Furthermore, since $AI_i \leq 1$ for $i = 1, \ldots, n$, reducing the completion time of a job such that $AI_i = 1$ cannot improve AI_{avg}

either. Therefore, the neighbourhood $\mathcal{N}_{AI_{avg}}$ is obtained by reversing machine arcs that belong to a critical path in the set $\{CP_i : AI_i < 1, 1 \leq i \leq n\}$.

In the case that the objective function is AI_{min}, a smaller neighbourhood structure can be considered. Indeed, for the minimum aggregation, reducing the completion time $\widehat{c}_i(\pi)$ of any job such that $AI_i > AI_{min}$ does not improve the objective function. Therefore we obtain a neighbourhood $\mathcal{N}_{AI_{min}} \subset \mathcal{N}_{AI_{avg}}$ by reversing an arc if and only if that arc is in a critical path in the set $\{CP_i : AI_i = AI_{min} < 1, 1 \leq i \leq n\}$.

In summary, the local search component consists in a simple hill climbing procedure using one of the neighbourhood structures $\mathcal{N}_{AI_{avg}}$ or $\mathcal{N}_{AI_{min}}$, depending on the objective function considered. This results in quite a fast local search procedure which is applied to all the individuals that are evaluated by the genetic algorithm.

4 Experimental Results

To provide an empirical evaluation of the proposed memetic algorithm, called MA hereafter, we perform a series of experiments with a C++ implementation running on a PC with Xeon processor at 2,2Ghz and 24 Gb RAM with Linux (SL 6.0.1). The parameter values (obtained after a parametric analysis not reported here due to lack of space) are population size 100, crossover and mutation probability 1.0 and 0.05 respectively and $maxIter$=25 as stopping criterion. In all cases, results correspond to 30 runs of MA. We evaluate solutions in terms of $1 - AI_{agg}$ for both $AI_{agg} = AI_{min}$ and $AI_{agg} = AI_{avg}$, representing the distance between the obtained overall due date satisfaction (measured with AI_{agg}) and the ideal situation where all due dates are fully satisfied. This ideal value of 1 provides an upper bound for solution performance, but if due dates are too tight it may occur that this upper bound is actually unattainable.

In a first set of experiments, we compare MA with two methods from the literature which, to our knowledge, conform the state-of-the-art. The first method is a genetic algorithm, denoted SMGA, proposed in [15] to optimise AI_{min}. It was tested on two new instances widely used in the literature thereafter (see the review [12]): S6.4 of size 6×6 and S10.4 of 10×10. On these instances, SMGA compared favourably to an alternative simulated annealing method SMSA. A second approach is a random key genetic algorithm (RKGA) from [8], also maximising AI_{min}. Both RKGA and the author's own implementation of SMGA were tested on a total of 10 instances: S6.4 and S10.4 above, 6 more instances from the literature and 2 new ones. We find 3 instances of size 6×6, denoted S6.1-3, and 3 of size 10×10, S10.1-3, originally proposed for a multiobjective approach in [14]. The two new instances of size 15×10, denoted Lei01 and Lei02 [12], are meant to provide more challenging scenarios. It must be noted that the results reported in [8] correspond to a different approximation for the maximum of fuzzy numbers which may lead to smaller completion times (cf. [12]). Figure 1 shows the performance of all three algorithms— SMGA, RKGA and MA using AI_{min} as objective function— on this test bed of 10 instances. The comparison is made

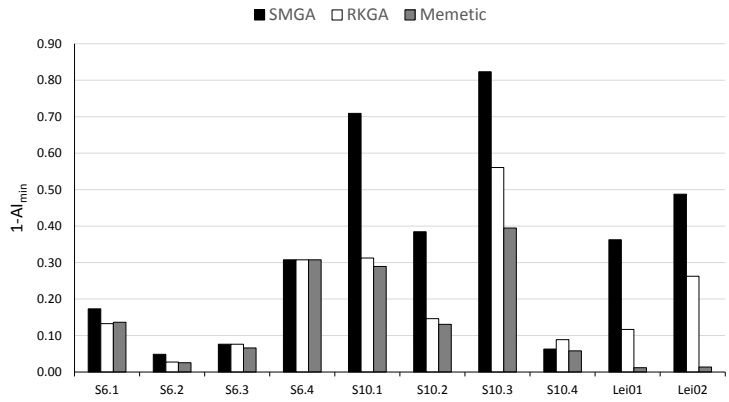

Fig. 1. Comparison with algorithms SMGA and RKGA maximising AI_{min}.

in terms of average values of $1 - AI_{min}$ across all runs of MA and RKGA (30 and 20 respectively) and the best average value for SMGA between the two values reported in [15] and [8].

We can appreciate that differences among methods are almost negligible in small instances (S6). In fact, MA obtains the same AI_{min} value on all runs, which suggests that this value is very close to or is actually the optimum for these instances. For the larger instances S10.1-4 and Lei01-02, we can see that MA yields the best results. While improvement with respect to the other methods is small on instances S10.1 and S10.2, on S10.3 MA is 29.6% better in average values than RKGA and on S10.4 it is 34.6% better than RKGA and 7.9% better than SMGA. Differences grow on the largest instances: for instances Lei01 and Lei02, the results obtained by MA are respectively 89.9% and 94.8% better than RKGA. Moreover, the obtained average difference is close to 0, which shows the potential of MA to solve these instances.

Additionally, we consider another method that maximises AI_{min}, proposed in [5]. In a multiobjective setting, a lexicographic genetic algorithm is proposed to optimise makespan, AI_{min}, and AI_{avg} and tested on five instances, obtained by fuzzifying well-known deterministic instances: FT06 (6×6), La11, La12, La13 and La14 (20×5). The proposed method always obtains full due-date satisfaction on all instances, with $AI_{min} = AI_{avg} = 1$. For the sake of completeness, we have run our method on these instances, first optimising AI_{min} and then AI_{avg}. In both cases, the obtained results reach the optimal value for all instances.

A second set of experiments is conducted on a set of more challenging instances from [10]. These are obtained from 12 well-known benchmark problems for deterministic job shop which are considered hard to solve: FT10 (size 10×10), FT20 (20×5), La21, La24, La25 (15×10), La27, La29 (20×10), La38, La40 (15×15), and ABZ7, ABZ8, ABZ9 (20×15). The deterministic processing times from the original instances have been transformed into symmetric TFNs so the original duration is the modal value, and flexible due dates have been intro-

Instance	$1 - AI_{avg}$ Best	$1 - AI_{avg}$ Avg.	$1 - AI_{min}$ Best	$1 - AI_{min}$ Avg.	Runtime
ABZ7	0.335	0.355 (0.018)	1.000	1.000 (0.000)	132.9
ABZ8	0.312	0.333 (0.012)	1.000	1.000 (0.000)	128.0
ABZ9	0.300	0.325 (0.015)	1.000	1.000 (0.000)	199.1
FT10	0.243	0.246 (0.008)	1.000	1.000 (0.000)	5.4
FT20	0.701	0.714 (0.010)	1.000	1.000 (0.000)	9.3
La21	0.358	0.381 (0.010)	1.000	1.000 (0.000)	23.9
La24	0.334	0.362 (0.015)	1.000	1.000 (0.000)	23.2
La25	0.319	0.342 (0.010)	1.000	1.000 (0.000)	25.7
La27	0.501	0.536 (0.021)	1.000	1.000 (0.000)	68.4
La29	0.457	0.479 (0.018)	1.000	1.000 (0.000)	58.3
La38	0.156	0.167 (0.007)	1.000	1.000 (0.000)	48.8
La40	0.116	0.137 (0.013)	1.000	1.000 (0.000)	56.6

Table 1. Results obtained using AI_{avg} as objective function.

duced. We refer the interested reader to [10] and references therein for further information on the fuzzification process.

The results obtained on each benchmark instance when the objective function is AI_{avg} are summarised in Table 1. After a first column containing the name of the instance, the second and third columns contain the value for the best solution and the average and standard deviation (the latter between brackets) of $1 - AI_{avg}$ across the 30 runs. The fourth and fifth columns are analogous, but measuring overall due-date satisfaction with $AI_{agg} = AI_{min}$. Finally, the last column shows the average runtime in seconds across the 30 runs.

Since MA is run using AI_{avg} as objective value, the most relevant data are those in the second and third columns. We can appreciate that the distance to full due-date satisfaction varies significantly across the different instances, ranging from less than 0.2 in instances La38 and La40 to an extreme value of 0.714 in instance FT20. We believe this is related to the method used in [10] to generate due-date values. For FT20, with many jobs but just a few tasks per job, due dates are very tight and there is little flexibility to schedule the tasks of every job in such a way that due dates are met. On the other hand, for instances like La38 and La40, where both the number of jobs and tasks per job is large, due dates result less rigid so the obtained satisfaction values are much closer to the upper bound.

The fourth and fifth column of Table 1 illustrate how AI_{min} turns out to be too restrictive as aggregation method on this set of instances: even for those solutions with an average agreement index AI_{avg} relatively close the upper bound, at least one of the due dates cannot be satisfied, resulting in $AI_{min} = 0$. This is due to the difficulty of the proposed instances, with very tight due dates that make it very unlikely to find a solution such that $AI_{min} > 0$. Indeed, when running MA on this test bed using AI_{min} as fitness function, the error of the obtained solutions is always very close to 1 (1 in many instances). In fact, the

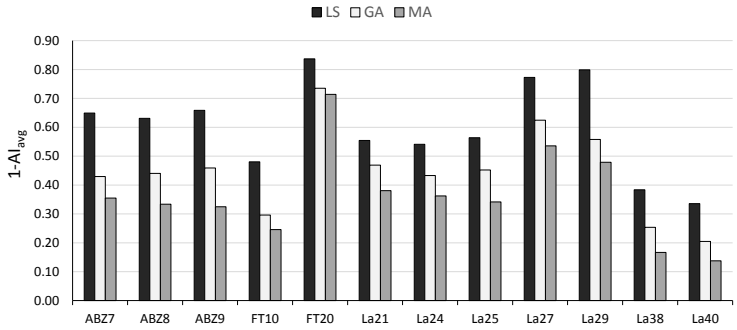

Fig. 2. Performance of the different components of MA

initial population consists of random solutions for which $AI_{min} = 0$ in most cases, so there is no good solution to guide the algorithm to promising areas of the search space. On the other hand, using AI_{avg} as fitness value, the initial population already contains many individuals with fitness greater than 0, allowing MA to converge.

In a final set of experiments, we assess if both components of our memetic algorithm MA are actually contributing to the obtained results. To this end, the genetic component, GA, and the local search, LS, are run independently on the second set of more challenging instances. For a fairer comparison, LS is run as a multi-start local search with as many restarts as the average number of evaluations performed by MA on each instance. Analogously, GA is run with the same setup as MA for as long as the latter takes to converge. Due to the issues we have outlined regarding the optimisation of AI_{min} in the harder set of test instances, we take AI_{avg} as objective function in all cases. The multi-start local search starting from random solutions obtains the worst results, not only in performance, Figure 2, but also in runtime which is 44% larger than MA. On the other hand, GA performs much better than the local search. Still, we can appreciate a synergy effect when combining both strategies, with MA obtaining much better results in the same running time than GA. This shows that MA benefits from the exploration of GA and also from the intensification of LS.

5 Conclusions

We have tackled the job shop scheduling problem with uncertain durations and flexible due dates modelled as fuzzy numbers. We have proposed a memetic algorithm, combining a genetic algorithm with a purpose-built local search. Experimental results compare favourably with the state-of-the-art methods, showing the potential of the proposed method.

Acknowledgements

This research has been supported by the Spanish Government under research grant TIN2016-79190-R.

References

1. Abdullah, S., Abdolrazzagh-Nezhad, M.: Fuzzy job-shop scheduling problems: A review. Information Sciences 278, 380–407 (2014)
2. Bierwirth, C.: A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectrum 17, 87–92 (1995)
3. Błażewicz, J., Domschke, W., Pesch, E.: The job shop scheduling problem: Conventional and new solution techniques. European Journal of Operational Research 93, 1–33 (1996)
4. Dubois, D., Fargier, H., Fortemps, P.: Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. European Journal of Operational Research 147, 231–252 (2003)
5. González Rodríguez, I., Puente, J., Vela, C.R.: A multiobjective approach to fuzzy job shop problem using genetic algorithms. CAEPIA 2007, Lecture Notes in Artificial Intelligence 4788, 80–89 (2007)
6. González Rodríguez, I., Vela, C.R., Puente, J., Varela, R.: A new local search for the job shop problem with uncertain durations. In: Proc. of ICAPS-2008. pp. 124–131. AAAI Press (2008)
7. Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 4, 287–326 (1979)
8. Lei, D.: Solving fuzzy job shop scheduling problems using random key genetic algorithm. International Journal of Advanced Manufacturing Technologies 49, 253–262 (2010)
9. Ono, I., Yamamura, M., Kobayashi, S.: A genetic algorithm for job-shop scheduling problems using job-based order crossover. In: Proc. of IEEE International Conference on Evolutionary Computation, 1996. pp. 547–552. IEEE (1996)
10. Palacios, J.J., Derbel, B.: On maintaining diversity in MOEA/D: Application to a biobjective combinatorial FJSP. In: Proc. of GECCO '15. pp. 719–726. ACM (2015)
11. Palacios, J.J., González, M.A., Vela, C.R., González-Rodríguez, I., Puente, J.: Genetic tabu search for the fuzzy flexible job shop problem. Computers & Operations Research 54, 74–89 (2015)
12. Palacios, J.J., Puente, J., Vela, C.R., González-Rodríguez, I.: Benchmarks for fuzzy job shop problems. Information Sciences 329, 736–752 (2016)
13. Palacios, J.J., Vela, C.R., González-Rodríguez, I., Puente, J.: Schedule generation schemes for job shop problems with fuzziness. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) Proc. of ECAI 2014. pp. 687–692. IOS Press (2014)
14. Sakawa, M., Kubota, R.: Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. European Journal of Operational Research 120, 393–407 (2000)
15. Sakawa, M., Mori, T.: An efficient genetic algorithm for job-shop scheduling problems with fuzzy processing time and fuzzy duedate. Computers & Industrial Engineering 36, 325–341 (1999)