Combinatorial artificial bee colony algorithm hybridised with a new release of iterated local search for job shop scheduling problem

Amaria Ouis Khedim

Manufacturing Engineering Laboratory of Tlemcen (MELT),
Department of Electrical and Electronic Engineering,
University of Tlemcen,
PB 230, Tlemcen, 13000, Algeria
Email: khedim3@gmail.com
Email: a_ouis@mail.univ-tlemcen.dz

Mehdi Souier*

Manufacturing Engineering Laboratory of Tlemcen (MELT),
University of Tlemcen,
PB 230, Tlemcen, 13000, Algeria
and
High School of Management of Tlemcen,
PB 1085, Tlemcen, 13000, Algeria
Email: souier.mehdi@gmail.com
Email: m_souier@mail.univ-tlemcen.dz
*Corresponding author

Zaki Sari

Manufacturing Engineering Laboratory of Tlemcen (MELT),
University of Tlemcen,
PB 230, Tlemcen, 13000, Algeria
and
Ecole Supérieure en Sciences Appliquées Tlemcen (ESSAT),
PB 165, Tlemcen, 13000, Algeria
Email: zaki_sari@yahoo.com
Email: z_sari@mail.univ-tlemcen.dz

Abstract: Job shop scheduling problem (JSP) is recognised as an attractive subject in production management and combinatorial optimisation. However, it is known as one of the most difficult scheduling problems. The present paper investigates the job shop scheduling problem in order to minimise the Makespan with a new hybrid combinatorial artificial bee colony algorithm. Firstly, the proposed combinatorial version integrates a position based crossover for the updating of solutions and the rank-based selection for selecting solutions to be updated in the onlooker bees phase. Another purpose of this study consists to highlight the impact of its sequential hybridisation with a new release of iterated local search method called ‘simple iterated local search (SILS)’. The proposed approaches are tested on many benchmark problems taken from the Operations Research Library (OR-Library). The simulation results show that the hybrid CABC performs the best in most of the studied cases.

Keywords: job shop scheduling problem; JSP; metaheuristics; artificial bee colony algorithm; iterated local search.

Reference to this paper should be made as follows: Ouis Khedim, A., Souier, M. and Sari, Z. (xxxx) ‘Combinatorial artificial bee colony algorithm hybridised with a new release of iterated local search for job shop scheduling problem’, *Int. J. Operational Research*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Amaria Ouis Khedim is currently an Assistant Professor at Tlemcen University and member of the Manufacturing Engineering Laboratory of Tlemcen (MELT), Algeria. She obtained her Engineer degree in Automatic from University of Tlemcen, Algeria in 1998 and Magister degree in Signals and Systems from University of Tlemcen Algeria, in 2001. She is currently working towards her PhD degree at the Manufacturing Engineering Laboratory of Tlemcen (MELT). Her research interests include planning, scheduling, metaheuristics algorithms, discrete optimisation methods and optimisation problems in manufacturing systems.

Mehdi Souier received his Engineering degree in 2007 in Computer Sciences and Master’s degree in 2009 in Manufacturing Engineering from University of Tlemcen, Algeria. He obtained his Doctorate degree in Manufacturing Engineering in 2012 and the University Habilitation in 2015 from Tlemcen University. He is currently an Associate Professor in Tlemcen High School of Management and member of the Manufacturing Engineering Laboratory of Tlemcen (MELT). He advised several masters and one doctorate thesis. He is a regular reviewer of many international journals and IPC member of many international conferences. His main research interest concern: planning and scheduling, heuristic and metaheuristic (simulated annealing, genetic algorithm, ant colony, …), discrete optimisation methods, optimisation problems in flexible manufacturing systems and maintenance of industrial systems.

Zaki Sari is currently a Senior Consultant for Industry and a Manufacturing Engineering Professor at Ecole Superieure en Sciences Appliquées Tlemcen (ESSAT), Algeria; he is the former Director of Manufacturing Engineering Laboratory of Tlemcen (MELT), and the former Head of the National Curriculum of Manufacturing Engineering. He obtained his Engineer degree in Electrical Engineering from the National Institute of Electrical Engineering, Boumerdes, Algeria in 1987; Magister degree in Power Engineering from the National Polytechnic School of Algiers, Algeria in 1990; and Doctorate degree in Manufacturing Engineering from Tlemcen University in 2003. In 2004, he became an Associate Professor then a Full Professor in 2009. His teaching skills include project management, factory physics, CIM, AS/RS. His main domain of interest concerns the design, modelling, optimisation, simulation and control of automated storage and retrieval system. He made several investigations on non-conventional AS/RS systems. He advised several Magister and Doctorate thesis.

1 Introduction

To survive in a modern and competitive world, which requires lower cost products with shorter life cycle, companies must respond quickly and accurately to customer inquiries. In such situation, it is clear that an effective scheduling would have a very important role in achieving these objectives in various production systems of goods and services such as cellular manufacturing system (Dehnavi-Arani et al., 2019), flow shop (Kumar et al., 2019; Ramezanian and Rahmani, 2017; Seidgar et al., 2017), parallel machine (Hung et al., 2019; Shokoufi et al., 2019), single machine (Rostami et al., 2019; Zoulfaghari and Nematian, 2019), project shop (Tabrizi et al., 2019), airport gate scheduling (Khatibi et al., 2019), scheduling of surgeries (Soudi et al., 2019).

Furthermore, different problems arising in industry, computing, business, and even in the social services can be structured and modelled as a job shop scheduling problem (JSP) which is considered as one of the most popular scheduling problems.

The classical job shop problem consists of a set of independent jobs to be processed through several machines (resources). Each job has an ordered set of operations, to be treated in a predefined order depending on their technological constraints. One of the main objectives of JSP is to minimise the makespan or completion time (C_{max}) for improving throughputs and the system productivity.

In terms of computational complexity, this problem is known to be NP-hard in the strong sense (Garey et al., 1976). Due to its wide applicability and inherent difficulty, the JSP has attracted the attention of many researchers, which has led to the development of several.

The artificial bee colony (ABC) is a metaheuristic introduced by Karaboga (2005), it is inspired from the intelligent behaviour of honey bees for seeking quality food source in nature. The ABC algorithm is a population-based algorithm. Every artificial bee generates one solution. Population of artificial bees searches for the optimal solution. As the ABC algorithm is inspired from the foraging process in the natural bee colony, it is considered that each solution is called a food source, whereas the fitness of the solution corresponds to the amount of nectar of the associated food source. The computing agents of this algorithm are given by three kinds of artificial bees, namely: employed bees, onlooker bees and scout bees. A bee that is currently exploiting a food source (solution) is called an employed bee. A bee waiting in the hive for making a decision to choose a food source is named an onlooker. A bee carrying out a random search for new food source is called a scout.

This metaheuristic has the advantage of regulating the trade-off between exploitation and exploration. It has also the advantage of employing fewer control parameters in the continuous space. These advantages make the ABC algorithm very competitive to other population-based algorithms. Indeed, in the last decade, several works tried to evaluate the performance of ABC in comparison with others. It has been proven that this algorithm has better performances in several problems such as waste collection problem (Wei et al., 2019), numerical optimisation (Bajer and Zorić, 2019), wireless sensors network (Yue et al., 2019), supply chain network management (Jiang et al., 2019), multi-robot path planning (Faridi et al., 2018).

Due to the ABC ability to find good solutions in different research areas and industries, we integrate this approach in our strategy for solving the JSP. In this work, we firstly adapt the continuous version of the ABC algorithm to the combinatorial problem of the job shop. The proposed combinatorial artificial bee colony (CABC) algorithm uses a position based crossover for the updating of solutions and the Rank-Based Selection for selecting solutions to be updated in the onlooker bees phase. After that, this algorithm is improved using a sequential hybridisation with a new local search method that we called ‘simple iterated local search (SILS)’. SILS iteratively applies local search to refine the current best solution found by some CABC iterations. In order to preserve the quality of this solution, SILS aims to exploit the current best solution only in its basin of attraction. Hence, the resulting solution is injected into the next local search without any prior modification. This made the proposed SILS a simplified release of the ILS metaheuristic (Lourenço et al., 2003). The new hybrid algorithm is called: ‘simple iterated local search combinatorial artificial bee colony (SILS_CABC)’ algorithm.

This paper is organised as follows: Section 2 presents briefly the most relevant literature. Section 3 is reserved for the job shop scheduling formulation. The fundamental ABC algorithm is presented in Section 4. In Section 5, the proposed CABC algorithm for JSP is given in detail. Then, Section 6 describes the hybrid release SILS_CABC. Based on the benchmark problems, Section 7 presents the experimental results, where the performances of the proposed algorithms are analysed and compared. Finally, conclusions and future research directions are provided in Section 8.

2 Literature review

The current marketing context is characterised by various challenges related to customers needs with high quality, low costs, short lead times... In such conditions, manufacturing systems require efficient management tools able to deal with different customers’ requests. Among these tools, scheduling systems can play an important role for managing efficiently different kinds of production systems.

Job shop scheduling is among the famous problems that are investigated by production managers and optimisation researchers. However, the scheduling problems in job shop are known as NP-Hard combinatorial optimisation problems.

The existing literature on the job shop manufacturing systems presents many strategies of scheduling decisions that are classified into two groups: exact methods and approximate methods (heuristics and metaheuristics...).

Among the main exact methods for solving the JSP, we can find branch and bound method (Benttaleb et al., 2018; Artigues and Feillet, 2008), the dynamic programming (Ozolins, 2018) and integer programming (Masmoudi et al., 2019; Roshanaei et al., 2010). However, exact methods may be inefficient when the problem size grows. For this reason, many works are focused on approximate methods such as heuristics and metaheuristics.

The approximate methods cannot guarantee the achievement of the global optimal, but they can find near-optimal solutions for problems of large sizes in moderate computing time. Recently, there are considerable researches aiming to develop scheduling solutions for job shop based on heuristics such as dispatching rules (Zhang and Roy, 2018), shifting bottleneck procedure (Tan et al., 2016).

Unlike heuristics that are constructive methods designed and applicable to a particular problem, metaheuristics are stochastic algorithms often inspired by analogies with natural phenomena, such as: physics (simulated annealing,), biology (Tabu search, evolutionary algorithms, artificial immune systems,) or ethology (ant colonies, particle swarm optimisation, bee colony, bat algorithm). They are iterative methods, applicable to a large variety of optimisation problems in science and engineering such as portfolio rebalancing (Zandieh and Mohaddesi, 2019), multi-dimensional knapsack problem (Abubaker et al., 2019), graph sum colouring (Mohammadnejad and Eshghi, 2019), facility layout (Tayal and Singh, 2019), hydraulic analysis (Moeini, 2018), capacitated clustering (Khambhampati et al., 2018).

In recent years, metaheuristics have become extremely popular as practical optimisation methods for solving the JSP. Among these approaches, the genetic algorithm (GA) and evolutionary algorithms are the most applied to solve JSP by different researchers such as: Tan et al. (2019), Kundakcı and Kulak (2016), Lei (2012), Zhang et al., (2011). Other metaheuristics are known as successful techniques in job shop scheduling domain such as simulated annealing (Tamssaouet et al., 2018; Suresh and Mohanasundaram, 2006; Aydin and Forgarty, 2004), Tabu search (Tamssaouet et al., 2018; González et al., 2013; Zhang et al., 2007), particle swarm optimisation (Dao et al., 2018; Singh and Mahapatra, 2016; Lin et al., 2010; Lian et al., 2006), ant colony optimisation (Chaouch et al., 2019; Huang and Yu, 2017; Huang et al., 2013), bee colony algorithm (Sundar et al., 2017; Gao et al., 2016; Zhang et al., 2013). Furthermore, many investigations based on other artificial intelligence techniques are proposed. Most of them use multi agent approaches (Nouri et al., 2016; Guizzi et al., 2018), neural network (Nayak et al., 2019). For more details about the different investigations on job shop scheduling, the reader can refer to the survey of Çaliş and Bulkan (2015) and Zhang et al. (2019).

The literature indicates that there is a significant interest in metaheuristics applications to deal with JSPs. However, it is noticed that the ABC metaheuristic is poorly invested in solving the important problems of job shop. On the other hand, the ABC is recognised as among the most successful techniques to solve various optimisation problems. For this instance, the present paper deals with a combinatorial and a hybrid ABC versions for different benchmarks of JSP.

3 Job shop scheduling problem

3.1 Problem statement and assumptions:

The classical job shop problem considers a set of n jobs, $J = \{J_1,\dots, J_n\}$ to be processed on a set of m machines, $M = \{M_1, \dots, M_m\}$ (resources). Each job J_i is composed of a set of m_i operations denoted $O_i = \{O_{i1}, O_{i2},, \dots, O_{imi},\}$. Each job J_i visits machines in a specific order. Figure1 illustrates an example of 3-job × 5-machine job shop. The jobs machine sequence of this example are given in Table 1. For example, the job J_1 must follow the sequence:

$$m_1 - m_2 - m_3 - m_5 - m_4.$$

For the case of a classical job shop, we retain some standard assumptions that can be presented as follows:

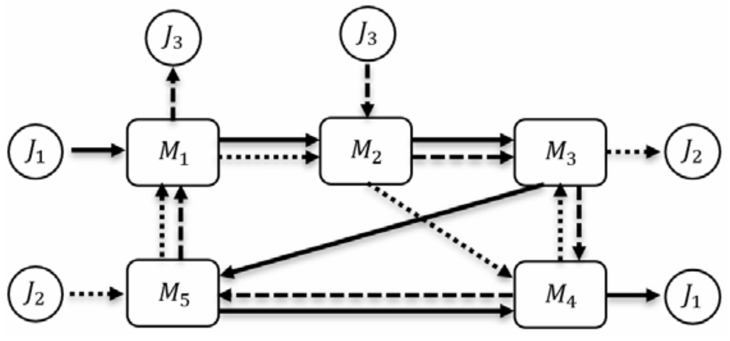
- all jobs are equally important
- each job is available to be processed at any time

- there are precedence relations (conjunctive constraints) between the operations of the same job
- the routing (processing order) for each job is defined by its operations sequence
- there is no due date for any job
- each machine is continuously available without any breakdown
- no alternative machines
- each machine processes at most one job (one operation) at a time
- each job is processed by only one machine at a time
- no preemption, no recirculation and no cancellation of orders are allowed
- all processing times are known in advance.

Table 1 Jobs machine sequence for a 3-job × 5-machine job shop

	Operations				
Job	1	2	3	4	5
	Machine sequence				
j_1	m_1	m_2	m_3	m_5	m_4
j_2	m_5	m_1	m_2	m_4	m_3
j_3	m_2	m_3	m_4	m_5	m_1

Figure 1 a 3-job×5-machine job shop



3.2 The mathematical model

The following notations are used for the formulation of JSP:

Sets and indices:

i	indices for jobs, i = 1,…, n
j	indices for operations, $j = 1,\dots, m_i$
k	indices for machines, $k = 1,\dots, m$

O_{ij}	j^{th} operation of the job J_i
J	set of n jobs to be scheduled, $J = \{J_1,\dots, J_i,\dots,J_n\}$
M	set of m machines $M = \{M_1,\dots, M_k,\dots M_m\}$
O_i	ordered set of all operations of job J_i, $O_i = \{O_{i1},\dots,O_{ij},\dots O_{im_i}\}$
OM_k	set of all operations executed on machine M_k.

Parameters:

p_{ij}	processing time of operation O_{ij}
L	a large number.

Variables:

$$\alpha_{ijk} = \begin{cases} 1 & \text{if } O_{ij} \text{ is excuted on machine } M_k \\ 0 & \text{otherwise} \end{cases}$$

$$\beta_{iji'j'} = \begin{cases} 1 & \text{if } O_{ij} \text{ precede } O_{i'j'};\ \textit{for } O_{ij}, O_{i'j'} \in OM_k,\ i \neq i' \text{ and } j \neq j' \\ 0 & \text{otherwise} \end{cases}$$

r_{ij}	release date of operation O_{ij}
s_{ij}	starting time of operation O_{ij}
C_{ij}	completion time of operation O_{ij}, ($C_{ij} = s_{ij} + p_{ij}$)
C_i	completion time of job j_i
C_{max}	makespan, ($C_{max} = max\ \{C_1, C_2,\dots, C_n\}$).

Actually, the overall cost of production also depends on the time required for the manufacturing of various products. Hence, the main objective of the JSP is to reduce the overall manufacturing time, called 'makespan' and denoted $C_{max} \cdot C_{max}$ is the completion time of the last job to leave the system.

Therefore, mathematically, the most often studied optimality criterion in the JSP is the minimisation of C_{max}. A minimum makespan usually implies a good utilisation of machines; because a given number of jobs are to be completed in the shortest possible time.

To solve the JSP, we have to determine the feasible schedule for the operations of all jobs by respecting the processing order of each job and the capacity of each machine. The goal is to find a particular feasible schedule that has the lowest possible Makespan.

Based on Manne formulation (Manne, 1960), we formulate the problem of the job shop scheduling by the following mixed integer programming model:

$$C^*_{\max} = \underset{\text{feasible schedules}}{\text{Min}} \left\{ C_{\max} = \max_{i=1,\dots,n} \{C_i\} \right\} \tag{1}$$

Equivalent to:

$$\underset{\text{feasible starting times}}{\text{Min}} \left\{ \max_{i=1,\dots,n} \left\{ \left(s_{ij} + p_{ij}\right); \text{for } j = 1,\dots,m_i \right\} \right\} \tag{2}$$

Subject to:

$$r_{i(j+1)} \geq s_{ij} + p_{ij}; \quad \text{for } i = 1,\ldots,n \;\&\; j = 1,\ldots,m_i - 1 \tag{3}$$

$$\sum_{O_{ij} \in OM_k} \alpha_{ijk} = 1; \quad \text{for } k = 1,\ldots,m \tag{4}$$

$$s_{i'j'} + L\left(1 - \beta_{iji'j'}\right) - s_{ij} \geq p_{ij}; \quad \text{for } k = 1,\ldots,m \;\&\; O_{ij}, O_{i'j'} \in OM_k \tag{5}$$

$$s_{ij} + L\beta_{iji'j'} - s_{i'j'} \geq p_{i'j'}; \quad \text{for } k = 1,\ldots,m \;\&\; O_{ij}, O_{i'j'} \in OM_k \tag{6}$$

$$\sum_{k=1}^{m} \alpha_{ijk} = 1; \quad \text{for } i = 1,\ldots,n \;\&\; j = 1,\ldots,m_i \tag{7}$$

$$\sum_{j=1}^{m_i} \alpha_{ijk} \leq 1; \quad \text{for } i = 1,\ldots,n \;\&\; k = 1,\ldots,m_i \tag{8}$$

$$\sum_{j=1}^{m_i} p_{ijk} \leq C_i; \quad \text{for } i = 1,\ldots,n \tag{9}$$

$$s_{ij} \geq r_{ij}; \quad \text{for } i = 1,\ldots,n \;\&\; j = 1,\ldots,m_i \tag{10}$$

$$\beta_{iji'j'} + \beta_{i'j'ij} \leq 1; \quad \text{for } k = 1,\ldots,m \;\&\; O_{ij}, O_{i'j'} \in OM_k \tag{11}$$

$$s_{ij} \geq 0; \quad \text{for } i = 1,\ldots,n \;\&\; j = 1,\ldots,m_i \tag{12}$$

$$\alpha_{ijk}, \beta_{iji'j'} \in \{0,1\}; \quad \text{for } k = 1,\ldots,m \;\&\; O_{ij}, O_{i'j'} \in OM_k \tag{13}$$

The objective function is given by (1). The first set of constraints (3) ensures both precedence and no-preemption constraints, where the operation $O_{i(j+1)}$ cannot be released before the end of operation O_{ij}. The second set of constraints (4) imposes the capacity constraints where a machine can process only one operation at a time. It can be reinforced by the disjunctive constraints (5) and (6). These constraints present the relations between the operations of different jobs to be processed on the same machine. For a machine M_k, a feasible schedule must either satisfy $s_{i'j'} - s_{ij} \geq p_{ij}$ or $s_{i'j'} \geq p_{i'j'}$; for all $O_{ij}, O_{i'j'} \in OM_k$. Actually, if O_{ij} precedes $O_{i'j'}$ on machine M_k, then $\beta_{iji'j'} = 1$ and (5) becomes $s_{i'j'} - s_{ij} \geq p_{ij}$ while (6) becomes redundant because of a large positive value of L. On the other hand, if $O_{i'j'}$ precedes O_{ij}, then $\beta_{iji'j'} = 0$ and (6) becomes $s_{i'j'} - s_{ij} \geq p_{i'j'}$ while (5) becomes redundant. The value of L must be large enough to satisfy $L \geq s_{i'j'} + p_{i'j'} - s_{ij}$ for all $M_k \in M$ and $O_{ij}, O_{i'j'} \in M_k$. For this requirement, $L = \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{ij}$ is sufficient. Constraint (7) ensures that an operation can be processed by only one machine. The assumption of the no-recirculation is ensured by constraints (8) and that of no-cancellation is given by constraint (9). Constraint (10) ensures that each operation can only be released after its acceptable release date. The last two sets of constraints define the domain for each variable.

Before trying to develop an algorithm which solves a particular problem, it is important to have an idea on the problem complexity. For certain restricted cases of the n-job, m-machine JSP, Garey and Johnson (1979) showed that there are polynomial algorithms that rapidly find the optimal schedules. However, Lenstra and Rinnooy Kan (1979) proved that the general JSP is an NP-Hard problem when the number of machines is greater than three. In this case, finding an optimum schedule could be time consuming and sometimes it can be impossible to achieve. Therefore, when the JSP problem becomes strongly NP-hard, it would be necessary to use heuristics or metaheuristics to solve it. With these methods, the guarantee of finding optimal solutions is sacrificed for the sake of (hopefully) getting acceptable solutions with a significantly reduced computational time.

4 ABC algorithm

ABC is one of the recent swarm intelligence optimisation techniques. It is a metaheuristic inspired from the intelligent behaviour of honey bees for seeking quality food source in nature. The idea and principle of this algorithm have been introduced for the first time by Karaboga (2005). However, the details of its different steps were published in Karaboga and Basturk (2007).

The ABC is a population based metaheuristic. Therefore, a population of artificial bees searches the optimal solution and every artificial bee generates one solution. As the ABC algorithm is inspired from the foraging process in the natural bee colony, it is considered that each solution to the problem to be solved is called a food source, whereas the fitness of the solution corresponds to the amount of nectar of the associated food source. The computing agents of this algorithm are given by three kinds of artificial bees, namely: employed bees, onlooker bees and scout bees.

A bee that is currently exploiting a food source (solution) is called an employed bee, whereas bee waiting in the hive for making decision to choose a food source is named as an onlooker. Finally, bee carrying out a random search for a new food source is called a scout (Teodorović et al., 2015). Both onlookers and scouts are also called unemployed bees (Karaboga et al., 2014).

Following the foraging mechanism in the natural bee colony, in the ABC algorithm, scout bees can be visualised as performing exploration, whereas employed and onlooker bees can be visualised as performing exploitation. Hence, this metaheuristic formulated by the ABC algorithm has the advantage of regulating the trade-off between exploitation and exploration. It has also the advantage of employing fewer control parameters in continuous space. These advantages make the ABC algorithm very competitive to other population based algorithms (Karaboga and Basturk, 2007, 2008; Karaboga and Akay, 2009).

The main steps of the fundamental ABC algorithm are summarised in Algorithm 1.

The fundamental ABC algorithm starts with an initialisation phase, where the population is randomly generated, and then, at each iteration: the employed bees phase, onlooker bees phase and scout bees phase are repeated until a termination condition is met. The details of these steps can be found in Karaboga and Akay (2009).

The fundamental ABC algorithm, as presented above, was originally designed for continuous optimisation problems and cannot be used directly for combinatorial cases.

Therefore, for solving a combinatorial problem some modifications to the fundamental ABC algorithm must be done.

Algorithm 1 Fundamental ABC algorithm

```
1 Initialise
2 REPEAT
3         Employed bees phase: the employed bees try to update their food sources.
4         Onlooker bees phase: the onlooker bees select the more interesting food sources
          given in Employed bees phase and furthermore try to update them.
5         Scout bees phase: send scouts to search new food sources that will replace the
          abandoned ones.
6         Memorise the best food source found so far.
7 UNTIL (termination criteria is satisfied).
```

5 The proposed CABC algorithm

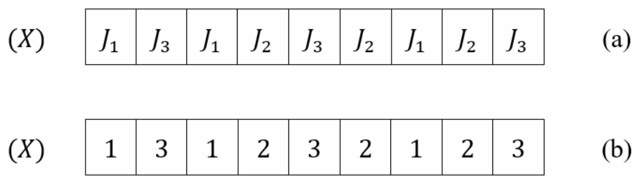
Mapped for the resolution of the JSP, the CABC algorithm is proposed in this section as a combinatorial release of the ABC algorithm. The aim is to find the job operation scheduling list that minimises the Makespan value.

The pseudo-code of the CABC algorithm proposed for solving the JSP is given in Algorithm 2 and its general steps are detailed in the following subsections:

5.1 Solution representation

In the CABC algorithm, each solution corresponds to a food source '*X*' exploited by one bee. The first step in problem solving is the solution representation according to the problem environment. Let us consider that our JSP to be solved is an instance of *n* jobs and *m* machines; where each job consists of m ordered operations. The solution representation adopted in this work is the '*operation-based representation*' with '*job repetition*'. Hence, the solution of this JSP is an operation scheduling list, which is represented in our CABC algorithm as food source '*X*'. This food source '*X*' is a vector with ($n \times m$) dimensions where each dimension stands for one operation of a job. According to the solution representation considered, in the food source '*X*', each job appears exactly *m* times. Figure 2 illustrates an example of a solution representation of job shop.

Figure 2 Example of food source representation for (3-job×3-machine) JSP



In the solution (X) of Figure 2(a), J_i stands for the operation of job i. Since each job has three operations, it occurs three times in this operation scheduling list. The first J_1

corresponds to the first operation of job J_1. The first J_3 corresponds to the first operation of job J_3. The second J_1 corresponds to the second operation of job J_1, and so on.

In fact, in the computing algorithm, we use for (X), the form given by Figure 2(b), where a repetition of jobs numbers is used.

5.2 Initialisation phase

In this phase, we start by setting the initial parameters of the algorithm, such as: the colony size (*CS*), the number of employed bees, onlooker bees (number of food sources *SN=CS/2*), the parameter *Limit* and the maximum number of iterations (*NI_Max*). Furthermore, the job shop parameters such as the number of jobs, number of machines, the job processing time on each machine and the job machine sequence will be given.

Next, the initial solutions are randomly generated. Although the solutions are constructed by generating a random suite of jobs numbers (according to the size of the JSP), they will be feasible solutions (feasible schedules). Indeed, when the '*operation-based representation*' with '*job repetition*' is used for solution representation, the precedence constraints in jobs processing are always respected.

The ABC algorithm seeks to find the solution that maximises the fitness. However, our objective is to solve the JSP with minimisation of the Makespan (C_{max}: maximum completion time). Therefore, the fitness in the CABC will be calculated as follow:

$$fit_i = \frac{1}{C_{\max_i}} = \frac{1}{F(X_i)} \tag{14}$$

where fit_i is the fitness value of the food source X_i and $F(X_i)$ is the objective function value of the food source X_i; (noted $C_{max}(X_i)$ or C_{maxi}).

Algorithm 2 Combinatorial artificial bee colony algorithm

```
Input: $CS$, $\beta$, $Limit$, $NI\_Max$, JSP parameters.
Output: Global best solution $X_{Gbest}$, $C^*_{max}$
1   $SN = CS/2$
2   For i =1 to $SN$
3       Generate randomly the food source $X_i$
4       Initialise the invalid trials counter $trial_i = 0$
5   End For
6   $it = 1$
7   repeat
8       Employed bees phase
9       Onlooker bees phase
10      Scout bees phase
11      Memorise the best solution found so far.
12      $it = it + 1$
13  Until $it = NI\_Max + 1$
14  Return $X_{Gbest}$
```

In CABC algorithm, another variable $trial_i$ is assigned to each food source (X_i). $trial_i$ is a counter of unsuccessful trials where the food source (X_i) is not improved. It is an indicator to find food sources (solutions) to be abandoned in the next iterations after *Limit* fruitless trials. First, as initial value, each $trial_i$ is set to 0, where, $i = 1,2,\ldots,SN$.

5.3 Employed bees phase

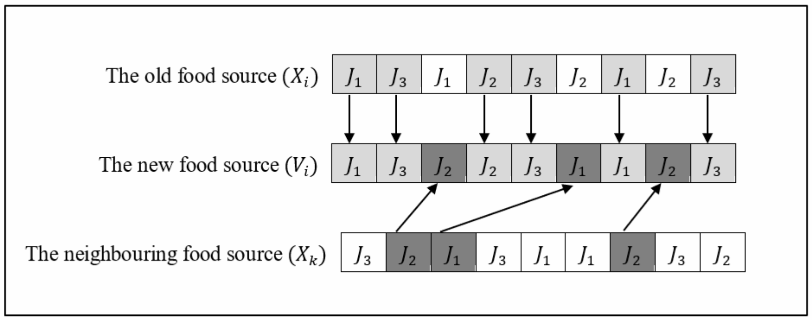
In this phase, the employed bee generates a new food source (V_i), by *updating* the old food source already exploited (X_i), with a neighbouring food source (X_k) randomly taken from other food sources in the population of solutions. The pseudo code of this phase is given by Algorithm 3.

V_i is the result of a crossover between (X_i) and (X_k). In order to obtain a new valid solution (V_i), the position base crossover (PBX) mechanism is used (Syswerda, 1991). The PBX procedure is described in the following steps with an example illustrated in Figure 3.

Step A Initially the new food source is created by copying in a new vector (solution) (V_i) all the job operations of the old food source (X_i) except those that will be randomly selected to be changed. The number of selected operations to be changed is noted *N_ch*. For our JSP with *n* jobs and *m* machines, the solution is given by the '*operation-based representation*' with '*job repetition*'. So, in the solution (X) each job appears exactly *m* times. Hence, to avoid redundant changes, the number of changes *N_ch* is limited to the number of operations required for each job. Which means that: *N_ch* = *m* (the number of machines). The *N_ch* empty positions in (V_i) will be filled from (X_k) as given in the next step.

Step B The job operations on the neighbouring food source (X_k) will be taken from the left to the right and placed into empty positions of the operation scheduling list in the new food source (V_i), from the left to the right also. To ensure that each job will be included exactly *m* times in the new food source (V_i), if any job has already been selected for *m* times [from either (X_i) or (X_k)], it will be skipped and the next job will be considered.

Figure 3 Updating solution by PBX crossover


The old food source (X_i)
J_1 J_3 J_1 J_2 J_3 J_2 J_1 J_2 J_3
The new food source (V_i)
J_1 J_3 J_2 J_2 J_3 J_1 J_1 J_2 J_3
The neighbouring food source (X_k)
J_3 J_2 J_1 J_3 J_1 J_1 J_2 J_3 J_2

Once the new food source (V_i) is obtained by updating the old food source by the PBX method, a greedy selection is applied. The old food source (X_i) in the employed bees memory will be replaced by the new candidate food source (V_i) if this latter has a better fitness value. In such enhancing update, $trial_i$ is set to '0'. Otherwise, if the employed bee does not change its food source (X_i), the trial counter $trial_i$ is increased by 1.

If after several trials, $trial_i$ exceeds the value defined by '*Limit*', its related employed bee will turn into a scout bee and after doing a random search, it will turn back to be an employed bee again.

Algorithm 3 Employed bees phase procedure

```
1   For all Employed Bees $X_i$ ; i = 1 to $SN$
2       Select randomly a neighbour $X_k$ from the colony
3       Produce new solution $V_i$ by updating $X_i$ with $X_k$ using a $PBX$ crossover
4       If $C_{max}(V_i) < C_{max}(X_i)$ then
5           $X_i \leftarrow V_i$
6           $trial_i = 0$
7           $X_{Gbest} = update\ (X_{Gbest}, V_i)$
8       Else
9           $trial_i = trial_i + 1$
10      End if
11  End For
```

5.4 Onlooker bees phase

This phase starts by evaluating the quality of all employed bees food sources. After that, *SN* onlooker bees will be recruited, (*SN* = number of employed bees). Indeed, *SN* onlooker bees will exploit new food sources by *selecting* and *updating* the *more interesting* employed bees food sources. The pseudo code of this phase is given by Algorithm 4 where the details are as follows:

5.4.1 Selection

The selection principle used in the CABC is different from that of the ABC. In the fundamental ABC algorithm, the selection is given by using a '*roulette wheel selection*' method (Goldberg, 1989). For this method, the food source is selected depending on its probability value p_i calculated by expression (15).

$$p_i = \frac{fit_i}{\sum_{k=1}^{SN} fit_k} \tag{15}$$

p_i is compared with the value of $\propto$ randomly generated between [0,1]. If $p_i > \propto$, the corresponding (X_i) is selected. However, for the JSP, the fitness fit_i expressed as the reverse of $C_{max}(X_i)$, takes relatively very small values. Then, the probability p_i will be small as well. p_i could be 1,000 times smaller than $\propto$. Therefore, in the case of the JSP, the '*roulette wheel selection*' may not be a suitable method of selection for the onlooker bees.

In the CABC, we keep the fact that, for better exploitation, the onlooker bees should select only the more interesting food sources among those proposed by the employed bees. However, the selection will not be based on the probability but on the C_{max} value. Since our purpose is to minimise C_{max}, it is considered that the solution is better when the corresponding C_{max} value is lower. Based on the fact that the possibility of finding better solutions are high in the neighbourhood of a good solution, most of onlooker bees will select to move to good solutions with the lowest C_{max}.

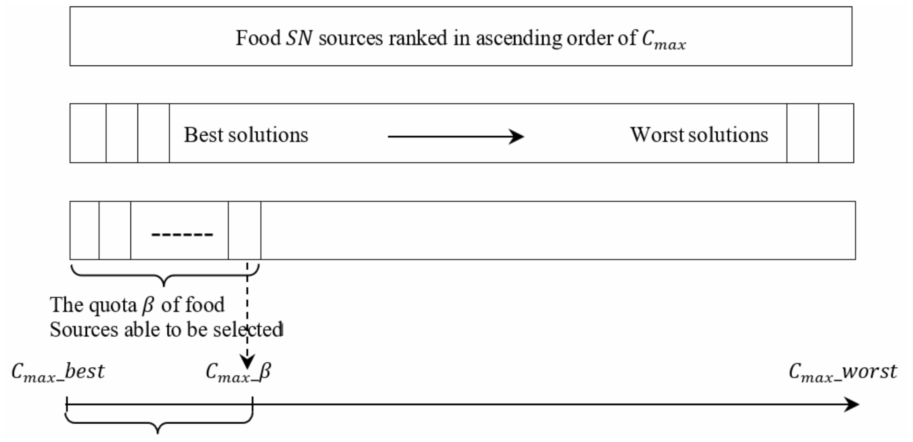
Our rank-based selection procedure can be described as follows:

- Step 1: As shown in Figure 4, the food sources of the employed bees are ranked in ascending order of C_{max} values.
- Step 2: Only a quota of the first good ranked food sources could eventually be selected. The estimation of this quota is given by a percentage β. For example, for $\beta = 25\%$, the higher C_{max} of the 25% of good ranked food sources is noted $C_{max\_}\beta$.
- Step 3: The onlooker bees can select only the food sources (X_i) having a $C_{max}(X_i) \leq C_{max\_}\beta$.

5.4.2 Updating

The updating of the selected food source (X_i) will be done in the same manner as in the employed bee phase. A greedy selection is also applied between the selected food source (X_i) and the new generated food source (V_i) in order to keep the best solution. The *trial*$_i$ counters are also updated.

Figure 4 Food sources candidates for selection


Food SN sources ranked in ascending order of C_{max}
Best solutions
Worst solutions

The quota β of food
Sources able to be selected
$C_{max\_}best$
$C_{max\_}\beta$
$C_{max\_}worst$

In the CABC, the onlooker bees phase provides the intensification of local search on the relatively promising chosen solutions. This means that only the best food sources proposed by the employed bees will be candidates for an updating in the onlooker bee phase. The aim is to further improve the quality of the solution found by the employed bees.

Algorithm 4 Onlooker bees phase procedure

```
1  For all Employed Bees $X_i$; i = 1 to $SN$
2  Evaluate $C_{max}(X_i)$
3  End For
4  Rank all $C_{max}(X_i)$; i = 1 to $SN$, in ascending order
5  Calculate $C_{max\_}\beta$ corresponding to quota $\beta$ of relatively good solutions.
6  $t = 0, i = 1$
7  While $t < SN$
8        If $C_{max}(X_i) < C_{max\_}\beta$ then
9            $t = 1 + 1$
10       Select randomly a neighbour $X_k$ from the colony
11       Produce new solution $V_i$ by updating $X_i$ with $X_k$ using a $PBX$ crossover
12       If $C_{max}(V_i) < C_{max}(X_i)$ then
13           $X_i \leftarrow V_i$
14           $trial_i = 0$
15           $X_{Gbest} = update\ (X_{Gbest}, V_i)$
16       Else
17           $trial_i = trial_i + 1$
18       End if
19    End if
20    $i = i + 1$
21    If $i$ = = SN then
22        $i = 1$
23    End if
24 End while
```

5.5 Scout bees phase

After carrying out the employed bees and onlooker bees phases, the solutions that have not been improved after many trials may be trapped in local optima. The scout bees phase aims to deal with this situation. Indeed, if a food source (solution) cannot be improved for a predetermined number of trials, denoted *Limit*, then the employed bee associated with this food source becomes a scout bee. This scout bee finds randomly a new food source and becomes an employed bee again.

The parameter *Limit* plays an important role in CABC by providing a balance between exploration and exploitation. A small value of *Limit* parameter favours exploration over exploitation, whereas the reverse is true for its large value.

In the fundamental ABC, only one scout is used. Whereas in CABC algorithm, the number of scout bees is not a fixed number. At each iteration, *all* the employed bees for which the corresponding food source has not been improved after *Limit* trials will become scout bees and their counters of trials are reset to zero. This step is detailed by Algorithm 5.

Algorithm 5 Scout bees phase procedure

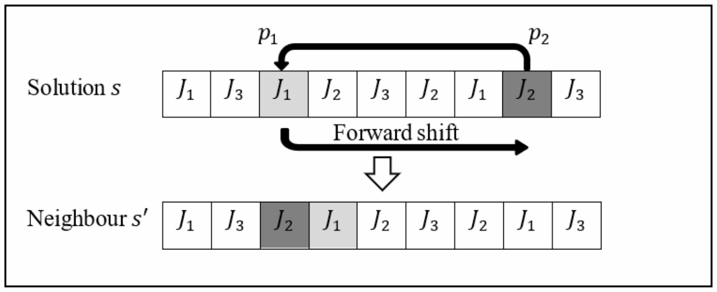
```
1  For all Bees $X_i$ ; i = 1 to $SN$
2      If $trial_i \geq Limit$ then
3          Generate randomly new $X_i$
4          Evaluate the new solution $X_i$
5          $trial_i = 0$
6      End if
7  End For
```

6 Hybrid combinatorial artificial bee colony (SILS_CABC) algorithm

The effectiveness of a search process in all population-based nature-inspired algorithms depends on two components: exploration and exploitation (Črepinšek et al., 2013). For the CABC algorithm, the exploration is well ensured by the scout bees phase. However, it is still poor at exploitation which is done by the employed and onlooker bees phases. In fact, at each iteration, each solution is updated one time at the employed bees phase and if this solution is good enough it could be selected for another update at the onlooker bees phase. Nevertheless, this is not enough for a good exploitation.

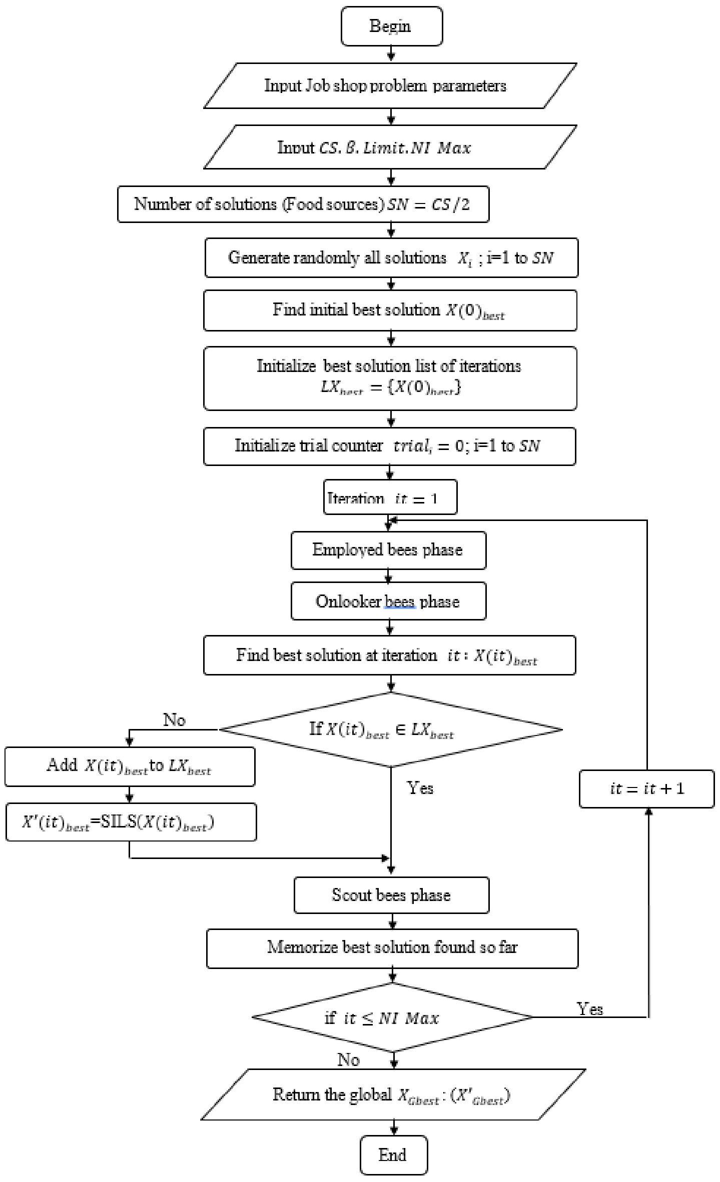
In order to enhance the exploitation of the CABC algorithm, a sequential hybridisation with a new procedure of local search is done. We called this proposed procedure: 'SILS'. SILS is applied to update the best solution of each iteration after the employed and onlooker bees phases. If at the iteration '*t*' the best solution $X(t)_{best}$ was already found and exploited by SILS in one of the precedent iterations (< *t*), the procedure SILS is skipped in this case. This will avoid redundant local searches. The hybrid release is noted SILS_CABC, and its flowchart is given in Figure 6.

Figure 5 Insert process for *Local Search* procedure of SILS algorithm


p_1
p_2
Solution s
J_1 J_3 J_1 J_2 J_3 J_2 J_1 J_2 J_3
Forward shift
Neighbour s'
J_1 J_3 J_2 J_1 J_2 J_3 J_2 J_1 J_3

SILS is a simple metaheuristic that iteratively applies local search to refine the current best solution $X(t)_{best}$ to its local optima $X'(t)_{best}$. The SILS procedure is given by Algorithm 6 and it operates as follows. Let consider that $X(t)_{best}$, the best solution done at the iteration '*t*', is the input solution for the SILS procedure. After some iterated improvement, given by some local search cycles, the output solution will be $X'(t)_{best}$. So, $X'(t)_{best}$ = *SILS_procedure* ($X(t)_{best}$). Each local search cycle starts with the input (seed) solution, noted s_{in} . s^* notes the output solution, s^* = *local search*(s_{in}). It is the best solution found by a local search in the neighbourhood of s_{in}. The first cycle starts by $s_{in} = X(t)_{best}$. The first solution found s^* will be the input solution in the second local search cycle, and so one for the next cycles, where the s_{in} will be the solution found in the precedent cycle. These cycles are stopped when the local search cannot improve the solution s_{in}. Indeed, when in the same cycle $s^* = s_{in}$, we consider that the local minimum is reached. Therefore, s^* is the final solution of SILS procedure, $X'(t)_{best} = s^*$.

Figure 6 Flow chart of the hybrid algorithm SILS_CABC for JSP


Begin
Input Job shop problem parameters
Input $CS, \beta, Limit, NI\ Max$
Number of solutions (Food sources) $SN = CS/2$
Generate randomly all solutions X_i ; i=1 to SN
Find initial best solution $X(0)_{best}$
Initialize best solution list of iterations $LX_{best} = \{X(0)_{best}\}$
Initialize trial counter $trial_i = 0$; i=1 to SN
Iteration $it = 1$
Employed bees phase
Onlooker bees phase
Find best solution at iteration $it : X(it)_{best}$
If $X(it)_{best} \in LX_{best}$
No
Yes
Add $X(it)_{best}$ to LX_{best}
$X'(it)_{best}$=SILS($X(it)_{best}$)
$it = it + 1$
Scout bees phase
Memorize best solution found so far
if $it \leq NI\ Max$
Yes
No
Return the global X_{Gbest} : (X'_{Gbest})
End

SILS seams similar to iterated local search (ILS) metaheuristic but it differs from the latter in some details. In fact, for the ILS four components have to be specified: *generate initial solution, modification, local search, and acceptance criterion* (Lourenço et al., 2003). Whereas, SILS is based only on the *Local Search* component. For SILS the initial solution is already done by the solution of CABC algorithm ($s_0 = X(t)_{best}$). In order to preserve the quality of the solution given by CABC, SILS aims to exploit the current best solution only in its basin of attraction. Hence, the resulting solution is injected into the next local search without any prior modification.

Furthermore, as there is no modification for the solution to exploit ($s_{in} = s^*$), the *acceptance criterion* test is not needed; because the solution s^* obtained by *Local Search*, will be surely better than s_{in} or at worst the same. SILS differs also from ILS in the number of runs. ILS runs for a fixed number of iterations but SILS continues to generate a chain of candidate solutions until reaching the local minimum ($s^* = s_{in}$).

The SILS algorithm, as presented by Algorithm 6 seems very simplistic; however, in the case of our study, it proved that it can be very effective and even more efficient than ILS approach.

Algorithm 6 SILS procedure

```
1  Input : $s_0 = X(t)_{best}$
2  $s^* = s_0$
3  Repeat
4      $sin = s^*$
5      $s^* = Local\ Search\ (s_{in})$
6  Until $s^* = s_{in}$
7  Output : $X'(t)best = s^*$
```

Algorithm 7 Local Search procedure

```
1  Input: $s = s_{in}$
2  $s^* = s$
3  For $p_1 = 1$ to $D - 1$
4      For $p_2 = p_1 + 1$ to $D$
5          $s' = Insert\_process(p_2, p_1, s)$
6          If $C_{max}(s') < C_{max}(s^*)$ then
7              $s^* = s'$
8          End if
9      End For
10 End For
11 Return $s^*$
```

Regarding the local search component given by the Algorithm 7 of *local search* procedure, we consider the insertion neighbourhood structure. The *Insert_process*(p_2, p_1, s) means removing the job operation in solution s from the position p_2 and inserting it in the position p_1. All the job operations from the position p_1 to $p_2 - 1$ will undergo a forward shift to fill the lack at the position p_2, as shown in Figure 5. All the neighbours will be considered. So, each job operation removed from its original position p_2, is inserted into all possible positions p_1 from 1 to $D - 1$, where D denotes the dimension of the solutions and is expressed by $D = n \times m$ (the number of all operations to be executed in the ($n \times m$) job shop).

For selecting the best neighbour, the best-improvement strategy is considered. Hence, all the possible neighbours (s') are exhaustively explored, but only the best solution (s^*) with the lowest C_{max} is returned.

7 Experiment results

In order to investigate the efficiency of CABC and SILS_CABC algorithms, numerical simulations are performed on many instances of JSP benchmarks taken from the Operations Research Library (OR-Library) (Beasley, 1990). All these considered benchmarks are strongly NP-hard problems because the size of their Instances ranges from 6 to 50 jobs and 5 to 20 machines. (The machines number $m \geq 3$).

In this section, we have focused only on the instances for which we achieved the best known solution. The instances under consideration are those of Fisher and Thompson (1963), referred as Ft06, and a part of those introduced by Lawrence (1984), referred as La01–La15. In order to have a comparison basis for these benchmarks, the survey of Jain and Meeran (1999) presents their structures and how the best solutions known previously were found. It even provides a classification of hard and easy problems.

Based on a sensitivity analysis on several combinations of parameters values, the algorithm parameters are set as follows:

- *Colony Size (CS)* = 1000, then the size of the population of solutions is given by *SN* = *CS/2* = 500 food sources
- 25% of ranked solutions are able to be selected for updating, then $\beta = 0.25$
- for abandonment criteria, *Limit* = 20 trials
- for termination criteria, *NI_Max* = 1,000 iterations.

Table 2 reports the experimental results of the combinatorial artificial bee colony algorithm CABC and its hybrid release SILS_CABC. Both algorithms are run 10 times on each problem instance. This table shows the problem name (Instance), the problem size (n × m) with n jobs and m machines (Size), the best known solution (BKS) for the instance, the best solution found by each algorithm ($Best\_C_{max}$), the average calculated on all C_{max} obtained by the 10 runs ($Avrg\_C_{max}$), the relative percent deviation of best solution with respect to the BKS (RPD_{Best}) calculated by equation (16), the relative percent deviation of average solution with respect to the BKS (RPD_{Avrg}) calculated by equation (17) and finally the minimal number of iteration to reach optimal solution ($NIROS_{min}$).

$$RPD_{Best} = \frac{\left(Best\_C_{\max} - BKS\right)}{BKS} \times 100 \tag{16}$$

$$RPD_{Avrg} = \frac{\left(Avrg\_C_{\max} - BKS\right)}{BKS} \times 100 \tag{17}$$

When RPD_{Best} = 0 for an instance this means that its optimal solution is obtained and furthermore, if RPD_{Avrg} = 0, then this optimal solution is obtained with 10 replications of simulation. The dash '-' in the $NIROS_{min}$ column means that the optimal solution is not reached for this algorithm.

It can be observed from Table 2, that the SILS_CABC algorithm leads to the BKS for all instances from La01 to La15. Furthermore, it performs better in all the replications for the instances Ft06 and La01 as well as La05-La15. We can also conclude that even the CABC can deal with the NP-hard problem of JSP, because it solves optimally almost all the considered instances, except La03. Comparing both algorithms, it is clear that the hybrid release SILS_CABC brings interesting improvements in terms of solution quality and convergence speed. Indeed, we can notice a significant reduction in the parameter $NIROS_{min}$ of SILS_CABC compared with CABC.

In this paper we fixed the population size *SN*= 500 food sources and the maximum number of iteration *NI_Max* = 1,000 iterations. However, in our simulation tests it is possible to get the BKSs for some instances with parameters values less than those fixed here. For example for the small instance La05 the SILS_CABC algorithm leads the BKS at the first iteration, with only *SN* = 5 and a simulation duration less than 1 second.

Furthermore, following the size criteria, the simulation tests showed that the instances could be classified into h*ard* and *harder* problems. Indeed, comparing the sizes of instance La03 (10x5) and the instance La14 (20x5), La14 is considered larger than La03. However the simulation shows that it is easier to solve La14 than La03. Solving La14 is less time consuming and can be done with only a small population size (*SN*). The same observations can be done for some instances having the same size. Example: comparing La05 with La03, or La14 with La15.

In Table 2, the parameter $NIROS_{min}$ highlights the possibility to classify the test benchmarks into *hard* or *harder* problems. With SILS_CABC algorithm: for La03, $NIROS_{min}$ = 780; for La04, $NIROS_{min}$ = 94 and for La05, $NIROS_{min}$ = 1. This means that the BKS could be obtained for La03 after 780 iterations, it can also be found for La04 after 94 iterations. However, it is reached for La05 at the first iteration. Hence, it is relatively easier to solve La05 than La03. Consequently, La05 can be classified as hard JSP benchmark and La03 can be viewed as harder one.

Further, to illustrate the convergence characteristics of proposed SILS_CABC, the Figure 7 shows the Gap of the 5 instances La1-La5 given by equation (18). In the present study, it is expressed by the relative percent deviation between the solutions obtained at all iterations generated by the algorithm and the Best Known optimal Solution.

$$GAP = \frac{\left(c_{\max_{iter}} - BKS\right)}{BKS} \times 100; \; iter = 1 \;\; to \;\; NI\_Max \tag{18}$$

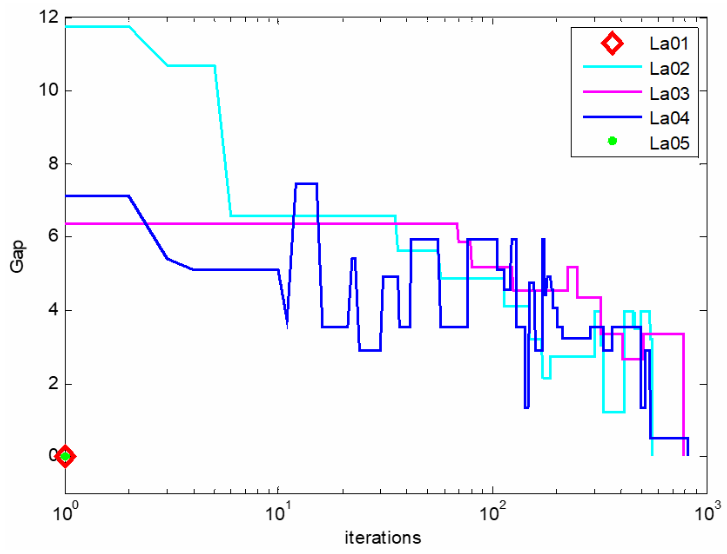
The Gap depicted in Figure 7 confirms that there are some hard and harder instances to solve. Indeed, the Gap of La01 and La05 is given by only one point because it happens that the algorithm converges very rapidly to the optimal solution with only one iteration and even with population size *SN* = 5.

Furthermore, concerning the other instances of JSP benchmarks in OR-library that are not presented in Table 2, simulations are also done, and we found that the RPD_{Best} range is from [0–5%], hence the proposed algorithms are quite efficient even for large instances.

Table 2 Comparison between results of CABC and SILS_CABC

Instance	*Size*	*BKS*	*Best_C$_{max}$*		*Avrg_C$_{max}$*		*RPD$_{Best}$ (%)*		*RPD$_{Avrg}$ (%)*		*NIROS$_{min}$*	
			CABC	*SILS CABC*	*CABC*	*SILS CABC*	*CABC*	*SILS CABC*	*CABC*	*SILS CABC*	*CABC*	*SILS CABC*
Ft06	6x6	55	55	55	55	55	0	0	0	0	1	1
La01	10x5	666	666	666	666	666	0	0	0	0	10	1
La02	10x5	655	655	655	661.3	657.3	0	0	0.9618	0.3511	349	154
La03	10x5	597	604	597	609.4	606,3	1.1725	0	2.0771	1.5578	-	780
La04	10x5	590	590	590	596.3	591.3	0	0	1.0678	0.2203	839	94
La05	10x5	593	593	593	593	593	0	0	0	0	1	1
La06	15x5	926	926	926	926	926	0	0	0	0	3	1
La07	15x5	890	890	890	890	890	0	0	0	0	75	4
La08	15x5	863	863	863	863	863	0	0	0	0	1	1
La09	15x5	951	951	951	951	951	0	0	0	0	2	1
La10	15x5	958	958	958	958	958	0	0	0	0	1	1
La11	20x5	1,222	1,222	1,222	1,222	1,222	0	0	0	0	23	1
La12	20x5	1,039	1,039	1,039	1,039	1,039	0	0	0	0	8	1
La13	20x5	1,150	1,150	1,150	1,150	1,150	0	0	0	0	13	1
La14	20x5	1,292	1,292	1,292	1,292	1,292	0	0	0	0	1	1
La15	20x5	1,207	1,207	1,207	1,212.4	1,207	0	0	0.4474	0	701	4

Figure 7 Convergence characteristics for instances: La01-La05 (see online version for colours)



Moreover, in order to analyse the impact of the improvement obtained by SILS_CABC compared with CABC in term of Makespan, both algorithms averages values are subjected to statistical analysis using t.test at a level of significance of 5%. Therefore, there are two hypothesis: the null hypothesis (H_0) which means that the algorithms results means are equals and the alternative hypothesis (H_1) representing the case where $CABC_{Avrg\_C_{\max}} > SILS\_CABC_{Acvrg\_C_{\max}}$.. After running the test, the obtained *p*-value is about 0,023. Since it is less than 5% (the significance level), the null hypothesis should be rejected. Therefore, there are significant differences between the two algorithms performances. As a result, SILS_CABC improves the Makespan compared with CABC.

8 Conclusions and future works

In this study, we address the JSP with the objective of minimising the Makespan to further improve throughputs and the system productivity. JSP is known to be NP-hard in the strong sense. Hence, due to its wide applicability and inherent difficulty, it is considered as one of the most popular scheduling problems.

To solve JSP, we opted for the Artificial Bee Colony metaheuristic (ABC). The ABC algorithm has been recently introduced and tested on various optimisation problems. However, most of them are of a continuous nature. Therefore, our first contribution consists to adapt the continuous version of the ABC algorithm to the combinatorial problem of the job shop. The proposed metaheuristic formulated by the Combinatorial Artificial Bee Colony (CABC) algorithm, is based on the intelligent behaviour of honey bees for seeking a quality food source in nature. It is articulated in three phases: the employed bees phase and onlooker bees phases that ensure the exploitation and the scout bees phase for a good exploration.

At the employed and onlooker bees phases, we update the current solution by performing a Position Based Crossover with another solution chosen randomly. Regarding the selecting of solutions to be updated in the onlooker bees phase, we use the Rank-Based Selection.

At each iteration of CABC algorithm, each solution is firstly updated at the employed bees phase. In addition, it could be selected for a second update at the onlooker bees phase if this solution is good enough. Hence, we noticed that the proposed concept with only two updating processes is poor in exploitation.

Furthermore, in order to enhance the exploitation side of the CABC algorithm, a sequential hybridisation with a new procedure of local search has been introduced. The proposed procedure, called: 'SILS' is a simple metaheuristic that iteratively applies local search to refine the current iteration best solution of CABC algorithm, but only in the case where this solution has not been subjected to the SILS procedure in the previous iterations.

The numerical simulations confirmed the numerical correctness and the efficiency of both algorithms CABC and SILS_CABC. Indeed, with these algorithms, the optimal solution was obtained for many job shop benchmarks problems drawn from the literature.

The experimental results subjected to statistical analysis t.test demonstrated also that the hybrid release SILS_CABC effectively improved the exploitation of the combinatorial proposed algorithm. Hence, it performs the best in terms of solution quality evaluated by the Makespan and in terms of convergence speed.

For a better adaptation of the used bee colony metaheuristic, the proposed algorithms require further research and improvements to propose other practical solutions. Therefore, our work can be enriched following several research directions:

1 It could be interesting to propose an optimal parameter setting by a well-known tuning approach based on other artificial intelligence techniques such as Artificial neural network.

2 An enhancement can be done in the updating and the selecting steps of the algorithms in order to solve rapidly the JSP with other complex instances.

3 It would be useful to extend the ideas proposed to different objective functions or multi-objective JSPs.

4 It is also worthwhile to apply the proposed algorithms to other kind of combinatorial optimisation problems.

References

Abubaker, A., Baharum, A. and Alrefaei, M. (2019) 'A pruned Pareto set for multi-objective optimisation problems via particle swarm and simulated annealing', *International Journal of Operational Research*, Vol. 35, No. 1, pp.67–86.

Artigues, C. and Feillet, D. (2008) 'A branch and bound method for the job-shop problem with sequence-dependent setup times', *Annals of Operations Research*, Vol. 159, No. 1, pp.135–159.

Aydin, M.E. and Fogarty, T.C. (2004) 'A simulated annealing algorithm for multi-agent systems: a job-shop scheduling application', *Journal of intelligent manufacturing*, Vol. 15, No. 6, pp.805–814.

Bajer, D. and Zorić, B. (2019) 'An effective refined artificial bee colony algorithm for numerical optimisation', *Information Sciences*, Vol. 504, pp.221–275.

Beasley, J.E. (1990) 'OR-Library: distributing test problems by electronic mail', *Journal of the Operational Research Society*, Vol. 41, No. 11, pp.1069–1072.

Benttaleb, M., Hnaien, F. and Yalaoui, F. (2018) 'Two-machine job shop problem under availability constraints on one machine: makespan minimization', *Computers & Industrial Engineering*, Vol. 117, pp.138–151.

Çaliş, B. and Bulkan, S. (2015) 'A research survey: review of AI solution strategies of job shop scheduling problem', *Journal of Intelligent Manufacturing*, Vol. 26, No. 5, pp.961–973.

Chaouch, I., Driss, O.B. and Ghedira, K. (2019) 'A novel dynamic assignment rule for the distributed job shop scheduling problem using a hybrid ant-based algorithm', *Applied Intelligence*, Vol. 49, No. 5, pp.1903–1924.

Črepinšek, M., Liu, S. H. and Mernik, M. (2013) 'Exploration and exploitation in evolutionary algorithms: a survey', *ACM Computing Surveys (CSUR)*, Vol. 45, No. 3, p.35.

Dao, T.K., Pan, T.S. and Pan, J.S. (2018) 'Parallel bat algorithm for optimizing makespan in job shop scheduling problems', *Journal of Intelligent Manufacturing*, Vol. 29, No. 2, pp.451–462.

Dehnavi-Arani, S., Saidi-Mehrabad, M. and Ghezavati, V. (2019) 'An integrated model of cell formation and scheduling problem in a cellular manufacturing system considering automated guided vehicles' movements', *International Journal of Operational Research*, Vol. 34, No. 4, pp.542–561.

Faridi, A.Q., Sharma, S., Shukla, A., Tiwari, R. and Dhar, J. (2018) 'Multi-robot multi-target dynamic path planning using artificial bee colony and evolutionary programming in unknown environment', *Intelligent Service Robotics*, Vol. 11, No. 2, pp.171–186.

Fisher, H. and Thompson, G.L. (1963) 'Probabilistic learning combinations of local job-shop scheduling rules', in Muth, J.F. and Thompson, G.L. (Eds.): *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, pp.225–251.

Gao, K.Z., Suganthan, P.N., Pan, Q.K., Chua, T.J., Chong, C.S. and Cai, T.X. (2016) 'An improved artificial bee colony algorithm for flexible job-shop scheduling problem with fuzzy processing time', *Expert Systems with Applications*, Vol. 65, pp.52–67.

Garey, M.R., Johnson, D.S. and Sethi, R. (1976) 'The complexity of flowshop and jobshop scheduling', *Mathematics of Operations Research*, Vol. 1, No. 2, pp.117–129.

Garey, M.R. and Johnson, D.S. (1979) *A Guide to the Theory of the NP-Completeness*, Computers and Intractability, Freeman, New York.

Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, USA.

González, M.A., Vela, C.R., González-Rodríguez, I. and Varela, R. (2013) 'Lateness minimization with Tabu search for job shop scheduling problem with sequence dependent setup times', *Journal of Intelligent Manufacturing*, Vol. 24, No. 4, pp.741–754.

Guizzi, G., Revetria, R., Vanacore, G. and Vespoli, S. (2019) 'On the open job-shop scheduling problem: a decentralized multi-agent approach for the manufacturing system performance optimization', *Procedia CIRP*, Vol. 79, pp.192–197.

Huang, R.H. and Yu, T.H. (2017) 'An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting', *Applied Soft Computing*, Vol. 57, pp.642–656.

Huang, R.H., Yang, C.L. and Cheng, W.C. (2013) 'Flexible job shop scheduling with due window a two-pheromone ant colony approach', *International Journal of Production Economics*, Vol. 141, No. 2, pp.685–697.

Hung, H.C., Lin, B.M., Posner, M.E. and Wei, J.M. (2019) 'Preemptive parallel-machine scheduling problem of maximizing the number of on-time jobs', *Journal of Scheduling*, Vol. 22, No. 4, pp.413–431.

Jain, A.S. and Meeran, S. (1999) 'Deterministic job-shop scheduling: past, present and future'. *European Journal of Operational Research*, Vol. 113, No. 2, pp.390–434.

Jiang, J., Wu, D., Chen, Y., Yu, D., Wang, L. and Li, K. (2019) 'Fast artificial bee colony algorithm with complex network and naive Bayes classifier for supply chain network management', *Soft Computing*, Vol. 23, No. 24, pp.13321–13337.

Karaboga, D. and Akay, B. (2009) 'A comparative study of artificial bee colony algorithm', *Applied Mathematics and Computation*, Vol. 214, No. 1, pp.108–132.

Karaboga, D. and Basturk, B. (2007) 'A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm', *Journal of Global Optimization*, Vol. 39, No. 3, pp.459–471.

Karaboga, D. and Basturk, B. (2008) 'On the performance of artificial bee colony (ABC) algorithm', *Applied Soft Computing*, Vol. 8, No. 1, pp.687–697.

Karaboga, D., Gorkemli, B., Ozturk, C. and Karaboga, N. (2014) 'A comprehensive survey: artificial bee colony (ABC) algorithm and applications', *Artificial Intelligence Review*, Vol. 42, No. 1, pp.21–57.

Karaboga, D. (2005) *An Idea based on Honey Bee Swarm for Numerical Optimization*, Technical report-TR06, Erciyes University, Kayseri, Turkey.

Khambhampati, S., Calyam, P. and Zhang, X. (2018) 'A tabu search algorithm for a capacitated clustering problem', *International Journal of Operational Research*, Vol. 33, No. 3, pp.387–412.

Khatibi, S., Bafruei, M.K. and Rahmani, M. (2019) 'Modelling a bi-objective airport gate scheduling with controllable processing time using hybrid NSGA-II and VNS algorithm', *International Journal of Operational Research*, Vol. 34, No. 1, pp.1–27.

Kumar, H., Kumar, P. and Sharma, M. (2019) 'A genetic algorithm for a flow shop scheduling problem with breakdown interval, transportation time and weights of jobs', *International Journal of Operational Research*, Vol. 35, No. 4, pp.470–483.

Kundakcı, N. and Kulak, O. (2016) 'Hybrid genetic algorithms for minimizing Makespan in dynamic job shop scheduling problem', *Computers & Industrial Engineering*, Vol. 96, pp.31–51.

Lawrence, S. (1984) *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

Lei, D. (2012) 'Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling', *Applied Soft Computing*, Vol. 12, No. 8, pp.2237–2245.

Lenstra, J.K. and Rinnooy Kan, A.H.G. (1979) 'Computational complexity of discrete optimization problems', *Annals of Discrete Mathematics*, Vol. 4, pp.121–140.

Lian, Z., Jiao, B. and Gu, X. (2006) 'A similar particle swarm optimization algorithm for job-shop scheduling to minimize Makespan', *Applied Mathematics and Computation*, Vol. 183, No. 2, pp.1008–1017.

Lin, T.L., Horng, S.J., Kao, T.W., Chen, Y.H., Run, R.S., Chen, R.J. et al. (2010) 'An efficient job-shop scheduling algorithm based on particle swarm optimization', *Expert Systems with Applications*, Vol. 37, No. 3, pp.2629–2636.

Lourenço, H. R., Martin, O. C. and Stützle, T. (2003) 'Iterated local search', in Glover, F. and Kochenberger, G.A. (Eds.): *Handbook of Metaheuristics*, Springer, Boston, MA, pp.320–353.

Manne, A.S. (1960) 'On the job-shop scheduling problem', *Operations Research*, Vol. 8, No. 2, pp.219–223.

Masmoudi, O., Delorme, X. and Gianessi, P. (2019) 'Job-shop scheduling problem with energy consideration', *International Journal of Production Economics*, Vol. 216, pp.12–22.

Moeini, R. (2018) 'Different hydraulic analysis conditions for sewer network design optimisation problem using three different evolutionary algorithms', *International Journal of Operational Research*, Vol. 33, No. 4, pp.512–537.

Mohammadnejad, A. and Eshghi, K. (2019) 'An efficient hybrid meta-heuristic ant system for minimum sum colouring problem', *International Journal of Operational Research*, Vol. 34, No. 2, pp.269–284.

Nayak, A., Lee, S. and Sutherland, J.W. (2019) 'Dynamic load scheduling for energy efficiency in a job shop with on-site wind mill for energy generation', *Procedia CIRP*, Vol. 80, pp.197–202.

Nouri, H.E., Driss, O.B. and Ghédira, K. (2016) 'Hybrid metaheuristics for scheduling of machines and transport robots in job shop environment', *Applied Intelligence*, Vol. 45, No. 3, pp.808–828.

Ozolins, A. (2018) 'Bounded dynamic programming algorithm for the job shop problem with sequence dependent setup times', *Operational Research*, https://doi.org/10.1007/s12351-018-0381-6.

Ramezanian, R. and Rahmani, D. (2017) 'MILP formulation and genetic algorithm for flow shop scheduling problem with missing operations', *International Journal of Operational Research*, Vol. 30, No. 3, pp.321–339.

Roshanaei, V., Balagh, A.K.G., Esfahani, M.M.S. and Vahdani, B. (2010) 'A mixed-integer linear programming model along with an electromagnetism-like algorithm for scheduling job shop production system with sequence-dependent set-up times', *The International Journal of Advanced Manufacturing Technology*, Vol. 47, Nos. 5–8, pp.783–793.

Rostami, S., Creemers, S. and Leus, R. (2019) 'Precedence theorems and dynamic programming for the single-machine weighted tardiness problem', *European Journal of Operational Research*, Vol. 272, No. 1, pp.43–49.

Seidgar, H., Rad, S.T. and Shafaei, R. (2017) 'Scheduling of assembly flow shop problem and machines with random breakdowns', *International Journal of Operational Research*, Vol. 29, No. 2, pp.273–293.

Shokoufi, K., Rezaeian, J., Shirazi, B. and Mahdavi, I. (2019) 'Preemptive just-in-time scheduling problem on uniform parallel machines with time-dependent learning effect and release dates', *International Journal of Operational Research*, Vol. 34, No. 3, pp.339–368.

Singh, M.R. and Mahapatra, S.S. (2016) 'A quantum behaved particle swarm optimization for flexible job shop scheduling', *Computers & Industrial Engineering*, Vol. 93, pp.36–44.

Soudi, A., Heydari, M. and Mazdeh, M.M. (2019) 'A new approach for integrated surgical procedure scheduling with arrival uncertainty', *International Journal of Operational Research*, Vol. 34, No. 3, pp.430–449.

Sundar, S., Suganthan, P.N., Jin, C.T., Xiang, C.T. and Soon, C.C. (2017) 'A hybrid artificial bee colony algorithm for the job-shop scheduling problem with no-wait constraint', *Soft Computing*, Vol. 21, No. 5, pp.1193–1202.

Suresh, R.K. and Mohanasundaram, K.M. (2006) 'Pareto archived simulated annealing for job shop scheduling with multiple objectives', *The International Journal of Advanced Manufacturing Technology*, Vol. 29, Nos. 1–2, pp.184–196.

Syswerda, G. (1991) 'Scheduling optimization using genetic algorithms', in Davis, L. (Eds.): *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, pp.332–349, Chapter 21.

Tabrizi, B.H., Ghaderi, S.F. and Haji-Yakhchali, S. (2019) 'Net present value maximisation of integrated project scheduling and material procurement planning', *International Journal of Operational Research*, Vol. 34, No. 2, pp.285–300.

Tamssaouet, K., Dauzère-Pérès, S. and Yugma, C. (2018) 'Metaheuristics for the job-shop scheduling problem with machine availability constraints', *Computers & Industrial Engineering*, Vol. 125, pp.1–8.

Tan, C.J., Neoh, S.C., Lim, C.P., Hanoun, S., Wong, W.P., Loo, C.K. and Nahavandi, S. (2019) 'Application of an evolutionary algorithm-based ensemble model to job-shop scheduling', *Journal of Intelligent Manufacturing*, Vol. 30, No. 2, pp.879–890.

Tan, Y., Hildebrandt, T. and Scholz-Reiter, B. (2016) 'Configuration and the advantages of the shifting bottleneck procedure for optimizing the job shop total weighted tardiness scheduling problem', *Journal of Scheduling*, Vol. 19, No. 4, pp.429–452.

Tayal, A. and Singh, S.P. (2019) 'Analysis of simulated annealing cooling schemas for design of optimal flexible layout under uncertain dynamic product demand', *International Journal of Operational Research*, Vol. 34, No. 1, pp.85–103.

Teodorović, D., Šelmić, M. and Davidović, T. (2015) 'Bee colony optimization part II: the application survey', *Yugoslav Journal of Operations Research*, Vol. 25, No. 2, pp.185–219.

Wei, Q., Guo, Z., Lau, H.C. and He, Z. (2019) 'An artificial bee colony-based hybrid approach for waste collection problem with midway disposal pattern', *Applied Soft Computing*, Vol. 76, pp.629–637.

Yue, Y., Cao, L. and Luo, Z. (2019) 'Hybrid artificial bee colony algorithm for improving the coverage and connectivity of wireless sensor networks', *Wireless Personal Communications*, Vol. 108, No. 3, pp.1719–1732.

Zandieh, M. and Mohaddesi, S.O. (2019) 'Portfolio rebalancing under uncertainty using meta-heuristic algorithm', *International Journal of Operational Research*, Vol. 36, No. 1, pp.12–39.

Zhang, C., Li, P., Guan, Z. and Rao, Y. (2007) 'A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem', *Computers & Operations Research*, Vol. 34, No. 11, pp.3229–3242.

Zhang, G., Gao, L. and Shi, Y. (2011) 'An effective genetic algorithm for the flexible job-shop scheduling problem', *Expert Systems with Applications*, Vol. 38, No. 4, pp.3563–3573.

Zhang, H. and Roy, U. (2018) 'A semantics-based dispatching rule selection approach for job shop scheduling', *Journal of Intelligent Manufacturing*, pp.1–21.

Zhang, J., Ding, G., Zou, Y., Qin, S. and Fu, J. (2019) 'Review of job shop scheduling research and its new perspectives under Industry 4.0', *Journal of Intelligent Manufacturing*, Vol. 30, No. 4, pp.1809–1830.

Zhang, R., Song, S. and Wu, C. (2013) 'A hybrid artificial bee colony algorithm for the job shop scheduling problem', *International Journal of Production Economics*, Vol. 141, No. 1, pp.167–178.

Zoulfaghari, H. and Nematian, J. (2019) 'Minimising total weighted tardiness with considering compulsory idle times on a single machine', *International Journal of Operational Research*, Vol. 35, No. 3, pp.340–354.