Applied Mathematical Modelling 38 (2014) 1111–1132

Contents lists available at ScienceDirect

Applied Mathematical Modelling

journal homepage: www.elsevier.com/locate/apm

A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities

Jun-Qing Li [a,b], Quan-Ke Pan [a,b,*], M. Fatih Tasgetiren [c]

[a] State Key Laboratory of Synthetic Automation for Process Industries, Northeastern University, Shenyang 110819, PR China
[b] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[c] Industrial Engineering Department, Yasar University, Izmir, Turkey

ARTICLE INFO

Article history:
Received 10 February 2012
Received in revised form 21 June 2013
Accepted 26 July 2013
Available online 19 August 2013

Keywords:
Flexible job-shop scheduling problem with maintenance activities
Multi-objective optimization
Artificial bee colony algorithm
Tabu search

ABSTRACT

This paper presents a novel discrete artificial bee colony (DABC) algorithm for solving the multi-objective flexible job shop scheduling problem with maintenance activities. Performance criteria considered are the maximum completion time so called makespan, the total workload of machines and the workload of the critical machine. Unlike the original ABC algorithm, the proposed DABC algorithm presents a unique solution representation where a food source is represented by two discrete vectors and tabu search (TS) is applied to each food source to generate neighboring food sources for the employed bees, onlooker bees, and scout bees. An efficient initialization scheme is introduced to construct the initial population with a certain level of quality and diversity. A self-adaptive strategy is adopted to enable the DABC algorithm with learning ability for producing neighboring solutions in different promising regions whereas an external Pareto archive set is designed to record the non-dominated solutions found so far. Furthermore, a novel decoding method is also presented to tackle maintenance activities in schedules generated. The proposed DABC algorithm is tested on a set of the well-known benchmark instances from the existing literature. Through a detailed analysis of experimental results, the highly effective and efficient performance of the proposed DABC algorithm is shown against the best performing algorithms from the literature.

1. Introduction

Scheduling is one of the most important key issues in planning and controlling the manufacturing systems [1,2]. The classical job-shop problem is one of the most difficult problems in this area. It is basically concerned with scheduling a set of jobs through a set of machines in order to minimize a creation performance criterion such that each job is consisted of a sequence of consecutive operations, each operation demands only one machine which is continuously available and can process one operation at a time without interruption.

On the other hand, the classical JSP can be generalized into the flexible job shop scheduling problem (FJSP) where two sub-problems are solved: the first one is called a routing sub-problem that assigns each operation to a machine selected from any of a set of suitable machines, the second one is called a scheduling sub-problem consisting of

* Corresponding author at: State Key Laboratory of Synthetic Automation for Process Industries, Northeastern University, Shenyang 110819, PR China. Tel.: +1 5063528919.
E-mail addresses: lijunqing.cn@gmail.com (J.-Q. Li), panquanke@gmail.com (Q.-K. Pan).

0307-904X/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.apm.2013.07.038

sequencing all the assigned operations on all machines in order to yield a feasible schedule to minimize a predefined performance criterion. In the classical JSP, job routings are known in advance and each operation is performed on a predefined machine. However, in the FJSP, each operation can be performed on any among a set of available machines. Therefore, the FJSP is more difficult than the classical JSP due to the consideration of both job routing and job scheduling. The general JSP is strongly NP-hard, while the FJSP is a much more complex version of the JSP, so the FJSP is strongly NP-hard [3]. Due to the complexity of the FJSP, no exact method has so far been developed to tackle the problem within a reasonable amount of time [2]. Thus, meta-heuristic algorithms have become a practical tool for solving this problem. For the FJSP with makespan criterion, tabu search (TS) has been proved to an efficient algorithm such as in Brandimarte [4], Mastrolilli et al. [5], Fattahi et al. [3], Fattahi et al. [6], Ennigrou and Ghedira [7], Li et al. [8], and Bozejko et al. [9], whereas the genetic algorithm (GA) has also been applied by many researchers, such as Kacem et al. [10], Ho et al [11], Pezzella et al. [12] and Gao et al. [13]. The particle swarm optimization (PSO) has also been investigated by Xia and Wu [14], Gao et al. [15], Liu et al. [16] and Zhang et al. [17]. In addition to the above, some other meta-heuristics have been introduced for the problem on hand such as the parallel variable neighborhood search (PVNS) algorithm [18], the knowledge-based ant colony optimization (KBACO) algorithm [19], the artificial immune algorithm (AIA) [2], and the climbing depth-bounded discrepancy search (CDDS) algorithm [20]. Recently, Wang et al. [21] proposed an effective artificial bee colony (ABC) algorithm for solving the single-objective FJSP, where different crossover and mutation approaches are conducted, and local search based on moving critical operations is also embedded.

Unlike the FJSP with a single objective, the FJSPs with multiple objectives have attracted attention by the researchers very recently. For example, Kacem et al. [22] proposed a hybrid approach combining the GA with a local search (AL + CGA). Xia and Wu [14] developed a hybrid method employing PSO with the simulated annealing (SA). Tay and Ho [23] developed a genetic programming with evolving dispatching rules. Gao et al. [13] provided a hybrid GA (hGA). Xing et al. [24,25] presented some local search algorithms, too. Zhang et al. [17] introduced a hybrid PSO with a TS algorithm. Li et al. [26] investigated the TS algorithm with some efficient neighborhood structures (HTSA).

Most of the literature for the multiple-objective FJSPs adopts the aggregation approach where a deterministic weight is assigned to each objective and these objectives are aggregated into a single objective. The drawback of the above approach is to generate only a single solution at each run. Thus, little information can be provided to the decision maker about the quality of each performance criterion. The Pareto-based approach has recently become a practical tool for solving the multi-objective optimization problems [27]. Kacem et al. [22] proposed a Pareto-based algorithm which combines evolutionary algorithms with fuzzy logic. Tay and Ho [23] developed an approach called MOEA-GLS by combining the evolutionary algorithm with a guided local search. Moslehi and Mahnam [28] conducted a Pareto approach using PSO with a local search. Very recently, Li et al. [29] presented a discrete ABC for solving the multi-objective FJSP, where a crossover operator is developed for information sharing among employed bees, the Pareto archive set is used to record the non-dominated solutions, a fast Pareto set update function is designed, and several local search methods are introduced.

Nowadays, production scheduling and maintenance planning have been received considerable attention because of their importance both in the fields of manufacturing and combinatorial research [30]. Schmidt [30] has outlined most of the literature related to deterministic scheduling problems with machine availability constraints until 1998 whereas Ma et al. [31] surveyed the scheduling problems with deterministic machine availability constraints very recently. It can be concluded from these survey papers that most of the literature considered machine availability constraints in solving single machine problems, parallel machine problems, flow shop scheduling problems, and job shop scheduling problems. However, it was pointed out that there are a few literatures considering the availability constraints in the FJSPs. Regarding the availability constraints, Gao et al. [32] proposed a hybridization of GA with a local search algorithm for solving the multi-objective FJSPs with preventive maintenance (PM) activities whereas Zribi et al. [33] considered the classical JSP with preventive maintenance (PM) activities. Chan et al. [34] also studied the distributed flexible manufacturing system (FMS) with availability constraints. Wang and Yu [35] proposed a filtered beam search (FBS) for solving the single-objective FJSP with at most one maintenance task for each machine.

By simulating the behavior of honey bee swarm intelligence, an artificial bee colony (ABC) algorithm is proposed by Karaboga [36–38] to optimize multi-variable and multi-modal continuous functions. Experimental comparisons demonstrated that performance of the ABC algorithm is competitive to other swarm intelligent algorithms. Due to its fewer control parameters and ease of implementation, researchers have adopted the ABC algorithm to solve many practical optimization problems [39]. In this study, we develop a novel discrete ABC (DABC) algorithm for solving the multi-objective FJSP with preventive maintenance activities. Both machine availability case and non-machine availability case are considered, respectively. The main features of the proposed DABC are as follows: (1) several problem-related neighborhood structures are designed; (2) a self-adaptive neighborhood structure strategy is conducted to balance the exploitation and exploration capability; (3) a TS-based local search heuristic is applied to enhance the exploitation performance; (4) a well-designed initialization method is embedded; (5) a decoding strategy considering the maintenance tasks is developed; (6) a Pareto archive set is constructed to record the non-dominated solutions. The rest of this paper is organized as follows: Section 2 briefly describes the problem formulation. Then, the artificial bee colony algorithm is presented in Section 3. The proposed DABC algorithm is given in detail in Section 4 whereas Section 5 gives the experimental results and compares to the best performing algorithms from the existing the literature to demonstrate the superiority of the DABC algorithm. Finally, Section 6 gives the concluding remarks and future research direction.

2. Problem formulation

In the FJSP, there are a set of machines $M = \{M_1, M_2, \ldots, M_k, \ldots, M_m\}$ and a set of jobs $J = \{J_1, J_2, \ldots, J_i, \ldots, J_n\}$. Each job consists of a sequence of operations O_{ij}, $j = 1, \ldots, n_i$ where O_{ij} and n_i denote the jth operation of job i and the number of operations required for job i, respectively. Each operation O_{ij} is to be processed on a machine denoted as M_k out of a set of available machines called $M_{ij} \subseteq M$.

Let PM_{kl} be the lth maintenance task on machine k.
Let L_k be the total number of preventive maintenance tasks on machine k.
Let d_{kl} be the duration of the maintenance task PM_{kl}.
Let $[w_{kl}^E, w_{kl}^L]$ be the time window associated with PM_{kl} where w_{kl}^E represents the earliest time of the window whereas w_{kl}^L is the latest time of the window.
Let z_{kl} be the completion time of the maintenance task PM_{kl}. Let p_{ijk} be the predefined fixed processing time of O_{ij} on machine M_k.
Let C_{ij} be the completion time of O_{ij}.

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

The mathematical model for the FJSP without maintenance activities can be referred to [8,40]. On the other hand, the mathematical model for the FJSP with maintenance activities is defined as follows:

$$\min f_1 = \max_{1 \leqslant i \leqslant n} \{c_{in_i}\}, \tag{1}$$

$$\min f_2 = \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n_i} x_{ijk} \cdot p_{ijk} + \sum_{k=1}^{m} \sum_{l=1}^{L_k} d_{kl}, \tag{2}$$

$$\min f_3 = \max_{1 \leqslant k \leqslant m} \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n_i} x_{ijk} \cdot p_{ijk} + \sum_{l=1}^{L_k} d_{kl} \right\}, \tag{3}$$

s.t.

$$[(z_{kl} - d_{kl} - c_{ij}) \cdot x_{ijk} \geqslant 0] \vee [(c_{ij} - z_{kl} - p_{ijk}) \cdot x_{ijk} \geqslant 0], \quad \forall (i,j)\ (k,l), \tag{4}$$

$$w_{kl}^E + d_{kl} \leqslant z_{kl} \leqslant w_{kl}^L, \quad \forall (k,l), \tag{5}$$

$$w_{kl}^E, w_{kl}^L \geqslant 0, \quad \forall (k,l). \tag{6}$$

Constraint (4) forces the non-overlapping constraints between PM tasks and operations whereas constraint (5) ensures that the PM tasks have to be completed within their time windows.

3. Artificial bee colony algorithm

3.1. The basic concept of ABC algorithm

In the basic ABC algorithm [36–39], there are two components: the foraging artificial bees and the food sources. The position of a food source represents a possible solution to the optimization problem and the nectar amount of a food source corresponds to the quality or fitness of the associated solution. The basic ABC classifies foraging artificial bees into three groups, namely, *employed bees*, *onlookers*, and *scouts*. An *employed bee* is responsible for flying to and making collections from the food source the bee swarm is exploiting. An *onlooker* waits in the hive and decides on whether a food source is acceptable or not. This is done by watching the dances performed by the employed bees. A *scout* randomly searches for new food sources by means of some internal motivation or possible external clue. In the ABC algorithm, each solution to the problem under consideration is called a *food source* and represented by an n-dimensional real-valued vector where the fitness of the solution corresponds to the *nectar amount* of the associated food resource. As with other intelligent swarm-based approaches, the ABC algorithm is an iterative process. The approach begins with a population of randomly generated solutions (or food sources); then, the following steps are repeated until a termination criterion is met [36–39]:

Initialize the foraging process.
Send the employed bees to exploit the discovered food sources.
Using the onlooker bees, choose the food sources and determine their nectar amounts.
Send scouts to search for new food sources.

Remember the best food source found so far.
If a termination criterion has not been satisfied, go to step 2; otherwise stop the procedure and report the best food source found so far.

3.2. Control parameters

There are three control parameters in the basic ABC algorithm, i.e., the number of food sources (*SN*), the number of cycles through which a food source cannot be improved further and then the food source is assumed to be abandoned (*limit*), and a termination criterion.

3.3. Initial population

In the basic ABC algorithm, the initial population is generated by a random approach. Let $v_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$ represents the ith food source in the population where n is the problem dimension. Each food source is randomly and uniformly generated as follows:

$$v_{ij} = v_{ij}^{\min} + \left(v_{ij}^{\max} - v_{ij}^{\min}\right) \times r, \quad j = 1, \ldots, n; \quad i = 1, \ldots, SN, \tag{7}$$

where $v_{ij}^{\max}$ and $v_{ij}^{\min}$ are the lower and upper bounds for the dimension j, respectively and r is a uniform random number in $[0, 1]$.

3.4. Employed bee phase

In the employed bee phase, the ith food source v_i is given to the ith artificial employed bee, who generates a new neighboring solution around the given food source as follows:

$$v_{new,j} = v_{ij} + U(-1, 1) \times (v_{ij} - v_{kj}), \tag{8}$$

where $i \in \{1, \ldots, SN\}$, and $k \in \{1, \ldots, SN\} \wedge k \neq i$ is randomly chosen food source. After obtaining the new solution v_{new}, it will be evaluated and compared to v_i, then the solution with the higher fitness value will be the winner.

3.5. Onlooker bee phase

The onlookers are the bees who wait in their hive for making decisions to select food sources after the employed bees carrying the food source back. The onlooker bees use the probability values to select the food source for discovering promising regions in the search space. The winning probability value for each food source is calculated as follows:

$$p_i = \frac{fit_i}{\sum_{i=1}^{SN} fit_i}, \tag{9}$$

where fit_i is the fitness value of the ith food source.

3.6. Scout bee phase

If a food source cannot be further improved through a limited cycles, then the food source is assumed to be abandoned and a randomly generated food source will be replaced with it.

However, the above ABC algorithm, originally designed for the continuous nature of optimization problems, cannot be used for discrete/combinatorial cases; therefore, in this work, some modifications to the above ABC algorithm have been made for the discrete version, as described below.

4. The proposed DABC algorithm

4.1. Solution representation

In the proposed algorithm, each food source/solution contains two vectors, i.e., the routing vector and the scheduling vector. Each dimension in the routing vector represents the selected machine for the corresponding operation whereas each dimension in the scheduling vector denotes the related job that their operations belong to. Each vector contains the total number of operations denoted as op_num. For example, consider an FJSP instance with three machines and three jobs where each machine has one maintenance task as shown in Table 1. In addition, Table 2 gives the operation processing time for the problem. The solution representation for this example is given in Fig. 1.

In Fig. 1, the routing vector indicates the assigned machine for each operation. For example, M_2 is selected for O_{21} whereas M_3 is selected for O_{32}. On the other hand, the scheduling vector indicates processing sequence for all operations. Reading jobs from left to right, the schedule of all operations can be found very easily, which is depicted in Fig. 2.

Table 1
Preventive maintenance tasks.

PM tasks		Time window		Duration (d_{kl})
		w^E_{kl}	w^L_{kl}	
M_1	PM_{11}	0	13	3
M_2	PM_{21}	2	14	1
M_3	PM_{31}	0	20	2

Table 2
Operation processing time.

		M_1	M_2	M_3
J_1	O_{11}	4	5	–
	O_{12}	3	2	1
	O_{13}	7	9	2
J_2	O_{21}	8	3	5
	O_{22}	7	6	4
	O_{23}	7	3	2
J_3	O_{31}	1	2	3
	O_{32}	3	2	1

Operation O_{ij}	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{31}	O_{32}
Machine k	1	2	3	2	3	2	1	3
Job i	2	1	1	3	2	2	1	3

Fig. 1. Solution representation.

Operation O_{ij}	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}	O_{31}	O_{32}
Machine k	1	2	3	2	3	2	1	3
Job i	2	1	1	3	2	2	1	3
Schedule job 2	O_{21}				O_{22}	O_{23}		
Schedule job 1	O_{21}	O_{11}	O_{12}		O_{22}	O_{23}	O_{13}	
Schedule job 3	O_{21}	O_{11}	O_{12}	O_{31}	O_{22}	O_{23}	O_{13}	O_{32}

Fig. 2. Sequence of operations.

4.2. Decoding with maintenance task

It is important to note that the solution representation given above contains no scheduling information for the maintenance tasks. In this study, maintenance tasks are scheduled dynamically by using the following heuristics:

Schedule the maintenance tasks on each machine at the begin of their time window, that is, set $z_{kl} = w^E_{kl} + d_{kl}$, for each $k \in (1, m)$.

When scheduling an operation O_{ij} on machine M_k, denote the possible start time and end time of O_{ij}, without considering the PM tasks, s_{ij} and c_{ij}, respectively.

If each maintenance task PM_{kl} does not overlap with the operation O_{ij}, then schedule O_{ij} at duration $[s_{ij}, c_{ij}]$. Otherwise, perform step 4.

If $[s_{ij}, c_{ij}]$ is overlapped with a maintenance task PM_{kl}, shift PM_{kl} to the right as possible as to schedule O_{ij} before PM_{kl}. If O_{ij} can be schedule before PM_{kl}, then schedule it. Otherwise, shift PM_{kl} to the left as possible as compact, then schedule O_{ij} after PM_{kl}.

Fig. 3 gives a Gantt chart for the example instance in Tables 1 and 2 with PM tasks.

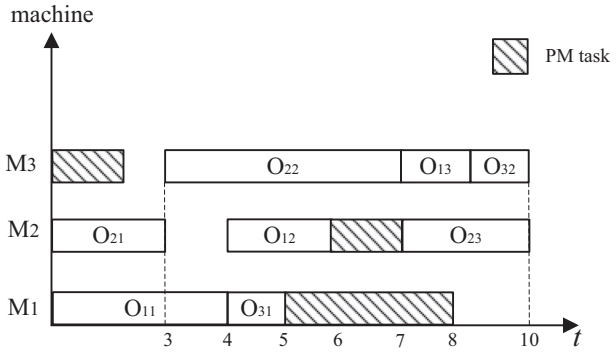

Fig. 3. Gantt chart for the example solution.

4.3. Neighborhood structures

There are two types of neighborhood structures in the DABC algorithm. In the first one, neighboring solutions are generated from the routing vector by considering some workload rules whereas in the second one, neighboring solutions are generated from scheduling vector by applying some traditional swap, insert and reverse moves.

4.4. Neighborhoods for routing vector

The neighborhood structures for the routing vector are given as follows:
Random method denoted as N_{r1}. This neighborhood can be referred to [26], which is generated as follows:

Randomly select an operation with more than one candidate machines;
From the candidate machine set, randomly select another machine different with the current one for the selected operation.

Top $k-q$ most workload neighborhood denoted as N_{r2}. This neighborhood is generated as follows:

- Select top-k machines with relatively heavy workload denoted as M_t^k;
- Randomly select q $(1 \leqslant q \leqslant k)$ machines form M_t^k denoted as M_k^q;
- For each machine M_q in M_k^q, randomly select an operation being operated on M_q denoted as O_{ij}, select another machine M_k from M_{ij} which is not in M_t^k, and then schedule O_{ij} on machine M_k.

Workload considered neighborhood denoted as N_{r3}.This neighborhood is generated by following steps:

- Randomly select an operation O_{ij} with more than two candidate machines and denote the machine which is processing O_{ij} as M_o;
- Select a machine M_k from M_{ij} different with M_o which satisfies one of the following condition:
- $p_{ijk} < p_{ijo}$, in this case, the total workload can be optimized;
- $(W_o = f_3) \wedge (W_k + p_{ijk} < f_3)$, where W_k means the workload of M_k, in this case, the maximal workload can be improved;
- Replace the current machine with M_k at the position of O_{ij}.

4.5. Neighborhoods for scheduling vector

Swap neighborhood.
Randomly select two operations which do not belong to the same job;
Swap the two selected operations in the scheduling vector.
Insert neighborhood.
Randomly select two operations which do not belong to the same job;
Erase the second operation and then insert it before the first one.
Reverse neighborhood.
Randomly select two different positions in the scheduling vector;
Reverse each operation between the two positions, that is, the last one in the current scheduling vector become the first one in the new vector while the first one in the pre-vector become the last one.

4.6. A self-adaptive strategy to produce neighboring solutions

A self-adaptive strategy is proposed in the proposed algorithm to utilize different neighborhood structures in different stages. The proposed self-adaptive strategy, which is similar to the one in [39], is presented as follows: At the beginning, two initial neighborhood structure vectors are defined as *NVR* for local search in routing vector and *NVS* for local search in scheduling vector with their length equal to the number of neighboring solutions needed to search (*numNS*). These two vectors are generated by filling them one by one randomly from the neighborhood structures explained before. Then the algorithm is started. During the search process, neighboring solutions are generated by using the routing neighboring approaches taken out from the *NVR* and the scheduling approaches selected from the *NVS* one by one, respectively. If the new neighboring solution is a non-dominated one, the corresponding approaches will enter into the winning neighborhood structure vectors called *WNR* for routing vector and *WNS* for scheduling vector, respectively. For the next iteration, the current *NVR* and *NVS* will be filled with the elements of the current *WNR* and *WNS*, respectively. Meanwhile, the two vectors *WNR* and *WNS* will be set empty for the following new generation. Once the length of the new *NVR* and *NVS* is less than *numNS*, the empty positions will be filled as follows: 75% is refilled from the *WNR* and *WNS*, respectively, and then the rest of 25% is refilled by a random selection from the neighborhood structures explained before. If the *WNR* or *NVS* is empty (this may happen when the search is stuck at local optima), the new *NVR* or *NVS* will be filled as follows: 50% is refilled from the latest *NVR* or *NVS*, and the rest 50% is randomly selected from the neighborhood structures explained before. The above process is repeated until a termination criterion is reached. We refer to [39,40] for details of the procedure.

4.7. TS-based local search heuristic

The tabu search algorithm proposed by Glover [41] has been successfully applied to a large number of combinatorial optimization problems [41–44]. In the proposed algorithm, TS was used to conduct a local search applied to the food sources generated by neighboring structure. The main steps of the TS based local search denoted as *TS_LocalSearch*(s) is given as follows:

Set the system parameters and set s as the current solution.
Set $j = 0$, perform steps 3 to 9 until $j \geqslant T_U$ where T_U is the maximum number of iterations for which the external *PAS* has not been improved.
For $i = 1$ *numNS*, perform sub-steps from 3.1 to 3.2.
Select the ith neighborhood structure from the NVR vector explained before.
Produce a neighboring solution by using the selected neighborhood structure and insert it in the queue denoted as *qNS*.
For $i = 1$ to *numNS*, perform sub-steps from 4.1 to 4.2.
Select the ith neighborhood structure from the *NVS* vector explained before.
Produce a neighboring solution by using the selected neighborhood structure and insert it in the queue denoted as *qNS*.
Apply the Pareto non-dominate sorting function barrowed from [45] to *qNS* and then select the solutions in the first Pareto level front (denoted as Γ_1) to update the Pareto archive set *PAS*. If the *PAS* is improved, then set $j = 0$, otherwise, increase j by $j + 1$.
Update the corresponding winning neighborhood structure vector *WNR* or *WNS* with the winning neighborhood structure if the corresponding neighboring solution is a non-dominated one.
Record each solution that has updated the *PAS* in a new vector called *BS*.
Select the best neighboring solution as the current solution satisfying one of the following conditions:
the solution in *BS* which is non-tabu;
if it does not exist, then select the solution in Γ_1 which is non-tabu;
if both 8.1 and 8.2 do not exist, then select the first solution satisfying the aspiration rule in *BS* or Γ_1. If there exist more than one solution satisfying the above conditions, randomly select one of them as the current solution.
Update the tabu list by adding the selected neighboring solution and remove the oldest solution if the tabu list is overflowed or the duration of the oldest solution in the tabu list exceeds the termination criterion T_U.

4.8. Initial population

In order to construct the initial population, four priority/heuristic rules are employed for the routing vector. These are operation minimum processing time rule (OPT) in [12], the longest processing time rule (LPT) in [8], the random rule, and the workload considered rule (WCR). The adoption of a mixture of these four rules to produce the routing vector may enrich the initial population to solve FJSP problems. In this study, 20% of the initial population was generated by LPT, 20% by WCR, 20% by OPT, and 40% by the random rule.

Regarding the scheduling vector, four priority rules are employed in the initial population. These are the random rule, the most work remaining rule (MWR) in [4], the most number of operations remaining (MOR) rule in [12], and the shortest processing time rule (SPT) in [4]. In this study, the initial population is constructed with a mixture of these four rules, i.e., 20% by MWR, 20% by MOR, 20% by SPT, and 40% by the random rule.

Main steps of the WCR approach are given below whereas Fig. 4 shows an example of the WCR approach for the example instance discussed in Section 4.1.

Create a vector to record the machine workload for each machine (hereafter called *MW*) with the length equals to *m*. Set each element in *MW* to zero.

Randomly sequence each job and select them one by one to perform step 3.

For each selected job *i* in the queue, find the unscheduled operation with minimum processing time O_{ij}^{*}, then perform substeps from 3.1 to 3.5.

Select each candidate machine $M_k (1 \leqslant k \leqslant m,\ M_k \in M_{ij})$ for operation O_{ij}.

Find the most suitable machine M_{k^*} which satisfies $k^* = \min\{p_{ijk} + MW_k | k \in M_{ij}\}$ where MW_k denotes the value of *k*th element in *MW*.

If more than one machine satisfies the above condition, select the one with minimum processing time for O_{ij}^{*}.

Add the processing time p_{ijk^*} to the position k^* in *MW*.

Assign M_{k^*} for processing O_{ij}^{*}.

4.9. Framework of the DABC algorithm

For multi-objective FJSPs in general, the makespan is typically the most important criterion and heavily affects the other two objectives [46]. The size of the external Pareto archive set is in general relatively large for large scale instances which has a significant impact on the convergence speed of the algorithm. In order to find the near-optimal solutions with relatively small makespan values as early as possible, we propose a two-stage algorithm in this paper. In the first stage, TS-based local search was utilized to carry out the traditional approach for solving the multi-objective FJSP. In other words, three performance criteria were aggregated into a single objective by assigning a deterministic weight value to each objective [26], i.e., $\min f = w_1 f_1 + w_2 f_2 + w_3 f_3$. On the other hand, in the second stage, the Pareto-based DABC algorithm was integrated into the proposed algorithm to solve the multi-objective FJSP.

The main steps of the proposed DABC algorithm are as follows:

Initialization phase.

Set the system parameters.

Generate the initial population by using the initial heuristics.

Apply the Pareto non-dominated sorting function, barrowed from [45], on the population. Then update the external Pareto archive set *PAS* by using the solutions in the first Pareto level front.

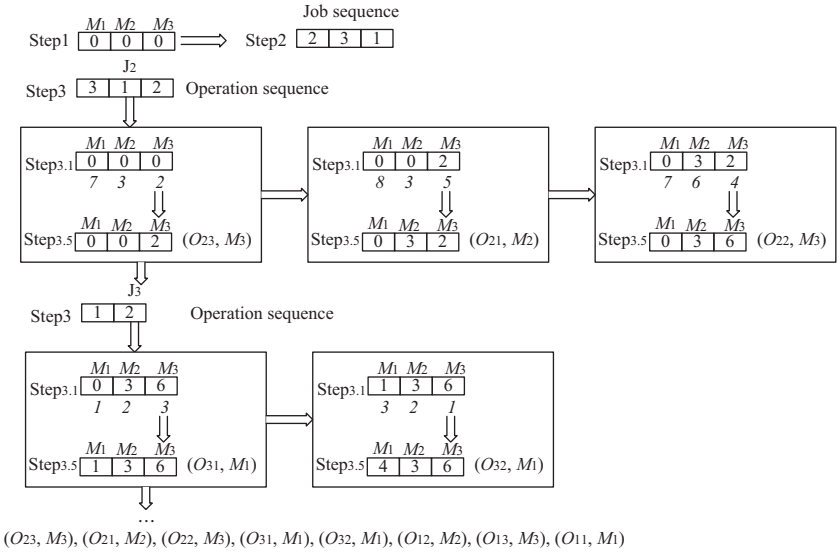

Fig. 4. An example process for the workload considered rule.

If the termination criterion is satisfied, report the non-dominated solutions in the PAS; otherwise, perform the steps from 4 to 5.

First stage-tabu search based-local search

Perform the sub-steps from 4.1 to 4.4 until the termination criterion is satisfied.
Set parameters for the first stage TS- based local search.
Randomly select a non-dominated solution from the external *PAS* as the current solution *s*.
For the current solution, apply the TS-based local search, i.e., *TS_LocalSearch(s)*.
If one of the following conditions satisfies, then perform step 5, otherwise, go back to step 4.2: the *PAS* has not been improved through P_U iterations;
the total number of iterations exceeds the maximum number of iterations T_{MAX}.

Second stage-DABC-based search

Perform the steps from 6 to 9 until the termination criterion is satisfied.
Employed bee phase.
If the size of the *PAS* exceeds the number of employed bees, for each employed bee, randomly select a recently improved solution, which must be different than the employed bees, from the external *PAS*; otherwise, put each employed bee on each non-dominated solution in the *PAS*.
For each employed bee, perform the TS-based local for the chosen solution s, i.e., *TS_LocalSearch(s)*.
If a non-dominated solution has not been improved through *limit* cycles, the corresponding employed bee becomes a scout bee.
Onlooker bee phase.
Each onlooker bee randomly selects three non-dominated solutions from the *PAS* and selects the solution.
Which is improved most recently as the food source.
Each onlooker bee, perform the TS-based local for the chosen solution *s*, i.e., *TS_LocalSearch(s)*.
Scout bee phase.
From the *PAS*, randomly select a member solution as the food source.
Each scout bee, perform the TS-based local for the chosen solution *s*, i.e., *TS_LocalSearch(s)*.
If one of the following conditions satisfies, report the non-dominated solutions in the external PAS: the *PAS* has not been improved through P_U iterations;
the total number of iteration exceeds the maximum number P_{MAX}. otherwise, go back to step 6.

5. Computational results

This section discusses the computational results to evaluate the performance of the proposed DABC algorithm with the best performing algorithms from the existing literature. The DABC algorithm was written and implemented in C++ on an AMD Athlon 64 X2 Notebook computer running at Dual-Core 1.7 GHz with 1 G memory.

5.1. Parameter setting

Each instance can be characterized by the following parameters: number of jobs (n), number of machines (m), and the number of operations (op_num). The other parameters are given as follows:

Number of food sources: $SN = 10$.
Number of employed bees: $N_{EB} = 10$.
Number of onlookers: $N_{OB} = 10$.
Number of scout bees: $N_{SB} = 2$.
Number of cycles through which a food source cannot be further improved: lim$it = 10$.
Number of neighboring solutions needed to search: $NumNS = 3n$.
For top $k - q$ most workload neighborhood structure, $k = 3$ and $q = 2$.
For the first stage TS heuristic, the weight values for the aggregated objective functions are set to $w_1 = 3/7$, $w_2 = 1/7$ and $w_1 = 3/7$ for f_1, f_2 and f_3, respectively.
The maximum duration of the TS-based local search for which the external *PAS* has not been improved: $T_U = 2n$.
The maximum number of consecutive iterations without improvement of the *PAS*: $P_U = 30$.
The maximum number of iterations $T_{MAX} = 150$ and $P_{MAX} = 300$.
Tabu tenure: $T_{TENURE} = op\_num/2$.
Tabu list length: $T_{LENGTH} = op\_num/2$.

Table 3
PM tasks of $8 \times 8 - m$.

		PM_{11}	PM_{21}	PM_{31}	PM_{41}	PM_{51}	PM_{61}	PM_{71}	PM_{81}
Time window	w_{kl}^E	1	3	5	6	0	3	1	3
	w_{kl}^L	10	9	15	17	10	16	14	13
Duration		4	3	5	3	3	5	3	4

Table 4
PM tasks of $10 \times 10 - m$.

		PM_{11}	PM_{21}	PM_{31}	PM_{41}	PM_{51}	PM_{61}	PM_{71}	PM_{81}	PM_{91}	$PM_{10,1}$
Time window	w_{kl}^E	0	1	0	0	2	0	0	0	0	0
	w_{kl}^L	4	7	6	5	7	6	5	7	5	6
Duration		2	1	1	2	1	2	2	3	2	3

Table 5
PM tasks of $15 \times 10 - m$.

		PM_{11}	PM_{21}	PM_{31}	PM_{32}	PM_{41}	PM_{51}	PM_{61}	PM_{71}	PM_{81}	PM_{82}	PM_{91}	$PM_{10,1}$
Time window	w_{kl}^E	1	2	0	3	0	0	0	2	0	6	1	2
	w_{kl}^L	5	7	3	11	10	8	6	7	5	11	5	8
Duration		1	1	1	2	3	2	1	1	1	1	1	1

5.2. Comparisons of the three FJSPs with PM tasks

In this section, we employed three representative FJSP instances with PM tasks, denoted as $n \times m - m$, ranging from 27 operations to 56 operations in [32,35]. The small scale instance $8 \times 8 - m$ and the medium scale instance $10 \times 10 - m$ have exactly one PM activity on each machine in the planning horizon whereas the large scale instance $15 \times 10 - m$ has two PM tasks for two machines and one PM task for the others. The non-fixed availability constraint is set as the same as in [32,35] and the time window and duration of maintenance tasks are shown in Tables 3–5, respectively.

Two schedules for the $8 \times 8 - m$ instance obtained by the DABC algorithm are shown in Figs. 5 and 6, respectively. In addition, Figs. 7 and 8 show another two schedules for the $10 \times 10 - m$ instance, respectively whereas the schedule for the $15 \times 10 - m$ instance is given in Fig. 9. In these five figures, the pair of numbers (in the form of [job, operation]) inside the blocks is the operation to be processed on the corresponding machine. The two numbers just below the block represent the start time and end time of the operation, respectively. The block marked with "PM" denotes the maintenance task for the corresponding machine.

Table 6 shows the comparisons on these instances with the other two algorithms, i.e., the hGA in [32] and the FBS-based algorithm in [35]. The first column in Table 6 gives the compared algorithms. Following observations can be derived from Table 6:

For solving the $8 \times 8 - m$ and $10 \times 10 - m$ instances, the DABC algorithm can obtain two non-dominated solutions whereas the other algorithms obtained only a single solution;

The DABC algorithm is also competitive to the hGA for solving the largest instance $15 \times 10 - m$ even with having more than one maintenance tasks.

5.3. Comparisons of the five Kacem instances

In this section, we compare to the best performing algorithms from the existing literature on the five Kacem instances [10,22]. These algorithms compared are summarized below:

The PSO + SA algorithm by Xia et al. [14],
The PSO + TS algorithm by Zhang et al. [17],
The X-LS by Xing et al. [24],
The HTSA algorithm by Li et al. [26],
The P-ABC algorithm by Li et al. [29],
And the MOPSO + LS algorithm by Moslehi et al. [28].

The computational results are given in Table 7. It can be seen from Table 7 that DABC algorithm is more efficient than the other approaches for solving the five Kacem instances. For example, for solving the 8×8, 10×10, and 15×10 instances, the

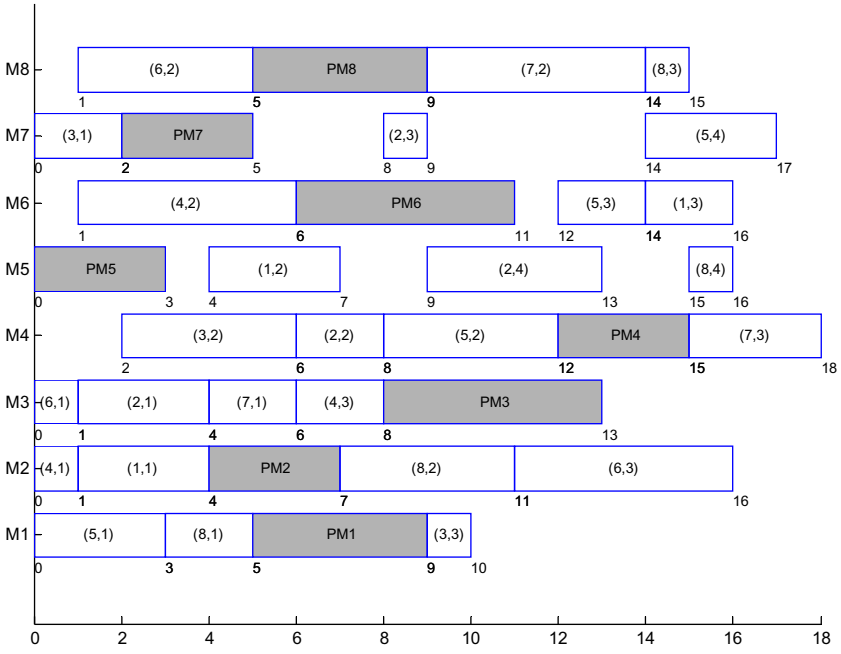

Fig. 5. Optimal solution of the 8 × 8 instance with maintenance constraint ($f_1 = 18, f_2 = 103, f_3 = 16$).

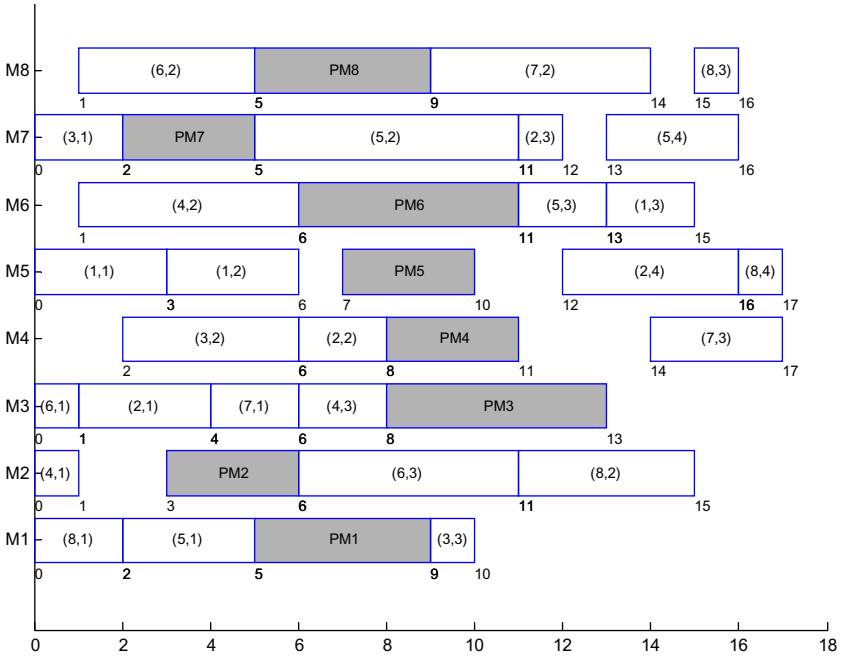

Fig. 6. Optimal solution of the 8 × 8 instance with maintenance constraint ($f_1 = 17, f_2 = 105, f_3 = 15$).

DABC algorithm either obtains superior solutions or gets richer (non-dominated and more than one) optimal solutions than the PSO + SA algorithm. In comparison with the very recently published PSO + TS algorithm, the DABC algorithm again obtains much richer (non-dominated and more than one) optimal solutions than the PSO + TS algorithm, especially for the large scale instances such as instances 10 × 10 and 15 × 10.

In addition to the above, we carried out a detailed comparison between DABC and X-LS on the five instances. Table 7 shows that our approach obtains dominated solutions in solving the 8 × 8 and 10 × 10 instances. For other three instances, i.e., 4 × 5, 10 × 7, and 15 × 10, the DABC algorithm also obtains richer optimal solutions or the same solutions. In comparison

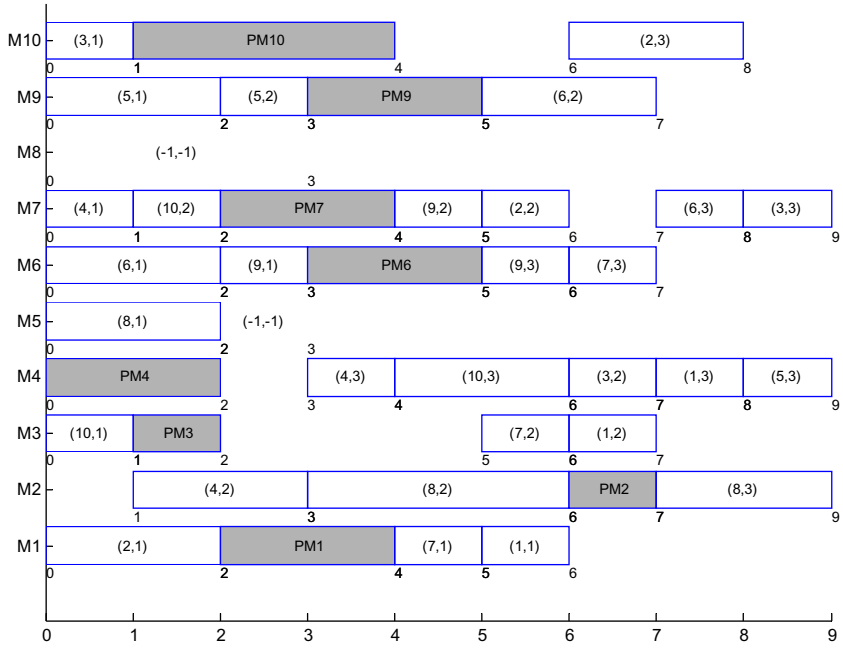

Fig. 7. Optimal solution of the 10×10 instance with maintenance constraint ($f_1 = 9, f_2 = 60, f_3 = 8$).

with the HTSA, the proposed DABC algorithm can obtain richer optimal solutions in solving four small and medium scale instances. The P-ABC and MOPSO + LS algorithms are the two methods in the six published compared algorithms which embed a Pareto archive set and thus can provide multiple solutions at one run. It can be seen from Table 7 that the MOPSO + LS algorithm can obtain all non-dominated solutions only for the 10×10 instance whereas the DABC algorithm can obtain all optimal solutions for the five instances. The P-ABC algorithm can obtain three non-dominated solutions for instances 4×5, 8×8, and 10×10, while only one non-dominated solution for instance 10×7. For solving the large scale instance 15×10, P-ABC can only obtain two relative worse solutions than the proposed algorithm. Based on the above comparisons, it can be concluded that DABC is superior to the above six algorithms in terms of solution quality and efficiency.

5.4. Comparisons of the BRdata problems

In this study, we select seven representative instances from the BRdata set [4], namely, MK01, MK02, MK03, MK04, MK05, MK07 and MK08 with the length of the reference Pareto solution set S^P for each instance less than 40 to verify the efficiency of the proposed DABC algorithm. The scale of these instances ranges from 55 operations to 225 operations.

There is much less literature considered the BRdata problems with multi-objectives. To make a detailed comparison, we select four efficient algorithms from the current literature. These are the X-SM algorithm by Xing et al. [25], the AIA algorithm by Bagheri et al. [2], the hGA by Gao et al. [46], and the HTSA by Li et al. [26].

The comparisons of the computational results are shown in Tables 8–10. In these tables, the solution number marked with '*' means that the corresponding solution is a non-dominated solution. Table 8 gives the computational results for the MK01 and MK02 instances. From Table 8, we can see that the DABC obtain more optimal solutions at one run instead of generating only a single solution as other algorithms did. For example, the proposed DABC algorithm obtains ten non-dominated solutions for MK01 and eight for MK02. Furthermore, all the results obtained by the proposed DABC algorithm dominate those results generated by the X-SM and AIA. The comparisons for MK03 and MK07 are shown in Table 9. It can be seen from Table 9 that only the hGA and the DABC algorithms can obtain optimal solutions for these two instances. In addition, our proposed algorithm obtains 17 non-dominated solutions for MK03 and 19 optimal solutions for MK07. Table 10 reports the comparison of the computational results for MK05 and MK08. From Table 10, we can see that for solving MK05, even though all three algorithms show promising performance, the proposed DABC algorithm can obtain eleven optimal solutions; on the other hand, for solving MK08, the four algorithms except the AIA can obtain non-dominated solutions. However, the proposed DABC algorithm is able to obtain more optimal solutions. The computational results for the MK04 instance are given in Table 11. It is obvious that only the proposed DABC algorithm obtains optimal solutions for the instance, hence it is clear winner by providing 27 optimal solutions.

It should be noted that all non-dominated solutions presented in Tables 8–11 are obtained by performing the DABC algorithm for 30 runs. To the best of our knowledge, all these solutions are the best ones for the considered multi-objective problems in the present literature. Thus, we can construct each reference Pareto solution set S^P by using the non-dominated

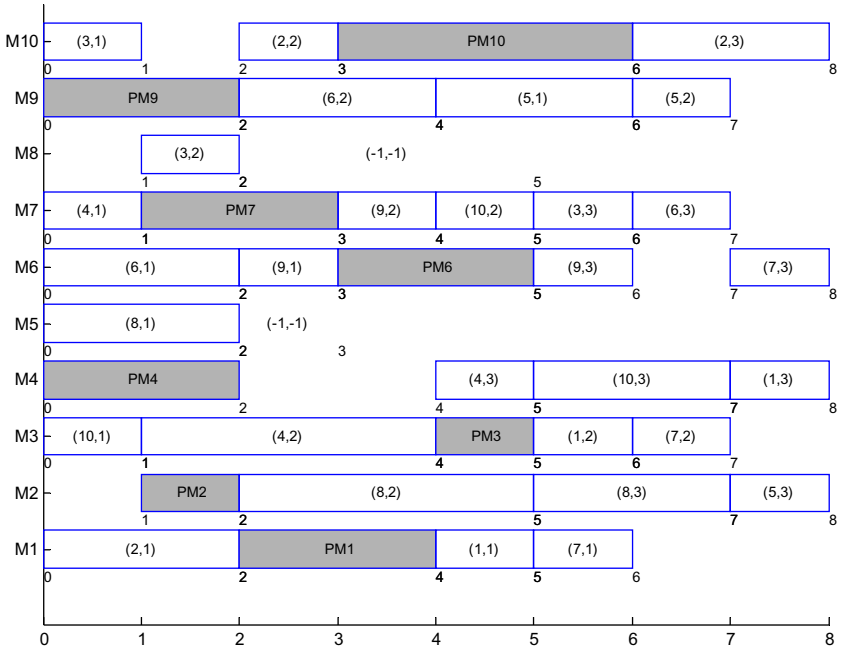

Fig. 8. Optimal solution of the 10×10 instance with maintenance constraint ($f_1 = 8, f_2 = 61, f_3 = 7$).

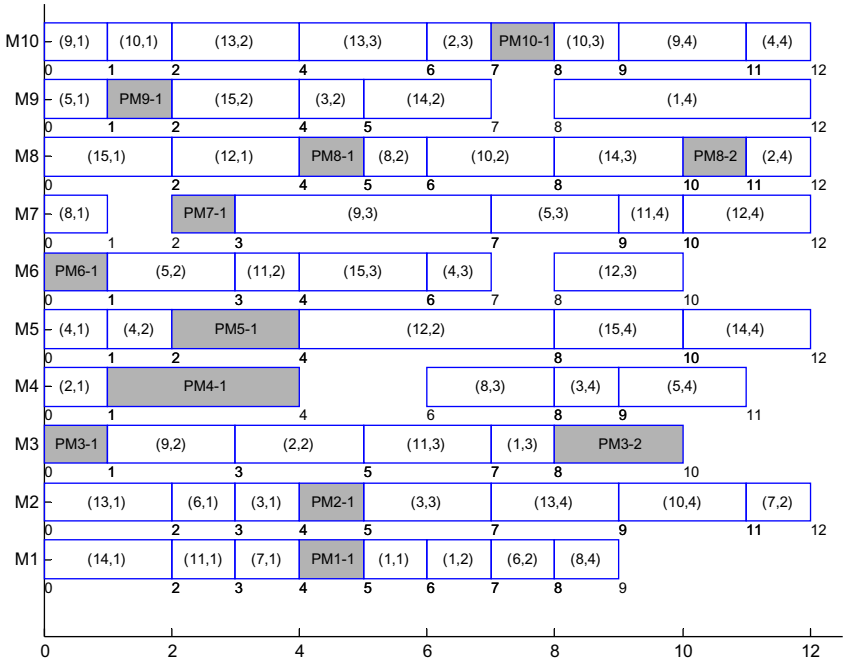

Fig. 9. Optimal solution of the 15×10 instance with maintenance constraint ($f_1 = 12, f_2 = 107, f_3 = 12$).

solutions given in the five tables for each corresponding instance. It is important to note that all these Pareto front sets will be used to make performance analysis in the Section 5.5.

5.5. Performance analysis

According to Ref. [27], three main metrics are commonly considered for evaluating the quality and diversity of the obtained non-dominated solutions in the Pareto archive set. That is, the number of the obtained non-dominated solutions

Table 6
Comparison of the three FJSPs with PM tasks.

Algorithm	$8 \times 8 - m$			$10 \times 10 - m$			$15 \times 10 - m$		
	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
hGA	17	105	15	8	61	7	12	107	12
FBS-based algorithm	18	103	16	9	60	8		*n/a*	
DABC	17	105	15	8	61	7	12	107	12
	18	103	16	9	60	8			

-n/a means not given by the author

Table 7
Comparison of results on the five Kacem instances.

Algorithm	4×5			8×8			10×7			10×10			15×10		
	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
PSO + SA		*n/a*		15	75	12		*n/a*		7	44	6	12	91	11
				16	73	13									
PSO + TS	11	32	10	14	77	12		*n/a*		7	43	6	11	93	11
				15	75	12									
X-LS	12	32	8	14	77	12	11	61	11	7	42	6	11	91	11
				15	76	12	11	62	10	8	42	5	11	93	10
HTSA	11	32	10	14	77	12	11	61	11	7	42	6	11	91	11
	12	32	8	15	75	12	11	62	10	8	42	5	11	93	10
										7	43	5			
P-DABC	11	32	10	14	77	12	12	61	11	7	43	5	12	91	11
	12	32	8	15	75	12	11	63	11	8	41	7	11	93	11
	13	33	7	16	73	13	12	60	12	8	42	5			
MOPSO + LS				14	77	12				7	42	6	11	91	11
				15	75	12				8	42	5	12	93	10
		n/a		16	73	13		*n/a*		7	43	5			
				16	78	11				8	41	7			
				17	77	11									
DABC	11	32	10	14	77	12	11	61	11	7	42	6	11	91	11
	12	32	8	15	75	12	11	62	10	8	42	5	11	93	10
	11	34	9	16	73	13	12	60	12	7	43	5			
	13	33	7	16	77	11				8	41	7			

n/a means not given by the author.

Table 8
Comparison of results on MK01 and MK02.

Algorithm	Solutions	MK01			Solutions	MK02		
		Makespan	Total workload	Max workload		Makespan	Total workload	Max workload
X-SM	1	42	162	42	1	28	155	28
AIA	1	40	171	36	1	26	154	26
hGA	1*	40	167	36	1*	26	151	26
HTSA	1*	40	167	36	1*	26	151	26
DABC	1*	40	167	36	1*	26	151	26
	2*	40	162	38	2*	27	145	27
	3*	40	164	37	3*	28	144	28
	4*	41	160	38	4*	29	143	29
	5*	41	163	37	5*	29	150	26
	6*	42	165	36	6*	30	142	30
	7*	42	158	39	7*	31	141	31
	8*	42	156	40	8*	33	140	33
	9*	43	154	40				
	10*	45	153	42				

'*' Means the corresponding solution is a non-dominated solution.

($N_{\prec}$), average distance of the obtained non-dominated front to Pareto front ($D_{\prec}$), and the ratio of the obtained non-dominated solutions ($R_{\prec}$). Suppose that S^P represents the reference Pareto solution set, and S^K the solution set obtained by the proposed algorithm, then the performance metrics are listed as follows:

Average distance of the obtained non-dominated front to Pareto front ($D_{\prec}$): Let $d_p(S^K)$ denotes the shortest normalized distance from a reference solution $p \in S^P$ to the solution set S^K, which is given by the following formula:

Table 9
Comparison of results on MK03 and MK07.

Algorithm	Solutions	MK07			Solutions	MK03		
		Makespan	Total workload	Max workload		Makespan	Total workload	Max workload
X-SM	1	150	717	150	1	204	852	204
AIA	1	140	695	140	1	204	1207	204
hGA	1*	139	693	139	1*	204	850	204
HTSA	1	140	695	140	1	204	852	204
DABC	1*	139	693	139	1*	204	850	204
	2*	140	685	140	2*	210	848	210
	3*	143	683	143	3*	213	844	213
	4*	144	674	144	4*	221	842	221
	5*	146	673	144	5*	222	838	222
	6*	150	669	150	6*	231	834	231
	7*	151	667	151	7*	240	832	240
	8*	156	664	156	8*	249	830	249
	9*	157	662	157	9*	258	828	258
	10*	161	660	161	10*	267	826	267
	11*	162	659	162	11*	276	824	276
	12*	166	657	166	12*	285	822	285
	13*	178	655	178	13*	294	820	294
	14*	179	655	175	14*	303	818	303
	15*	190	653	190	15*	312	816	312
	16*	202	651	202	16*	321	814	321
	17*	204	654	187	17*	330	812	330
	18*	208	653	187				
	19*	217	649	217				

$$d_p(S^K) = \min_{x \in S^K} \left\{ \sqrt{\sum_{i=1}^{w} \left(\frac{f_i(x) - f_i(p)}{f_i^{\max}(\cdot) - f_i^{\min}(\cdot)} \right)^2} \right\}, \tag{10}$$

where w is the objective number of the problem, $f_i^{\max}(\cdot)$ and $f_i^{\min}(\cdot)$ are the maximum and minimum value of the ith objective in the reference Pareto set S^P, respectively and $f_i(\cdot)$ represents the ith objective value. In this study, if $f_i^{\max}(\cdot) - f_i^{\min}(\cdot) = 0$, then we set $f_i^{\max}(\cdot) - f_i^{\min}(\cdot) = 0.5$.

The average distance is the average of those shortest normalized distances from all the reference solutions to the obtained solutions set S^K, which is calculated as follows:

$$D_{\prec}(S^K) = \frac{1}{|S^P|} \sum_{p \in S^P} d_p\left(S^K\right). \tag{11}$$

It is obvious that a smaller $D_{\prec}(\cdot)$ value denotes a better distribution of the obtained solution set.

Number of non-dominated solution $N_{\prec}$: The total number of non-dominated solutions in the obtained solution set represents the efficiency of the proposed algorithm. It is clear that the larger the metric, the better the obtained solution set. The metric $N_{\prec}$ is calculated as follows:

$$N_{\prec}(S^K) = \left| S^K - \left\{ x \in S^K \middle| \exists y \in S^P : y \prec x \right\} \right|. \tag{12}$$

Ratio of non-dominated solution $R_{\prec}$: The ratio of non-dominated solutions is used to evaluate the quality of the solutions in the obtained solutions set. It is obvious that the higher the $R_{\prec}$ is, the better the set S^K is

$$R_{\prec}(S^K) = \frac{N_{\prec}(S^K)}{\left|S^K\right|}. \tag{13}$$

Table 12 shows the average performance of our proposed algorithm for solving all instances explained before. For each instance, ten independent runs were preformed. The first column in Table 12 gives the considered problems. The next four columns report the maximum, minimum, average and standard deviation (SD) of the number of non-dominated solution $N_{\prec}$ for the corresponding problem, respectively. The maximum, minimum, average, and standard deviation of the ratio of non-dominated solution $R_{\prec}$ are given from 6th column to 9th column in the table, respectively. The following four columns list the result for the average distance of the obtained non-dominated front to Pareto front $D_{\prec}$. The standard deviation (SD) is computed as follows:

$$SD = \sqrt{\frac{\sum_{i=1}^{w} (x_i - \overline{x})^2}{w}}, \tag{14}$$

Table 10
Comparison of results on MK05 and MK08.

Algorithm	Solutions	MK05			Solutions	MK08		
		Makespan	Total workload	Max workload		Makespan	Total workload	Max workload
X-SM	1	177	702	177	1*	523	2524	523
AIA	1	173	686	173	1	523	2723	523
hGA	1*	172	687	172	1*	523	2524	523
HTSA	1*	172	687	172	1*	523	2524	523
DABC	1*	172	687	172	1*	523	2524	523
	2*	173	683	173	2*	524	2519	524
	3*	175	682	175	3*	533	2514	533
	4*	178	680	178	4*	542	2509	542
	5*	179	679	179	5*	551	2504	551
	6*	183	677	183	6*	560	2499	560
	7*	185	676	185	7*	569	2494	569
	8*	191	675	191	8*	578	2489	578
	9*	197	674	197	9*	587	2484	587
	10*	203	673	203				
	11*	209	672	209				

Table 11
Comparison of results on MK04.

Algorithm	Solutions	Makespan	Total workload	Max workload
X-SM	1	68	352	67
AIA	1	60	403	60
hGA	1	60	375	60
HTSA	1	61	366	61
DABC	1*	60	374	60
	2*	61	368	60
	3*	61	365	61
	4*	62	360	61
	5*	62	363	60
	6*	62	357	62
	7*	63	354	62
	8*	63	357	61
	9*	63	360	60
	10*	64	353	62
	11*	65	349	63
	12*	66	348	63
	13*	66	345	66
	14*	68	344	66
	15*	68	347	65
	16*	70	343	67
	17*	72	340	72
	18*	78	337	78
	19*	84	334	84
	20*	90	331	90
	21*	98	330	98
	22*	106	329	106
	23*	114	328	114
	24*	122	327	122
	25*	130	326	130
	26*	138	325	138
	27*	146	324	146

where $\bar{x}$ is the average value of the w runs. The last two columns give the average computational times for obtaining one solution $A_1(s)$ and the average computational times for all solutions $A_2(s)$ after ten independently runs. The $A_1(s)$ and $A_2(s)$ are computed as follows:

$$A_2(s) = \frac{\sum_{i=1}^{w} t_i}{w}, \tag{15}$$

$$A_1(s) = \frac{A_2(s)}{\sum_{i=1}^{w} |S_K^i| / w}, \tag{16}$$

Table 12
Performance of the DABC for solving all considered instances.

Problems	$N_{\prec}$				$R_{\prec}$				$D_{\prec}$				$A_1(s)^*$	$A_2(s)^*$
	MAX	MIN	AVG	SD	MAX	MIN	AVG	SD	MAX	MIN	AVG	SD		
$8 \times 8 - m$	2.00	2.00	2.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.36	0.71
$10 \times 10 - m$	2.00	1.00	1.60	0.49	1.00	0.50	0.94	0.16	1.22	0.00	0.37	0.56	6.39	10.88
$15 \times 10 - m$	1.00	0.00	0.83	0.37	1.00	0.00	0.83	0.37	2.00	0.00	0.36	0.77	26.02	26.02
4×5	4.00	4.00	4.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
8×8	4.00	3.00	3.90	0.30	1.00	1.00	1.00	0.00	0.13	0.00	0.01	0.04	6.71	26.18
10×7	3.00	3.00	3.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	3.74	11.21
10×10	4.00	4.00	4.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	6.86	27.43
15×10	2.00	0.00	0.88	0.93	1.00	0.00	0.63	0.34	2.22	0.00	1.31	1.02	45.57	63.65
MK01	9.00	2.00	4.90	2.34	0.90	0.14	0.44	0.26	0.16	0.02	0.08	0.04	17.21	191.64
MK02	6.00	3.00	4.33	0.94	0.67	0.22	0.47	0.14	0.17	0.05	0.11	0.03	27.63	254.56
MK03	17.00	14.00	16.33	0.94	1.00	0.74	0.94	0.08	0.00	0.00	0.00	0.00	17.83	309.67
MK04	21.00	13.00	17.40	2.10	1.00	0.50	0.81	0.16	0.12	0.04	0.05	0.02	37.32	801.74
MK05	11.00	7.00	8.67	1.05	1.00	0.47	0.63	0.14	0.03	0.00	0.01	0.01	23.48	323.14
MK07	14.00	8.00	11.00	1.61	0.93	0.26	0.63	0.20	0.05	0.03	0.04	0.00	56.54	987.20
M/K08	9.00	9.00	9.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	2.86	25.73

*Time unit: second.

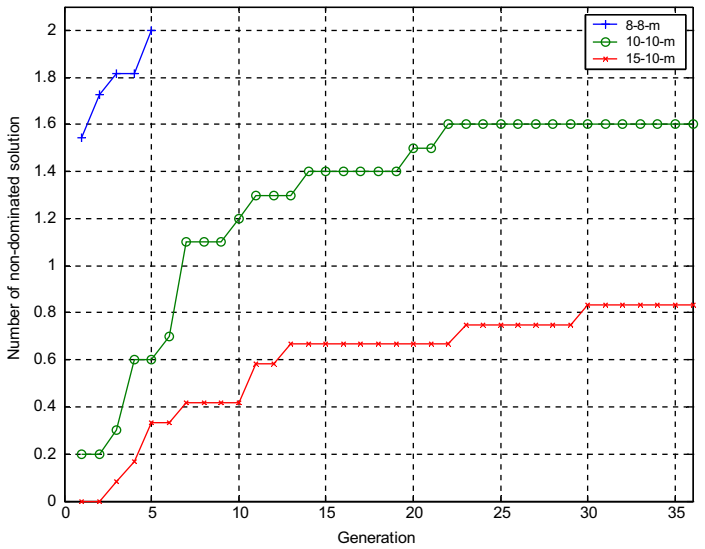

Fig. 10. Evolutionary curves of the $N_{\prec}$ for the three Kacem instances with PM tasks.

where $|S_K^i|$ represents the number of solutions obtained in the ith run, t_i denotes the computational times consumed for the ith run, w is the running times.

Following observations can be derived from Table 12:

For solving the three multi-objective problems with maintenance tasks, the proposed algorithm shows efficient performance, especially for the largest scale $15 \times 10 - m$ instance. The computational times for each instance with PM tasks are also competitive to other algorithms;
For solving the five Kacem instances, the proposed DABC algorithm can obtain all non-dominated solutions and the average performances are also efficient;
The standard deviation of the $D_{\prec}$ for each BRdata instance shows that the resulted solutions obtained by the DABC algorithm are all in the Pareto front or very near to the Pareto front. The standard deviation of the $R_{\prec}$ for each instance also verifies the robustness of the proposed algorithm;
The average values of $N_{\prec}$, $R_{\prec}$, and $D_{\prec}$ for each instance show that the proposed algorithm holds effectiveness and efficiency in solving the considered problems;
The average computational times for most instances are competitive to other algorithms.

To deeply verify the performance of the proposed algorithm, Figs. 10–12 illustrate the evolutionary curves of the $N_{\prec}$, $D_{\prec}$, and $R_{\prec}$, respectively, for the three Kacem instances with PM tasks. Meanwhile, Figs. 13–15 show the evolutionary curves of

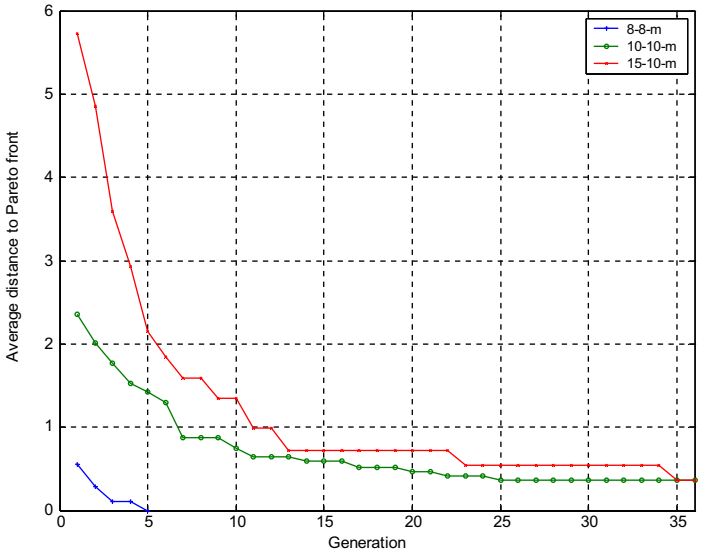

Fig. 11. Evolutionary curves of the $D_{\prec}$ for the three Kacem instances with PM tasks.

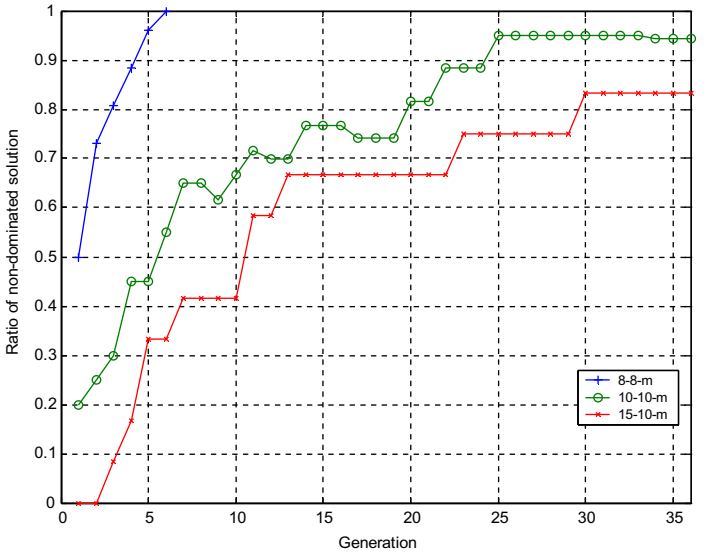

Fig. 12. Evolutionary curves of the $R_{\prec}$ for the three Kacem instances with PM tasks.

the $N_{\prec}$, $D_{\prec}$, and $R_{\prec}$, respectively, for MK01, MK02, MK04, and MK07. It can be concluded from these Figures that the proposed algorithm has an efficient performance during the search process.

5.6. Comparisons of makespan on the seven BRdata problems

Table 13 illustrates the comparison of the makespan results on the seven BRdata instances of the DABC algorithm with six recent algorithms, i.e., the GA by Pezzella et al. [12] (hereafter called P.M.C.), the KBACO algorithm by Xing et al. [19], the PVNS by Yazdani et al. [18], the HTSA [26], the CDDS algorithm by Hmida et al. [20], and the ABC by Wang et al. [21]. The relative improvement of our algorithm with respect to the corresponding algorithm is defined as follows:

$$imp = [(MK_c - MK_{our})/MK_c] \times 100\%, \tag{17}$$

where MK_{our} is the makespan obtained by our algorithm and MK_c is the makespan of the algorithm being compared to.

Following observations can be derived from Table 13:

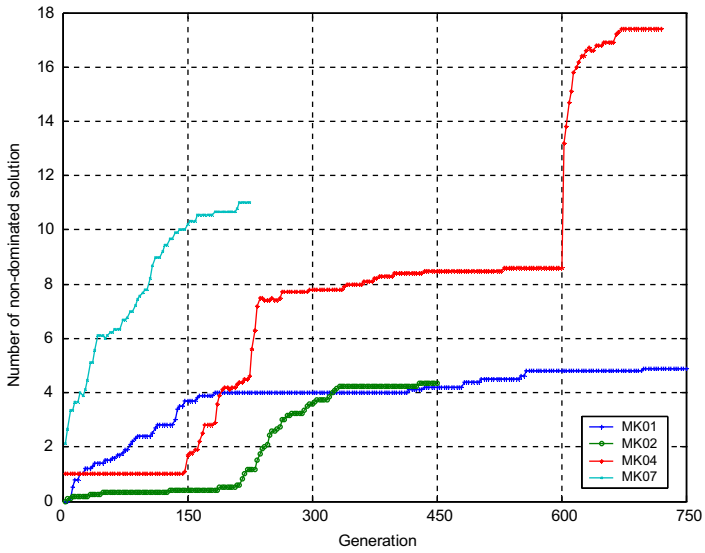

Fig. 13. Evolutionary curves of the $N_{\prec}$ for MK01, MK02, MK04, and MK07.

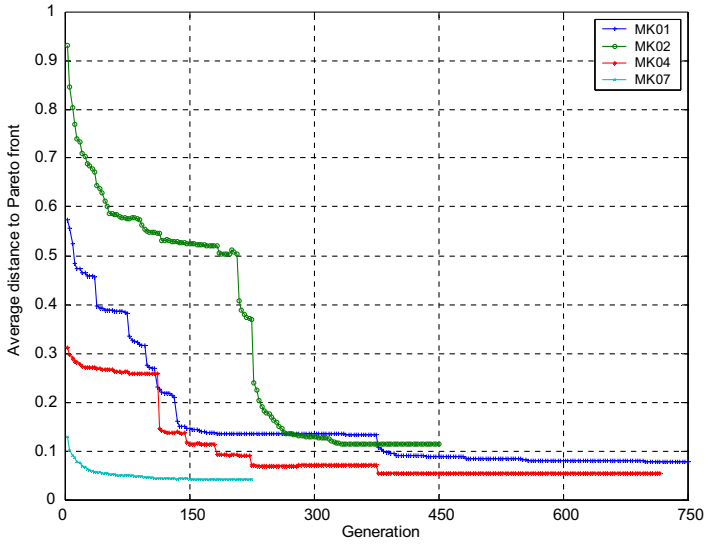

Fig. 14. Evolutionary curves of the $D_{\prec}$ for MK01, MK02, MK04, and MK07.

For solving the instance MK05, the DABC algorithm outperforms the P.M.C., the KBACO, the PVNS, and the CDDS algorithms;
For MK07, the DABC algorithm outperforms the KBACO, the PVNS, and the HTSA algorithms;
The DABC algorithm outperforms the KBACO algorithm in 4 out of 7 problems and outperforms the PVNS approach and the HTSA algorithm in 2 out of 7 problems;
When compared to the CDDS algorithm, the DABC algorithm can obtain either better result in solving MK05 or the same results for other instances;
When compared to the very recently published ABC algorithm, our proposed algorithm can obtain the same best makespan results, which are the best results among all the compared algorithms.
The only slightly worse result obtained by the DABC algorithm is for solving MK01 where the makespan obtained by the KBACO algorithm is 39 while the DABC algorithm generated the makespan of 40;
The last row reports the average improvement of our algorithm with respect to other compared algorithms. It should be noted that the objective functions in the five algorithms, i.e., the P.M.C., the KBACO, the PVNS, the CDDS, and the ABC, are

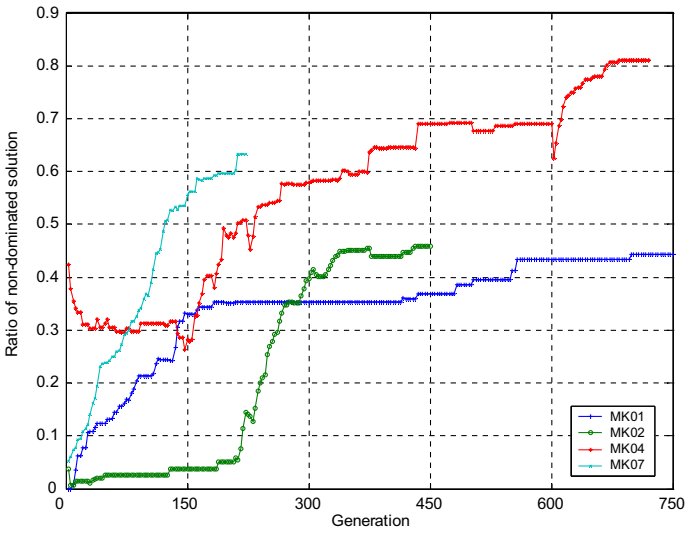

Fig. 15. Evolutionary curves of the $R_{\prec}$ for MK01, MK02, MK04, and MK07.

Table 13
Comparison of makespan on the seven BRdata instances.

Name	Size	op_num	LB	DABC	P.M.C.	*imp* (%)	KBACO	*imp* (%)	PVNS	*imp* (%)	HTSA	*imp* (%)	CDDS	*imp* (%)	ABC	*imp* (%)
MK01	10 × 6	55	36	40	40	0.00	39	-2.56	40	0.00	40	0.00	40	0.00	40	0.00
MK02	10 × 6	58	24	26	26	0.00	29	+10.34	26	0.00	26	0.00	26	0.00	26	0.00
MK03	15 × 8	150	204	204	204	0.00	204	0.00	204	0.00	204	0.00	204	0.00	204	0.00
MK04	15 × 8	90	48	60	60	0.00	65	+7.69	60	0.00	61	+1.64	60	0.00	60	0.00
MK05	15 × 4	106	168	172	173	+0.58	173	+0.58	173	+0.58	172	0.00	173	+0.58	172	0.00
MK07	20 × 5	100	133	139	139	0.00	144	+3.47	141	+1.42	140	+0.71	139	0.00	139	0.00
MK08	20 × 10	225	523	523	523	0.00	523	0.00	523	0.00	523	0.00	523	0.00	523	0.00
Avg imp						+0.08		+2.79		+0.29		+0.34		+0.08		0.00

single objective while the problems considered in this study are multi-objective. Thus, the comparisons of makespan results on the BRdata instances also verify the efficiency of the DABC.

6. Conclusions

This paper aims at solving the multi-objective FJSPs with minimization of the maximal completion time, the total workload, the maximal workload. We considered the problem under both the preventive maintenance constraints and non-maintenance constraints cases and presented a discrete artificial bee colony algorithm. In the proposed algorithm, the food sources were represented as discrete machine number and job permutation. The ABC based searching mechanism with an effective population initialization approach and a TS based local search with a self-adaptive neighboring food source generation strategy were developed to perform exploration and exploitation for promising solutions within the entire solution space. Due to the reasonable hybridization of the ABC search and TS based local search, the proposed DABC algorithm had the ability to obtain promising solutions for the problem considered. Computational simulations and comparisons demonstrated the effectiveness and efficiency of the proposed algorithm. Our future work is to investigate the other meta-heuristics for the multi-objective flexible job shop scheduling problems and generalize the application of the ABC algorithms to solve other combinatorial optimization problems.

Acknowledgements

This research is partially supported by National Science Foundation of China under Grant 61104179 and 61174187, Basic scientific research foundation of Northeast University under Grant N110208001, starting foundation of Northeast University under Grant 29321006, Science Foundation of Liaoning Province in China (2013020016), and Science Research and Development of Provincial Department of Public Education of Shandong under Grant (J08LJ20, J09LG29, and J10LG25). In addition, it is also partially supported by TUBITAK project 110M622.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.apm.2013.07.038.

References

[1] B. Naderi, S.M.T. Fatemi Ghomi, M. Aminnayeri, M. Zandieh, Scheduling open shops with parallel machines to minimize total completion time, J. Comput. Appl. Math. 235 (2011) 1275–1287.
[2] A. Bagheri, M. Zandieh, I. Mahdavi, M. Yazdani, An artificial immune algorithm for the flexible job-shop scheduling problem, Future Gener. Comput. Syst. 26 (2010) 533–541.
[3] P. Fattahi, M. Saidi, F. Jolai, Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, J. Intel. Manuf. 18 (2007) 331–342.
[4] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Ann. Oper. Res. 41 (1993) 157–183.
[5] M. Mastrolilli, L.M. Gambardella, Effective neighborhood functions for the flexible job shop problem, J. Scheduling 3 (2000) 3–20.
[6] P. Fattahi, F. Jolai, J. Arkat, Flexible job shop scheduling with overlapping in operations, Appl. Math. Model. 33 (2009) 3076–3087.
[7] M. Ennigrou, K. Ghedira, New local diversification techniques for flexible job shop scheduling problem with a multi-agent approach, Auton. Agents Multi-Agent 17 (2008) 270–287.
[8] J.Q. Li, Q.K. Pan, P.N. Suganthan, T.J. Chua, A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem, Int. J. Adv. Manuf. 52 (2011) 683–697.
[9] W. Bożejko, M. Uchroński, M. Wodecki, Parallel hybrid metaheuristics for the flexible job shop problem, Comput. Ind. Eng. 59 (2010) 323–333.
[10] I. Kacem, S. Hammadi, P. Borne, Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems, IEEE Trans. Syst. Man Cybern. C 32 (2002) 408–419.
[11] N.B. Ho, J.C. Tay, E.M.K. Lai, An effective architecture for learning and evolving flexible job-shop schedules, Eur. J. Oper. Res. 179 (2007) 316–333.
[12] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Comput. Oper. Res. 35 (2008) 3202–3212.
[13] J. Gao, L. Sun, M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, Comput. Oper. Res. 35 (2008) 2892–2907.
[14] W.J. Xia, Z.M. Wu, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, Comput. Ind. Eng. 48 (2005) 409–425.
[15] L. Gao, C.Y. Peng, C. Zhou, P.G. Li, Solving flexible job shop scheduling problem using general particle swarm optimization, in: Proceedings of the 36th CIE Conference on, Computers & Industrial Engineering, 2006, pp. 3018–3027.
[16] H.B. Liu, A. Abraham, C. Grosan, A novel variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems, in: Proceeding of the second IEEE International Conference on Digital Information Management ICDIM, 2007, pp. 138–145.
[17] G.H. Zhang, X.Y. Shao, P.G. Li, L. Gao, An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, Comput. Ind. Eng. 56 (2009) 1309–1318.
[18] M. Yazdani, M. Amiri, M. Zandieh, Flexible job-shop scheduling with parallel variable neighborhood search algorithm, Expert. Syst. Appl. 37 (2010) 678–687.
[19] L.N. Xing, Y.W. Chen, P. Wang, Q.S. Zhao, J. Xiong, A knowledge-based ant colony optimization for flexible job shop scheduling problems, Appl. Soft Comput. 10 (2010) 888–896.
[20] A.B. Hmida, M. Haouari, M.J. Huguet, P. Lopez, Discrepancy search for the flexible job shop scheduling problem, Comput. Oper. Res. 37 (2010) 2192–2201.
[21] L. Wang, G. Zhou, Y. Xu, S.Y. Wang, An effective artificial bee colony algorithm for the flexible job-shop scheduling problem, Int. J. Adv. Manuf. 60 (2012) 303–315.
[22] I. Kacem, S. Hammadi, P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems, hybridization of evolutionary algorithms and fuzzy logic, Math. Comput. Simulat. 60 (2002) 245–276.
[23] J.C. Tay, N.B. Ho, Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems, Comput. Ind. Eng. 54 (2008) 453–473.
[24] L.N. Xing, Y.W. Chen, K.W. Yang, Multi-objective flexible job shop schedule, design and evaluation by simulation modeling, Appl. Soft Comput. 9 (2009) 362–376.
[25] L.N. Xing, Y.W. Chen, K.W. Yang, An efficient search method for multi-objective flexible job shop scheduling problems, J. Intel. Manuf. 20 (2009) 283–293.
[26] J.Q. Li, Q.K. Pan, Y.C. Liang, An effective hybrid tabu search algorithm for multi-objective flexible job shop scheduling problems, Comput. Ind. Eng. 59 (2010) 647–662.
[27] Q.K. Pan, L. Wang, B. Qian, A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems, Comput. Oper. Res. 36 (2009) 2498–2511.
[28] G. Moslehi, M. Mahnam, A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search, Int. J. Prod. Econ. 129 (2010) 14–22.
[29] J.Q. Li, Q.K. Pan, K.G. Gao, Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems, Int. J. Adv. Manuf. 55 (2011) 1159–1169.
[30] G. Schmidt, Scheduling with limited machine availability, Eur. J. Oper. Res. 121 (2000) 1–15.
[31] Y. Ma, C.B. Chu, C.R. Zuo, A survey of scheduling with deterministic machine availability constraints, Comput. Ind. Eng. 58 (2010) 199–211.
[32] J. Gao, M. Gen, L.Y. Sun, Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm, J. Intel. Manuf. 17 (2006) 493–507.
[33] N. Zribi, A.E. Kamel, P. Borne, Minimizing the makespan for the MPM job-shop with availability constraints, Int. J. Prod. Econ. 112 (2008) 151–160.
[34] F.T.S. Chan, S.H. Chung, L.Y. Chan, G. Finke, M.K. Tiwari, Solving distributed FMS scheduling problems subject to maintenance, genetic algorithms approach, Robot. Cim-Int. Manuf. 22 (2006) 493–504.
[35] S.J. Wang, J.B. Yu, An effective heuristic for flexible job-shop scheduling problem with maintenance activities, Comput. Ind. Eng. 59 (2010) 436–447.
[36] D. Karaboga, An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
[37] D. Karaboga, B. Basturk, On The performance of artificial bee colony (ABC) algorithm, Appl. Soft Comput. 8 (2008) 687–697.
[38] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Appl. Math. Comput. 214 (2009) 108–132.
[39] Q.K. Pan, M.F. Tasgetiren, P.N. Suganthan, T.J. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Inform. Sci. 181 (2010) 2455–2468.
[40] C. Özgüven, L. Özbakır, Y. Yavuz, Mathematical models for job-shop scheduling problems with routing and process plan flexibility, Appl. Math. Model. 34 (2010) 1539–1548.
[41] F. Glover, Tabu Search, A Tutorial, Interfaces, 20 (1990) pp. 74–94.
[42] M. Dell'Amico, M. Trubian, Applying tabu search to the job-shop scheduling problem, Ann. Oper. Res. 41 (1993) 231–252.
[43] E. Nowicki, C. Smutnicki, A fast taboo search algorithm for the job-shop problem, Manage. Sci. 42 (1996) 797–813.

[44] C.Y. Zhang, P.G. Li, Z.L. Guan, Y.Q. Rao, A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem, Comput. Oper. Res. 34 (2007) 3229–3242.
[45] K. Deb, A. Paratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm, NSGA-II, IEEE Trans. Evol. Comput. 6 (2002) 182–197.
[46] J. Gao, M. Gen, L.Y. Sun, X.H. Zhao, A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems, Comput. Ind. Eng. 53 (2007) 149–162.