# An Elitist Seasonal Artificial Bee Colony Algorithm for the Interval Job Shop

Hernán Díaz [a], Juan J. Palacios [a], Inés González-Rodríguez [b], Camino R. Vela [a,*],
[a] *Dep. of Computing, University of Oviedo, Gijón, Spain*
*E-mails: diazhernan@uniovi.es, palaciosjuan@uniovi.es, crvela@uniovi.es*
[b] *Dep. Matemáticas, Estadística y Computación, Universidad de Cantabria, Santander, Spain*
*E-mail: gonzalezri@unican.es*

**Abstract.** In this paper, a novel Artificial Bee Colony algorithm is proposed to solve a variant of the Job Shop Scheduling Problem where only an interval of possible processing times is known for each operation. The solving method incorporates a diversification strategy based on the seasonal behaviour of bees. That is, the bees tend to explore more at the beginning of the search (spring) and be more conservative towards the end (summer to winter). This new strategy helps the algorithm avoid premature convergence, which appeared to be an issue in previous papers tackling the same problem. A thorough parametric analysis is conducted and a comparison of different seasonal models is performed on a set of benchmark instances from the literature. The results illustrate the benefit of using the new strategy, improving the performance of previous ABC-based methods for the same problem. An additional study is conducted to assess the robustness of the solutions obtained under different ranking operators, together with a sensitivity analysis to compare the effect that different levels of uncertainty have on the solutions' robustness.

Keywords: Artificial Bee Colony, Job Shop Scheduling, Makespan, Interval Uncertainty, Robustness

## 1. Introduction

The job shop scheduling problem (JSP) in its different variants is considered one of the most relevant problems in scheduling, in part because it is used to model many practical engineering and social applications [1]. It consists in organising the execution of a set of jobs on a set of resources under a set of given constraints in an optimal way. In the literature, this most commonly translates into minimising the project's execution timespan, also known as makespan. Solving this problem improves the efficiency of chain production processes and has a positive impact on costs and environmental sustainability [2].

Real-world applications of JSP can be generally found in industries where customer orders may differ and have their own parameters; this applies obviously to many manufacturing industries, but also to service industries. For instance, a very common application is semiconductor manufacturing; this includes wafer fabrication, where layers of metal and wafer material are built up in patterns on wafers of silicon or gallium arsenide to produce the circuitry and where each layer requires a number of operations. Another classical example of a job shop is a hospital, where patients can be seen as jobs, so each patient has to follow a given route and has to be treated at a number of different stations while going through the system [3]. A railway scheduling problem can also be modelled as a job shop by dividing the railway network into segments, so each job corresponds to a train trip and operations in a job correspond to the track segments on the train route [4]. Steel mills also have workshops that function as job shops; they produce heavy-duty parts which are processed in different machines, including cutting machines, shaper machines, grinding machines, milling machines, lathes and special turning machines, drilling/boring machines, polishing ma-

*Corresponding author. E-mail: crvela@uniovi.es.

chines, and painting and drying machines [5]. Another example can be found in the apparel industry, where the job shop is used to model both a progressive bundle system and a unit production system [6]. A wide list of applications of the job shop scheduling in its multiple variants can be found in the survey from [1]. Additionally, real-life applications of the flexible variant of the job shop are reviewed in [7].

To increase its applicability, we must take into account that in real-world situations the available information is often imprecise. Interval uncertainty arises as soon as information is incomplete, and contrary to the case of stochastic and fuzzy scheduling, it does not assume any further knowledge [8, 9]. 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 relevant scheduling decisions and to produce robust solutions. Intervals can also be provided if there is some uncertainty about numerical data, as done, for instance, in [10].

Solving scheduling problems with interval uncertainty can also contribute to the active field of fuzzy scheduling [11]. Here, it is common to use fuzzy intervals to represent both uncertainty and preference in parameters, such as ill-known processing times and flexible due dates [12]. Fuzzy intervals are fuzzy sets in the real line whose level-cuts are intervals. Since fuzzy arithmetic is defined via the extension principle for fuzzy sets, operations between fuzzy intervals extend the usual interval analysis into membership functions [13, 14]. Hence, solving a scheduling problem where pure intervals are used to represent ill-known durations provides a first step towards solving problems in the framework of fuzzy scheduling.

Additionally, intervals are inherent to interval-valued fuzzy sets, where the membership degree of an element of the set corresponds to a value in a considered membership interval [15]. Interval-valued fuzzy sets provide an alternative model for ill-known processing times where decision makers find it tough to quantify their judgement about the membership of a possible duration value as a number in the interval $[0, 1]$ and prefer instead to reflect their opinions by a range [16, 17].

Contributions to scheduling with interval uncertainty are not abundant in the literature. A recent review of publications where intervals are used to model either setup or processing times (or both) in different scheduling problems can be found in [9]. For the job shop, a genetic algorithm is proposed in [18] to minimize the total tardiness when both durations and due dates are intervals. In [19] a different genetic algorithm is applied to the same problem and [20] includes a study of the influence of using different interval ranking methods on the robustness of the resulting schedules. In [21], a hybrid between particle swarm optimisation and a genetic algorithm is used to solve a flexible JSP with interval durations within a more complex integrated planning and scheduling problem. Finally, the JSP minimising makespan with interval durations is tackled using a population-based neighbourhood search in [22], a genetic algorithm in [23] and, more recently, an artificial bee colony method in [24], with the latter achieving the results that constitute the current state of the art and serving as basis for this paper.

The above methods for solving scheduling problems with interval uncertainty belong to the nature-inspired computing paradigm. Indeed, nature-inspired methods have proved very successful in solving complex optimisation [25]. Evolutionary algorithms, inspired in biological evolution, have been widely used for solving complex optimisation problems, many of them with engineering applications [26]. Another well-known group of bio-inspired algorithms are swarm intelligence methods, mimicking the collective behaviour of decentralised, self-organized systems in nature, such as flocks of birds or ant colonies [27]. Together with biology-based algorithms, we find methods inspired in physics [28]. Perhaps the best-known is simulated annealing (SA), based on the principle of thermodynamics [29]. Other example of physics-based method is the harmony search algorithm (HSA), which is a music-inspired population-based metaheuristic algorithm [30]. The gravitational search algorithm (GSA) has its roots in gravitational kinematics [31] while the water drop algorithm (WDA), in hydrology and hydrodynamics [32]. Finally, we can also find chemical reaction optimisation, a population-based meta-heuristic algorithm based on the principles of chemistry [33].

Examples abound in the literature where swarm intelligence methods have been proposed to tackle complex optimisation problems. For instance, cat swarm optimisation, inspired by the biological behaviour of domestic cats, performs very well on binary combinatorial optimisation problems such as 0/1 knapsack [34]. Spider monkey optimisation, based on the social behaviour of spider monkeys is successfully applied to the travelling salesman problem [35]. A particle swarm optimisation (PSO) method with multiple swarms helps improving transfer learning methods in [36], while an ensemble of five particle swarm optimisation strategies is applied to designing the structure of echo state networks in [37]. Another PSO-based method with selective search helps to find good solutions to the university course scheduling problem [38].

There is also an extensive record of successful applications of different metaheuristic methods, most of them nature-inspired, to solving complex engineering problems. For instance, modified versions of the ant colony optimisation algorithm have been proposed to solve high-speed railway alignment and vertical alignment within highway geometric design [39, 40]. A hybrid metaheuristic algorithm that combines harmony search, flower pollination, teaching-learning-based optimisation and Jaya algorithm has been used for the optimization process of active tuned mass dampers, used in structures in the reduction of structural responses resulted from earthquakes [41]. A coronavirus optimisation algorithm combined with long short term memory deep learning has served to forecast deformations of a hydropower dam in [42]. The neural dynamic model of Adeli and Park has been successful in optimising large steel structures [43, 44]. More recently, game theory-based strategies have been incorporated to a Jaya algorithm to improve the computation efficiency and effectiveness in design optimization of civil engineering structures [45].

In particular, artificial bee colony (ABC), a swarm intelligence optimisation template inspired in the foraging behaviour of honeybees, has shown very competitive performance on deterministic JSP with makespan minimisation. For instance, [46] proposes an evolutionary computation algorithm based on ABC that includes a state transition rule to construct the schedules. Taking some principles from genetic algorithms, an improved ABC (IABC) is proposed in [47] that uses a mutation operation to explore the search space, thus enhancing the search performance of the algorithm. An effective ABC approach based on updating the population using the information of the best-so-far food source can be found in [48]. More recently, [24] introduces the idea of having an elite set of bees instead of a queen to improve diversification.

In this work, we tackle the JSP with makespan minimisation and intervals modelling uncertain durations. The problem is presented in Section 2. Building upon our former work in [24], where the employed bee phase was modified, in Section 3 we propose to include a new strategy based on the seasonal behaviour of bees. These variants are compared in Section 4, where the most successful one is also compared with the state-of-the-art. In that section, a robustness analysis is conducted to compare different interval ranking methods, together with a sensitivity analysis on the performance of the algorithm over scenarios with different levels of uncertainty.

## 2. The Job Shop Problem with Interval Durations

In the *job shop scheduling problem* we have several machines or resources $M = \{M_1, \ldots, M_m\}$ where a set of jobs $J = \{J_1, \ldots, J_n\}$ need to be processed. Each job $J_j$ is composed of $m_j$ tasks $(o_{j,1}, \ldots, o_{j,m_j})$ that must be sequentially executed. We can uniquely identify every task with a number between 1 and $N = \sum_{j=1}^{n} m_j$, so task $o_{j,l}$ maps to $o = l$ if $j = 1$ and $o = \sum_{i=1}^{j-1} m_i + l$ if $j > 1$. In this way, the set of all tasks can be denoted as $O = \{1, \ldots, N\}$. Besides the precedence constraints within a job, there exist resource constraints, in the sense that each task $o \in O$ needs to be executed in a machine $M(o) \in M$ for its whole processing time $p_o$ without interruptions and without the possibility of simultaneously executing other tasks in $M(o)$.

A *schedule* $\boldsymbol{s}$ is an assignment of starting times for all tasks. The schedule is said to be *feasible* if it does not violate any of the constraints. The *makespan* $C_{max}$ of a schedule is the time span between the start and the completion of the whole project. A solution to the job shop is a feasible schedule minimising the makespan.

### 2.1. Interval Uncertainty

Uncertainty in the processing times of tasks is modelled using closed intervals as done, for instance, in [20, 22]. This approach is appropriate when the lack of historical data does not allow to estimate probabilities and only an upper and lower bound of the likely duration can be provided or when there is uncertainty in numerical data [9, 10]. Under these circumstances, the processing time of a 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 $p_o$. Obviously, any known crisp value $p$ can be seen as a trivial interval $\mathbf{p} = [p, p]$.

The interval JSP (IJSP) with makespan minimisation requires two arithmetic operations: addition and maximum. Given two intervals $\mathbf{a} = [\underline{a}, \overline{a}]$ and $\mathbf{b} =$

$[\underline{b}, \overline{b}]$, the addition and the maximum are given by

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

$$\max(\mathbf{a},\mathbf{b})=[\max(\underline{a},\underline{b}), \max(\overline{a},\overline{b})]. \tag{2}$$

Also, given the lack of a natural linear order in the set of closed intervals, to determine the schedule with the "minimal" makespan, we need an interval ranking method. We follow [20] and consider the rankings $\leqslant_{Lex1}, \leqslant_{Lex2}, \leqslant_{YX}, \leqslant_{MP}$ described therein. The first three rankings are admissible orders, that is, they are linear orders and they extend the partial order of intervals $\leqslant_2$ induced by the usual partial order in $\mathbb{R}^2$ defined as $[\underline{a}, \overline{a}] \leqslant_2 [\underline{b}, \overline{b}]$ if and only if $\underline{a} \leqslant \underline{b} \wedge \overline{a} \leqslant \overline{b}$ [49]. The ranking $\leqslant_{MP}$ is used in [23, 24] and it is equivalent to the ranking method used in [18]. It corresponds to the Hurwicz criterion for interval comparisons when $\gamma = 0.5$, which can be interpreted as a comparison halfway between pessimism and optimism [50]. Unlike the rest of rankings it is not an admissible linear order: it is coherent with $\leqslant_2$, but it is only a linear preorder, since it fails to be antisymmetric. It is also interesting to notice that $\leqslant_{Lex1}$ and $\leqslant_{Lex2}$ extend, respectively, the interval preorders Maximim and Maximax, which can be interpreted as corresponding to pessimistic and optimistic attitudes in a decision maker [50].

### 2.2. Interval Schedules

Given a schedule $\boldsymbol{s}$ for the IJSP, there exists a relative order $\pi$ for all tasks executed in the same machine. Conversely, from a processing order of tasks $\pi$ it is possible to obtain a schedule $\boldsymbol{s}$ as follows.

For a task $o \in O$, let $\mathbf{s_o}(\pi)$ and $\mathbf{c_o}(\pi)$ denote the starting and completion times of $o$ respectively, let $PM_o(\pi)$ and $SM_o(\pi)$ denote the tasks immediately preceding and succeeding $o$ in the machine $M(o)$ according to $\pi$, and let $PJ_o$ and $SJ_o$ denote the tasks immediately preceding and succeeding $o$ in its job. If $o$ were to be the first task in its machine or its job, we take its predecessor to be a dummy task 0 with $\mathbf{c_0}(\pi) = [0, 0]$. Then $\mathbf{s_o}(\pi)$ and $\mathbf{c_o}(\pi)$ are given by:

$$\mathbf{s_o}(\pi) = \max(\mathbf{s}_{PJ_o} + \mathbf{p}_{PJ_o}, \mathbf{s}_{PM_o(\pi)} + \mathbf{p}_{PM_o(\pi)}) \tag{3}$$

$$\mathbf{c_o}(\pi) = \mathbf{s_o}(\pi) + \mathbf{p_o}. \tag{4}$$

The makespan is given by the completion time of the last task to be processed according to $\pi$, that is, $\mathbf{C_{max}}(\pi) = \max_{o\in O}\{c_o(\pi)\}$. If there is no possible confusion regarding the processing order, we may simplify notation by writing $\mathbf{s_o}$, $\mathbf{c_o}$ and $\mathbf{C_{max}}$.

### 2.3. MILP Model

A mixed integer formulation of the IJSP can be derived by introducing a binary decision variable $x_{uv}$ for each pair of tasks $u, v \in O$ requiring the same machine, that is, such as that $M(u) = M(v)$. Variable $x_{uv}$ specifies whether $u$ precedes $v$ in the machine (in which case $x_{uv} = 1$) or not (in which case $x_{uv} = 0$).

Let $W$ be an arbitrarily large integer, and let $SJ_o$ denote the immediate successor of any task $o \in O$ in its job, that is, if $o = o_{j,l}$ is the $l$-th task of job $J_j$ for some $j \in \{1, \dots, n\}$ with $1 \leqslant l \leqslant m_j - 1$, then $SJ_o = o_{j,l+1}$. Then the IJSP can be formulated as follows:

$$\min[\underline{C}_{max}, \overline{C}_{max}] \tag{5}$$

s.t.

$$\underline{s}_o \geqslant 0 \qquad \forall o \in O \tag{6}$$

$$\overline{s}_o \geqslant 0 \qquad \forall o \in O \tag{7}$$

$$\underline{s}_o \leqslant \overline{s}_o \qquad \forall o \in O \tag{8}$$

$$\underline{s}_o + \underline{p}_o \leqslant \underline{C}_{max} \qquad \forall o \in O \tag{9}$$

$$\overline{s}_o + \overline{p}_o \leqslant \overline{C}_{max} \qquad \forall o \in O \tag{10}$$

$$\underline{s}_u + \underline{p}_u \leqslant \underline{s}_v \qquad \forall u, v \in O, v = SJ_u \tag{11}$$

$$\overline{s}_u + \overline{p}_u \leqslant \overline{s}_v \qquad \forall u, v \in O, v = SJ_u \tag{12}$$

$$\underline{s}_u + \underline{p}_u - W(1 - x_{uv}) \leqslant \underline{s}_v \qquad \forall u, v \in O, M(u) = M(v) \tag{13}$$

$$\overline{s}_u + \overline{p}_u - W(1 - x_{uv}) \leqslant \overline{s}_v \qquad \forall u, v \in O, M(u) = M(v) \tag{14}$$

$$\underline{s}_v + \underline{p}_v - W x_{uv} \leqslant \underline{s}_u \qquad \forall u, v \in O, M(u) = M(v) \tag{15}$$

$$\overline{s}_v + \overline{p}_v - W x_{uv} \leqslant \overline{s}_u \qquad \forall u, v \in O, M(u) = M(v) \tag{16}$$

$$x_{uv} \in \{0, 1\} \qquad \forall u, v \in O, M(u) = M(v) \tag{17}$$

The first constraints (6) and (7) ensure that it is not possible for the starting time of each task to take negative values, while constraint (8) ensures that the starting time $\mathbf{s}_o = [\underline{s}_o, \overline{s}_o]$ is an interval. Constraints (9) and (10) determine the interval makespan

$\mathbf{C_{max}} = [\underline{C}_{max}, \overline{C}_{max}]$. Constraints (11) and (12) ensure the precedence relations between consecutive tasks of the same job, while constraints (13) to (16) prevent the overlapping of tasks on the machine where they are to be executed: the first two constraints ensure that task $v$ starts after $u$ has finished if $u$ is to be processed before $v$ in their machine while the second pair of constraints ensure that task $u$ starts after $v$ is finished in the case that $u$ is not to be processed before $v$ in their machine. Finally, notice that the minimization of the objective function in (5) should be understood with respect to one of the ranking methods described in Section 2.1.

### 2.4. Robustness of solutions

The makespan obtained for the IJSP under uncertainty is an interval $\mathbf{C_{max}} = [\underline{C}_{max}, \overline{C}_{max}]$. This interval contains all the possible values for the makespan if tasks are executed in the relative order established by the schedule. In fact, when the solution is executed on a real scenario we obtain exact processing times for tasks $P^{ex} = \{p_o^{ex} \in [\underline{p}_o, \overline{p}_o], o \in O\}$ so after the execution the actual makespan $C_{max}^{ex} \in [\underline{C}_{max}, \overline{C}_{max}]$ can be known. Clearly, it is desirable that this executed makespan $C_{max}^{ex}$ does not differ much from the estimation provided by the a-priori makespan $\mathbf{C_{max}}$, and in particular, from its expected value.

This is the idea underlying the concept of $\epsilon$-robustness, first proposed in [51] in the context of stochastic scheduling and adapted to the IJSP in [23]. For a given $\epsilon \geqslant 0$, a schedule with makespan $\mathbf{C_{max}}$ is said to be *$\epsilon$-robust* in a real scenario $P^{ex}$ if the relative error made by the expected makespan $E[\mathbf{C_{max}}]$ with respect to the executed makespan $C_{max}^{ex}$ is bounded by $\epsilon$, that is:

$$\frac{|C_{max}^{ex} - E[\mathbf{C_{max}}]|}{E[\mathbf{C_{max}}]} \leqslant \epsilon, \tag{18}$$

where $E[\mathbf{C_{max}}]$ is the expected value of the uniform distribution on the interval $\mathbf{C_{max}}$, given by $E[\mathbf{C_{max}}] = 0.5(\underline{C}_{max} + \overline{C}_{max})$. Clearly, the interval schedule is considered to be more robust with a tighter bound $\epsilon$.

This robustness measure depends on a particular configuration $P^{ex}$ of task durations obtained in one execution of the predictive schedule $\boldsymbol{s}$. If no real data are available, as is the case with the usual synthetic benchmark instances for job shop, we may turn to Monte-Carlo simulations. We simulate $K$ possible configurations $P^k = \{p_o^k \in [\underline{p}_o, \overline{p}_o], o \in O\}$ using uniform probability distributions to sample durations for every task. For each configuration $k = 1, \ldots, K$ we compute the exact makespan $C_{max}^k$ that results from executing tasks according to the ordering provided by $\boldsymbol{s}$. Then, we can calculate the average $\epsilon$-robustness of the predictive schedule across the $K$ possible configurations, denoted $\overline{\epsilon}$, as follows:

$$\overline{\epsilon} = \frac{1}{K} \sum_{k=1}^{K} \frac{|C_{max}^k - E[\mathbf{C_{max}}]|}{E[\mathbf{C_{max}}]}, \tag{19}$$

This provides an estimate of the robustness of the solution $\boldsymbol{s}$ across different processing times configurations.

Simulation techniques such as the one proposed to compute the average robustness are not rare in complex optimisation settings. Simulation allows for modelling and artificially reproducing complex systems in a natural way within affordable computational effort [52]. In particular, our proposal is related to the dominant use of simulation in management science and operations research as a means for system analysis, where the intent is to mimic behaviour to understand or improve system performance [53].

## 3. ESABC: An Elitist Seasonal Artificial Bee Colony Algorithm

The Artificial Bee Colony (ABC) algorithm is a swarm-based metaheuristic search schema based on the foraging behaviour of honey bees which has been successfully applied with different modifications to a variety of optimisation problems [54]. However, when applied in its standard form to the interval job shop with makespan minimisation, it seems to lead to premature convergence, so a modification was introduced in [24] to improve its diversification capabilities and hence avoid this phenomenon. This method, denoted $ABC_{E3}$, has become the state-of-the-art method for this problem.

In the following we build on $ABC_{E3}$ to obtain further diversity in the search process which will eventually result in better solutions. This improvement is inspired by the influence of the season of the year and the temperature on the honeybees' behaviour. It also bears similarities with the mechanism used to escape from local optima in simulated annealing, one of the most popular metaheuristic search techniques with multiple applications in engineering [29].

In standard ABC, a hive of bees exploits a changing set of food sources (representing solutions to the

**Algorithm 1** Pseudocode of the Elitist Seasonal ABC

```
Require: An IJSP instance
Ensure: A schedule
  Generate a pool P_0 of food sources
  Best ← Best solution in P_0
  numIter ← 0
  k ← 0
  while numIter < maxIter do
     /* Employed bee phase */
     E ← B best solutions in P_k
     for each food source fs in P_k do
        fs' ← Select a solution from E at random
        fs_new ← Combine fs and fs' with probability p_emp
        if fs_new is better than fs and fs_new ≠ Best then
           fs ← fs_new
           if fs_new is better than Best then
              Best ← fs_new
              numIter ← 0
        else
           fs.numTrials ← fs.numTrials + 1
     /* Onlooker bee phase */
     for each food source fs in P_i do
        if fs.numTrials ⩽ NT_max then
           fs_new ← Apply onlooker op. to fs with prob. p_on
           if fs_new is better than fs and fs_new ≠ Best then
              fs ← new_fs
              if fs_new is better than Best then
                 Best ← fs_new
                 numIter ← 0
           else
              T_k ← Monotonic(k, α, NC, T_0) /* see Section 3.4.1 */
              μ ← Adaptive(fs, fs_new) /* see Section 3.4.2 */
              T ← μ · T_k
              r ~ U(0, 1)
              if r < T then
                 fs ← fs_new
              else
                 fs.numTrials ← fs.numTrials + 1
     /* Scout bee phase */
     for each food source fs in P_i do
        if fs.numTrials > NT_max then
           fs ← create new random food source
           fs.numTrials ← 0
           if fs_new is better than Best then
              Best ← fs_new;
              numIter ← 0
     numIter ← numIter + 1
     k ← k + 1
  return Best
```

problem) with two leading models of behaviour: recruiting rich food sources (i.e. keeping promising solutions) and abandoning poor ones (i.e. discarding bad solutions). It starts by generating and evaluating an initial pool $P_0$ of random food sources, and the best food source in the pool is assigned to the hive queen. Then, it iterates over a number of cycles, each consisting of three phases mimicking the behaviour of three types of foraging bees: employed, onlooker and scout bees. In the employed bee phase, each food source is assigned to an employed bee. This bee explores a new candidate food source between its current one and the best food source found so far, which is always assigned to the queen. After evaluating the candidate food source, if it is equivalent to the queen's one, it is discarded by the bee in order to maintain diversity in the pool. If the employed bee does not discard the candidate food source and it is better than the bee's current one, then the bee moves to the new food source. 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 newly found food source receives the same treatment as in the previous phase. In the scout bee phase, if the number of improvement trials of a food source reaches a maximum number $NT_{max}$, a scout bee finds a new food source to replace the former one in the pool. Finally, the algorithm terminates after a specific stopping criterion is met.

In our proposal, each food source *fs* encodes an IJSP solution using permutations with repetition [55]. The decoding of a food source follows an insertion strategy, consisting in iterating along the permutation and scheduling each task at its earliest feasible insertion position [23]. Thus, the creation of the initial pool of food sources $P_0$, or initial hive, consists on generating a set of random permutations with repetition that lead to feasible solutions according to the encoding and decoding strategies from [23]. The nectar amount of each food source, representing the quality of that solution, is inversely proportional to the makespan of the schedule it represents, so in terms of comparisons, a food source $fs$ is considered to be better than another $fs'$ if $\mathbf{C_{max}}(fs) \leqslant_R \mathbf{C_{max}}(fs')$ for a given ranking $R$ on intervals. As stopping criterion, the search terminates after a number *maxIter* of consecutive iterations without finding a food source that improves the queen's one (denoted *Best*). The pseudo-code of the resulting ABC is given in Algorithm 1. The following subsections explain each of the three phases in more detail.

### 3.1. Employed Bee Phase

Originally, the employed bees' search is always guided by the queen's food source. That is, each bee combines its food source with the best one in the hive to find a new one. However, this strategy may in some occasions cause a lack of diversity and lead to premature convergence [48]. To address this issue, a modification of the original strategy was proposed in [24] so the guiding food source was instead selected from an elite group. Three different ways of defining the elite group were considered and evaluated. $\mathsf{Elite}_1$ consists in selecting only the best food source in the hive, thus representing the standard employed bee phase. $\mathsf{Elite}_2$ selects the best food source from those food sources with the highest number of trials, which represent local optima and therefore promising areas to explore. In $\mathsf{Elite}_3$, the elite group is made up of the best $B$ food sources in the current hive and, for each employed bee, a guiding solution from this group is chosen at random. This strategy is equivalent to the standard ABC if $B = 1$ and chooses a random food source from the hive if $B$ is as large as the maximum number of food sources. Therefore, $B$ is a parameter of the algorithm ranging between 1 and the maximum number of food sources that allows to adjust the diversity. The experimental results presented in [24] suggest that the third strategy is the most interesting one.

Once an elite solution *fs*$'$ is selected, a recombination operator is applied with probability $p_{emp}$ to the bee's current food source *fs* and the selected elite one *fs*$'$. In our case, we use three well-known operators for our encoding: Generalised Order Crossover (GOX), Job-Order Crossover (JOX) and Precedence Preservative Crossover (PPX). If the resulting food source $fs_{new}$ is better than the bee's current one *fs* and distinct from the best-so-far source $Best$, the bee leaves its current food source and adopts the new one for the next iteration. Otherwise, the bee remains at its original source, increasing in one the counter *fs.numTrials* of improvement trials.

### 3.2. Onlooker Bee Phase with Seasonal Behaviour

In this phase, onlooker bees with a food source that is not exhausted (i.e. it has not reached the maximum number of improvement trials) search in the neighbourhood of their food source with a probability $p_{on}$. Typically, the neighbouring food sources are obtained by performing a small change on the original one. In our case, food sources encode solutions as permutations, so we use one of the following operators for permutations: Swap, Inversion or Insertion. In the original ABC, if the neighbour is better than the original food source, it will replace it becoming the new food source. Otherwise, the onlooker bee will remain in the original food source, increasing in one the counter *fs.numTrials* of improvement trials. If *fs.numTrials* reaches a maximum $\mathrm{NT}_{max}$, then the food source *fs* will be considered as exhausted.

However, a behaviour where bees only move to improving food sources is likely to reduce the search ability of onlooker bees by getting them stuck in local optima. The Scout Bee Phase (Section 3.3) is supposed to solve this issue by discarding those food sources that have not improved after a number of attempts, and replacing them by new random ones. Consequently, new diversity is introduced in the population, but it is a diversity which initial quality might not be adequate to make an effective contribution to the resolution of the problem. Therefore, it seems advisable to allow for certain moves to worsen food sources in the onlooker bee phase with the goal of escaping local optima and avoiding premature convergence.

In nature, honeybees already have this mechanism in place. Seasons have an influence on the honeybees' thermal behaviour, which might be connected with seasonal shifts of temperature regulated by the honeybee colony. When spring comes, new sources of nectar become available. At that moment, the bees increase their activity to explore larger areas and find the best food sources. This is the time when they accumulate nectar in big amounts and reproduce. By summer, when the daylight period is the longest, bees have already located the best food sources and they make use of the longer days for maximum foraging of nectar. In this period, they focus on honey production and prepare for winter. When freezing temperatures arrive in winter, bees cease their activity in wait for the new spring. In an optimisation context, this resembles a dynamic strategy: at the first stages (spring and reproduction) the swarm focuses on exploration, and as time advances the focus shifts towards exploitation (summer) until it reaches a freezing state (winter). This is analogous to simulated annealing, where a *temperature* parameter allows for more exploration at the beginning and reduces it as the algorithm advances. The *inertia* in particle swarm optimisation also has a similar effect, enabling more movement in the first stages, and slowing down when the particles converge to promising areas. A similar idea is also found in memetic algorithms, where local search is combined with population-based evolutionary algorithms and applied with certain probability to balance exploration and exploitation [29, 56].

To mimic this behaviour in our algorithm, we take the food source *fs* of an onlooker bee and a neighbouring one $fs_{new}$ currently being explored. As in the standard onlooker bee phase, if $fs_{new}$ is better than *fs*, it will automatically replace it. However, if $fs_{new}$ is equal or worse than *fs*, the replacement will depend on a probability that is proportional to the quality of $fs_{new}$ and the environment's temperature, which decreases with each new iteration or temperature cycle $k$ (from spring to winter). This replacement strategy is similar to the Metropolis algorithm for simulation of physical systems subject to a heat source [57]. As a side effect, allowing for more substitutions and with higher quality solutions in the second phase of ABC implies less replacements in the Scout Bee Phase, with a positive impact on the average quality of the new solutions. The different seasonal models considered for temperature changes are discussed in Section 3.4.

### 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* $= 0$. Random food sources are generated by creating a random permutation with repetitions that leads to feasible solutions according to the encoding and decoding strategy from [23], as explained above for the initial population.

### 3.4. Seasonal Strategies

To model how temperatures change in our seasonal environment, we get inspiration from simulated annealing algorithms, where temperature is also used to lead the system from states of high energy to states of low energy. To simulate that behaviour, these methods start from an initial temperature $T_0$, which then decreases along the evolution of the algorithm following a predefined cooling strategy. It is desirable that $T_0$ is set to a high enough value, so that solutions generated in the first iterations of the algorithm have a high probability of being accepted independently of their quality, thus encouraging exploration. Regarding the decrease in temperature, several strategies have been proposed in the literature [29, 58, 59]. In this work we apply two types of strategy: monotonic and adaptive, selected from the most promising ones from [60]. This is reflected in Algorithm 1 with two functions used to update the temperature at each iteration, *Monotonic* and *Adaptive*.

#### 3.4.1. Monotonic Strategies

We consider two subcategories of monotonic cooling strategies: multiplicative and additive. In multiplicative strategies, the temperature $T_k$ at iteration $k$ is obtained as the product of the initial temperature $T_0$ and a decreasing factor $\Delta_k$. This factor starts with a value equal to 1, so the system starts with temperature $T_0$ and decreases over time depending on a parameter $\alpha$. The following four cooling strategies belong in this category (in each case, we indicate the most typical $\alpha$ values between brackets):

- Exponential multiplicative cooling (ExpM):

$$T_k = \alpha^k \cdot T_0 \qquad (0.8 \leqslant \alpha \leqslant 0.9) \tag{20}$$

- Logarithmic multiplicative cooling (LogM):

$$T_k = \frac{1}{1+\alpha \log(1+k)} \cdot T_0 \qquad (\alpha \geqslant 1) \tag{21}$$

- Linear multiplicative cooling (LinM):

$$T_k = \frac{1}{1+\alpha k} \cdot T_0 \qquad (\alpha \geqslant 0) \tag{22}$$

- Quadratic multiplicative cooling (QadM):

$$T_k = \frac{1}{1+\alpha k^2} \cdot T_0 \qquad (\alpha \geqslant 0) \tag{23}$$

In additive strategies, two new values need to be taken into account: the number of cooling cycles $NC$ and the final temperature $T_{NC}$. To find the value of $T_k$ at each iteration $k$, a value $\Delta_k$ that decreases over time is added to the final temperature $T_{NC}$. Ideally, $\Delta_0 = (T_0 - T_{NC})$, so the algorithm starts with temperature $T_0$ and decreases over time until it reaches $T_{NC}$. In this category, we consider three cooling strategies.

- Linear additive cooling (LinAd):

$$T_k = T_{NC} + (T_0 - T_{NC})(\frac{NC-k}{NC}) \tag{24}$$

- Quadratic additive cooling (QadAd):

$$T_k = T_{NC} + (T_0 - T_{NC})(\frac{NC-k}{NC})^2 \tag{25}$$

- Trigonometric additive cooling (TrigAd):

$$T_k = T_{NC} + \frac{1}{2}(T_0 - T_{NC})(1+cos(\frac{k\pi}{NC})) \tag{26}$$

Independently of the strategy type, values $k$ and $T_0$ are necessary. In addition, a parameter $\alpha$ is needed for multiplicative strategies and a parameter $NC$, for additive ones. This is indicated in Algorithm 1 by including these 4 values as parameters of *Monotonic* function. In the experimental analysis, we shall determine which monotonic strategy works better for our problem.

#### 3.4.2. Adaptive strategies

Monotonic cooling strategies only take into account the number of iterations of the algorithm. In our case, the temperature of the environment and, with it, the bees behaviour will only depend on the time of the year if a monotonic strategy is used. However, it is reasonable to assume that onlooker bees will be more willing to explore if the food sources around them are poor and they will be more conservative if their neighbourhood is reasonably rich. This would correspond to a non-monotonic adaptive cooling where the temperature $T_k$ is multiplied by an adaptive factor $\mu$ that depends on the difference between the amount of nectar in the current food source and the nectar of the best food source found so far. In our case, $\mu$ will depend on the difference between the quality of the neighbouring food source $fs_{new}$ and the current food source $fs$ of the bee. The introduction of $\mu$ implies that onlooker bees whose neighbouring food sources are much worse that their current one will have a higher temperature, and therefore more exploration capabilities than those that are already in promising neighbourhoods. Typically, the $\mu$ factor is calculated as follows, so $(1 \leqslant \mu \leqslant 2)$:

$$T = \mu T_k = (1 + \frac{f(S) - f^*}{f(S)})T_k, \tag{27}$$

where $f(S)$ denotes the fitness of the new solution ($fs_{new}$ in our case) and $f^*$ is the best fitness found by the bee so far ($fs$ in our case). The need of these values is expressed in Algorithm 1 by including them as parameters of *Adaptive* function.

To enhance this feature, we propose a new variant of the previous one where $(1 \leqslant \mu \leqslant 4)$, meaning that the temperature can get even higher if the quality of the current solution is too bad in comparison with the best achieved solution:

$$T = \mu T_k = (1 + \frac{f(S) - f^*}{f(S)})^2 T_k \tag{28}$$

In the experimental analysis, we shall determine which adaptive strategy works better for our problem.

## 4. Experimental Results

In this section, the different seasonal strategies are evaluated and compared with respect to the best-known solutions in the literature. Also, a study is carried out to determine the advantage (if any) of considering uncertainty during the optimisation process. Different ranking methods for intervals are compared to assess which one yields more robust solutions. Finally, a sensitivity analysis tries to assess the impact of having larger intervals on the solutions.

The experimental analysis is performed on benchmark instances from the literature. Two sets of benchmark instances for IJSP have been proposed in the past: 17 instances in [22] and 12 instances in [23]. The first set of instances is an extension to the IJSP of the well-known crisp instances ORB1 to ORB5 (size $10 \times 10$), La16 to La20 ($10 \times 10$), La21 to La25 ($15 \times 10$), and ABZ5, ABZ6 ($10 \times 10$). Here, the values between brackets of the form $n \times m$ refer to the instance size, where $n$ is the number of jobs and $m$ is the number of resources. To adapt the instances to the interval framework, an interval $\mathbf{p_o} = [p_o, p_o + \delta_o]$ was generated from each original deterministic processing time $p_o$ by adding a number $\delta_o \in [3, 8]$. However, the original deterministic instances (with the exception of La21, La24 and La25) were already labelled as *easy* more than thirty years ago [61]. More recently, fuzzy versions of these instances have also proved to be easy to solve with current metaheuristic techniques [62]. The second set of instances from [23] is also obtained by extending classical deterministic benchmark instances to the interval framework. However, in this case it contains larger instances: FT10 ($10 \times 10$), FT20 ($20 \times 5$), La21, La24, La25 ($15 \times 10$), La27, La29 ($20 \times 10$), La38, La40 ($15 \times 15$), ABZ7, ABZ8, and ABZ9 ($20 \times 15$). The interval $\mathbf{p_o} = [(1 - \delta_o)p_o, (1 + \delta_o)p_o]$ was generated from each deterministic processing time $p_o$, with $\delta_o$ a random value in the interval $[0, 0.15]$. This set contains the 10 instances considered as *tough* in [61], making it more suitable for evaluating our solving method. Thus we shall use this second set in the following experimental analysis.

All experiments are done using a C++ implementation on a PC with Intel Xeon Gold 6240 processor at 2.6 Ghz and 128 Gb RAM with Linux (CentOS v6.10). For every experiment, we consider 30 runs of the method on each instance, so the resulting data are representative of the method's performance.

### 4.1. Parameter analysis

A preliminary parametric study is conducted to find the best parameter configuration for $ESABC$. To compare the behaviour of the seven proposed monotonic seasonal strategies, we consider seven different variants of $ESABC$, namely $ESABC_{ExpM}$, $ESABC_{LogM}$, $ESABC_{LinM}$, $ESABC_{QadM}$, $ESABC_{LinAd}$, $ESABC_{QadAd}$ and $ESABC_{TrigAd}$. The stopping criterion for all variants is set to $maxIter = 25$ consecutive iterations without improving the best solution found so far. Following the results obtained in [24] for $ABC_{E3}$, the hive size is set to 250 food sources. For fairer comparisons, we use the $\leqslant_{MP}$ ranking method, which is the ranking used in [22–24]. The following values are tested for each parameter:

- Employed bee phase operator: GOX, JOX, PPX
- Employed bee phase probability: 0.5, 0.75, 1.0
- Onlooker bee phase operator: Insertion, Inversion, Swap
- Onlooker bee probability: 0.5, 0.75, 1.0
- Max. number of tries: 10, 15, 20
- Size of the elite: 40, 50, 60

In addition, every seasonal strategy has its own specific parameters that need to be tuned: the cooling constant $\alpha$ for multiplicative strategies, the number of cooling cycles $NC$ for additive ones and the initial temperature $T_0$ for all of them. Since these parameters have different impact on the seasonal strategies (i.e. logarithmic vs. quadratic), the range of values to test is chosen accordingly and disclosed in Table 1. Each row corresponds to the variant of $ESABC$ obtained by incorporating a seasonal strategy as explained above and each column indicates the range of values considered in the parametric analysis for each parameter of the seasonal strategy (in the table, the acronym "N.A." in a cell indicates that the parameter labelling the corresponding column is not applicable to the seasonal strategy labelling the row).

Finally, the adaptive strategies from Section 3.4.2 are also tested on each variant of $ESABC$. Table 2 reports the best parameter values for each variant of $ESABC$. Each row corresponds to one parameter and each column, to one variant of the algorithm. The values in the sixth row, corresponding to the adaptive strategy, are $Standard$ if equation (27) is used, *Quadratic*, if equation (28) is used, or *Disabled*, if no adaptive strategy is used. It would seem that the *Quadratic adaptive strategy* proposed in this work as a variant of the one in [59] finds better results when

Table 1
Range of parameter values for each seasonal strategy.

| $ESABC$ Variant | $\alpha$ | $NC$ | $T_0$ |
|---|---|---|---|
| $ESABC_{ExpM}$ | 0.80, 0.85, 0.90 | N.A. | 1.0, 2.0, 10.0 |
| $ESABC_{LogM}$ | 1.50, 2.00, 2.50 | N.A. | 1.0, 1.5, 2.0 |
| $ESABC_{LinM}$ | 0.02, 0.50, 1.00 | N.A. | 1.0, 1.5, 2.0 |
| $ESABC_{QadM}$ | 0.02, 0.05, 0.10 | N.A. | 1.0, 1.5, 2.0 |
| $ESABC_{LinAd}$ | N.A. | 100, 150, 200 | 0.9, 1.0, 1.1 |
| $ESABC_{QadAd}$ | N.A. | 100, 150, 200 | 0.9, 1.0, 1.1 |
| $ESABC_{TrigAd}$ | N.A. | 25, 50, 100 | 0.8, 1.0, 1.2 |

combined with the different multiplicative monotonic cooling variants, appearing to be the best option for all of them.

The results obtained with the best setup of each variant are detailed in Table 3. Each row corresponds to one of the instances; the first column contains the name of the instance and the remaining seven columns correspond to the results obtained by $ESABC$ using the different seasonal strategies presented in Section 3.4, so for each seasonal strategy $se$, the heading of the column $ESABC_{se}$ denotes the variant of $ESABC$ using $se$. For each variant, the average $E[C_{max}]$ across 30 runs is reported. Values in bold highlight the best variant for each instance. An automated statistical test is performed to find significant differences between the variants. If the samples pass a Shapiro-Wilk test of normality, an analysis of variance model (ANOVA) is carried out, followed by Tukey's Honest Significant Difference to display the results of all pairwise comparisons in the tested groups. If the test of normality is not passed, a Kruskall-Wallis rank sum test is performed followed by a multiple comparison to determine which groups are different. The tests are configured with a $p$-value of 0.05. The results show no significant differences between the different variants, except on instance $ABZ8$ where $ESABC_{QadM}$ and $ESABC_{QadAd}$ are significantly different than the others. For the sake of choosing one of the variants for further tests, $ESABC_{LinM}$ obtains the best average result in 5 out of 12 instances, so it is the one we shall consider in the following sections. For the sake of clarity, in further sections we will refer to $ESABC_{LinM}$ simply as $ESABC$.

### 4.2. Comparison with the state of the art

Table 4 reports the comparison of our method with the $GA$ and $ABC_{E3}$ methods from [23] and [24] respectively, which to the best of our knowledge represents the most successful methods in the literature for our problem. $GA$ was shown in [23] to outperform the population-based neighbourhood search algorithm from [22], which was the most competitive method for this problem up to that moment. Regarding $ABC_{E3}$, the experimental results from [24] already showed that it obtained better results than a standard ABC and was comparable, if not better, than $GA$. We therefore conclude that $GA$ and $ABC_{E3}$ can be considered to be the state of the art for IJSP.

The comparison is made in terms of Relative Errors ($RE$) with respect to the best-known lower bound of the problem, shown in the second column of the table. For the sake of clarity, Table 4 shows these errors as percentages. Columns $Best$ and $Avg$. show the best and average relative errors obtained in 30 runs of each method w.r.t. the given lower bound. Values in brackets represent the standard deviation (SD). Finally, the column $Time$ contains the average runtime of each algorithm. Best average values of each instance are in bold. In terms of the best found values, $ABC_{E3}$ is outperformed or matched by $ESABC$ on every instance except $La24$ and $La40$. However, what is more representative is the average behaviour of each method. In this case, $ESABC$ obtains the best results in 11 out of 12 instances, reducing the Relative Error w.r.t. LB in 13% when compared to $ABC_{E3}$ and 43% when compared to $GA$. Following the procedure explained in Section 4.1 for statistical tests, a Kruskal-Wallis test reveals that the differences between $ESABC$ and the other methods are significant in all instances except $FT20$, $La38$ and $La40$, where there is no significant difference between $ESABC$ ad $ABC_{E3}$. Regarding the runtime, $ESABC$ appears to take a bit longer than $ABC_{E3}$ to converge, which in turn takes longer than $GA$. One may think that the different runtimes come from a difference in algorithmic complexity, but the three approaches have similar asymptotic complexity, which is given by the most costly operation. In this case, the evaluation of a new solution, that being an individual (GA) or a food source (ABC). So time differences might be related to the algorithm's behaviour. This is expected, and good in a sense, since the main idea of including seasonal strategies is to avoid premature convergence and therefore spend more time exploring to reach more promising areas of the search space. $ESABC$ seems to achieve this, taking slightly more time but reaching better solutions than $ABC_{E3}$. This is illustrated on Figure 1, where both $ABC_{E3}$ and $ESABC$ are left to converge for 150 iterations on instance $ABZ7$. We can see that $ABC_{E3}$ converges too quickly and stops improving, while $ESABC$ takes a

Table 2
Best parameter setup for variants of $ESABC$ with different seasonality strategies.

| *Parameter/Variant* | $ESABC_{ExpM}$ | $ESABC_{LogM}$ | $ESABC_{LinM}$ | $ESABC_{QadM}$ | $ESABC_{LinAd}$ | $ESABC_{QadAd}$ | $ESABC_{TrigAd}$ |
|---|---|---|---|---|---|---|---|
| Employed operator | JOX | JOX | JOX | JOX | JOX | JOX | JOX |
| Employed prob. $p_{emp}$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Onlooker operator | Insertion | Insertion | Insertion | Insertion | Insertion | Insertion | Swap |
| Onlooker prob. $p_{on}$ | 0.5 | 1 | 0.75 | 0.5 | 0.75 | 0.5 | 0.75 |
| Max. tries | 20 | 20 | 20 | 20 | 15 | 20 | 10 |
| Elite size | 40 | 50 | 50 | 50 | 50 | 40 | 40 |
| Adaptive strategy | Quadratic | Quadratic | Quadratic | Quadratic | Quadratic | Disabled | Standard |
| $\alpha$ | 0.9 | 2 | 0.5 | 0.02 | N.A. | N.A. | N.A. |
| $NC$ | N.A. | N.A | N.A. | N.A. | 150 | 150 | 100 |
| $T_0$ | 1 | 2 | 1 | 1 | 0.9 | 1 | 1 |
| $T_{NC}$ | N.A. | N.A. | N.A. | N.A. | 0 | 0 | 0 |

Table 3
Average $E[C_{max}]$ values obtained with different seasonality strategies for $ESABC$.

| *Instance* | $ESABC_{ExpM}$ | $ESABC_{LogM}$ | $ESABC_{LinM}$ | $ESABC_{QadM}$ | $ESABC_{LinAd}$ | $ESABC_{QadAd}$ | $ESABC_{TrigAd}$ |
|---|---|---|---|---|---|---|---|
| ABZ7 | **698.55** | 702.23 | 700.12 | 699.75 | 703.67 | 698.90 | 700.88 |
| ABZ8 | 717.37 | 719.47 | **715.62** | 716.45 | 716.07 | 720.18 | 717.13 |
| ABZ9 | **733.82** | 734.67 | 734.98 | 734.73 | 734.35 | 737.85 | 736.28 |
| FT10 | 959.42 | 957.57 | 957.97 | **956.23** | 957.82 | 962.43 | 960.83 |
| FT20 | 1186.78 | **1184.08** | 1185.73 | 1184.68 | 1185.73 | 1187.68 | 1185.72 |
| La21 | 1089.48 | 1088.05 | **1087.47** | 1087.57 | 1087.70 | 1090.80 | 1090.98 |
| La24 | 977.90 | **977.50** | 981.28 | 980.07 | 981.35 | 978.57 | 978.98 |
| La25 | 1004.58 | 1003.90 | **1003.78** | 1004.12 | 1005.78 | 1004.58 | 1007.38 |
| La27 | 1288.57 | 1285.87 | **1285.85** | 1289.03 | 1288.18 | 1288.05 | 1294.13 |
| La29 | 1237.50 | 1239.30 | 1232.98 | 1233.93 | 1235.80 | **1231.98** | 1235.70 |
| La38 | 1272.57 | 1271.95 | **1265.68** | 1272.63 | 1274.27 | 1271.50 | 1273.85 |
| La40 | 1271.98 | **1270.85** | 1272.23 | 1273.55 | 1273.98 | 1271.87 | 1271.88 |

bit longer, but reaches better results. In any case, the elapsed time can be considered reasonable taking into account that it remains below 10 seconds for all instances.

### 4.3. Comparison between different ranking methods

After comparing the different versions of $ESABC$ among themselves and with a state-of-the-art method, we study the effect of using different ranking methods for intervals during the optimisation process. For this study, we focus on $ESABC$ and create four variants of it using the four rankings introduced in section 2.1. A crisp version of $ESABC$, $ESABC_c$ is also considered, where the algorithm is fed a version of the instances where the intervals are replaced by their expected value, thus becoming crisp instances. The idea is to see if it is worth considering the uncertainty during the optimisation process, or if solving the associated crisp instance yields similar results. For a fair comparison, $ESABC$ parameter values are tuned to obtain the best setup for each ranking method following the same process than in Section 4.1. Table 5 contains the final parameter values that result from this process. Each row corresponds to one parameter and there are five columns, the first one corresponding to the parameter setting for the crisp version of $EASBC$ and the remaining four, to the parameter setting for $ESABC$ using each of the four ranking methods (indicated as sub-index of the name of the variant). The table shows that despite the change of ranking, the best values for the seasonal strategies are the same, as well as the operator for the employed bee phase. Only the elite size

Table 4

Comparison between $GA$, $ABC_{E3}$ and $ESABC$ in terms of best and average RE (%) w.r.t. LB and runtime

| | | $GA$ | | | $ABC_{E3}$ | | | $ESABC$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *Instance* | *LB* | *Best* | *Avg.* *(SD)* | *Time* | *Best* | *Avg.* *(SD)* | *Time* | *Best* | *Avg.* *(SD)* | *Time* |
| ABZ7 | 656 | 6.33 | 12.50 (1.96) | 1.8 | 5.26 | 7.32 (1.11) | 4.5 | 4.19 | **6.73** (1.74) | 6.5 |
| ABZ8 | 645 | 11.32 | 18.48 (2.14) | 1.8 | 8.99 | 12.06 (1.16) | 4.2 | 8.68 | **10.95** (1.74) | 7.7 |
| ABZ9 | 661 | 13.01 | 17.97 (2.41) | 2.2 | 9.68 | 13.13 (1.57) | 6.0 | 8.55 | **11.19** (1.55) | 8.7 |
| FT10 | 930 | 1.83 | 5.23 (2.12) | 0.5 | 1.08 | 4.11 (1.28) | 1.6 | 0.59 | **3.01** (1.21) | 1.8 |
| FT20 | 1165 | 1.46 | 4.35 (1.36) | 0.7 | 0.69 | **1.73** (0.66) | 2.7 | 0.69 | 1.78 (0.63) | 2.9 |
| La21 | 1046 | 3.15 | 5.01 (1.29) | 1.1 | 2.58 | 5.01 (1.29) | 1.8 | 2.06 | **3.96** (0.72) | 2.9 |
| La24 | 935 | 4.06 | 6.34 (1.57) | 0.8 | 2.25 | 5.06 (1.25) | 2.7 | 3.53 | **4.95** (0.96) | 2.6 |
| La25 | 977 | 1.94 | 5.11 (2.39) | 1.0 | 1.94 | 3.88 (0.89) | 2.4 | 1.33 | **2.74** (0.95) | 3.0 |
| La27 | 1235 | 4.57 | 10.22 (2.00) | 1.3 | 2.75 | 4.66 (0.99) | 4.1 | 2.75 | **4.12** (1.02) | 5.8 |
| La29 | 1152 | 11.11 | 14.23 (1.62) | 1.1 | 5.51 | 8.65 (1.31) | 4.4 | 4.99 | **7.03** (1.09) | 6.7 |
| La38 | 1196 | 6.02 | 9.16 (2.28) | 1.4 | 4.52 | 6.88 (1.47) | 6.0 | 3.01 | **5.83** (1.40) | 7.2 |
| La40 | 1222 | 5.07 | 8.74 (2.33) | 1.2 | 1.88 | 4.21 (1.13) | 3.0 | 2.74 | **4.11** (0.91) | 3.7 |

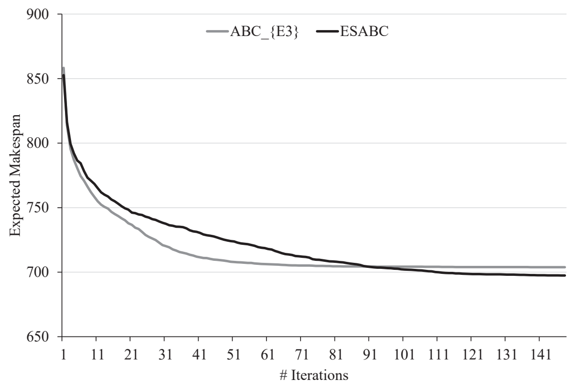



Fig. 1. Best solution found by $ABC_{E3}$ and $ESABC$ at each iteration on instance $ABZ7$

and the onlooker bee phase operators get a finer tuning depending on the ranking method in use.

To establish a comparison between all variants, we must take into account that comparisons between intervals depend on the chosen ranking. Since we are comparing the different rankings themselves, choosing a specific one as basis for comparison would be unfair and favour the $ESABC$ variant that used that ranking during the optimisation process. To avoid this problem, the $\overline{\epsilon}$-robustness measure is adopted to compare solutions based on their quality as predictive schedules. Every variant of $ESABC$ returns an expected makespan value and a task processing order per instance and run. This order can then be evaluated on 1000 deterministic realisations of the instance to find the $\overline{\epsilon}$ value. Table 6 reports the $\overline{\epsilon}$ robustness value obtained with $ESABC$ using the different ranking method on each instance. Next to the average $\overline{\epsilon}$, it also reports the standard deviation between brackets. To improve the clarity of the table, the original $\overline{\epsilon}$ values are rescaled multiplying them by 1000. For each instance, values in bold highlight the most robust method (that is, that with the smallest $\overline{\epsilon}$) and grey cells correspond to methods with no significant differences with the best method after running a Kruskal-Wallis statistical test as explained above.

It is clear that the solutions to the associated deterministic problem are considerably less robust than the solutions obtained incorporating the knowledge about interval uncertainty to the search. In comparison with $ESABC_{Lex2}$, which can be seen as the most robust of $ESABC$ variants, solving the crisp instance gets solutions that are 28% less robust. When comparing the use of different ranking methods, $ESABC_{Lex2}$ obtains the most robust solutions, reaching the best values in 11 out of the 12 analysed instances. $ESABC_{YX}$ obtains very similar results, not having significant differences with $ESABC_{Lex2}$ in 9 of the instances. These results are aligned to the ones obtained by the Genetic Algorithm in [23], showing that independently on the optimisation method, using $\leqslant_{Lex2}$ tends to be more robust. To better illustrate the results, Figure 2 shows the boxplots of the resulting $\overline{\epsilon}$ values on one representative instance of each group: $ABZ7$, where no significant difference is found between the ranking methods, $ABZ9$ where $ESABC_{Lex2}$ is significantly better than the oth-

Table 5

Best parameter setup for $ESABC$ using different interval ranking methods

| *Parameter* | $ESABC_c$ | $ESABC_{MP}$ | $ESABC_{Lex1}$ | $ESABC_{Lex2}$ | $ESABC_{YX}$ |
|---|---|---|---|---|---|
| Employed operator | JOX | JOX | JOX | JOX | JOX |
| Employed prob. $p_{emp}$ | 1 | 1 | 1 | 1 | 1 |
| Onlooker operator | Swap | Insertion | Swap | Insertion | Insertion |
| Onlooker prob. $p_{on}$ | 1 | 0.75 | 0.5 | 1 | 1 |
| Max. tries | 20 | 20 | 20 | 20 | 20 |
| Elite size | 60 | 50 | 60 | 60 | 50 |
| Adaptive strategy | Quadratic | Quadratic | Quadratic | Quadratic | Quadratic |
| $\alpha$ | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 |
| $T_0$ | 1 | 1 | 1 | 1 | 1 |

Table 6

Average $\overline{\epsilon}(\times 1000)$ values for $ESABC$ using different ranking methods (standard deviation in brackets)

| *Instance* | $ESABC_c$ | $ESABC_{MP}$ | $ESABC_{Lex1}$ | $ESABC_{Lex2}$ | $ESABC_{YX}$ |
|---|---|---|---|---|---|
| ABZ7 | 12.26 (1.75) | 8.98 (1.29) | 8.99 (1.03) | **8.03** (1.46) | 8.89 (0.99) |
| ABZ8 | 9.86 (1.58) | 7.67 (1.09) | 7.81 (1.06) | 7.46 (1.19) | **7.37** (1.07) |
| ABZ9 | 10.61 (1.38) | 7.13 (0.92) | 7.67 (0.79) | **6.46** (1.02) | 7.13 (1.09) |
| FT10 | 12.00 (1.81) | 9.10 (1.20) | 9.81 (1.19) | **9.00** (1.11) | 9.61 (1.15) |
| FT20 | 9.03 (1.05) | 7.58 (0.43) | 7.80 (0.72) | 7.53 (0.44) | **7.52** (0.36) |
| La21 | 13.64 (1.42) | 10.28 (0.94) | 10.48 (0.78) | **9.12** (0.85) | 10.24 (1.23) |
| La24 | 15.08 (2.15) | 11.94 (1.38) | 12.42 (1.30) | **11.41** (1.81) | 11.97 (1.63) |
| La25 | 12.72 (1.71) | 10.20 (0.80) | 10.57 (0.66) | **9.56** (1.25) | 10.07 (1.06) |
| La27 | 13.01 (1.83) | 9.41 (1.15) | 9.70 (1.08) | **8.78** (1.15) | 9.33 (0.92) |
| La29 | 12.35 (1.63) | 9.52 (0.91) | 9.64 (1.24) | **8.57** (1.13) | 9.31 (1.26) |
| La38 | 13.78 (1.23) | 9.74 (1.69) | 10.58 (1.17) | **8.37** (1.79) | 9.30 (1.53) |
| La40 | 13.46 (2.24) | 10.17 (0.98) | 10.04 (0.83) | **9.21** (0.99) | 10.22 (1.15) |

ers and $La29$, where $ESABC_{Lex2}$ and $ESABC_{YX}$ behave similarly, but better than the others.

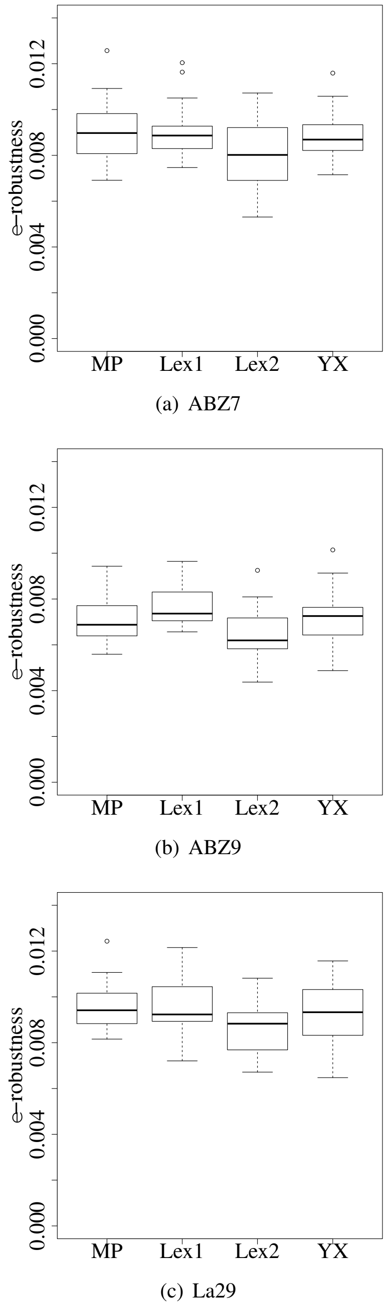
To understand why $ESABC_{Lex2}$ obtains the most robust results, Table 7 shows the expected makespan values across 30 runs of $ESABC_{Lex2}$ and $ESABC_{MP}$. For each of the obtained makespan intervals, we also measure the level of uncertainty of the solutions. As explained in Section 2, the makespan interval $\mathbf{C_{max}} = [\underline{C}_{max}, \overline{C}_{max}]$ represents the set of all possible values for $C_{max}$ in a real execution of the solution. In other words, we can define a possibility measure $Pos_{\mathbf{C_{max}}}$ as:

$$Pos_{\mathbf{C_{max}}}(x) = \begin{cases} 1 & \text{if } \underline{C}_{max} \leqslant x \leqslant \overline{C}_{max} \\ 0 & \text{otherwise} \end{cases} \quad (29)$$

In that setting, we use the $U$-uncertainty measure [63] to evaluate the uncertainty of $\mathbf{C_{max}}$:

$$\begin{aligned} H(Pos_{\mathbf{C_{max}}}) &= \log_2 |\mathbf{C_{max}}| \\ &= \log_2 |\overline{C}_{max} - \underline{C}_{max} + 1| \end{aligned} \quad (30)$$

The average $H(Pos_{\mathbf{C_{max}}})$ is also included in Table 7 together with the average runtime. Values in bold highlight the best $E[\mathbf{C_{max}}]$ and $H(Pos_{\mathbf{C_{max}}})$ value obtained for each instance. We can see how the level of uncertainty obtained by $ESABC_{Lex2}$ tends to be smaller, showing that using this ranking can obtain results that reduce the amount of uncertainty and therefore, real executions are more predictable. This is illustrated in Figure 3, where 1000 realisations of the best solution obtained by $ESABC_{Lex2}$ and $ESABC_{MP}$ are depicted in an histogram. Thick black lines show the expected makespan (predictive value) and thinner dotted




(a) ABZ7

(b) ABZ9

(c) La29

Fig. 2. $\overline{\epsilon}$-robustness of schedules obtained with the different variants of $ESABC_{LinM}$ on instances ABZ7, ABZ9 and La29.

lines the interval bounds. We can see that reducing the uncertainty measure translates into a thinner histogram in the figure. This eventually means that solutions are more robust, as we had seen in Table 6. On the other hand, one would expect $ESABC_{MP}$ to obtain better results in terms of expected makespan.

Table 7

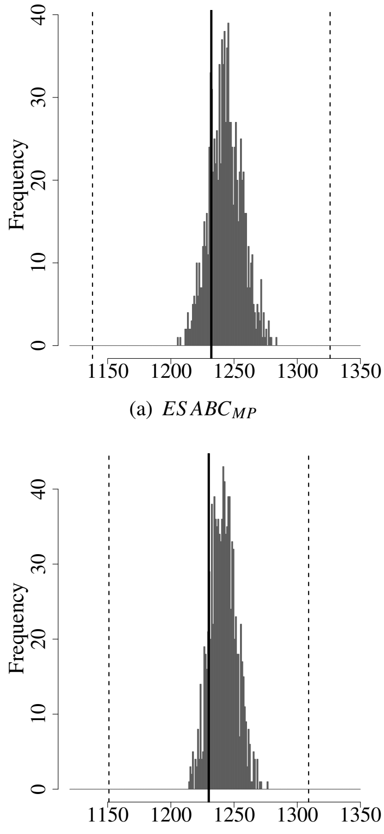
Average $E[\mathbf{C_{max}}]$ and $H(Pos_{\mathbf{C_{max}}})$ values obtained by $ESABC_{Lex2}$ and $ESABC_{MP}$.

| | $ESABC_{MP}$ | | | $ESABC_{Lex2}$ | | |
|---|---|---|---|---|---|---|
| *Instance* | $E[C_{max}]$ | $H(Pos_{C_{max}})$ | Time | $E[C_{max}]$ | $H(Pos_{C_{max}})$ | Time |
| ABZ7 | 700.1 (11.4) | 6.50 (0.07) | 6.5 | **699.7** (13.3) | **6.39** (0.14) | 9.4 |
| ABZ8 | **715.6** (11.2) | 6.29 (0.06) | 7.7 | 717.0 (11.6) | **6.25** (0.07) | 9.9 |
| ABZ9 | **735.0** (10.2) | 6.26 (0.11) | 8.7 | 739.3 (14.7) | **6.13** (0.15) | 10.2 |
| FT10 | 958.0 (11.3) | 6.86 (0.16) | 1.8 | **955.6** (12.0) | **6.82** (0.12) | 2.3 |
| FT20 | **1185.7** (7.3) | 7.16 (0.04) | 2.9 | 1186.9 (8.6) | **7.15** (0.06) | 3.7 |
| LA21 | 1087.5 (7.5) | 7.42 (0.13) | 2.9 | **1084.3** (10.8) | **7.19** (0.16) | 4.1 |
| LA24 | 981.3 (9.0) | 7.29 (0.06) | 2.6 | **974.9** (8.3) | **7.22** (0.07) | 3.4 |
| LA25 | **1003.8** (9.2) | 7.10 (0.06) | 3.0 | 1003.9 (10.9) | **6.99** (0.11) | 3.5 |
| LA27 | **1285.9** (12.6) | 7.38 (0.07) | 5.8 | 1288.5 (20.0) | **7.31** (0.10) | 8.1 |
| LA29 | 1233.0 (12.5) | 7.36 (0.07) | 6.7 | **1232.1** (23.2) | **7.26** (0.13) | 7.6 |
| LA38 | 1265.7 (16.7) | 7.47 (0.09) | 7.2 | **1263.1** (16.0) | **7.33** (0.14) | 5.6 |
| LA40 | 1272.2 (11.1) | 7.56 (0.07) | 3.7 | **1271.2** (12.1) | **7.46** (0.12) | 5.6 |

Surprisingly, the results obtained by $ESABC_{Lex2}$ are quite similar to those of $ESABC_{MP}$. In fact, a Kruskal-Wallis statistical test shows no significant difference between them. By paying attention to the runtime, we see that $ESABC_{Lex2}$ also takes longer to converge, which leads us to believe that it explores more of the search space before converging and therefore it eventually finds more promising solutions.

### 4.4. Sensitivity analysis

Here we asses the output of the proposed algorithm in environments with different degrees of uncertainty. In our setting, this would translate into having wider or narrower intervals, which would correspond to larger or smaller values of the measure of $U$-uncertainty. We propose a sensitivity analysis where, for each instance, two new variants are generated by modifying interval widths by +20% and +40%. Considering the importance of the middle point, the modifications are applied symmetrically making sure that no negative values are found in the intervals. Since $ESABC_{Lex2}$ provides similar result to $ESABC_{MP}$ in terms of expected makespan, but it obtains more robust solutions, it will be the tested variant. As a reference point, we also run $ESABC_c$ to check if, when uncertainty increases, modelling it during the optimisation process loses meaning. As before, the solutions obtained by each method are evaluated over $K = 1000$ deterministic realisations for each instance to find average $\overline{\epsilon}$ values. Table 8 summarizes the obtained results. For each increment in the interval widths, average $\overline{\epsilon}$ values obtained by $ESABC_c$ and $ESABC_{Lex2}$ are reported. The best result for each instance and interval increment is highlighted in bold. Based on the results, it is worth mentioning that in both




(a) $ESABC_{MP}$

(b) $ESABC_{Lex2}$

Fig. 3. 1000 deterministic realisations of the best solutions obtained by $ESABC_{MP}$ and $ESABC_{Lex2}$ on instance La38.

cases, increasing the interval widths in 20 and 40% respectively, $ESABC_{Lex2}$ keeps yielding the most robust solutions. Moreover, in 10 of the instances the solutions obtained by $ESABC_{Lex2}$ when the intervals are enlarged a 40% are still more robust than those obtained by $ESABC_c$ in the scenario with the unaltered intervals. As expected for both methods, the more uncertainty is present in the problem, the worse are the predictive schedules and therefore the robustness values get worse. However, there is also a difference in this aspect between considering uncertainty in the optimisation or not. For instance, average $\overline{\epsilon}$ values obtained with $ESABC_{Lex2}$ get 8.5% and 24.3% worse in average when intervals increase in 20 and 40% respectively. In the case of $ESABC_c$, these values get 14.0% and 39.4% worse respectively.

To graphically illustrate these different behaviours, Figures 4 and 5 show the histograms corresponding to the 1000 makespan values reached with the execution of the best solutions from $ESABC_c$ and $ESABC_{Lex2}$ on instance La27 with increasing interval width. We can see that the predictive makespan, represented by the thick black line, is usually located on the left side of the histogram in both cases; this suggests that the predictive is an optimistic estimate of the makespan on real executions. However, in the case of $ESABC_{Lex2}$ it is more centred in the histogram. As the width of the intervals increase in the instance, the histogram starts expanding rightwards, reaching almost 1350 in $LA27(+40\%)$ for $ESABC_c$. This movement is much less obvious with the solutions from $ESABC_{Lex2}$. In that case, real executions expand significantly less to the right and remain concentrated in a smaller range closer to the black thick line, showing that these solutions are more robust.

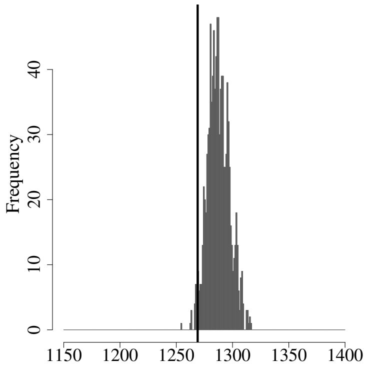
Table 8

Average $\overline{\epsilon}$ values ($\times 1000$) for $ESABC_c$ and $ESABC_{Lex2}$ increasing processing times' interval width in +0%, +20% and +40%

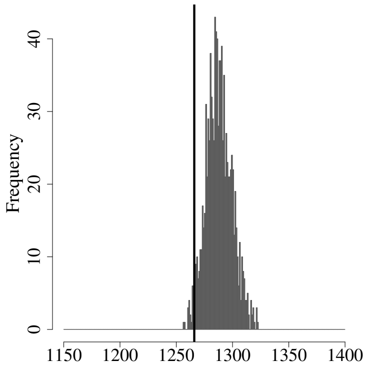
| | +0% | | +20% | | +40% | |
|---|---|---|---|---|---|---|
| *Instance* | $ESABC_c$ | $ESABC_{Lex2}$ | $ESABC_c$ | $ESABC_{Lex2}$ | $ESABC_c$ | $ESABC_{Lex2}$ |
| ABZ7 | 12.26 | **8.03** | 12.51 | **8.03** | 15.39 | **8.92** |
| ABZ8 | 9.86 | **7.46** | 11.07 | **6.98** | 13.59 | **7.93** |
| ABZ9 | 10.61 | **6.46** | 10.85 | **6.33** | 13.35 | **7.21** |
| FT10 | 12.00 | **9.00** | 13.79 | **10.10** | 16.72 | **11.53** |
| FT20 | 9.03 | **7.53** | 10.18 | **8.28** | 12.26 | **9.98** |
| La21 | 13.64 | **9.12** | 16.08 | **10.34** | 19.56 | **11.46** |
| La24 | 15.08 | **11.41** | 18.17 | **12.60** | 22.14 | **15.45** |
| La25 | 12.72 | **9.56** | 14.48 | **9.74** | 17.84 | **11.52** |
| La27 | 13.01 | **8.78** | 15.66 | **10.27** | 19.14 | **11.34** |
| La29 | 12.35 | **8.57** | 14.00 | **9.61** | 17.08 | **11.10** |
| La38 | 13.78 | **8.37** | 16.30 | **9.83** | 19.98 | **10.85** |
| La40 | 13.46 | **9.21** | 16.11 | **10.74** | 19.88 | **12.13** |

### 4.5. Additional results on larger instances

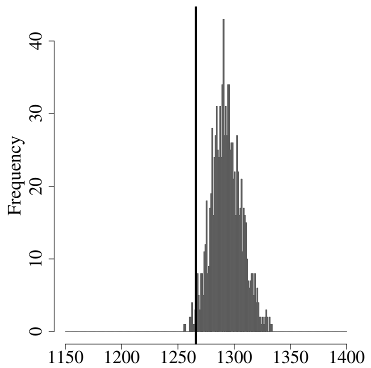
After comparing the algorithm using the instances previously defined in the literature for IJSP, we carry out a new set of experiments on a new set of instances of varied sizes. Our goal is to check the performance of the variant $ESABC_{LinM}$, selected as the one with best behaviour in the previous set of experiments. To achieve this goal we use Taillard's instances, consisting of 8 sets of 10 instances, with sizes ranging from 15 jobs and 15 machines to 100 jobs and 20 machines [64]. These are crisp instances for the Job Shop Problem, so we adapt them to the IJSP using the method from [23] described at the beginning of this section. Due to the significant difference in number of instances and sizes, a new parameter configuration has been obtained both for $ABC_{E3}$ and $ESABC_{LinM}$ following the same methodology as in Section 4.1. For both algorithms, the best parameter setup regarding the operators present in both methods is the following:




(a) $ESABC_c$ on $LA27$

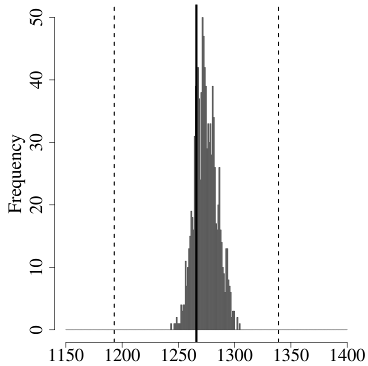



(b) $ESABC_c$ on $LA27(+20\%)$

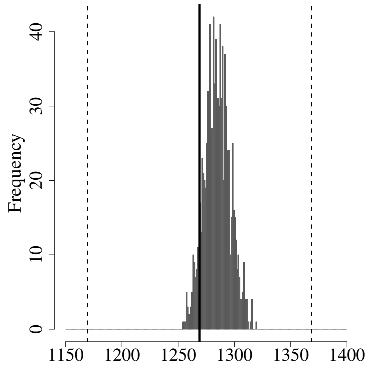



(c) $ESABC_c$ on $LA27(+40\%)$

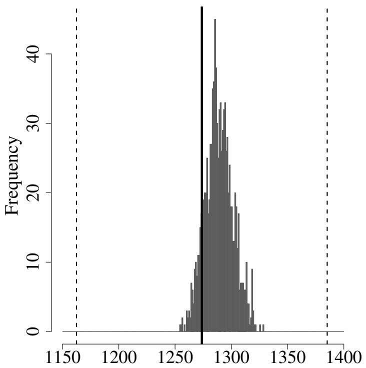
Fig. 4. Histograms of $C^{ex}_{max}$ values for $ESABC_c$




(a) $ESABC_{Lex2}$ on $LA27$




(b) $ESABC_{Lex2}$ on $LA27(+20\%)$




(c) $ESABC_{Lex2}$ on $LA27(+40\%)$

Fig. 5. Histograms of $C^{ex}_{max}$ values for $ESABC_{Lex2}$

- Employed operator: GOX with probability $p_{emp} = 1.0$
- Onlooker operator: Swap with probability $p_{on} = 1.0$
- Max. tries: 20
- Elite size: 60

For the parameters that are specific for $ESABC_{LinM}$, the best found values are as follows:

- Adaptive cooling: Quadratic
- $\alpha = 0.02$
- $T_0 = 1.5$

As with the previous set of instances, the best results are obtained with the quadratic adaptive cooling scheme. Detailed results on all 80 instances obtained after the parameter tuning can be found in the Appendix. There, Tables 9 and 10 report the best and average $E[\mathbf{C_{max}}]$ values together with the average runtime in seconds. Values in bold highlight the best average behaviour on each instance. In summary, $ESABC$ obtains better average results on 71 of the 80 instances, with an improvement in average $RE$ w.r.t. $LB$ of 8.92%. Figure 6 shows the average $RE$ values grouped by instance size. We can see that the best results are obtained for the largest instances, suggesting that the algorithm's efficiency increases with problem size. For instance, for the $100 \times 20$ instances $RE$ decreases from 3.99% to 2.91%, with an improvement of 26.97%, and in the $50 \times 15$ instances, $RE$ drops from 5.71% to 4.55%, with a decrease of 20.33%. Conversely, the lowest improvements are obtained in the $15 \times 15$ and $20 \times 20$ instances, where the average $RE$ is reduced by 0.13% and 4.06% respectively.

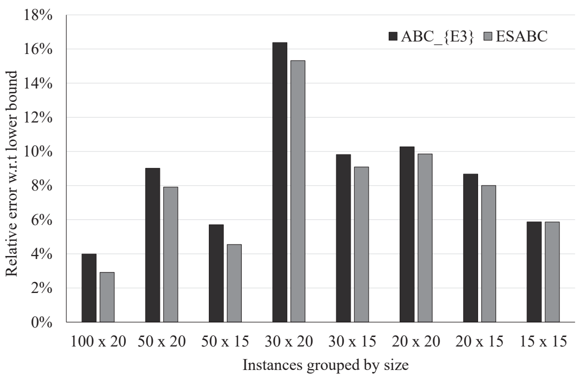



Fig. 6. Average relative error obtained by $ABC_{E3}$ and $ESABC$ w.r.t $LB$ on Taillard's instances depending on the instances size.

## 5. Conclusions

In this work we have confronted the IJSP, a version of the JSP that uses intervals to model the uncertainty on task durations often appearing in real-world problems, with the goal of minimising the makespan. In [24] we used an ABC approach as solving method, adapting the general scheme to our problem, and we tackled the problem of lack of diversity in the swarm by proposing a new ABC variant, $ABC_{E3}$ that enhances diversity in the employed bee phase. Now we have pointed out and addressed a new issue in the scout bee phase, where the discarded food sources are replaced by random new ones in an attempt to increase diversity. We have argued that the nectar amount or initial quality of these new food sources is not good enough to make an effective contribution to the search. Therefore, we have modified the onlooker bee phase incorporating a diversification strategy similar to the one used in simulated annealing and inspired in the seasonal behaviour of bees. This modification has a double effect. On the one hand, it allows the swarm to explore neighbouring food sources with lower nectar amount, thus preventing the algorithm from being prematurely trapped in local optima. On other hand, it has as a consequence a lower ratio of discarded food sources and replacements in the scout bee phase.

An experimental analysis has shown the potential of the seven variants of seasonal behaviour combined with $ABC_{E3}$, outperforming the state-of-the-art results from the literature. Particularly good results are obtained with the multiplicative monotonic cooling variants using the quadratic adaptive cooling scheme proposed for first time in this work. Also, one of the best performing variants, $ESABC_{LinM}$, has been selected to conduct a robustness analysis with different ranking operators. This has shown that ranking $\leqslant_{Lex2}$ yields the most robust solutions for makespan minimisation, a result that concurs with previous experiments using tardiness as objective function [20]. Finally, we have performed a sensitivity analysis increasing the amplitudes of the uncertain durations in problem instances, showing a better behaviour towards the increase in uncertainty in the interval version of $ESABC$ using the $\leqslant_{Lex2}$ ranking operator, than in its crisp counterpart.

In the future, the novel search strategy of $ESABC$ could be applied to combinatorial optimisation problems other than the IJSP. This could be achieved by changing the problem-specific components of the algorithm, that is, the coding of solutions as food sources and their decoding to evaluate the nectar amount, as well as the operators for food-source combination in the employed bee phase and for finding neighbouring food sources in the onlooker bee phase. The general search schema, including the seasonal strategy would remain unchanged.

Also, now that greater diversity is achieved with the seasonal strategy in the search, it would be possible to hybridise the population-based ABC with local search without risking premature convergence to local optima. In this way, the hybrid method could benefit from the synergies between global and local search, as is the case with memetic algorithms, to obtain solutions with higher quality [56].

Finally, it would be interesting to study the effect on the robustness of the solutions obtained with different ranking methods if the Monte-Carlo simulations where obtained using non-symmetric distributions, simulating more extreme scenarios where shorter or larger processing times than expected are more likely to occur.

## Acknowledgements

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

## References

[1] Xiong H, Shi S, Ren D, Hu J. A survey of job shop scheduling problem: The types and models. Computers & Operations Research. 2022;142:105731.

[2] Pinedo ML. Scheduling. Theory, Algorithms, and Systems. 5th ed. Springer; 2016.

[3] Pinedo ML. Planning and Scheduling in Manufacturing and Services. 2nd ed. Springer; 2009.

[4] Brucker P. The job-shop problem: Old and new challenges. In: 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications, Paris, France; 2007. p. 15-22.

[5] Meeran S, Morshed MS. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. Journal of Intelligent Manufacturing. 2012;23:1063-78.

[6] Guo ZX, Wong WK, Leung SYS, Fan JT, Chan SF. Mathematical model and genetic optimization for the job shop scheduling problem in a mixed- and multi-product assembly environment: A case study based on the apparel industry. Computers & Industrial Engineering. 2006;50(3):202-19.

[7] Xie J, Gao L, Peng K, Li X, Li H. Review on flexible job shop scheduling. IET Collaborative Intelligent Manufacturing. 2019;1(3):67-77.

[8] Allahverdi A, Aydilek H, Aydilek A. Single machine scheduling problem with interval processing times to minimize mean weighted completion time. Computers & Operations Research. 2014;51:200-7.

[9] Allahverdi A. A survey of scheduling problems with uncertain interval/bounded processing/setup times. Journal of Project Management. 2022;7(4):255-64.

[10] Bustince H, Pagola M, Barrenechea E, Fernandez J, Melo-Pinto P, Couto P, et al. Ignorance functions. An application to the calculation of the threshold in prostate ultrasound images. Fuzzy Sets and Systems. 2010;161(1):20-36. Special section: New Trends on Pattern Recognition with Fuzzy Models.

[11] Behnamian J. Survey on fuzzy shop scheduling. Fuzzy Optimization and Decision Making. 2016;15:331-66.

[12] Dubois D, Prade H, Sandri S. On possibility/probability transformations. In: Fuzzy Logic. vol. 12 of Theory and Decision Library. Kluwer Academic; 1993. p. 103-12.

[13] Lodwick WA, Jamison KD. Special issue: interfaces between fuzzy set theory and interval analysis. Fuzzy Sets and Systems. 2003;135(1):1-3.

[14] Dubois D, Kerre E, Mesiar R, Prade H. Fuzzy Interval Analysis. In: Dubois D, Prade H, editors. Fundamentals of Fuzzy Sets. The Handbooks of Fuzzy Sets. Boston/London/Dordrecht: Kluwer Academic Publishers; 2000. p. 483-583.

[15] Bustince H, Barrenechea E, Pagola M, Fernandez J, Xu Z, Bedregal B, et al. A Historical Account of Types of Fuzzy Sets and Their Relationships. IEEE Transactions on Fuzzy Systems. 2016;24(1):179-94.

[16] Lin FT. Fuzzy Job-Shop Scheduling Based on Ranking Level $(\lambda, 1)$ Interval-Valued Fuzzy Numbers. IEEE Transactions on Fuzzy Systems. 2002;10(4):510-22.

[17] Dorfeshan Y, Tavakkoli-Moghaddam R, Mousavi SM, Vahedi-Nouri B. A new weighted distance-based approximation methodology for flow shop scheduling group decisions under the interval-valued fuzzy processing time. Applied Soft Computing. 2020;91:106248.

[18] Lei D. Interval job shop scheduling problems. International Journal of Advanced Manufacturing Technology. 2012;60:291-301.

[19] Díaz H, Palacios JJ, Díaz I, Vela CR, González-Rodríguez I. Tardiness Minimisation for Job Shop Scheduling with Interval Uncertainty. In: de la Cal EA, Villar Flecha JR, Quintián H, Corchado E, editors. Hybrid Artificial Intelligent Systems. Springer International Publishing; 2020. p. 209-20.

[20] Díaz H, Palacios JJ, Díaz I, Vela CR, González-Rodríguez I. Robust schedules for tardiness optimization in job shop with interval uncertainty. Logic Journal of the IGPL. 2022.

[21] 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. Computers & Industrial Engineering. 2019;235:1036-46.

[22] Lei D. Population-based neighborhood search for job shop scheduling with interval processing time. Computers & Industrial Engineering. 2011;61:1200-8.

[23] Díaz H, González-Rodríguez I, Palacios JJ, Díaz I, Vela CR. A Genetic Approach to the Job Shop Scheduling Problem with Interval Uncertainty. In: Lesot MJ, Vieira S, Reformat MZ, Carvalho JP, Wilbik A, Bouchon-Meunier B, et al., editors. Information Processing and Management of Uncertainty in Knowledge-Based Systems. Springer; 2020. p. 663-76.

[24] Díaz H, Palacios JJ, González-Rodríguez I, Vela CR. Elite Artificial Bee Colony for Makespan Optimisation in Job Shop with Interval Uncertainty. In: Ferrández Vicente JM, Álvarez-Sánchez JR, de la Paz López F, Adeli H, editors. Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence. Springer International Publishing; 2022. p. 98-108.

[25] Siddique NH, Adeli H. Nature Inspired Computing: An Overview and Some Future Directions. Cognitive Computation. 2015;7(6):706-14.

[26] Slowik A, Kwasnicka H. Evolutionary algorithms and their applications to engineering problems. Neural Computing and Applications. 2020;32(16):12363-79.

[27] Slowik A, Kwasnicka H. Nature Inspired Methods and Their Industry Applications—Swarm Intelligence Algorithms. IEEE Transactions on Industrial Informatics. 2018;14(3):1004-15.

[28] Siddique N, Adeli H. Physics-Based Search and Optimization: Inspirations from Nature. Expert Systems. 2016;33(6):607–623.

[29] Siddique NH, Adeli H. Simulated Annealing, Its Variants and Engineering Applications. International Journal on Artificial Intelligence Tools. 2016;25(6):1630001.
[30] Siddique N, Adeli H. Harmony Search Algorithm and its Variants. International Journal of Pattern Recognition and Artificial Intelligence. 2015;29(08):1539001.
[31] Siddique N, Adeli H. Gravitational Search Algorithm and Its Variants. International Journal of Pattern Recognition and Artificial Intelligence. 2016;30(08):1639001.
[32] Siddique N, Adeli H. Water Drop Algorithms. International Journal on Artificial Intelligence Tools. 2014;23(06):1430002.
[33] Siddique N, Adeli H. Nature-inspired chemical reaction optimisation algorithms. Cognitive computation. 2017;9(4):411-22.
[34] Siqueira H, Santana C, Macedo M, Figueiredo E, Gokhale A, Bastos-Filho C. Simplified binary cat swarm optimization. Integrated Computer-Aided Engineering. 2021;28(1):35-50.
[35] Akhand MAH, Ayon SI, Shahriyar SA, Siddique N, Adeli H. Discrete Spider Monkey Optimization for Travelling Salesman Problem. Applied Soft Computing. 2020;86:105887.
[36] Liu H, Gu F, Lin Z. Auto-sharing parameters for transfer learning based on multi-objective optimization. Integrated Computer-Aided Engineering. 2021;28(3):295-307.
[37] Xue Y, Zhang Q, Neri F. Self-Adaptive Particle Swarm Optimization-Based Echo State Network for Time Series Prediction. International Journal of Neural Systems. 2021;31(12):2150057.
[38] Imran Hossain S, Akhand MAH, Shuvo MIR, Siddique N, Adeli H. Optimization of University Course Scheduling Problem using Particle Swarm Optimization with Selective Search. Expert Systems with Applications. 2019;127:9-24.
[39] Roy S, Maji A. Sampling-based modified ant colony optimization method for high-speed rail alignment development. Computer-Aided Civil and Infrastructure Engineering. 2022;37(11):1417-33.
[40] Sushma MB, Roy S, Maji A. Exploring and exploiting ant colony optimization algorithm for vertical highway alignment development. Computer-Aided Civil and Infrastructure Engineering. 2022;37(12):1582-601.
[41] Kayabekir AE, Nigdeli SM, Bekdaş G. A hybrid metaheuristic method for optimization of active tuned mass dampers. Computer-Aided Civil and Infrastructure Engineering. 2022;37(8):1027-43. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12790.
[42] Bui KTT, Torres JF, Gutiérrez-Avilés D, Nhu VH, Bui DT, Martínez-Álvarez F. Deformation forecasting of a hydropower dam by hybridizing a long short-term memory deep learning network with the coronavirus optimization algorithm. Computer-Aided Civil and Infrastructure Engineering. 2022;37(11):1368-86. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12810.
[43] Adeli HS Hojjat andPark. Optimization of space structures by neural dynamics. Neural Networks. 1995;8(5):769-81.
[44] Park HS, Adeli H. Distributed Neural Dynamics Algorithms for Optimization of Large Steel Structures. Journal of Structural Engineering. 1997;123(7):880-8.
[45] Mahjoubi S, Bao Y. Game theory-based metaheuristics for structural design optimization. Computer-Aided Civil and Infrastructure Engineering. 2021;36(10):1337-53.
[46] Wong LP, Puan CY, Low MYH, Chong CS. Bee Colony Optimization algorithm with Big Valley landscape exploitation for Job Shop Scheduling problems. In: 2008 Winter Simulation Conference; 2008. p. 2050-8.
[47] Yao B, Yang C, Hu J, Yin G, Yu B. An Improved Artificial Bee Colony Algorithm for Job Shop Problem. Applied Mechanics and Materials. 2010;26-28:657-60.
[48] Banharnsakun A, Sirinaovakul B, Achalakul T. Job Shop Scheduling with the Best-so-far ABC. Engineering Applications of Artificial Intelligence. 2012;25(3):583-93.
[49] Bustince H, Fernandez J, Kolesárová A, Mesiar R. Generation of linear orders for intervals by means of aggregation functions. Fuzzy Sets and Systems. 2013;220:69-77.
[50] Destercke S, Couso I. Ranking of fuzzy intervals seen through the imprecise probabilistic lens. Fuzzy Sets and Systems. 2015;278:20-39.
[51] Bidot J, Vidal T, Laboire P. A theoretic and practical framework for scheduling in stochastic environment. Journal of Scheduling. 2009;12:315-44.
[52] Juan AA, Faulin J, Grasman SE, Rabe M, Figueira G. A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. Operations Research Perspectives. 2015;2:62-72.
[53] Nance RE, Sargent RG. Perspectives on the Evolution of Simulation. Operations Research. 2002;50(1):161-72.
[54] Karaboga D, Gorkemli B, Ozturk C, Karaboga N. A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review. 2014;42:21-57.
[55] Bierwirth C. A Generalized Permutation Approach to Jobshop Scheduling with Genetic Algorithms. OR Spectrum. 1995;17:87-92.
[56] Gendreau M, Potvin JY, editors. Handbook of Metaheuristics. vol. 272 of International Series in Operations Research & Management Science. 3rd ed. Springer; 2019.
[57] Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E. Equation of state calculation by fast computing machines. Journal of Chemistry Physics. 1953;21:1087-91.
[58] Van Laarhoven P, Aarts E, Lenstra K. Job shop scheduling by simulated annealing. Operations Research. 1992;40:113-25.
[59] Locatelli M. Convergence of a Simulated Annealing Algorithm for Continuous Global Optimization. Journal of Global Optimization. 2000 01;18:219-33.
[60] Díaz Martín JF, Riaño Sierra JM. A comparison of cooling schedules for simulated annealing. In: Encyclopedia of Artificial Intelligence. IGI Global; 2009. p. 344-52.
[61] Applegate D, Cook W. A computational study of the jobshop scheduling problem. ORSA Journal of Computing. 1991;3:149-56.
[62] Palacios JJ, Puente J, Vela CR, González-Rodríguez I. Benchmarks for fuzzy job shop problems. Information Sciences. 2016;329:736-52.
[63] Klir GJ, Smith RM. On measuring uncertainty and uncertainty-based information: Recent developments. Annals of Mathematics and Artificial Intelligence. 2001;32:5-33.
[64] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research. 1993;64:278-85.

# Appendix A. Additional experimental results

Table 9
Best and average $E[\mathbf{C_{max}}]$ values, and average runtime in seconds obtained by $ABC_{E3}$ and $ESABC$ on Taillard's instances (I)

| | | | $ABC_{E3}$ | | | $ESABC$ | | |
|---|---|---|---|---|---|---|---|---|
| *Size* | *Instance* | *LB* | *Best* | *Avg. (SD)* | *Time* | *Best* | *Avg. (SD)* | *Time* |
| 15 x 15 | TA1 | 1231 | 1289.00 | **1313.82** (12.19) | 3.7 | 1286.50 | 1320.52 (12.91) | 5.4 |
| | TA2 | 1244 | 1277.00 | **1296.25** (10.37) | 3.9 | 1284.50 | 1304.58 (8.99) | 5.2 |
| | TA3 | 1218 | 1262.50 | 1286.58 (12.40) | 4.4 | 1263.50 | **1285.88** (13.10) | 5.7 |
| | TA4 | 1175 | 1226.50 | 1257.17 (17.39) | 4.0 | 1235.50 | **1256.83** (13.80) | 5.9 |
| | TA5 | 1224 | 1263.00 | 1283.80 (12.82) | 3.5 | 1254.50 | **1276.23** (12.93) | 5.0 |
| | TA6 | 1238 | 1280.00 | 1307.40 (13.05) | 4.2 | 1267.00 | **1298.55** (13.59) | 5.5 |
| | TA7 | 1227 | 1261.50 | 1286.60 (13.08) | 3.6 | 1258.50 | **1281.27** (12.22) | 5.3 |
| | TA8 | 1217 | 1248.00 | **1282.78** (12.03) | 3.9 | 1241.00 | 1286.10 (20.72) | 5.2 |
| | TA9 | 1274 | 1331.00 | **1379.12** (22.65) | 4.1 | 1343.50 | 1379.18 (20.15) | 5.7 |
| | TA10 | 1241 | 1280.00 | **1317.42** (17.60) | 3.8 | 1287.00 | 1320.95 (16.13) | 5.3 |
| 20 x 15 | TA11 | 1357 | 1442.00 | 1484.13 (16.66) | 6.6 | 1448.50 | **1470.63** (13.75) | 9.7 |
| | TA12 | 1367 | 1428.00 | 1448.93 (10.93) | 5.8 | 1407.00 | **1439.20** (11.60) | 8.5 |
| | TA13 | 1342 | 1426.00 | **1461.22** (17.93) | 7.1 | 1438.50 | 1466.07 (15.34) | 9.8 |
| | TA14 | 1345 | 1395.50 | 1412.52 (15.06) | 6.3 | 1378.50 | **1404.63** (12.01) | 8.6 |
| | TA15 | 1339 | 1418.50 | 1453.27 (17.80) | 6.3 | 1400.50 | **1433.42** (16.12) | 9.6 |
| | TA16 | 1360 | 1455.50 | **1482.08** (15.32) | 6.3 | 1453.00 | 1485.33 (28.44) | 8.6 |
| | TA17 | 1462 | 1546.00 | 1589.52 (20.08) | 6.1 | 1525.00 | **1579.77** (26.83) | 8.0 |
| | TA18 | 1377 | 1506.50 | 1539.35 (14.83) | 6.6 | 1506.00 | **1537.18** (12.66) | 9.6 |
| | TA19 | 1332 | 1445.50 | 1483.32 (18.01) | 5.7 | 1438.00 | **1461.22** (12.40) | 8.9 |
| | TA20 | 1348 | 1415.00 | 1457.00 (18.98) | 6.2 | 1419.00 | **1442.68** (14.96) | 10.2 |
| 20 x 20 | TA21 | 1642 | 1749.00 | 1774.48 (15.46) | 6.9 | 1731.00 | **1764.58** (19.09) | 11.2 |
| | TA22 | 1561 | 1696.00 | 1725.35 (17.30) | 8.1 | 1689.50 | **1713.57** (17.31) | 11.8 |
| | TA23 | 1518 | 1670.00 | 1693.55 (12.78) | 8.0 | 1662.00 | **1688.45** (16.61) | 11.8 |
| | TA24 | 1644 | 1728.50 | 1766.83 (20.59) | 7.8 | 1731.00 | **1757.65** (16.71) | 11.0 |
| | TA25 | 1558 | 1702.50 | **1728.90** (17.84) | 7.8 | 1683.50 | 1732.78 (29.44) | 11.2 |
| | TA26 | 1591 | 1758.00 | 1798.65 (18.78) | 8.4 | 1750.00 | **1796.65** (22.87) | 11.7 |
| | TA27 | 1652 | 1793.50 | 1824.07 (19.29) | 8.3 | 1775.00 | **1814.17** (21.12) | 11.6 |
| | TA28 | 1603 | 1706.50 | 1734.60 (17.04) | 7.5 | 1703.00 | **1723.52** (10.55) | 10.9 |
| | TA29 | 1583 | 1709.50 | 1744.63 (18.38) | 7.6 | 1704.00 | **1735.42** (14.39) | 12.0 |
| | TA30 | 1528 | 1677.00 | 1714.68 (21.00) | 7.7 | 1681.00 | **1712.03** (15.89) | 10.6 |
| 30 x 15 | TA31 | 1764 | 1870.00 | 1918.87 (21.63) | 12.5 | 1864.50 | **1901.78** (19.82) | 18.7 |
| | TA32 | 1774 | 1963.50 | 2005.80 (20.28) | 13.4 | 1943.00 | **1984.08** (26.79) | 19.8 |
| | TA33 | 1788 | 1956.00 | 1997.40 (20.46) | 13.3 | 1949.50 | **1977.20** (14.39) | 18.8 |
| | TA34 | 1828 | 1978.50 | 2004.02 (16.49) | 12.2 | 1961.00 | **1988.10** (16.25) | 17.3 |
| | TA35 | 2007 | 2036.50 | **2072.15** (21.09) | 11.7 | 2039.00 | 2085.75 (22.44) | 13.0 |
| | TA36 | 1819 | 1963.50 | 1994.12 (15.63) | 12.7 | 1932.50 | **1979.67** (19.55) | 18.2 |
| | TA37 | 1771 | 1915.50 | 1942.83 (17.27) | 12.7 | 1896.00 | **1932.35** (26.91) | 16.4 |
| | TA38 | 1673 | 1815.50 | 1851.15 (18.12) | 12.1 | 1795.00 | **1832.13** (40.40) | 18.9 |
| | TA39 | 1795 | 1905.00 | 1944.97 (19.65) | 13.2 | 1893.00 | **1935.37** (35.17) | 18.0 |
| | TA40 | 1651 | 1835.50 | 1872.52 (17.79) | 13.1 | 1836.00 | **1860.78** (14.38) | 17.8 |

Table 10

Best and average $E[\mathbf{C_{max}}]$ values, and average runtime in seconds obtained by $ABC_{E3}$ and $ESABC$ on Taillard's instances (II)

| | | | $ABC_{E3}$ | | | $ESABC$ | | |
|---|---|---|---|---|---|---|---|---|
| *Size* | *Instance* | *LB* | *Best* | *Avg. (SD)* | *Time* | *Best* | *Avg. (SD)* | *Time* |
| 30 x 20 | TA41 | 1906 | 2241.00 | 2297.70 (25.10) | 16.1 | 2217.50 | **2272.65** (25.48) | 22.4 |
| | TA42 | 1884 | 2150.50 | 2212.85 (23.41) | 15.3 | 2156.50 | **2201.58** (30.16) | 20.8 |
| | TA43 | 1809 | 2099.50 | 2136.97 (20.58) | 16.5 | 2059.50 | **2111.75** (22.40) | 24.2 |
| | TA44 | 1948 | 2179.50 | 2228.73 (26.44) | 16.8 | 2152.00 | **2196.88** (24.03) | 25.9 |
| | TA45 | 1997 | 2136.50 | 2185.62 (23.61) | 17.0 | 2132.00 | **2164.72** (17.97) | 24.2 |
| | TA46 | 1957 | 2211.50 | 2268.27 (29.82) | 18.2 | 2198.00 | **2248.08** (50.82) | 26.2 |
| | TA47 | 1807 | 2106.00 | 2176.27 (30.54) | 16.3 | 2084.00 | **2143.90** (31.96) | 24.0 |
| | TA48 | 1912 | 2139.50 | 2177.72 (20.44) | 15.1 | 2128.00 | **2177.45** (44.09) | 22.1 |
| | TA49 | 1931 | 2145.00 | 2185.63 (24.78) | 17.1 | 2135.00 | **2181.67** (39.33) | 21.3 |
| | TA50 | 1833 | 2136.50 | 2206.25 (29.39) | 16.3 | 2135.50 | **2178.07** (18.96) | 22.1 |
| 50 x 15 | TA51 | 2760 | 2912.00 | 2958.60 (31.68) | 27.3 | 2863.00 | **2918.87** (31.69) | 40.6 |
| | TA52 | 2756 | 2867.00 | 2919.47 (26.24) | 27.5 | 2835.00 | **2879.32** (18.23) | 43.2 |
| | TA53 | 2717 | 2816.50 | 2863.85 (27.06) | 27.5 | 2793.50 | **2826.80** (26.49) | 37.9 |
| | TA54 | 2839 | 2857.00 | 2922.90 (26.70) | 24.3 | 2848.50 | **2888.57** (24.60) | 34.3 |
| | TA55 | 2679 | 2840.00 | 2905.17 (26.14) | 29.3 | 2836.50 | **2880.35** (21.73) | 39.2 |
| | TA56 | 2781 | 2871.50 | 2921.12 (21.82) | 23.7 | 2864.50 | **2908.05** (22.78) | 36.9 |
| | TA57 | 2943 | 2991.00 | 3049.33 (29.99) | 26.9 | 2979.00 | **3016.08** (26.63) | 41.1 |
| | TA58 | 2885 | 2974.00 | 3027.40 (24.09) | 26.7 | 2941.00 | **2988.97** (35.95) | 38.4 |
| | TA59 | 2655 | 2804.50 | 2854.22 (24.07) | 27.8 | 2780.50 | **2830.77** (36.58) | 38.9 |
| | TA60 | 2723 | 2843.50 | 2888.35 (21.77) | 24.7 | 2822.00 | **2850.28** (19.27) | 34.9 |
| 50 x 20 | TA61 | 2868 | 3062.00 | 3121.43 (29.76) | 33.5 | 3034.50 | **3090.73** (29.15) | 49.2 |
| | TA62 | 2869 | 3139.00 | 3205.95 (29.12) | 33.7 | 3117.50 | **3167.33** (25.91) | 44.7 |
| | TA63 | 2755 | 2965.50 | 3008.20 (23.92) | 30.6 | 2930.00 | **2971.88** (27.00) | 45.8 |
| | TA64 | 2702 | 2889.50 | 2927.90 (20.47) | 33.6 | 2862.50 | **2885.18** (12.00) | 47.0 |
| | TA65 | 2725 | 2954.50 | 2996.90 (27.91) | 33.8 | 2924.00 | **2971.60** (43.21) | 47.5 |
| | TA66 | 2845 | 3038.50 | 3088.55 (21.51) | 32.1 | 3017.00 | **3059.80** (42.15) | 42.8 |
| | TA67 | 2825 | 3008.50 | 3077.80 (27.33) | 34.0 | 3008.50 | **3057.87** (50.11) | 45.4 |
| | TA68 | 2784 | 2969.50 | 3007.00 (22.07) | 32.7 | 2919.50 | **2974.65** (37.91) | 46.6 |
| | TA69 | 3071 | 3227.00 | 3273.53 (28.63) | 29.6 | 3184.50 | **3252.13** (48.22) | 45.0 |
| | TA70 | 2995 | 3247.50 | 3291.77 (20.81) | 30.9 | 3210.50 | **3256.28** (22.77) | 42.9 |
| 100 x 20 | TA71 | 5464 | 5645.50 | 5723.65 (44.87) | 94.2 | 5581.50 | **5652.82** (36.80) | 136.0 |
| | TA72 | 5181 | 5359.00 | 5409.03 (30.16) | 103.1 | 5283.50 | **5347.05** (47.32) | 132.4 |
| | TA73 | 5568 | 5683.00 | 5735.93 (41.97) | 95.3 | 5631.50 | **5669.68** (25.46) | 134.5 |
| | TA74 | 5339 | 5434.00 | 5512.33 (54.56) | 99.5 | 5391.00 | **5450.30** (26.78) | 121.7 |
| | TA75 | 5392 | 5649.50 | 5731.18 (33.34) | 105.0 | 5618.00 | **5700.20** (62.13) | 127.7 |
| | TA76 | 5342 | 5535.00 | 5586.72 (35.81) | 101.2 | 5481.50 | **5540.68** (26.52) | 126.5 |
| | TA77 | 5436 | 5529.50 | 5616.35 (65.01) | 107.0 | 5492.50 | **5537.63** (27.76) | 133.5 |
| | TA78 | 5394 | 5490.00 | 5535.08 (29.19) | 102.7 | 5426.00 | **5487.77** (46.05) | 125.5 |
| | TA79 | 5358 | 5446.00 | 5502.72 (33.76) | 107.8 | 5418.50 | **5447.00** (18.40) | 125.9 |
| | TA80 | 5183 | 5373.00 | 5440.45 (35.15) | 107.1 | 5339.00 | **5382.85** (27.95) | 138.5 |