Elite Artificial Bee Colony for Makespan Optimisation in Job Shop with Interval Uncertainty

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

[1] Department of Computing, University of Oviedo, Gijón, Spain
{diazhernan,palaciosjuan,crvela}@uniovi.es
[2] Departamento de Matemáticas, Estadística y Computación, Universidad de Cantabria, Santander, Spain
gonzalezri@unican.es

Abstract. This paper addresses a variant of the Job Shop Scheduling Problem with makespan minimisation where uncertainty in task durations is taken into account and modelled with intervals. Given the problem's complexity, we tackle it using a metaheuristic approach. Specifically, we propose a novel Artificial Bee Colony algorithm incorporating three different selection mechanisms that help in guiding the search towards more promising areas. A parametric analysis is conducted and a comparison of the different selection strategies is performed on a set of benchmark instances. The results illustrate the benefit of using the new guiding strategies, improving the behaviour of the ABC algorithm, which compares favourably to the state-of-the art in the problem. An additional study is conducted to assess the robustness of the solutions obtained under each guiding strategy.

Keywords: Job Shop Scheduling · Makespan · Interval Uncertainty · Artificial Bee Colony

1 Introduction

The job shop scheduling problem (JSP) is considered to be one of the most relevant scheduling problems. It consists in allocating a set of resources to execute a set of jobs under a set of given constraints, with the most popular objective in the literature of minimizing the project's execution timespan, also known as makespan. Solving this problem improves the efficiency of chain production processes, optimising the use of energy and materials [12] and having a positive impact on costs and environmental sustainability. However, in real-world applications, the available information is often imprecise. Interval uncertainty arises as soon as information is incomplete, and contrary to the case of stochastic and

Supported by the Spanish Government under research grant PID2019-106263RB-I00 and by the Asturias Government under research grant Severo Ochoa.

J. M. Ferrández Vicente et al. (Eds.): IWINAC 2022, LNCS 13259, pp. 98–108, 2022.
https://doi.org/10.1007/978-3-031-06527-9_10

fuzzy scheduling, it does not assume any further knowledge, thus representing a first step towards solving problems in other frameworks [1]. Moreover, intervals are a natural model whenever decision-makers prefer to provide only a minimal and a maximal duration, and obtain interval results that can be easily understood. Under such circumstances, interval scheduling allows to concentrate on significant scheduling decisions and to produce robust solutions.

Contributions to interval scheduling in the literature are not abundant. In [10], a genetic algorithm is proposed for a JSP minimizing the total tardiness with respect to job due dates with both processing times and due dates represented by intervals. In [5] a different genetic algorithm is applied to the same problem, including a study of different interval ranking methods based on the robustness of the resulting schedules. A population-based neighbourhood search for an interval JSP with makespan minimisation is presented in [9]. In [11], a hybrid between PSO and a genetic algorithm is used to solve a flexible JSP with interval processing times as part of a larger integrated planning and scheduling problem. Recently, in [6] a genetic algorithm is applied to the JSP with interval uncertainty minimizing the makespan and achieving the results that are the current state of the art.

Metaheuristic search methods are especially suited for job shop due to its complexity. In particular, artificial bee colony (ABC) is a swarm intelligence optimiser inspired by the intelligent foraging behaviour of honeybees that has shown very competitive performance on JSP with makespan minimisation. For instance, [13] propose an evolutionary computation algorithm based on ABC that includes a state transition rule to construct the schedules. Taking some principles from Genetic Algorithms, [14] present an Improved ABC (IABC) where a mutation operation is used for exploring the search space, enhancing the search performance of the algorithm. Later, [2] propose an effective ABC approach based on updating the population using the information of the best-so-far food source.

In the following, we consider the JSP with makespan minimisation and intervals modelling uncertain durations. The problem is presented in Sect. 2. In Sect. 3 we propose several variants of an ABC algorithm to address this problem. These variants are compared in Sect. 4, where the most successful one is also compared with the state-of-the-art and a robustness analysis is also included.

2 The Job Shop Problem with Interval Durations

The classical *job shop scheduling problem* consists of a set of resources $M = \{M_1, \ldots, M_m\}$ and a set of jobs $J = \{J_1, \ldots, J_n\}$. Each job J_j is organised in tasks $(o(j,1), \ldots, o(j,m_j))$ that need to be sequentially scheduled. We assume w.l.o.g. that tasks are indexed from 1 to $N = \sum_{j=1}^{n} m_j$, so we can refer to task $o(j,l)$ by its index $o = \sum_{i=1}^{j-1} m_i + l$ and denote the set of all tasks as $O = \{1, \ldots, N\}$. Each task $o \in O$ requires the uninterrupted and exclusive use of a machine $\nu_o \in M$ for its whole processing time p_o.

A solution to this problem is a *schedule* $\boldsymbol{s}$, i.e. an allocation of starting times for each task, which, besides being *feasible* (all constraints hold), is *optimal* according to some criterion, in our case, minimal makespan C_{max}.

2.1 Interval Uncertainty

Following [9] and [5], uncertainty in the processing time of tasks is modelled using a closed intervals. Therefore, the processing time of task $o \in O$ is represented by an interval $\mathbf{p_o} = [\underline{p}_o, \overline{p}_o]$, where $\underline{p}_o$ and $\overline{p}_o$ are the available lower and upper bounds for the exact but unknown processing time p_o.

The interval JSP (IJSP) with makespan mimisation requires two arithmetic operations: addition and maximum. Given two intervals $\mathbf{a} = [\underline{a}, \overline{a}], \mathbf{b} = [\underline{b}, \overline{b}]$, the addition is expressed as $[\underline{a} + \underline{b}, \overline{a} + \overline{b}]$ and the maximum as $[\max(\underline{a}, \underline{b}), \max(\overline{a}, \overline{b})]$. Also, given the lack of a natural order in the set of closed intervals, to determine the schedule with the "minimal" makespan, we need an interval raking method. For the sake of fair comparisons with the literature, we shall use the midpoint method: $\mathbf{a} \leq_{MP} \mathbf{b} \Leftrightarrow m(\mathbf{a}) \leq m(\mathbf{b})$ with $m(\mathbf{a}) = (\underline{a} + \overline{a})/2$. This is used in [5] and it is equivalent to the method used in [10]. Notice that $m(\mathbf{a})$ coincides with the expected value of the uniform distribution on the interval $E[\mathbf{a}]$.

A schedule $\boldsymbol{s}$ for the IJSP establishes a relative order π among tasks requiring the same machine. Conversely, given a task processing order π the schedule $\boldsymbol{s}$ may be computed as follows. For every task $o \in O$, let $\mathbf{s_o}(\pi)$ and $\mathbf{c_o}(\pi)$ denote respectively the starting and completion times of o, let $PM_o(\pi)$ and $SM_o(\pi)$ denote the predecessor and successor tasks of o in the machine ν_o according to π, and let PJ_o and SJ_o denote the tasks preceding and succeeding o in its job. Then the starting time of o is given by $\mathbf{s_o}(\pi) = \max(\mathbf{s}_{PJ_o} + \mathbf{p}_{PJ_o}, \mathbf{s}_{PM_o(\pi)} + \mathbf{p}_{PM_o(\pi)})$, and the completion time by $\mathbf{c_o}(\pi) = \mathbf{s_o}(\pi) + \mathbf{p_o}$. The makespan is computed as the completion time of the last task to be processed according to π thus, $\mathbf{C_{max}}(\pi) = \max_{o \in O}\{c_o(\pi)\}$. If there is no possible confusion regarding the processing order, we may simplify notation by writing $\mathbf{s_o}$, $\mathbf{c_o}$ and $\mathbf{C_{max}}$.

2.2 Robustness on Interval JSP

When uncertainty is present, solution robustness may become a concern. In fact, makespan values obtained for IJSP are not exact values, but intervals. It is only after the solution is executed on a real scenario that actual processing times for tasks $P^{ex} = \{p_o^{ex} \in [\underline{p}_o, \overline{p}_o], o \in O\}$ are known. Therefore, it is not until that moment that the actual makespan $C_{max}^{ex} \in [\underline{C}_{max}, \overline{C}_{max}]$ can be found. It is desirable that this executed makespan C_{max}^{ex} does not differ much from the expected value of the makespan according to the interval $\mathbf{C_{max}}$.

This is the idea behind the concept of ϵ-robustness first proposed in [3] for stochastic scheduling, and later adapted to the IJSP in [5]. For a given $\epsilon \geq 0$, a schedule with makespan $\mathbf{C_{max}}$ is considered to be *ϵ-robust* in a real scenario P^{ex} if the relative error made by the expected makespan $E[\mathbf{C_{max}}]$ with respect to the makespan C_{max}^{ex} of the executed schedule is bounded by ϵ, that is:

$$\frac{|C^{ex}_{max} - E[\mathbf{C_{max}}]|}{E[\mathbf{C_{max}}]} \leq \epsilon. \tag{1}$$

Clearly, the smaller the bound ϵ, the more robust the interval schedule is.

This measure of robustness is dependent on a specific configuration P^{ex} of task processing times obtained upon execution of the predictive schedule $\boldsymbol{s}$. In the absence of real data, as is the case with the usual synthetic benchmark instances for job shop, we may resort to Monte-Carlo simulations. We simmulate K possible configurations $P^k = \{p^k_o \in [\underline{p}_o, \overline{p}_o], o \in O\}$ using uniform probability distributions to sample durations for every task and compute for each configuration $k = 1, \ldots, K$ the exact makespan C^k_{max} that results from executing tasks according to the ordering provided by $\boldsymbol{s}$. Then, the average ϵ-robustness of the predictive schedule across the K possible configurations, denoted $\overline{\epsilon}$, can be calculated as:

$$\overline{\epsilon} = \frac{1}{K} \sum_{k=1}^{K} \frac{|C^k_{max} - E[\mathbf{C_{max}}]|}{E[\mathbf{C_{max}}]}. \tag{2}$$

This value provides an estimate of how robust the solution $\boldsymbol{s}$ is across different processing times configurations.

3 An Artificial Bee Colony Algorithm

The Artificial Bee Colony Algorithm is a bioinspired swarm metaheuristic for optimisation based on the foraging behaviour of honey bees. Since it was introduced in [7] it has been successfully adapted to a variety of problems [8]. In this paper, we adapt it to solve the Interval Job Shop Scheduling problem.

In ABC, a swarm of bees exploit a changing set of food sources with two leading models of behaviour: recruiting rich food sources and abandoning poor ones. In our case, each food source *fs* encodes an IJSP solution using permutations with repetition [4] and the decoding of a food source follows an insertion strategy, consisting in iterating along the food source and scheduling each task at its earliest feasible insertion position [5]. The richness or nectar amount of each food source is proportional to the makespan of the schedule it represents, so lower makespan values translate into richer food sources.

The ABC starts by generating and evaluating initial pool P_0 of random food sources, so the best food source in the pool is assigned to the hive queen. Then, ABC iterates over a number of cycles, each consisting of three phases mimicking the behaviour of three types of foraging bees: employed, onlooker and scout. In the employed bee phase, each food source is assigned to one employed bee, so this employed bee explores a new candidate food source between its own food source and the queen's one, evaluating the candidate and sharing this information with the rest of the hive. If the new food source is equivalent to queen's (i.e. the best food source found so far), it is discarded for the sake of maintaining diversity in the pool. If it is not discarded and it improves the original food source (i.e. smaller makespan value), it replaces it. Otherwise, the number of improvement trials *fs.numTrials* of the original food source is increased by one. In the next phase,

each onlooker bee chooses a food source and tries to find a better neighbouring one. The new food source receives the same treatment as in the previous phase. Finally, in the scout bee phase, if the number of improvement trials of a food source reaches a maximum number NT_{max}, the scout bee finds a new food source to replace the former one in the pool of solutions. Finally, the algorithm terminates after a number *maxIter* of consecutive iterations without finding a food source that improves the queen's one. The following subsections provide more detail on each of the phases; the pseudo-code of the resulting ABC is given in 1.

3.1 Employed Bee Phase

Originally, the employed bees search is always guided by the queen's food source. However, this strategy may in some occasions cause a lack of diversity in the swarm and lead to premature convergence [2]. To address this issue, we propose to modify the original algorithm and select the guiding food source from an elite group that will contain the most suitable food sources according to one of the following strategies. In the first strategy, denoted Elite_1, it only contains the best-found food source, so it is equivalent to the classical ABC. In the second strategy, denoted Elite_2, the elite group contains the food sources with the highest number of improvement trials at the beginning of the iteration and the best food source in the set is selected to guide the employed bee. Finally, in the third strategy, denoted Elite_3, the elite group contains the best B food sources in the current swarm and a solution from this group is chosen at random to guide the employed bee. B is a parameter of the algorithm that helps balancing diversity: when $B = 1$, this strategy is equivalent to Elite_1, and the larger B is, the more diversity is inserted into the phase.

Once two food sources are selected for each employed bee, a recombination operator is applied with probability p_{emp} to find a new food source to explore. Here, taking advantage of the solution encoding, we propose to use the following operators: Generalised Order Crossover (GOX), Job-Order Crossover (JOX) and Precedence Preservative Crossover (PPX).

3.2 Onlooker Bee Phase

In this phase, food sources are selected from those that have not reached the maximum number of improvement trials. Each selected food source is assigned to an onlooker bee that will explore a neighbouring solution with probability p_{on} to explore a neighbouring solution. Neighbours are obtained by performing a small change on the food source using one of the following operators for permutations: Swap, Inversion or Insertion.

Table 1. Final parameter setup for each variant of ABC

Instance	ABC_{E1}	ABC_{E2}	ABC_{E3}
Recombination operator	JOX	JOX	GOX
Employed probability p_{emp}	0.75	1	1
Neighbourhood operator	Insertion	Insertion	Swap
Onlooker probability p_{on}	0.75	0.75	1
Improvement trials $fs.numTrials$	10	15	20

3.3 Scout Bee Phase

In this last phase, a scout bee is assigned to each food source that has reached the maximum number of improvement trials. Since this food source has not been improved after the given number of attempts, it is discarded and the scout bee is in charge of finding a replacement. To implement this phase, every food source fs having $fs.numTrials > \mathrm{NT}_{max}$ is replaced by a random one fs' with $fs'.numTrials = 0$.

4 Experimental Results

The objective of this Section is to evaluate the performance of the three variants of the ABC algorithm in comparison with the state-of-the-art for interval JSP with makespan minimization, which, to our knowledge is the genetic algorithm from [5], referred to as *GA* hereafter.

We consider 12 well-known instances for the job shop problem (in brackets, the size $n \times m$): FT10 (10×10), FT20 (20×5), La21, La24, La25 (15×10), La27, La29 (20×10), La38, La40 (15×15), ABZ7, ABZ8, and ABZ9 (20×15). Processing times are modified to be intervals, so given the original deterministic processing time of a task p_o, the interval time is $\mathbf{p_o} = [p_o - \delta, p_o + \delta]$, where δ is a random value in $[0, 0.15p_o]$. The resulting IJSP instances are available online[1]. We use a PC with Intel Xeon Gold 6132 processor at 2.6 Ghz and 128 Gb RAM with Linux (CentOS v6.10) and a C++ implementation. Every variant of the algorithm is run 30 times on each instance to obtain representative data.

A parameter tuning process has been carried out for the three variants of ABC, namely ABC_{E1}, ABC_{E2} and ABC_{E3}, where ABC_{Ei} incorporates the strategy $\mathbf{Elite}_i$, $i = 1, 2, 3$, in the employed bee phase. In all cases, the population size is equal to 250 and the stopping criterion consists in $maxIter = 25$ consecutive iterations without improving the best solution found so far. For ABC_{E3}, the size of the elite set is $B = 50$ food sources. The final configuration for the remaining parameters for each variant of *ABC* is shown in Table 1.

Table 2 summarises the results obtained by the *GA* from [5] and the three *ABC* variants. For each algorithm and instance it reports the expected makespan (or midpoint) of the best-found solution ($m(\mathbf{Best})$), the average expected

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

Table 2. Computational results and times of GA and ABC

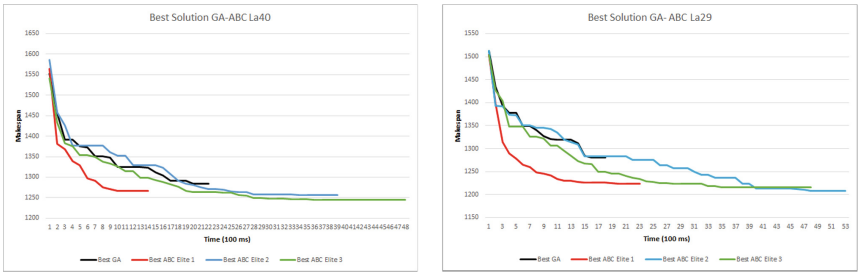
	GA				ABC_{E1}				ABC_{E2}				ABC_{E3}			
Instance	m(**Best**)	Avg.	σ	Time	m(**Best**)	Avg.	σ	Time	m(**Best**)	Avg.	σ	Time	m(**Best**)	Avg.	σ	Time
*ABZ*7	697.5	738.0	12.87	1.80	703.0	722.3	8.27	**1.66**	691.5	713.0	18.90	6.35	**690.5**	**704.0**	7.27	4.45
*ABZ*8	718.0	764.2	13.82	**1.76**	721.5	741.7	10.36	2.07	**702.0**	730.9	19.31	7.00	703.0	**722.8**	7.49	4.19
*ABZ*9	747.0	779.8	15.91	**2.19**	729.5	761.7	19.24	2.27	**715.0**	757.0	19.41	6.59	725.0	**747.8**	10.37	5.97
*FT*10	947.0	978.6	19.70	**0.48**	945.0	982.1	19.47	0.78	**939.0**	**966.7**	16.73	1.14	940.0	968.2	11.87	1.56
*FT*20	1182.0	1215.7	15.82	**0.69**	1177.0	1199.0	16.36	0.77	**1173.0**	1190.2	13.17	2.06	**1173.0**	**1185.1**	7.70	2.74
*LA*21	1079.0	**1098.4**	13.45	1.13	**1067.5**	1112.6	18.19	**0.90**	1069.5	**1098.4**	15.55	2.54	1073.0	**1098.4**	13.54	1.82
*LA*24	973.0	994.3	14.66	0.81	972.0	999.9	14.71	**0.70**	965.0	986.5	15.18	2.06	**956.0**	**982.3**	11.72	2.74
*LA*25	996.0	1026.9	23.39	0.97	1010.5	1037.5	18.00	**0.71**	**992.0**	1019.5	16.02	3.80	996.0	**1014.9**	8.71	2.44
*LA*27	1291.5	1361.2	24.67	**1.30**	1281.0	1319.8	16.95	1.38	**1268.5**	1300.5	18.49	4.55	1269.0	**1292.6**	12.25	4.11
*LA*29	1280.0	1315.9	18.63	**1.08**	1223.0	1281.1	27.14	1.35	**1208.0**	**1250.1**	26.03	5.00	1215.5	1251.6	15.13	4.40
*LA*38	1268.0	1305.5	27.26	1.41	**1249.5**	1304.5	27.50	**1.27**	1251.5	1289.4	21.17	3.41	1250.0	**1278.3**	17.61	5.96
*LA*40	1284.0	1328.8	28.50	**1.20**	1252.0	1302.1	21.70	1.43	1256.0	1283.4	17.94	4.04	**1245.0**	**1273.4**	13.81	3.01

makespan across all runs, the standard deviation, and the average CPU time in seconds. The best result for each instance is highlighted in bold. Additionally, ANOVA or Kruskall Wallis statistical tests have been performed on the results depending on the normality of the data, followed by a multi-variable analysis. Grey cells highlight those algorithms with no significant difference w.r.t. the best solution on that instance.

In terms of m(**Best**), GA is outperformed by ABC_{E2} and ABC_{E3} on every instance. ABC_{E2} improves GA 1.82% on average, being up to 5.63% for instance La29. If we pay attention to the average behaviour, ABC_{E3} obtains the best results on 10 out of 12 instances, while it is not significantly different from the best on the remaining 2 instances. On average, its results are 3.02% better than those of GA. However, it is not significantly different than ABC_{E2} on any instance. What is more interesting is that ABC_{E1} is never in the set of best methods, which reinforces the hypothesis that the standard ABC is not adequate for our problem and the proposed alternatives offer a significant improvement both w.r.t. the standard ABC and the state-of-the-art GA.

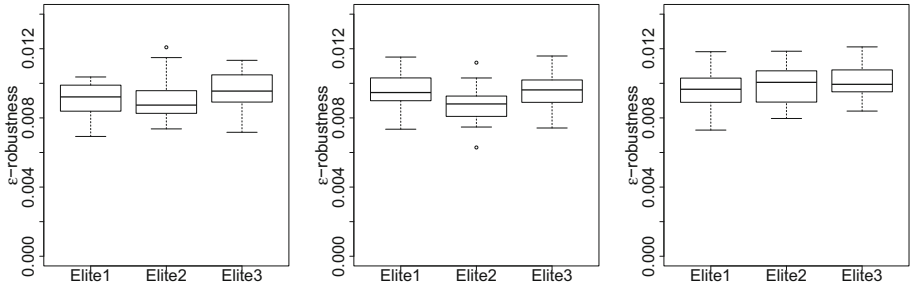
We can also observe that all ABC variants take longer running times than GA. The reason is that one iteration of ABC takes longer than an iteration of GA and also the dynamic stopping criterion translates into more iterations (hence, longer running times) for ABC. However, the efficiency of ABC_{Ei} per time unit is comparable to if not better than that of GA. Figure 1 depicts the evolution of the midpoint of the best makespan for representative instances La29 and La40. In both cases, ABC_{E1} (red line) and ABC_{E3} (green line) not only outperform GA (black line) in the final result, but also present a better makespan improvement rate per time unit. For ABC_{E2} (blue line) this improvement rate is very similar to that of GA, but ABC_{E2} achieves a better final result by taking longer to converge. This longer time to converge is also observed in ABC_{E3}, the version that achieves better results in average.

Finally, we perform a robustness analysis on the solutions obtained by each method. To do so, for each variant of ABC, each instance and each one of the 30 runs we take the expected makespan according to the obtained solution as well as the associated task processing order. This task order is then executed



(a) La40 (b) La29

Fig. 1. Evolution along time of the makespan's midpoint for the best schedules obtained with *GA* (in black) and the different variants of *ABC* on instances La40 and La29. (Color figure online)



(a) La29 (b) La38 (c) La40

Fig. 2. $\bar{\epsilon}$-robustness of schedules obtained with the different variants of *ABC* on instances La29, La38 and La40.

for $K = 1000$ deterministic realisations of each instance to calculate the $\bar{\epsilon}$ value. Figure 2 shows the boxplots of the resulting $\bar{\epsilon}$ values on three representative instances. Statistical tests on all instances allow to conclude that there is no significant difference between the robustness of the three variants of the ABC. This homogeneity shows that the newly proposed variants ABC_{E2} and ABC_{E3} can obtain better results than a standard ABC (ABC_{E1}) and *GA* without deteriorating the robustness of the solutions.

Algorithm 1. Schema of the ABC Algorithm

Require: An IJSP instance
Ensure: A schedule
  Generate a pool P_0 of food sources
  $Best \leftarrow$ Best solution in P_0
  $numIter \leftarrow 0$
  while $numIter < maxIter$ **do**
    /* *Employed bee phase*/*
    $E \leftarrow$ Elite group from P_i based on $\mathbf{Elite}_x$ strategy
    for each food source fs in P_i **do**
      $fs' \leftarrow$ Select food source from E using $\mathbf{Elite}_x$ strategy
      $new_{fs} \leftarrow$ Apply crossover to (fs, fs') with probability p_{emp}
      if new_fs is better than fs and different than $Best$ **then**
        $fs \leftarrow new_{fs}$
        if new_{fs} is better than $Best$ **then**
          $Best \leftarrow new_{fs}$
          $numIter \leftarrow 0$
      else
        $fs.numTrials \leftarrow fs.numTrials + 1$
    /* *Onlooker bee phase*/*
    for each food source fs in P_i **do**
      if $fs.numTrials < \mathrm{NT}_{max}$ **then**
        $new_{fs} \leftarrow$ Apply onlooker operator to fs with probability p_{on}
        if new_{fs} is better than fs and different than $Best$ **then**
          $fs \leftarrow new_{fs}$
          if new_{fs} is better than $Best$ **then**
            $Best \leftarrow new_{fs}$
            $numIter \leftarrow 0$
        else
          $fs.numTrials \leftarrow fs.numTrials + 1$
    /* *Scout bee phase*/*
    for each food source fs in P_i **do**
      if $fs.numTrials > \mathrm{NT}_{max}$ **then**
        $fs \leftarrow$ find new food source
        $fs.numTrials \leftarrow 0$
        if new_{fs} is better than $Best$ **then**
          $Best \leftarrow new_{fs}$;
          $numIter \leftarrow 0$
    $numIter \leftarrow numIter + 1$
  return $Best$

5 Conclusions

We have considered the IJSP, a version of the JSP that models the uncertainty on task durations appearing in real-world problems using intervals. We have used an ABC approach as solving method, adapting the general scheme to our problem, and we have tackled the issue of lack of diversity in the swarm by

redesigning certain aspects of the employed and onlooker bee phases. This has resulted in three variants of the *ABC* algorithm. An experimental analysis has shown the potential of these variants, especially those introducing more diversity, ABC_{E2} and ABC_{E3}, which outperform the results of a more standard ABC_{E1} as well as the state-of-the-art from the literature. This improvement is present not only when all methods are allowed to converge and stop after *maxIter* iterations without improvement, but it would also be the case if the stopping criterion were changed and they were given equal runtime to the state-of-the-art method. Finally, a robustness analysis has shown that the makespan improvement of the new methods is not obtained at the expense of deteriorating the solutions' robustness.

References

1. Allahverdi, A., Aydilek, H., Aydilek, A.: Single machine scheduling problem with interval processing times to minimize mean weighted completion time. Comput. Oper. Res. **51**, 200–207 (2014)
2. Banharnsakun, A., Sirinaovakul, B., Achalakul, T.: Job shop scheduling with the best-so-far ABC. Eng. Appl. Artif. Intell. **25**(3), 583–593 (2012)
3. Bidot, J., Vidal, T., Laboire, P.: A theoretic and practical framework for scheduling in stochastic environment. J. Sched. **12**, 315–344 (2009)
4. Bierwirth, C.: A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectrum **17**, 87–92 (1995)
5. Díaz, H., González-Rodríguez, I., Palacios, J.J., Díaz, I., Vela, C.R.: A genetic approach to the job shop scheduling problem with interval uncertainty. In: Lesot, M.-J., et al. (eds.) IPMU 2020. CCIS, vol. 1238, pp. 663–676. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50143-3_52
6. Díaz, H., Palacios, J.J., Díaz, I., Vela, C.R., González-Rodríguez, I.: Tardiness minimisation for job shop scheduling with interval uncertainty. In: de la Cal, E.A., Villar Flecha, J.R., Quintián, H., Corchado, E. (eds.) HAIS 2020. LNCS (LNAI), vol. 12344, pp. 209–220. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61705-9_18
7. Karaboga, D.: An idea based on honey bee swarm for numerical optimization, technical report - tr06. Technical report, Erciyes University, January 2005
8. Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artif. Intell. Rev. **42**(1), 21–57 (2012). https://doi.org/10.1007/s10462-012-9328-0
9. Lei, D.: Population-based neighborhood search for job shop scheduling with interval processing time. Comput. Ind. Eng. **61**, 1200–1208 (2011)
10. Lei, D.: Interval job shop scheduling problems. Int. J. Adv. Manuf. Technol. **60**, 291–301 (2012)
11. Li, X., Gao, L., Wang, W., Wang, C., Wen, L.: Particle swarm optimization hybridized with genetic algorithm for uncertain integrated process planning and scheduling with interval processing time. Comput. Ind. Eng. **235**, 1036–1046 (2019)
12. Pinedo, M.L.: Scheduling. Theory, Algorithms, and Systems. Springer, New York (2016). https://doi.org/10.1007/978-1-4614-2361-4

13. Wong, L.P., Puan, C.Y., Low, M.Y.H., Chong, C.S.: Bee colony optimization algorithm with big valley landscape exploitation for job shop scheduling problems. In: 2008 Winter Simulation Conference, pp. 2050–2058 (2008)
14. Yao, B., Yang, C., Hu, J., Yin, G., Yu, B.: An improved artificial bee colony algorithm for job shop problem. Appl. Mech. Mater. **26–28**, 657–660 (2010)