Hybrid Cooperative Coevolution for Fuzzy Flexible Job Shop Scheduling Problems

Juan José Palacios, Camino R. Vela, Jorge Puente
Dep. Computer Science
University of Oviedo (Spain)
{palaciosjuan,crvela,puente}@uniovi.es

Inés González-Rodríguez
University of Cantabria (Spain)
Dep. Maths, Stats and Computing
gonzalezri@unican.es

Abstract. In this paper we consider a variant of the flexible job shop scheduling problem with uncertain task durations modelled as fuzzy numbers. We propose a cooperative coevolutionary algorithm to minimise the schedule's makespan, with two different populations evolving the two main aspects that conform a solution: machine assignment and task relative order. Additionally, we incorporate a specific local search method for each population. The resulting hybrid algorithm, called CELS, is then evaluated on existing benchmark instances, comparing favourably with the state-of-the-art methods.

Keywords: flexible job shop scheduling, fuzzy durations, coevolutionary algorithm, local search.

1 Introduction

Research in scheduling is important, both because it poses complex combinatorial optimisation problems but also as a field with numerous real applications in industry, finance, welfare, etc. In particular, shop problems in their multiple variants— for instance, incorporating flexibility or operators— can model many situations which naturally arise in manufacturing environments. The applicability of scheduling can be enhanced using fuzzy sets as a means of handling flexible constraints and uncertain data, as well as improving solution robustness [10],[18].

In deterministic scheduling, the complexity of problems such as job shop means that practical approaches to solving them usually involve metaheuristic strategies. Some attempts have been made to extend such methods, mostly evolutionary algorithms, to the case where uncertain durations are modelled via fuzzy intervals. In particular, the fuzzy flexible job shop is receiving an increasing attention, with proposals including a genetic algorithm [12], a hybrid artificial bee colony algorithm [17] and a coevolutionary algorithm [13]. In general, coevolutionary algorithms [16] are proving to be very successful in solving complex problems, such as instance and feature selection [2] or parallelisation of multiobjective evolutionary algorithms [3].

In the sequel we tackle the fuzzy flexible job shop problem using a cooperative coevolutionary algorithm hybridised with local search. After introducing the problem, we shall see how it naturally lends itself to cooperative coevolution. We shall later propose neighbourhood structures for each population, so local search can be embedded in the coevolutionary algorithm. The experimental results will illustrate the synergy between the coevolution and the local search, as well as the competitiveness of our approach when compared to the state-of-the-art.

2 The fuzzy flexible job shop scheduling problem

The *flexible job shop scheduling problem*, fJSP in short, consists in scheduling a set of jobs $J = \{J_1, \ldots, J_n\}$ on a set of physical machines $M = \{M_1, \ldots, M_m\}$, subject to a set of constraints. There are *precedence constraints*, so each job $J_i, i = 1, \ldots, n$ consists of a sequence of n_i tasks $\Theta_i = \{\theta_{i1}, \ldots, \theta_{in_i}\}$ that must be sequentially scheduled. There are also *capacity constraints*, whereby each task θ_{ij} requires the uninterrupted and exclusive use of one machine from a subset $R_{ij} \subset M$ for a processing time p_{ijk}, which is dependent on the machine $M_k \in R_{ij}$.

A *feasible schedule* or solution consists of an assignment to machines of all $N = \sum_{i=1}^{n} n_i$ tasks in the set $\Theta = \cup_{1 \leq i \leq n} \Theta_i$ together with an allocation of starting times for each task such that all constraints hold. Alternatively, a solution consists of a feasible assignment of each task $\theta_{ij} \in \Theta$ to a machine $M_k \in R_{ij}$ and a task processing order for each machine in M. Indeed, given these, the starting time of θ_{ij}, denoted S_{ij}, is easily computed as the maximum between the completion times of the predecessor of θ_{ij} in its job and the predecessor of θ_{ij} in the machine M_k where it has been allocated, and the completion time is given by $C_{ij} = S_{ij} + p_{ijk}$. The objective is to find an *optimal* solution according to some criterion, in our case, minimise the *makespan*, which is the completion time of the last task to be executed, denoted $C_{max} = \max_{\theta_{ij} \in \Theta} C_{ij}$.

2.1 Uncertain processing times

In real-life applications, it is often the case that the exact processing time of tasks is not known in advance. However, based on previous experience, an expert may be able to estimate, for instance, an interval for the possible processing time or its most typical value. When there is little knowledge available, the crudest representation for uncertain processing times 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. The simplest model is a *triangular fuzzy number* or *TFN*, using an interval $[a^1, a^3]$ of possible values and a modal value a^2, and with the membership function taking a triangular shape. It is common to denote such TFN as $A = (a^1, a^2, a^3)$.

In the flexible job shop, we essentially need two operations on fuzzy numbers, the sum and the maximum. These are obtained by extending the corresponding operations on real numbers using the *Extension Principle*. In the case of the addition, it turns out that for any pair of TFNs A and B, $A + B = (a^1 + b^1, a^2 + b^2, a^3 + b^3)$. Unfortunately, computing the maximum can be cumbersome and, most importantly, the set of TFNs is not closed under this operation. For the sake of simplicity and tractability of numerical calculations, we follow [5] and approximate the maximum (a continuous isotonic function) by a TFN, evaluating only the operation on the three defining points. For any two TFNs A, B, if F denotes their maximum and $G = (\max\{a^1, b^1\}, \max\{a^2, b^2\}, \max\{a^3, b^3\})$ its approximated value, it holds that $\forall \alpha \in [0, 1], \quad \underline{f}_\alpha \leq \underline{g}_\alpha, \overline{f}_\alpha \leq \overline{g}_\alpha$, where $[\underline{f}_\alpha, \overline{f}_\alpha]$ denotes the α-cut of F or its support if $\alpha = 0$. In particular, F and G have identical support and modal value, that is, $F_0 = G_0$ and $F_1 = G_1$.

The interpretation of membership functions as possibility distributions on the real numbers allows to define the *expected value* of a fuzzy number [9], given for a TFN A by $E[A] = \frac{1}{4}(a^1 + 2a^2 + a^3)$. The expected value coincides with the the *neutral scalar substitute* of a fuzzy interval and can also be obtained as the centre of gravity of its mean value or using the area compensation method [4]. It induces a total ordering $\leq_E$ in the set of fuzzy intervals [5], where for any two fuzzy intervals A, B $A \leq_E B$ if and only if $E[A] \leq E[B]$.

When task durations are given as TFNs, the resulting problem is a *fuzzy flexible job shop problem*, FfJSP in short. Here, our objective will be to minimise the C_{max} according to $\leq_E$.

3 Cooperative coevolutionary algorithm for the FfJSP

Coevolutionary algorithms are advanced evolutionary techniques specially suited to solve complex problems which are decomposable. They handle two or more populations, each with its own coding schemes and recombination operators, that interact through evaluation.When all populations cooperate to build the problem solution, we talk about cooperative algorithms [16].

Cooperative coevolution seems specially suited for the FfJSP, due to the composite nature of its solutions. First, we need to assign the processing of each task θij to a machine $M_k \in R_{ij}$. Once this has been done, we need to establish the order in which tasks are to be processed in each machine. We thus propose a coevolutionary framework to separately evolve a "machine assignment population" P^M and a "task ordering population" P^T.

3.1 Genotype coding and decoding

Every individual from population P^M encodes a machine assignment as a vector $\alpha = \{\alpha_1, \ldots, \alpha_N\}$; task θ_{ij} is associated to the element in position $p = j + \sum_{l=1}^{i-1} n_l$, so $\alpha_p \in R_{ij}$ represents the machine assigned to θ_{ij}. On the other hand, an individual in P^T encodes a topological order of tasks as a permutation with repetition $\pi = \{\pi_1, \ldots, \pi_N\}$ such that $\forall l, 1 \leq \pi_l \leq n$ and $|\{\pi_l : \pi_l = i\}| = n_i, \forall i = 1, \ldots, n$. This is a permutation of the set of tasks as proposed in [1] for the JSP, where each task is represented by its job number. For example, the topological order θ_{21}, θ_{11}, θ_{22}, θ_{31}, θ_{32}, θ_{12} is encoded as (2 1 2 3 3 1).

Notice that the encoding of each population is completely independent of the other population, unlike other proposals from the literature, for instance [13]. This independence allows both populations to evolve separately, interacting only at the evaluation phase. Indeed, to calculate a schedule we require a full solution, combining an individual α from P^M and an individual π from P^T. Then, a pair $(\alpha, \pi) \in P^M \times P^T$ will be decoded using the following insertion strategy. For a task θ_{ij} assigned to machine M_k, a *feasible insertion interval* is defined as a time interval $[t_k^S, t_k^E]$ in which machine M_k is idle and such that $t_k^S + p_{ijk} \leq t_k^E$ and $t_k^S \geq C_{i(j-1)}$ (if $j = 0$, $C_{i(j-1)}$ is taken to be 0), that is θ_{ij} can be processed within that time interval without violating precedence constraints. When t_k^S, t_k^E and p_{ijk} are TFNs, we require that these inequalities hold in each of their three components (in accordance to the definition of maximum and addition). Then, the *earliest starting time* for operation θ_{ij} in machine M_k, denoted EST_{ijk}, is the smallest t_k^S that can be found. The insertion strategy essentially traverses the task sequence in the order given by π and, for each task θ_{ij}, if M_k is the machine assigned to this task by chromosome α, the task is scheduled in machine M_k with starting time $S_{ij} = EST_{ijk}$.

3.2 Initial populations

The simplest way to generate both initial populations is to do it randomly. Alternatively, we propose a heuristic seeding method based on the insertion decoding algorithm. The idea is to use this algorithm as a production rule to generate a full schedule for the FfJSP and then encode its task ordering as an individual for P^M and its machine assignment as an individual for P^R.

More precisely, let A denote the set of tasks that can be scheduled at a certain stage, initially the first task from each job. We iteratively select a random task $\theta_{ij} \in A$ and compute $C^\star = \min\{EST_{ijk} + p_{ijk} : M_k \in R_{ij}\}$, the earliest possible completion time for θ_{ij} in all machines where it can be processed. A machine M_{k*} is randomly selected from the set $K = \{M_k : EST_{ijk} + p_{ijk} = C^\star\}$ of machines where this earliest completion time can be achieved, so θ_{ij} is scheduled in M_{k*} with starting time EST_{ijk*}. θ_{ij} is removed from A and its successor in the job is added to A, provided it exists. The process finishes when A becomes empty.

3.3 Reproduction

For the machine assignment population P^M, we use the one-point crossover: given two genotypes $\alpha^A = \{\alpha_1^A, \ldots, \alpha_N^A\}$ and $\alpha^B = \{\alpha_1^B, \ldots, \alpha_N^B\}$ the operator chooses a random point $p \in (1, N)$ and builds two offsprings α^C, α^D such that $\alpha_i^C = \alpha_i^A$ and $\alpha_i^D = \alpha_i^B$ for $i \leq p$ and $\alpha_i^C = \alpha_i^B$ and $\alpha_i^D = \alpha_i^A$ for $i > p$. The mutation operator takes a random gene α_q in the genotype associated to task θ_{ij} and changes its value to a random machine in R_{ij}.

In the case of P^T, individuals are combined using the JOX operator. Given two genotypes π^A, π^B, JOX selects a random subset of jobs, copies their genes to one offspring in the same positions as in the first parent π^A, and fills the remaining genes from the second parent π^B so that they maintain their relative ordering. The second offspring is formed exchanging the role of the parents with their "unused" genes. The well-known insertion operator or PSM, is used for mutation. A random gene π_p in the genotype is chosen and changes its position to a random one, while keeping the relative order of the other tasks.

In both populations, all individuals are grouped in pairs for mating. Acceptance is carried out using tournament in each group of parents and offsprings, selecting the best two individuals from this group of four to pass to the next generation. Additionally, we introduce elitism, so the best individual from a population substitutes the global worst individual in the next generation.

3.4 Cooperative partners for evaluation

It is at the time of evaluation that populations need to cooperate: any individual only encodes part of a solution and needs to be complemented by an individual from the other population, the so-called *cooperative partner*, to conform a full solution which can be evaluated, using the decoding method above. Based on [11], we use three cooperative partners to evaluate each individual. Assuming all individuals in both populations are arbitrarily ordered, an individual in position p from one population, has as cooperative partners from the other population the best individual in the previous generation, a random individual and the individual in the same position p. The fitness value will be the best makespan from this group of three solutions.

4 Local Search

Evolutionary algorithms are often hybridised with local search to benefit from the synergy between both methods. Here, we propose to apply local search to each individual after its evaluation following a hill-climbing strategy. Local search is applied three times, one for each pair of cooperative partners. The best solution after the three local search processes per individual will be selected and the chromosome will be updated accordingly, thus introducing lamarckism.

It is common in the literature to represent solutions to shop problems using acyclic graphs and define neighbourhood structures based on critical paths in these graphs. Here, we adapt the *solution graph* model from [8] to incorporate machine flexibility. A solution can be represented by an acyclic directed graph G with a node for each task of the problem, labelled with the machine to which it has been assigned, plus two nodes representing fictitious tasks *start* and *end* with null processing times. There are *conjunctive arcs* representing job precedence constraints (including arcs from node *start* to the first task of each job and arcs from the last task of each job to node *end*) and *disjunctive arcs* representing machine processing orders. Each arc is weighted with the processing time of the task at the source node (a TFN in our case).

The starting and completion times of each task can be found by propagating constraints in the graph, and the makespan will be the completion time of task *end* (which may not coincide with the completion time of any job). In the crisp case, the makespan corresponds to the cost of a

critical path, which is defined as the longest path in a solution graph from node *start* to node *end*. It is not trivial to extend concepts and algorithms related to criticality to the problem with uncertain durations (cf [5], [4]). Here we adopt the definition from [8], where it is proposed that a solution graph G be decomposed into three *parallel solution graphs* G^i, $i = 1, 2, 3$, with identical structure to G but where the cost of any arc is the i-th component of the TFN labelling that arc in G. The union of all critical paths in G^i $i = 1, 2, 3$ will be the set of critical paths in G and *critical nodes and arcs* will be those within a critical path. Finally, a *critical block* is a maximal subsequence of tasks of a critical path assigned to the same machine. The makespan of the schedule is not necessarily the cost of a critical path in G, but it holds that each component C^i_{max} is the cost of a critical path in the corresponding solution parallel graph G^i.

For population P^M, representing machine assignments, based on the work of [15] and [6] for other variants of fJSP, we build a neighbour by taking a critical task θ_{ij} and assign it to a new machine $M_k \in R_{ij}$. The resulting neighbours are always feasible, so no repair strategy is needed. The evaluation of neighbours and, hence, the cost of the local search procedure, is optimised by using makespan estimates in the line of [6]

Regarding population P^T, aimed at finding good task orderings, the local search assumes a fixed machine assignment (provided by the cooperative partner). This allows to use the structure for fuzzy job shop from [7], where a neighbour is built by reversing a critical arc at the extreme of a critical block; the motivation for this definition is that reversing critical arcs preserves feasibility and, additionally, reversing arcs inside critical blocks does not improve the makespan. Again, the evaluation of neighbours and consequent cost of the local search procedure is optimised using makespan estimates.

5 Experimental study

For the experimental evaluation, we use the instances available in the literature for the FfJSP which, to our knowledge, are the four instances proposed in [12] (denoted 01–04), and the two instances proposed in [13] (denoted 05,06), these two being the largest ones. Our hybrid algorithm (denoted CELS hereafter) has been implemented in C++ on a PC with a Xeon E5520 processor and 24 Gb RAM. After some preliminary testing, the parameters have been set as follows: 50 individuals per population and 100 generations as stopping criterion, crossover probability equal to 0.90 and mutation probability equal to 0.05 for both populations. A reference for the quality of a solution is the lower bound of the expected makespan given by $LB_f = E[\max_j\{\sum_{i=1}^{n} p^*_{ij}\}, \}]$ with $p^*_{ij} = \min\{p_{ijk}, k \in R_{ij}\}$.

5.1 Analysis of algorithm's components

A first set of experiments is devoted to analysing the different components of our algorithm. To evaluate the heuristic seeding we generate two pairs of initial populations, $HP = (HP^M, HP^T)$ and $RP = (RP^M, RP^T)$, the first pair following Section 3.2 and the second pair at random. We then evaluate the quality of the resulting chromosomes in terms of expected makespan. A summary of the results can be seen in the first columns of Table 1. For reference, the lower bound LB_f is written in brackets under the id of each instance, and the best-known solution (pBKS) is included in the second column. The third and fourth columns report the best (average) expected makespan for both pairs of initial populations. We observe a considerable gain in quality for the heuristic solutions: the average makespan mean relative error (MRE) w.r.t. LB_f is reduced 76% across all instances.

We now evaluate the contribution of the cooperative coevolutionary algorithm (CCEA) and the local search procedure (LS). To this end, CCEA is run with no local search for the same time

Instance	pBKS	RP	HP	CCEA	LS	CELS
01	30.00	62.48	32.30	30.25	28.75	28.50
(28.50)		(83.53)	(37.18)	(31.15)	(29.51)	(28.53)
02	45.25	81.98	47.80	45.75	45.25	45.25
(45.00)		(107.39)	(54.53)	(46.60)	(45.38)	(45.25)
03	47.50	90.88	50.23	47.00	45.25	43.50
(43.50)		(114.11)	(56.95)	(47.63)	(45.86)	(44.18)
04	37.75	76.25	39.25	37.25	36.00	34.25
(33.50)		(95.22)	(44.85)	(38.28)	(36.51)	(35.08)
05	62.00	110.18	63.33	58.50	57.75	53.25
(37.50)		(133.92)	(69.58)	(60.43)	(58.73)	(55.07)
06	63.75	103.89	61.63	57.00	55.75	52.75
(40.25)		(125.26)	(68.04)	(58.50)	(57.41)	(53.93)

Table 1: Analysis of algorithm's components with best (average) expected makespan values obtained in each case.

taken by CELS. Additionally, since CELS uses two populations of size 50 and evolve for 100 generations, we evaluate LS by generating two populations of 500 individiduals and applying LS to the resulting populations (three searches per chromosome, one per cooperative partner). The last three colums in Table 1 report the best(average) expected makespan values obtained with the three methods CCEA, LS and CELS. We can see that CCEA improves the best expected makespan 18% w.r.t. the heuristic initial population, which means that the heuristic seeding provides a good starting point for the CCEA in terms of quality but also in terms of diversity, allowing for a proper evolution of the populations. In fact, the results of CCEA are already quite competitive, especially as the problem size increases. LS obtains even better results than CCEA, with a MRE equal to 17.57% in the best case and equal to 19.67% in average. More importantly, CELS, combining both CCEA and LS, improves the best and average expected makespan in every instance, with the exception of 02, where the best makespan is equal for CELS and LS. As an added value for CELS, the runtime of LS is in average 134% greater than the runtime of CELS. We conclude that there is a synergy effect between the two metaheuristics.

5.2 Comparison with the state-of-the-art in the FfJSP

To our knowledge, the most competitive approaches to FfJSP in the literature are the coevolutionary genetic algorithm (CGA) from [13], the swarm-based neighbourhood search algorithm (SNSA) from [14], and the hybrid artificial bee bolony algorithm (hABC) from [17]. Table 2 shows the results of 30 runs of CELS on each instance compared to these methods. For each method it includes the makespan of the best solution (with its expected value between brackets), the average expected makespan across all solutions found in several runs, the corresponding MRE values and the average runtime of a single run in seconds. The missing rows for instances 05 and 06 correspond to the cases when the original works do not report results on these instances. In bold we highlight the best solution from all methods, marked with "(*)" when it improves the previous best-known solution. We see that CELS improves the best and average values in all cases except for instance 02, where it obtains the same best expected value as SNSA. For instances 01–04 (for which all algorithms provide results), CELS reduces the MRE more than 77% in average. For instance 05, the reduction w.r.t. CGA and SNSA exceeds 38%, and on instance 06 it obtains a 46% reduction w.r.t. SNSA. Notice that solutions for instances 01 and 03 are indeed optimal, as they coincide with the lower bound.

Instance (LB_f)	Algor.	$Best(C_{max})$ ($E[Best(C_{max})]$)	$Avg(E[C_{max}])$	MRE		Time (s.)
				Best	Avg	
01 (28.50)	CGA	21,29,41 (30.00)	33.18	5.26	16.40	8.3
	SNSA	21,29,42 (30.25)	31.68	6.14	11.14	8.7
	hABC	19,30,43 (30.50)	32.15	7.02	12.81	9.9
	CELS	**21,28,37 (28.50)(*)**	**28.53**	**0.00**	**0.09**	1.9
02 (45.00)	CGA	32,47,57 (45.75)	47.45	1.67	5.44	8.3
	SNSA	**35,43,60 (45.25)**	47.05	**0.56**	4.56	8.9
	hABC	33,46,58 (45.75)	47.70	1.67	6.00	10.9
	CELS	32,46,57 **(45.25)**	**45.25**	**0.56**	**0.56**	2.3
03 (43.50)	CGA	34,47,63 (47.75)	51.00	9.77	17.24	10.7
	SNSA	36,46,62 (47.50)	51.25	9.20	17.82	11.4
	hABC	33,47,64 (47.75)	50.70	9.77	16.55	14.8
	CELS	**31,43,57 (43.50)(*)**	**44.18**	**0.00**	**1.55**	3.0
04 (33.50)	CGA	26,37,51 (37.75)	40.80	12.69	21.79	10.8
	SNSA	26,39,53 (39.25)	41.45	17.16	23.73	11.5
	hABC	23,38,53 (38.00)	40.45	13.43	20.75	13.9
	CELS	**24,33,47 (34.25)(*)**	**35.08**	**2.24**	**4.73**	2.7
05 (37.50)	CGA	42,62,82 (62.00)	65.95	65.33	75.87	23.9
	SNSA	40,65,93 (65.75)	68.53	75.33	82.73	14.2
	CELS	**35,53,72 (53.25)(*)**	**55.07**	**42.00**	**46.84**	6.7
06 (40.25)	SNSA	46,63,83 (63.75)	65.65	58.39	63.11	14.4
	CELS	**35,52,72 (52.75)(*)**	**53.93**	**31.06**	**34.00**	6.7

Table 2: Summary of results in FfJSP instances with best-known solutions in **bold**. (*) improves previous best-known solution.

Overall, CELS establishes new best solutions for all instances except for 02, where it obtains the same expected makespan as SNSA. Regarding the average expected makespan, not only is CELS significantly better than the others methods, but it also improves the previously known best value.

6 Conclusions

We have tackled the flexible job shop scheduling problem with fuzzy durations and have proposed a new cooperative coevolutionary algorithm hybridised with local search, named CELS, to solve it. The experimental results have assessed the quality of the initial heuristic seeding and the synergy between coevolution and local search. They have also shown that CELS outperforms the state-of-the-art methods, establishing new best known solutions and, in two cases, even finding the optimal solution.

Acknowledgements

This research has been supported by the Spanish Government under research grants FEDER TIN2010-20976-C02-02 and MTM2010-16051.

References

[1] C. Bierwirth. A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spectrum*, 17:87–92, 1995.

[2] J. Derrac, S. García, and F. Herrera. IFS-CoCo: Instance and feature selection based on cooperative coevolution with nearest neighbor rule. *Pattern Recognition*, 43:2082–2015, 2010.

[3] B. Dorronsoro, G. Danoy, A. J. Nebro, and P. Bouvry. Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution. *Computers & Operations Research*, 40:1552–1563, 2013.

[4] D. Dubois, H. Fargier, and P. Fortemps. Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, 147:231–252, 2003.

[5] P. Fortemps. Jobshop scheduling with imprecise durations: a fuzzy approach. *IEEE Transactions of Fuzzy Systems*, 7:557–569, 1997.

[6] M. González, C. R. Vela, and R. Varela. An efficient memetic algorithm for the flexible job shop with setup times. In *Proc. of ICAPS-2013*, 2013. AAAI Press.

[7] I. González Rodríguez, C. R. Vela, A. Hernández-Arauzo, and J. Puente. Improved local search for job shop scheduling with uncertain durations. In *Proc. of ICAPS-2009*, 154–161, 2009. AAAI Press.

[8] I. González Rodríguez, C. R. Vela, J. Puente, and R. Varela. A new local search for the job shop problem with uncertain durations. In *Proc. of ICAPS-2008*, 124–131, 2008. AAAI Press.

[9] S. Heilpern. The expected value of a fuzzy number. *Fuzzy Sets and Systems*, 47:81–86, 1992.

[10] F. Herrera and J. L. Verdegay. Fuzzy sets and operations research: Perspectives. *Fuzzy Sets and Systems*, 90:207–218, 1997.

[11] Z. Hong and W. Jian. A cooperative coevolutionary algorithm with application to job shop scheduling problem. In *Proc. of SOLI'06*, 746–751. IEEE, 2006.

[12] D. Lei. A genetic algorithm for flexible job shop scheduling with fuzzy processing time. *International Journal of Production Research*, 48(10):2995–3013, 2010.

[13] D. Lei. Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. *Applied Soft Computing*, 12:2237–2245, 2012.

[14] D. Lei and X. Guo. Swarm-based neighbourhood search algorithm for fuzzy flexible job shop scheduling. *International Journal of Production Research*, 50(6):1639–1649, 2012.

[15] M. Mastrolilli and L. Gambardella. Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20, 2000.

[16] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[17] L. Wang, G. Zhou, Y. Xu, and L. Min. A hybrid artificial bee colony algorithm for the fuzzy flexible job-shop scheduling problem. *International Journal of Production Research*, pages 1–16, 2013.

[18] B. K. Wong and V. S. Lai. A survey of the application of fuzzy set theory in production and operations management: 1998–2009. *International Journal of Production Economics*, 129:157–168, 2011.