ENGINEERING OPTIMIZATION, 2017
https://doi.org/10.1080/0305215X.2017.1353090

Taylor & Francis
Taylor & Francis Group

Estimation of distribution algorithm with path relinking for the blocking flow-shop scheduling problem

Zhongshi Shao[a], Dechang Pi[a,b] and Weishi Shao[a]

[a]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, PR China; [b]Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, PR China

ABSTRACT

This article presents an effective estimation of distribution algorithm, named P-EDA, to solve the blocking flow-shop scheduling problem (BFSP) with the makespan criterion. In the P-EDA, a Nawaz–Enscore–Ham (NEH)-based heuristic and the random method are combined to generate the initial population. Based on several superior individuals provided by a modified linear rank selection, a probabilistic model is constructed to describe the probabilistic distribution of the promising solution space. The path relinking technique is incorporated into EDA to avoid blindness of the search and improve the convergence property. A modified referenced local search is designed to enhance the local exploitation. Moreover, a diversity-maintaining scheme is introduced into EDA to avoid deterioration of the population. Finally, the parameters of the proposed P-EDA are calibrated using a design of experiments approach. Simulation results and comparisons with some well-performing algorithms demonstrate the effectiveness of the P-EDA for solving BFSP.

ARTICLE HISTORY
Received 15 November 2016
Accepted 22 June 2017

KEYWORDS
Flow-shop scheduling with blocking; estimation of distribution algorithm; path relinking; makespan

1. Introduction

The traditional permutation flow-shop scheduling problem (PFSP) with infinite buffer capacity has been extensively investigated over the past 60 years (Vallada and Ruiz 2010; Shao and Pi 2016; Wang *et al.* 2017). However, in practice, owing to technological requirements or process characteristics, the buffer capacity between two consecutive machines may be totally absent. The traditional PFSP becomes the blocking flow-shop scheduling problem (BFSP) (Ronconi and Henriques 2009). In such cases, the job has to be blocked on its machine until its next machine is free (Ribas, Companys, and Tort-Martorell 2011; Han, Gong, Jin, *et al.* 2016). Many modern production systems can be modelled as BFSPs when no buffers exist between consecutive machines, such as in the iron and steel industry (Gong, Tang, and Duin 2010), serial manufacturing processes (Koren, Wang, and Gu 2017), robotic cells (Ribas, Companys, and Tort-Martorell 2015) and the chemical industry (Merchan and Maravelias 2016). Hence, it is necessary to develop effective and efficient approaches for such problems.

BFSP with the makespan criterion is a typical NP-hard problem when the number of machines is larger than two (Hall and Sriskandarajah 1996). With increasing problem size, the BFSP becomes more and more complicated and is difficult to solve completely. Therefore, to tackle this challenge, many constructive heuristics, such as profile fitting (PF), Nawaz–Enscore–Ham (NEH) and MinMax

CONTACT Dechang Pi nuaacs@126.com

Supplemental data for this article can be accessed at https://doi.org/10.1080/0305215X.2017.1353090

(MM), which use some specific rules to assign a priority index to each job to construct a scheduling permutation, were proposed. Moreover, Ronconi (2004) presented two constructive heuristics, *i.e.* MME and PFE, which combined NEH with PF and MM, respectively. The author showed that the performances of both MME and PFE outperformed the NEH heuristic over the instances with 500 jobs and 20 machines. Companys, Ribas, and Mateo (2010) improved the NEH heuristic by using several different priority rules and tie-breaking strategies. Wang *et al.* (2012) proposed a modified NEH heuristic based on the average value and standard deviation of the processing time. Pan and Wang (2012) introduced the concept of weight value into PF and presented two new constructive heuristics, *i.e.* profile fitting (wPF) and the Pan–Wang (PW) heuristic. They also combined them with NEH and proposed three improved constructive heuristics, namely PF-NEH, wPF-NEH and PW-NEH. Wang, Pan, and Tasgetiren (2010) proposed an improved NEH heuristic named NEH_WPT, which sorted jobs in non-decreasing order of the sum of their processing times on all machines.

Some metaheuristics have also been proposed to solve the BFSP with the makespan criterion to obtain better solutions. Wang *et al.* (2010) proposed a hybrid discrete differential evolution (HDDE) algorithm in which a new mutation and crossover operator were developed. To improve the efficiency of the whole algorithm, the authors also presented a speed-up method to evaluate the insert neighbourhood. Lin and Ying (2013) presented a revised artificial immune system (RAIS) algorithm based on the features of artificial immune systems and the annealing process of simulated annealing algorithms. Pan *et al.* (2013) proposed a high-performing memetic algorithm (MA). In the MA, a path-relinking-based crossover operator, a referenced local search (RLS) and a procedure for controlling diversity were used. Han, Gong, and Sun (2014) proposed a hybrid algorithm named DE-ABC, which combined the discrete artificial bee colony (ABC) with differential evolution (DE). In DE-ABC, the mutation and crossover operators of discrete DE were used by the employed bee operator of ABC. Ding *et al.* (2016) proposed an iterated greedy algorithm based on some new block properties of BFSP. Han *et al.* (2016) proposed a modified fruit fly optimization (MFFO) algorithm, which employed three key operators, *i.e.* a problem-specific heuristic, a neighbourhood strategy and a speed-up insert-neighbourhood-based local search. Besides the above metaheuristics, other high-performing metaheuristics have been proposed to solve the problem under consideration. These include two tabu searches (Grabowski and Pempera 2007), a dynamic multi-swarm particle swarm optimization (DMS-PSO) (Liang *et al.* 2011), an iterated greedy algorithm (IG) (Ribas, Companys, and Tort-Martorell 2011), an improved artificial bee colony (IABC) (Han *et al.* 2012), a hybrid modified global-best harmony search (hmgHS) (Wang, Pan, and Tasgetiren 2011), a three-phase algorithm (TPA) (Wang *et al.* 2012), a discrete particle swarm optimization (DPSO) (Wang and Tang 2012), a discrete self-organizing migrating algorithm (DSOMA) (Davendra and Bialic-Davendra 2013) and variable neighbourhood searches (VNSs) (Ribas, Companys, and Tort-Martorell 2013). From the previous research, it can be seen that the superiority of these high-performing approaches can be attributed to an effective constructive heuristic to generate initial solutions with good quality, and the combination of global search and local search strategies to achieve an appropriate balance between exploration and exploitation. However, these approaches have their shortcomings. On the one hand, if the metaheuristics adopt a single individual as the population, such as in IG, TPA and VNS, a great deal of time will be consumed for the quality of solutions to reach a high level. On the other hand, several metaheuristics, such as DMS-PSO, MA, IABC, hmgHS and the genetic algorithm (GA), employ the idea of swarm intelligence to find the global optimum. This mechanism can effectively enrich the search directions by means of multiple individuals, but it takes a lot of effort to evolve the whole population through different evolutionary operators. In particular, the metaheuristics based on swarm intelligence often have poor performance on large-scale discrete optimization problems. Therefore, a well-known metaheuristic called the estimation of distribution algorithm (EDA) is introduced in this article and adapted with some distinctive evolutionary strategies to overcome the above problems.

EDA is a stochastic optimization technique based on statistical learning (Hauschild and Pelikan 2011) which has been developed to solve a variety of optimization problems, such as arc routing problems (Wang *et al.* 2015), production scheduling (Shao, Pi, and Shao 2017), water distribution

network optimization (Qi, Li, and Potter 2016) and vehicle routing (Pérez-Rodríguez and Hernández-Aguirre 2016). Unlike GAs, which use mutation and crossover operators to generate new individuals, EDA generates offspring through building and sampling explicit probabilistic models of promising candidate solutions. EDA is good at the automatic discovery and exploration of problem regularities. Until now, only Jarboui *et al.* (2009) have proposed a hybrid EDA to solve the BFSP with the makespan criterion. In this EDA, a probabilistic model is built based on both the order of the jobs in the sequence and the similar blocks of jobs. However, compared with the state-of-the-art metaheuristics, this EDA is uncompetitive. Moreover, some drawbacks of this EDA have been pointed out by Pan and Ruiz (2012a). Therefore, an effective EDA (named P-EDA) is proposed in this article to solve the BFSP with the makespan criterion. In the proposed algorithm, an NEH-based heuristic and the random method are combined to initialize the population. A modified linear rank selection is used to select the superior individuals from the parental population. An effective probabilistic model is constructed to guide the algorithm to search the promising solution space. To enhance the convergence property of EDA and avoid blindness of the search, the path relinking method is incorporated into EDA. A modified referenced local search (mRLS) based on a random trajectory, a speed-up method and a pruning procedure is used to enhance the exploitation capability of the proposed algorithm. Finally, a diversity-maintaining scheme is adopted to avoid the deterioration of the population.

The contributions of this work can be summarized as follows. (1) The path relinking technique is incorporated into EDA to guide the search process. The advantages of using path relinking were confirmed by the experimental results over the benchmark instances of BFSP. (2) The problem that some elite solutions may not be selected in traditional linear rank selection, owing to randomness of sampling, is solved. (2) The shortcoming of the probabilistic model proposed by Pan and Ruiz (2012a) is improved. (3) An mRLS is proposed to enhance the exploitation ability of the EDA. Its effectiveness and contribution to the proposed algorithm are confirmed by experiments. (4) The performance of the proposed P-EDA is evaluated and compared with other well-performing algorithms. The reported computational results show that the proposed algorithm is superior to all of the compared algorithms.

The rest of this article is organized as follows. Section 2 gives a detailed description of the BFSP with the makespan criterion. Section 3 elaborates on the proposed P-EDA. Section 4 shows the computational evaluation of the algorithms with statistical analyses. Finally, conclusions and suggestions for further research are presented in Section 5.

2. Problem description

The BFSP, denoted as $Fm|blocking|C_{\max}$ according to the notation proposed by Graham *et al.* (1979), can be briefly described as follows: n jobs have to be processed on m machines. Each job has m operations processed on m machines sequentially in the same order. Each job can be only processed on one machine at a time. Each machine can process one job at a time. There are no intermediate buffers between adjacent machines. This means that upon completion of its operation on one machine, the job cannot leave its machine until the next machine is available for processing it. Pre-emption is not allowed. Each job j has a deterministic positive processing time on every machine i. Jobs and machines are usable from time zero. The objective usually considered in $Fm|blocking|C_{\max}$ is to determine a sequence for processing all jobs on all machines so that the maximum completion time (called makespan) is minimized.

Let $\pi = \{\pi(1), \pi(2), \ldots, \pi(k), \ldots, \pi(n)\}$ denote a permutation sequence, in which $\pi(k)$ represents the job at the kth position of π. Let $p_{\pi(i),k}$ denote the processing time of job $\pi(i)$ on machine k. The departure time of job $\pi(k)$ on machine k is denoted as $D_{\pi(j),k}$. According to the literature (Ronconi 2004), $C_{\max}$ can be calculated with the following recursive formulations:

$$D_{\pi(1),0} = 0 \tag{1}$$

$$D_{\pi(1),k} = D_{\pi(1),k-1} + p_{\pi(1),k} \quad k = 1, 2, \ldots, m-1 \tag{2}$$

$$D_{\pi(j),0} = D_{\pi(j-1),1} \quad j = 2,3,\ldots,n \tag{3}$$

$$D_{\pi(j),k} = \max\{D_{\pi(j),k-1} + p_{\pi(j),k-1}, D_{\pi(j-1),k+1}\} \quad j = 2,3,\ldots,n \quad k = 1,2,\ldots,m-1 \tag{4}$$

$$D_{\pi(j),m} = D_{\pi(j),m-1} + p_{\pi(j),m} \quad j = 1,2,\ldots,n \tag{5}$$

Then, the makespan is $C_{\max}(\pi) = D_{\pi(n),m}$. Therefore, the BFSP with the makespan criterion is to find a permutation sequence for processing the jobs such that $C_{\max}(\pi^*) = \min C_{\max}(\pi), \forall \pi \in \Pi$. To clearly explain the process of calculating makespan, a detailed example is provided in the online supplementary material.

3. The proposed P-EDA for the BFSP

3.1. Initialization

In the scheduling literature, it is very common to construct a few good initial individuals by effective constructive heuristics and to produce others randomly (Pan *et al.* 2014; Karthikeyan *et al.* 2015). NEH is a famous heuristic with excellent performance, which has been used in many algorithms to produce initial solutions in the traditional PFSP (Shao and Pi 2016). However, the longer total processing time may lead to a higher probability of occurrence of blocking in BFSP (Ding *et al.* 2016), so that the original NEH may not produce good solutions. Hence, an effective heuristic, *i.e.* PF-NEH, proposed by Pan and Wang (2012) is used to initialize part of the initial solutions to intensify the quality of the initial population. Meanwhile, other initial solutions are randomly generated in the search space to maintain the diversity of the initial population. In the PF-NEH heuristic, the well-known PF heuristic first determines an initial sequence of all jobs. Then, the NEH enumeration procedure is performed on the last λ jobs of this sequence. Following Pan and Wang (2012), the parameter λ is set to 25 when the problem is larger than 25 jobs and set to n (the number of jobs) for problems with fewer than 25 jobs. The procedure of initializing the population is described in Algorithm 1.

Algorithm 1: Initialization of population

Step 1: Let Pc denote the population. Set $Pc = \emptyset$ and initialize λ and the population size PS.
Step 2: Generate a job sequence $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ by sorting the total processing time of all jobs in ascending order. Set $l = 1$.
Step 3: Apply the PF-NEH heuristic to produce an initial individual through the following steps.
  Step 3.1: Let $\beta = \{\alpha_l\}$. Construct a completed job sequence $\beta = \{\beta_1, \beta_2, \ldots, \beta_n\}$ through the PF heuristic.
  Step 3.2: Let $\pi^* = \{\beta_1, \beta_2, \ldots, \beta_{n-\lambda}\}$.
  Step 3.3: For the job in sequence $\beta = \{\beta_1, \beta_2, \ldots, \beta_n\}$ from position $k = n - \lambda + 1$ to n, find the best position j^* by inserting β_k into the current partial sequence π^* to minimize the partial makespan. Then, insert β_k into sequence π^* at position j^*.
  Step 3.4: Put π^* into the population, *i.e.* $Pc = Pc \cup \pi^*$.
Step 4: If $|Pc| == 0.1 \times PS$ or $l == n$, then go to Step 5; otherwise, $l = l + 1$, return to Step 3.
Step 5: Randomly generate an individual π in the solution space. If there are no identical individuals in the current population, add it to the population, *i.e.* $Pc = Pc \cup \pi$; otherwise, discard this individual. Note that the similarity between two individuals is defined as follows:

$$S(\pi_1, \pi_2) = \sum_{i=1}^{n} x_i(\pi_{1,i}, \pi_{2,i}) \tag{6}$$

$$x_i(\pi_{1,i}, \pi_{2,i}) = \begin{cases} 0 & \text{if } \pi_{1,i} = \pi_{2,i} \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

If $S(\pi_1, \pi_2) < \varepsilon$, it means that these two individuals are similar. To adequately search the whole solution space, ε is set to zero to make the population distribute in the whole solution space as much as possible.

Step 6: Repeat Step 5 until the population has *PS* individuals.

In the above procedure, the jobs with different total processing times are used as the initial job of the PF-NEH heuristic to generate $0.1 \times PS$ different initial solutions. These initial solutions are not only of good quality but also different from each other. This strategy is chosen since if most individuals were randomly generated in the search space and only one individual was much better than the others, it would spend too much time in exploiting the area around these individuals to make them achieve a high level of quality in one iteration. As a result, the ability of the algorithm to perform more iterations would diminish. In particular, improvement of these random individuals using an exhausted local search method would consume a great deal of time and effort. The main computational burden of the PF-NEH can be attributed to the insertion process of Steps 3.1– 3.4. In the insertion procedure of PF-NEH, $(2n - \lambda + 1)\lambda/2$ partial sequences should be evaluated, which gives a total computational complexity of $O(mn^3)$. In this article, the speed-up method (Wang *et al.* 2010) is used to evaluate these partial sequences, so the PF-NEH can be completed in $O(mn^2)$ computational efforts compared with $O(mn^3)$ for the original PF-NEH.

3.2. Selection

The construction of a probabilistic model of EDA depends on an elite set of individuals which are selected from the parental population using a selection operator. The linear rank selection operator has been extensively used in the selection procedure of EDA (Jarboui *et al.* 2009). In the traditional linear rank selection operator, the individuals are selected according to their rank in the population. The rank of the individuals is first transformed to the cumulative probability. Then, the individuals in the population are selected through sampling. However, there is a problem in this selection procedure. If only a few individuals are selected to construct the probabilistic model, some superior individuals in the population may not be selected since the random numbers (pseudo-random numbers) generated by the computer probably will not fall into the cumulative probabilistic intervals of these superior individuals. That is to say, the best individual in the population may not be selected or participate in the construction phase of the probabilistic model, which would result in an imprecise probabilistic model. Therefore, to choose those individuals with good quality, a modified linear rank selection operator is presented in this article. Two selection methods are used in this selection operator. One is based on elite selection, which ensures that the best individual or the top few best individuals will certainly be selected. The other is based on traditional linear rank selection, which ensures that those individuals participating in construction of the probabilistic model have a certain level of diversity so that the offspring individuals are different from each other. Hence, this modified linear rank selection operator can be used to construct an accurate probabilistic model and thereby obtain an offspring population with high quality and diversity. Algorithm 2 displays the procedure of the new selection operator.

Algorithm 2: Modified linear rank selection

Step 1: Let $\Pi_e = \emptyset$ and $l = PS$. *PS* is the population size. The individuals in population $Pc = \{\pi_1, \pi_2, \ldots, \pi_{PS}\}$ are sorted in descending order of their makespan. The selection probability of each individual can be calculated as follows:

$$p(\pi_i) = \frac{r_i}{\sum_{\pi_i \in Pc} r_i} \tag{8}$$

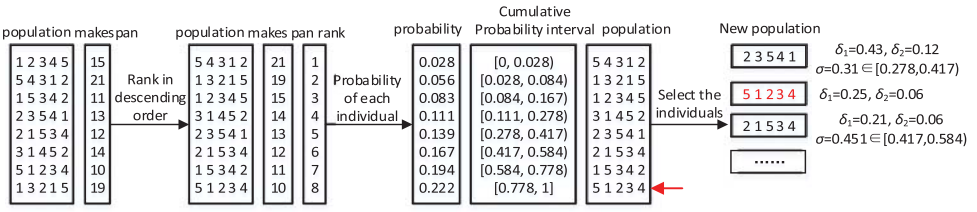

Figure 1. The procedure of the modified linear rank selection operator.

where π_i denotes the ith individual and r_i denotes its rank. Obviously, π_i with $r_i = PS$ is the best one in the current population, which has a high probability of being selected.

Step 2: Calculate the cumulative probabilistic *sumProbability* for the ith individual in the population according to $sumProbability\,(i) = sumProbability\,(i-1) + p\,(\pi_i)$, where $sumProbability\,(0) = 0$.

Step 3: Generate two uniformly random numbers δ_1 and δ_2 in the interval [0, 1].

Step 4: If $\delta_1 < \delta_2$, then select the individual π_i with $r_i = l$, put it into Π_e, *i.e.* $\Pi_e = \pi_i \cup \Pi_e$, set $l = l + 1$, and go to Step 5; otherwise, generate a uniformly random number σ in the interval [0, 1], and find the cumulative probability interval which satisfies $sumProbability\,(i-1) \leq \sigma < sumProbability\,(i)$, $i = 1, 2, 3, \ldots, n$ according to this random number. Set $\pi_k = \pi_i$ and put π_k into Π_e, *i.e.* $\Pi_e = \pi_k \cup \Pi_e$.

Step 5: Repeat Steps 3–4 until Π_e has PS individuals.

To further describe the procedure of the modified linear rank selection operator, an example (number of jobs $= 5$, $PS = 8$) is given in Figure 1. In Figure 1, eight individuals are first ranked according to their makespan and then the cumulative probability for each individual is calculated according to its rank. Afterwards, the individuals in the population are selected based on the cumulative probability or rank. For example, generate two random numbers, *e.g.* $\delta_1 = 0.43$ and $\delta_2 = 0.12$, $\delta_1 < \delta_2$, and one individual is selected based on the cumulative probability. A uniformly random number $\sigma = 0.31$ is generated, which is in [0.278, 0.417), so the individual {2, 3, 5, 4, 1} is selected and saved into the new population. Then, generate two random numbers again, *e.g.* $\delta_1 = 0.25$ and $\delta_2 = 0.06$, $\delta_1 > \delta_2$, and one individual is selected based on rank. So, the first best individual is selected and saved into the new population. Constantly switch the two cases above in a random way until the new population has PS individuals.

3.3. Probabilistic model and sampling method

The conventional EDAs use the probabilistic model to guide the exploration of the search space. Therefore, it is crucial for a high-powered EDA to build an effective probabilistic model. To address lot-streaming flow-shop scheduling problems with set-up times, Pan and Ruiz (2012a) proposed an effective probabilistic model with consideration of both job orders in the sequence and similar job blocks of the selected elite individuals. In their probabilistic model, the probability of placing job i at position k of the offspring, *i.e.* $\xi_{i,k}$, is calculated as follows:

$$\xi_{i,k} = \begin{cases} \dfrac{\rho_{i,k}}{\sum_{l\in\Omega_k}\rho_{j,l}} & k = 1 \\ \dfrac{1}{2}\left(\dfrac{\rho_{i,k}}{\sum_{l\in\Omega_k}\rho_{j,l}} + \dfrac{\tau_{j',i}}{\sum_{l\in\Omega_k}\tau_{j',l}}\right) & k = 2, 3, \ldots, n \end{cases} \tag{9}$$

where $\rho_{i,k}$ denotes the number of times job i appears before or at position k in the selected individuals, $\tau_{j',i}$ denotes the number of times that job i appears immediately after job j' in position $k-1$, and Ω_k is the set of jobs not scheduled until position k.

The above method of constructing a probabilistic model has a disadvantage when it is applied to solve the flow-shop scheduling problem. When the diversity of the population becomes poor and the individuals in the population are very similar, the matrix of job blocks tends to be sparse. This may mean that the number of times that job i in Ω_k appears immediately after job j' in position $k-1$ equals zero, *i.e.* $\tau_{j',i}=0$, so that there exists an illegal case, *i.e.* $\sum_{l\in\Omega_k}\tau_{j',l}=0$. However, the authors did not mention how to handle this illegal case. Therefore, to overcome this shortcoming, a modified probabilistic model is presented to determine $\xi_{i,k}$ as

$$\xi_{i,k}=\begin{cases}\dfrac{\rho_{i,k}}{\sum_{l\in\Omega_k}\rho_{j,l}} & k=1\\ \dfrac{1}{2}\left(\dfrac{\rho_{i,k}}{\sum_{l\in\Omega_k}\rho_{j,l}}+\eta^{i}_{j',k}\right) & k=2,3,\ldots,n\end{cases}\tag{10}$$

where $\eta^{i}_{j',k}$ is the probability of job i in unscheduled job set Ω_k appearing immediately after job j' in position $k-1$, which is defined as follows:

$$\eta^{i}_{j',k}=\begin{cases}\dfrac{\tau_{j',i}}{\sum_{l\in\Omega_k}\tau_{j',l}} & \sum\limits_{l\in\Omega_k}\tau_{j',l}=0\\ \dfrac{1}{|\Omega_k|} & \sum\limits_{l\in\Omega_k}\tau_{j',l}\neq 0\end{cases}\tag{11}$$

where $|\Omega_k|$ is the size of unscheduled jobs. In Equation (11), when $\sum_{l\in\Omega_k}\tau_{j',l}=0$, a block is artificially allocated for each unscheduled job, *i.e.* $\tau_{j',l}=1$, $l\in\Omega_k$, so that each unscheduled job has the same probability of appearing after job j' in position $k-1$, *i.e.* $\frac{1}{\sum_{l\in\Omega_k}\tau_{j',l}}=\frac{1}{|\Omega_k|}$. This treatment would effectively avoid the loss of information of similar job blocks caused by poor diversity and enhance the self-disturbance of the probabilistic model. Based on Equation (10), the job assigned to a specific position k in the sequence can be determined by the probability of $\xi_{i,k}$.

To generate a job sequence with high quality, a new sampling method is developed based on the probabilistic model above. After determining the assignment probability of each job according to Equation (10), one specific job can be selected by a roulette wheel selection method for the current position. The detailed procedure of generating a new individual is shown in Algorithm 3.

Algorithm 3: Sampling method

Step 1: Set the position index $k=1$, $\pi=\emptyset$. Let $\Omega=\{\Omega(1),\Omega(2),\ldots,\Omega(n)\}$ denote the unassigned jobs set.

Step 2: Calculate the probabilistic vector $P=\{\xi_{\Omega(1),k},\xi_{\Omega(2),k},\ldots,\xi_{\Omega(n-k+1),k}\}$ for position k using Equation (10).

Step 3: Construct a roulette wheel vector rw based on P, *i.e.* $rw(0)=0$, $rw(j)=rw(j-1)+\xi_{\Omega(j),k}$, $j=1,2,\ldots,n-k+1$.

Step 4: Generate a rand number r in the interval [0, 1]. If $rw(j-1)<r<rw(j)$, then $\pi(k)=\Omega(j)$, and remove $\Omega(j)$ from Ω, *i.e.* $\Omega=\Omega\backslash\Omega(i)$.

Step 5: If $k==n$, then output π; otherwise, $k=k+1$ and return to Step 2.

3.4. Path relinking

EDA generates new individuals using the probabilistic model. Although it has been proven that EDA can converge to the global optimum (Zhang and Mühlenbein 2004), two weaknesses mean that the

convergence property of EDA is not excellent. First, it is very difficult to build an exact probabilistic model based on a limited population. Secondly, owing to the randomness of sampling, the new individuals may deviate from the centre of the probabilistic model so that the fitness of these individuals is very poor. This leads to blindness of the search when using the probabilistic model to guide the search process. To overcome these drawbacks, a search technique named path relinking, proposed by Glover and Laguna (2007), is used to guide the search process. Path relinking is mostly used as a local search procedure. It can generate a set of new individuals or offspring between two good individuals. In this work, path relinking is used to revise the individuals that deviate from the centre of the probabilistic model and improve the convergence property of EDA. In path relinking, given two selected individuals, the original individual α and the guide individual β, a series of movements (*i.e.* swap moves or insertion moves) is performed on these two individuals to transform α to β. Each time, a movement is carried out and an intermediate individual is obtained. With increasing movements, the obtained intermediate individual becomes very similar to β and very different from α. Finally, all obtained intermediate individuals are evaluated and the individual with the lowest makespan will be selected. Note that the set of moves from individual α to β is not the same as from β to α (Vallada and Ruiz 2010). However, if α is very similar to β, there exists one exceptional case that needs only one move from α to β. No intermediate individuals are obtained. To avoid this situation, Pan *et al.* (2013) proposed a method that performed a random insertion or swap move on β to ensure that a new individual is generated. Therefore, in the path relinking in the present study, to ensure the diversity of the search, the swap move is used as the main transforming tool, while the insertion move is used to ensure that at least one new individual is generated. It should be noted that the guide individual, *i.e.* β, comes from the best individual in the population, while the original individual, *i.e.* α, is generated by sampling from the probabilistic model. The procedure of path relinking is described in Algorithm 4.

Algorithm 4: Path relinking

Step 1: Set path set $\Pi_p = \emptyset$, $k = 1$, $\pi = \alpha$.
Step 2: For each position $k = 1$ to n, if $\pi(k) \neq \beta(k)$, find the position p of job $\beta(k)$ in the individual π and swap the job in position k with p for the individual π. If $\pi \neq \beta$, save the individual π into Π_p; otherwise, go to Step 3.
Step 3: If Π_p is empty, randomly select two positions p_1 and p_2 in β, $p_1 \neq p_2$, and insert the job in position p_1 before the job in position p_2. Then, put the generated individual into Π_p; otherwise, go to Step 4.
Step 4: Evaluate all individuals in Π_p and return the one with the lowest makespan.

It can be observed from the procedure of path relinking above that its time complexity depends on the differences between the original and the guide individual. In the worst case, the sequence of the original individual is completely contrary to the guide individual. A total of $n/2$ intermediate individuals will be generated and their evaluation will consume $O(mn^2/2)$. In the best case, the original individual only needs to generate an intermediate individual to obtain the guide individual, so the time complexity is $O(mn)$. In the exceptional case, no intermediate individuals are generated and only one mutation individual is generated by executing a random insertion move, which needs $O(n)$ to execute the insertion move and $O(mn)$ for evaluation. However, the path relinking would not consume much computational time. This is because, in the proposed algorithm, the original individual is generated by sampling from the probabilistic model. The guide individual is the best individual in the population. The probabilistic model has extracted the features of the superior individuals in the population. Therefore, the individuals generated by sampling from the probabilistic model are similar to the best solution in the population but different from each other. Hence, only a few intermediate individuals are generated between the original and the guide individual. Meanwhile, the best individual

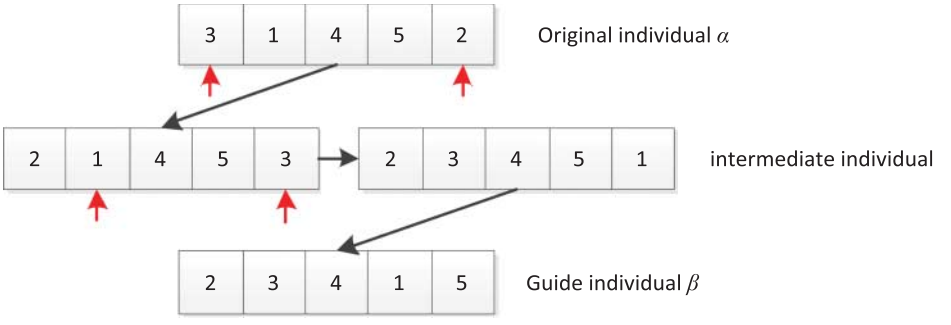

Figure 2. An example of path relinking.

in the population is selected as the guide individual, which ensures that the generated intermediate individuals are of good quality.

To clearly illustrate the procedure of path relinking, an example with two permutations $\alpha = \{3, 1, 4, 5, 2\}$ and $\beta = \{2, 3, 4, 1, 5\}$ is considered in Figure 2. According to the rule of path relinking, for the individual α, job 3 on position 1 will be swapped with job 2 on position 5. Then, a new intermediate individual {2,1,4,5,3} is generated. At the same time, swap job 1 on position 2 with job 3 on position 5, and another individual {2,3,4,5,1} is generated. Lastly, swap job 5 on position 4 with job 1 on position 5, generating β. Two intermediate individuals, *i.e.* {2,1,4,5,3} and {2,3,4,5,1}, are obtained.

3.5. Local search

To enhance the exploitation of EDA, a local search is implemented to quickly lead the current new individual towards the promising area. To solve the BFSP with the makespan criterion, Wang *et al.* (2010) presented an RLS, which has been adopted in other algorithms (Wang, Pan, and Tasgetiren 2011; Pan *et al.* 2013). In RLS, $\pi_R = \{\pi_R(1), \pi_R(2), \ldots, \pi_R(n)\}$ denotes the reference sequence. Usually, the best individual or a random permutation of all jobs is selected as the reference sequence. The process of RLS is shown as follows. Suppose that the current individual is $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$. First, the RLS removes job $\pi_R(i)$, $i = 1, 2, \ldots, n$, from the current individual π and inserts it into all possible slots of π. Then, find the slot j with the lowest makespan and insert job $\pi_R(i)$ into the slot j of π to generate a new individual π_2. Finally, if π_2 is better than π, π will be replaced by π_2. Repeat this procedure until no better solutions are obtained.

Such a local search has the disadvantage that once the reference sequence π_R is given, the search path of RLS becomes deterministic. That is, all movements among jobs are known. This mechanism makes RLS lack an effective self-disturbance ability and has difficulty changing the search path by itself. Therefore, an mRLS is developed in this article. In the mRLS, a random trajectory $rt = \{rt(1), rt(2), \ldots, rt(n)\}$ (*i.e.* a random permutation of n jobs) is generated to replace the reference sequence. After searching all the track points in rt, the search trajectory will be updated. The Fisher–Yates shuffle method (https://en.wikipedia.org/wiki/Fisher-Yates_shuffle) is adopted to update the current trajectory. Therefore, this mechanism can effectively enhance the diversity of the search path. The procedure of the mRLS is shown in Algorithm 5.

Algorithm 5: Modified RLS

Step 1: Set $count = 0$, $j = 0$ and input the individual π.
Step 2: Randomly generate a trajectory $rt = \{rt(1), rt(2), \ldots, rt(n)\}$.
Step 3: Set $j = j + 1$. If $j > n$, then let $j = \mathrm{mod}(j, n)$ and update the trajectory $rt = \{rt(1), rt(2), \ldots, rt(n)\}$ using the Fisher–Yates shuffle method. $\pi' \leftarrow$ Remove job $\pi(rt(j))$ from π, $bestfit = \infty$.

Step 4: For $i = 1$ to $n, \pi_t \leftarrow$ the individual by reinserting job $\pi(rt(j))$ in position i of π'. If $C_{max}(\pi_t) < bestfit$, set $bestpos = j$ and $bestfit = f(\pi_t)$.

Step 5: If $bestfit < C_{\max}(\pi)$, then $\pi \leftarrow$ insert job $\pi(rt(j))$ at position $bestpos$ of π', $count = 0$; otherwise, $count = count + 1$.

Step 6: If $count \leq n$, then go back to Step 3; otherwise, stop the procedure and return π.

It can be seen from Algorithm 5 that the main difference between the mRLS and the original RLS is whether the referenced sequence (or the search trajectory) is updated. For the original RLS, the referenced sequence is fixed. When all jobs in the referenced sequence have been considered, the local search will begin again from the first job of the current referenced sequence. This mechanism may cause repeated searches in the same neighbourhoods and result in low efficiency. For the mRLS, the search trajectory is dynamically updated. When all jobs in the search trajectory have been completed but the whole local search is not finished (*i.e.* $count \neq n$), the search trajectory will be updated at this time. This mechanism can prevent the neighbourhoods from always being explored in the same order. In addition, the Fisher–Yates shuffle method is adopted to update the search trajectory since it can produce unbiased permutations, *i.e.* every permutation is equally likely. Hence, compared with the original RLS, the mRLS can dynamically and efficiently exploit the search space.

Moreover, to reduce the complexity of evaluating n new individuals generated by reinserting the $\pi(rt(j))$ job into all possible n positions of π' in Algorithm 5, two accelerating algorithms, *i.e.* a speed-up evaluation (Wang *et al.* 2010) and a pruning procedure (Wang *et al.* 2012), are adopted. The speed-up evaluation first calculates the departure time $D'_{k,j}$ and the tail time $F'_{k,j}$ of job i on machine j in π'. Note that the tail time is the duration between the latest starting time of job i on machine j and the end of the operations. Then, insert the job $\pi(rt(j))$ into the kth position of π' and calculate the departure time $D''_{k,j}$ of $\pi(rt(j))$ on each machine. A new individual π_k is obtained, whose makespan is calculated by

$$C_{\max}(\pi_k) = \max_{j=1,2,\ldots,m} D''_{k,j} + F''_{k,j} \tag{12}$$

Based on the pruning procedure, if one of $D''_{k,j} + F''_{k,j}$ is larger than the makespan of π, π_k must be no better than π. So, it is unnecessary to further evaluate π_k in search of a better schedule. π_k can be discarded and the next insertion position can be considered. The speed-up method reduces the computational effort of evaluating each insertion from $O(mn)$ to $O(m)$ and the pruning procedure further reduces the computational effort. Therefore, the computational complexity of the mRLS is less than $O(mn^2)$ to evaluate the whole insertion neighbourhood of π.

3.6. Diversity-maintaining scheme

As the population evolves, its diversity gradually begins to degenerate and the individuals in the population become very similar. If this problem is detected during the evolution, some diversity-maintaining strategies can be used to overcome it, such as generating a new population, replacing some individuals in the current population or applying some moves over several individuals in the current population. Many diversity measures have been presented in the literature (Pan and Ruiz 2012a; Xu *et al.* 2014). In this work, a method proposed by Vallada and Ruiz (2010) is adopted to calculate the diversity value. In this method, the sum of all positions where two given solutions have different jobs is used to measure similarity. Using this metric, the diversity of the population (Div) can be calculated as follows:

$$Div = \frac{1}{n-1} \sum_{k=1}^{n} \sum_{\alpha=1}^{n} \frac{C_k(\alpha)}{PS} \left(1 - \frac{C_k(\alpha)}{PS}\right) \tag{13}$$

where $C_k(\alpha)$ is the number of times that job α appears at a given position k across the population. Therefore, a diversity value between 0 and 1 can be obtained from Equation (13).

Base on the above diversity value, if the diversity value (*Div*) of the current population falls below a given threshold value γ, $0 \leq \gamma \leq 1$, it means that all individuals are very similar and the population has deteriorated. The current population should be regenerated. In this work, the individuals of the population are first sorted in non-descending order according to their makespan. Then, three schemes are used to regenerate the population with high quality and diversity, as follows: (1) keep the top 20% best individuals from the sorted list; (2) apply one single random insert operation to each of the intermediate 40% individuals; and (3) randomly generate 40% new individuals to replace the last 40% individuals. To determine an appropriate γ value, a calibrated experiment is presented in Section 4.2.

3.7. Overview of the P-EDA

According to the above description, the complete procedure of the proposed P-EDA is described in Algorithm 6. Note that, in contrast to the traditional EDAs, only one new individual is generated in each iteration for the P-EDA. If the new individual is better than the worst individual of the current population and there are no other identical individuals in the current population, the worst individual is replaced by this new individual. Moreover, since only one individual is generated in each iteration, the probabilistic model will be updated for every *PS* iterations. To reduce the computational complexity, the diversity of the population is also checked every *PS* iterations.

Algorithm 6: P-EDA

Step 1: Initialize the population size *PS* and the diversity threshold γ and set $gen = 1$.
Step 2: Initialize the population *Pc* (described in Algorithm 1) and evaluate each individual of the population.
Step 3: Select *PS* individuals from the population with the modified linear rank selection (described in Algorithm 2) to generate the elite set Π_e.
Step 4: Estimate the probabilistic model based on the elite set Π_e.
Step 5: Sample the probabilistic model to generate a new individual π (described in Algorithm 3).
Step 6: Apply path relinking (described in Algorithm 4) to the new individual π (α) and the best individual (β) in the population to construct another complete individual π'.
Step 7: Perform the mRLS (described in Algorithm 5) over π'.
Step 8: If $C_{\max}(\pi') < C_{\max}(\pi_{worst})$ and $\pi' \cap Pc = \emptyset$, then replace the worst individual π_{worst} in the population *Pc* with π'; otherwise, discard it.
Step 9: If $\mathrm{mod}(gen, PS) = 0$, then check the diversity of the population with Equation (13). If $Div < \lambda$, then regenerate the population and update the probabilistic model with the procedure in Steps 3–4.
Step 10: If the termination criterion is reached, then return the best solution found so far and stop the algorithm; otherwise, $gen = gen + 1$, go back to Step 5.

3.8. Computational complexity of P-EDA

Suppose there are n jobs and m machines in the BFSP and the population size is p. From the process of P-EDA above, it can be learned that this algorithm is made up of seven parts. The first part is the generation and evaluation of the initial population. Since $0.1 \times p$ initial individuals are generated by PF-NEH, whose time complexity is $O(mn^2)$, and others are randomly generated in the search space, the time complexity of initialization is $O(0.1pmn^2) + O(0.9pmn)$. The second part is the modified linear rank selection, the time complexity of which is mainly decided by the sort method of the individuals in the population. Since the P-EDA is implemented in Java, which uses merging sort, the time complexity of the second part in the worst case is $O(plog(p))$. The third part is the construction of the probabilistic model, which needs to traverse all individuals in the population to construct an $n \times n$

probabilistic matrix. The time complexity of the third part is $O(n^2)$. The fourth part is the production of offspring, which needs to generate a new individual by sampling from the $n \times n$ probabilistic matrix. Its time complexity is also $O(n^2)$. The fifth part is the path relinking; its time complexity is $O(mn^2/2)$ in the worst case, as discussed in Section 3.4. For the sixth part, the local search phase consumes at least $O(mn^2)$ to exploit a solution, which can be seen in Section 3.5. For the last part, *i.e.* the diversity-maintaining scheme, all individuals in the population will be sorted in non-descending order, so its time complexity is $O(plog(p))$. Note that the local search phase has the maximum time complexity among all parts of the proposed algorithm. According to the analyses above, the time complexity of P-EDA under the condition of k times evolution can be calculated as follows:

$$\begin{aligned} O(k,n,m,p) &= O(0.1pmn^2) + O(0.9pmn) + O(k)^*[2^*O(p\log p) \\ &\quad + 2^*[O(n^2) + O(mn^2)] \\ &\approx O(0.1pmn^2) + O(k)^*[O(2n^2) + O(mn^2)] \\ &\approx O(0.1pmn^2) + O(k)^*O(mn^2) \\ &\approx O(pmn^2) + O(kmn^2) \\ &= O((p+k)mn^2) \end{aligned}$$

4. Computational evaluation

4.1. Experimental set-up

The proposed algorithm and the compared algorithms are coded in Java language. The computational experiments are conducted on a server with two 2.40 GHz Intel® Xeon® E5-2520 v3 processors (12 cores) and 64 GB of RAM running Windows Server 2008. To compare the performance of P-EDA with other methods, the well-known Taillard's benchmark data set (Taillard 1993) is used in the computational experiment. This benchmark contains 12 groups with size varying from 20 jobs and five machines to 500 jobs and 20 machines, where $n \in \{20, 50, 100, 200, 500\}$, $m \in \{5, 10, 20\}$. Each group consists of 10 instances. However, sets 200×5, 500×5 and 500×10 are missing, but they have been added by Pan and Ruiz (2012b) to maintain the orthogonality of the experiment. All implemented algorithms are executed in 10 independent replications for each instance. To make a fair comparison, the same elapsed central processing unit (CPU) time, $t = \rho \times m \times n$ ms, is adopted as the termination criterion, where ρ has three values: 30, 60 and 90.

To evaluate the performance of each algorithm, the average relative percentage deviation (ARPD) (Wang *et al.* 2010) is computed to measure the quality of the solutions. The ARPD is calculated as follows:

$$\text{ARPD} = \frac{1}{10}\sum_{i=1}^{10}\frac{C_i - C_R}{C_R} \times 100 \tag{14}$$

where C_i is the solution obtained by one of the algorithms on a given instance, and C_R is the upper bound from Ribas, Companys, and Tort-Martorell (2011). Since no one has provided the upper bounds for the above three supplementary test sets, the best solutions obtained by the IG of Ribas, Companys, and Tort-Martorell (2011) and implemented by the current authors are used as the upper bounds.

In addition, since the parameters have an important influence on the performance of P-EDA, the design of experiments method (Montgomery 2008) is implemented to determine the best parameters for the proposed algorithm. There are two main parameters for the proposed algorithm, *i.e.* the population size (PS) and the diversity threshold (λ). Interested readers can find the detailed results and analyses of the parameters' calibration in the online supplementary material. The final parameters used in the proposed P-EDA are set as follows: $PS = 50$ and $\lambda = 0.3$.

Table 1. Statistical results of P-EDA-WPT, P-EDA-R and P-EDA.

Instance	P-EDA-WPT	P-EDA-R	P-EDA
20 × 5	0.000	0.018	0.060
20 × 10	0.011	0.012	0.009
20 × 20	0.000	0.008	0.000
50 × 5	0.245	0.274	0.155
50 × 10	0.082	0.263	0.040
50 × 20	0.136	0.293	0.104
100 × 5	−0.076	−0.006	−0.647
100 × 10	−0.482	−0.329	−0.800
100 × 20	−0.467	−0.254	−0.616
200 × 5	0.571	0.539	−1.294
200 × 10	0.221	0.502	−0.939
200 × 20	−0.660	−0.267	−1.281
500 × 5	1.796	2.066	−2.115
500 × 10	0.062	0.391	−3.046
500 × 20	−0.498	0.040	−2.583
Average	0.063	0.237	**−0.864**

Note: P-EDA = estimation of distribution algorithm with path relinking; P-EDA-WPT = P-EDA with Wang, Pan and Tasgetiren heuristic; P-EDA-R = P-EDA without any heuristics.

4.2. Effectiveness of population initialization

The PF-NEH heuristic plays an important role in guiding the P-EDA to quickly converge to the promising area. Therefore, to confirm the contribution of the PF-NEH heuristic for P-EDA, the following experiments are carried out. In the first trial, another constructive heuristic, *i.e.* NEH_WPT, proposed by Wang, Pan, and Tasgetiren (2011), is used to replace the PF-NEH heuristic to generate an initial solution and other solutions are randomly generated in the search space. Compared with the original NEH, the NEH_WPT adopts the non-decreasing sums of their process time to sort the jobs; this is because jobs with a higher total processing time may block the successive jobs and yield a longer blocking time than jobs with less total processing time for BFSP. P-EDA combined with the NEH_WPT heuristic is referred to as P-EDA-WPT. In the second trial, P-EDA without any heuristics (P-EDA-R) is executed. Lastly, the complete P-EDA is executed. The termination time is set to $\rho = 30$. Each algorithm is executed 10 times for each instance. The statistical results are reported in Table 1.

As can be seen from Table 1, P-EDA outperforms P-EDA-R and P-EDA-WPT, which demonstrates that PF-NEH can effectively enhance the performance of the proposed P-EDA. Moreover, the results of P-EDA are better than for P-EDA-WPT, which reveals that PF-NEH can provide better starting points for the proposed P-EDA in comparison with NEH_WPT. This is because PF-NEH generates more than one excellent initial solutions, which can effectively guide the P-EDA to quickly converge to the promising area. NEH_WPT generates only one solution, which has a limited contribution to enhancing the convergence speed of P-EDA. Therefore, through the PF-NEH heuristic and random method, an initial population with high quality and diversity is obtained.

4.3. Effectiveness of path relinking

To investigate the effectiveness of path relinking for EDA, a computational experiment is designed as follows. In the first trial, P-EDA with path relinking is executed. In the second trial, P-EDA is executed without path relinking. By comparing the experimental results of these two trials, the contribution of path relinking can be clearly confirmed. The termination criterion for these two algorithms is identically set to $\rho = 30$. Each algorithm is run 10 times for each instance. Table 2 reports the comparison results of these two trials.

Table 2. Statistical results of P-EDA and EDA.

Instance	P-EDA	EDA
20 × 5	0.060	0.121
20 × 10	0.009	0.072
20 × 20	0.000	0.040
50 × 5	0.155	2.723
50 × 10	0.040	2.542
50 × 20	0.104	2.093
100 × 5	−0.647	1.425
100 × 10	−0.800	1.847
100 × 20	−0.616	2.367
200 × 5	−1.294	−0.283
200 × 10	−0.939	0.475
200 × 20	−1.281	0.527
500 × 5	−2.115	−1.550
500 × 10	−3.046	−2.407
500 × 20	−2.583	−1.664
Average	**−0.864**	0.555

Note: P-EDA = estimation of distribution algorithm with path relinking; EDA = estimation of distribution algorithm.

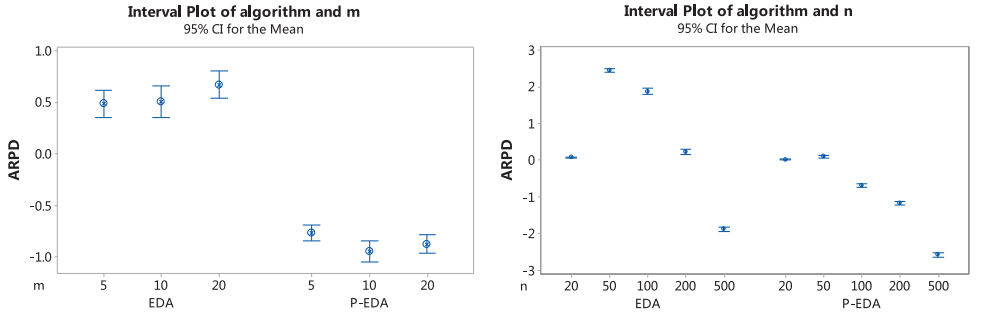

Figure 3. 95% confidence interval (CI) plot of average relative percentage deviation (ARPD) by algorithms *n* and *m*. EDA = estimation of distribution algorithm; P-EDA = proposed estimation of distribution algorithm.

As can be seen from Table 2, P-EDA obtains a smaller ARPD value of –0.864 compared to the corresponding value of 0.555 for EDA. It can also be seen from Table 2 that the performances of EDA and P-EDA gradually improve with increasing instance size. To illustrate this case, the interactions between *m* and *n* with EDA and P-EDA are shown in Figure 3. Notice in this figure that P-EDA is less influenced by *n*, while the behaviour of EDA is significantly different with respect to *n*. Especially for the instances with 50 and 100 machines, EDA performs relatively poorly. It can also be observed that P-EDA and EDA are both less influenced by *m*, but P-EDA performs slightly better when *m* increases. Therefore, these results demonstrate that path relinking makes a significant contribution to the effectiveness of EDA, especially for large instances.

4.4. Effectiveness of local search

To test the effectiveness of the proposed local search method in P-EDA, three variants of P-EDA are compared, whose abbreviations are defined as follows: (1) P-EDA represents P-EDA with the mRLS; (2) P-EDA/RLS represents P-EDA with the original RLS; and (3) P-EDA/NOL represents P-EDA without any local searches. From the experimental results of these three trials, two facts about the local search for P-EDA can be confirmed. One is the contribution of the local search, and the other is

Table 3. Statistical results of P-EDA, P-EDA/RLS and P-EDA/NOL.

Instance	P-EDA	P-EDA/RLS	P-EDA/NOL
20 × 5	0.060	0.363	2.644
20 × 10	0.009	0.295	2.442
20 × 20	0.000	0.181	1.658
50 × 5	0.155	0.644	2.885
50 × 10	0.040	0.496	3.073
50 × 20	0.104	0.593	3.989
100 × 5	−0.647	−0.338	1.030
100 × 10	−0.800	−0.535	1.472
100 × 20	−0.616	−0.373	2.724
200 × 5	−1.294	−1.138	−0.375
200 × 10	−0.939	−0.815	0.387
200 × 20	−1.281	−1.119	0.468
500 × 5	−2.115	−2.024	−1.568
500 × 10	−3.046	−2.961	−2.413
500 × 20	−2.583	−2.501	−1.673
Average	**−0.864**	−0.616	1.116

Note: P-EDA = estimation of distribution algorithm with path relinking; P-EDA/RLS = P-EDA with the original referenced local search; p-EDA/NOL = P-EDA without any local searches.

that the mRLS outperforms the original RLS. These three variants of P-EDA are tested independently with a termination criterion of $\rho = 30$. The results are reported in Table 3.

In Table 3, it can be observed that P-EDA achieves a smaller ARPD than P-EDA/RLS and P-EDA/NOL, while P-EDA/RLS outperforms P-EDA/NOL. It can be demonstrated that the local search is very important for the performance of P-EDA. Moreover, it also reveals that mRLS is more suitable for P-EDA than the original RLS. In particular, P-EDA/RLS and P-EDA/NOL give the worst performance on the small-scale instances (*i.e.* $n = 20$ and 50). The P-EDA without local search only has a capability for global search and its exploitation is relatively poor. If a solution is trapped in local optima, P-EDA without local search finds it very difficult to escape from local optima. The mRLS enriches the search directions by means of the random trajectory and compensates for the poor exploitation of P-EDA. Thus, it can be concluded that mRLS makes a significant contribution to the effectiveness of P-EDA.

4.5. Effectiveness of the diversity-maintaining scheme

The diversity-maintaining scheme can effectively restart the algorithm. Hence, the function of the adopted diversity-maintaining scheme is verified through experiments. In the experiments, P-EDA is executed with and without the diversity-maintaining scheme (*i.e.* P-EDA and P-EDA-ND, respectively). The termination time is set to $\rho = 30$. Each algorithm is executed 10 times for each instance. The statistical results are given in Table 4.

It can be seen from Table 4 that P-EDA with a diversity-maintaining scheme is better than P-EDA without a diversity-maintaining scheme for all scale instances. The superiorities of the diversity-maintaining scheme can be contributed to using three different strategies to build a new population with high quality and diversity. Meanwhile, considering the diversity of the population after some iterations can further save computational time. Therefore, the adopted diversity-maintaining mechanism can effectively ensure that the population has good diversity.

4.6. Comparison of existing algorithms

To verify the effectiveness and efficiency of the proposed P-EDA in searching for better solutions, it was compared with some well-performing algorithms for BFSP with the makespan criterion. These algorithms are HDDE (Wang *et al.* 2010), hmgHS (Wang, Pan, and Tasgetiren 2011), IABC (Han *et al.*

Table 4. Statistical results of P-EDA-ND and P-EDA.

Instance	P-EDA-ND	P-EDA
20 × 5	0.139	0.060
20 × 10	0.038	0.009
20 × 20	0.020	0.000
50 × 5	0.519	0.155
50 × 10	0.393	0.040
50 × 20	0.462	0.104
100 × 5	−0.382	−0.647
100 × 10	−0.509	−0.800
100 × 20	−0.287	−0.616
200 × 5	−1.122	−1.294
200 × 10	−0.750	−0.939
200 × 20	−1.052	−1.281
500 × 5	−2.024	−2.115
500 × 10	−2.963	−3.046
500 × 20	−2.495	−2.583
Average	−0.668	−**0.864**

Note: P-EDA = estimation of distribution algorithm with path relinking; P-EDA-ND = P-EDA without diversity-maintaining scheme.

2012), DE-ABC (Han, Gong, and Sun 2014), MFFO (Han *et al.* 2016), IG (Ribas, Companys, and Tort-Martorell 2011), TPA (Wang *et al.* 2012), RAIS (Lin and Ying 2013), serial variable neighbourhood search (SVNS) (Ribas, Companys, and Tort-Martorell 2013), MA (Pan *et al.* 2013) and iterated greedy algorithm with blocking properties (B-IG) (Ding *et al.* 2016). It is should be noted that since these algorithms were encoded in different programming languages and executed on different computer platforms, and the reported results came from different upper bounds, it is unreasonable to directly compare them according to their reported results in the literature. Thus, to make a fair and reasonable comparison, all compared algorithms are reimplemented with the same programming language and executed in the same computer environment. All algorithms have the same CPU power and time available. To test the efficiency of all the algorithms, the termination criterion ρ has three values: 30, 60 and 90. This termination criterion has been used in recent literature on scheduling (Vallada and Ruiz 2010; Ribas, Companys, and Tort-Martorell 2015; Pan and Ruiz 2012b). As suggested in Ribas, Companys, and Tort-Martorell (2013), the parameters of the SVNS are taken as $\beta = 0.5$, $\alpha = 0.75$ and $ds = 8$. In addition, the PF-NEH heuristic is used to generate the initial solution of SVNS. For the other algorithms, the parameters and initialization procedure are in accordance with the original articles. The results are shown in Tables 5–7, with the minimum ARPD values in bold.

As can be seen from Table 5, where $\rho = 30$, the total ARPD obtained by the P-EDA is −0.864, which is superior to the corresponding values obtained by other algorithms. The P-EDA outperforms the other algorithms for most instance sizes. MA obtains the best performance for the instance sizes 50 × 5, 50 × 10 and 50 × 20. B-IG achieves the best values on the instance sizes 100 × 20 and 200 × 20. Regarding the two improved ABC algorithms, the results of DE-ABC are better than IABC. In addition, P-EDA achieves the best performance on the large-scale instances (500 machines). For $\rho = 60$, it can be found from Table 6 that the results of P-EDA are much lower than those of the other algorithms. For the instance sizes 50 × 5, 50 × 10 and 50 × 20, MA also has the best performance and this is similar with $\rho = 30$. In Table 7, where $\rho = 90$, P-EDA is also substantially better than the other algorithms for most instance sizes. For the small-scale instances (20 machines), SVNS, B-IG, MA and P-EDA obtain the optimal solutions. It can also be found from these tables that the performance of RAIS, SVNS and B-IG is largely improved with additional CPU time, but this is not enough to compete with the proposed algorithm. Therefore, it can be concluded from the above results that the proposed P-EDA is better than the other algorithms for solving the BFSP with the makespan criterion.

To check whether the differences observed in Tables 5–7 are significant, a multi-factor analysis of variance (ANOVA) is carried out, where the number of jobs n, the number of machines m, the

Table 5. Statistical results of the compared algorithms ($\rho = 30$).

Instance	HDDE	hmgHS	IABC	DE-ABC	MFFO	IG	TPA	RAIS	SVNS	MA	B-IG	P-EDA
20 × 5	0.003	0.113	0.049	0.384	0.005	0.101	0.195	0.211	0.069	**0.000**	0.027	**0.000**
20 × 10	0.013	0.052	0.041	0.017	0.007	0.145	0.106	0.074	0.034	0.009	0.001	**0.000**
20 × 20	0.001	0.019	0.020	0.012	0.003	0.092	0.040	0.033	0.003	**0.000**	0.001	**0.000**
50 × 5	0.581	0.597	1.433	1.751	0.705	1.213	1.041	0.527	0.196	**0.137**	0.309	0.155
50 × 10	0.313	0.454	1.221	1.428	0.395	1.339	0.688	0.521	0.296	**−0.040**	0.090	0.040
50 × 20	0.280	0.413	0.995	1.165	0.342	1.344	0.467	1.591	0.561	**0.078**	0.171	0.104
100 × 5	1.455	0.868	2.270	2.272	0.916	1.040	0.603	0.264	−0.609	−0.266	0.429	**−0.647**
100 × 10	0.732	0.321	1.740	1.514	0.135	1.140	0.118	0.863	−0.465	−0.648	−0.471	**−0.800**
100 × 20	0.565	0.271	1.326	1.177	0.016	1.185	−0.129	2.534	0.120	−0.603	**−0.814**	−0.616
200 × 5	2.512	2.075	3.360	2.526	1.767	0.829	0.410	0.712	−0.247	−0.830	0.077	**−1.294**
200 × 10	2.059	1.735	2.774	2.117	1.180	2.000	0.484	2.692	−0.742	−0.644	−0.346	**−0.939**
200 × 20	0.940	0.663	1.568	1.012	−0.028	1.300	−0.427	3.579	−0.857	−1.197	**−1.450**	−1.281
500 × 5	3.484	3.141	2.382	2.852	2.548	0.925	0.662	2.708	−0.329	−1.801	0.103	**−2.115**
500 × 10	1.817	1.632	2.115	1.442	1.073	1.144	−0.495	4.126	−0.298	−2.783	−1.522	**−3.046**
500 × 20	1.169	1.027	1.631	0.761	0.038	1.020	−0.654	4.412	−2.425	−2.491	−1.895	**−2.583**
Average	1.062	0.892	1.528	1.362	0.607	0.988	0.207	1.656	−0.313	−0.738	−0.353	**−0.864**

Note: HDDE = hybrid discrete differential evolution; hmgHS = hybrid modified global-best harmony search; IABC = improved artificial bee colony; DE-ABC = differential evolution–artificial bee colony; MFFO = modified fruit fly optimization; IG = iterated greedy algorithm; TPA = three-phase algorithm; RAIS = revised artificial immune system; SVNS = serial variable neighbourhood search; MA = mimetic algorithm; B-IG = iterated greedy algorithm with blocking properties; P-EDA = estimation of distribution algorithm with path relinking.

Table 6. Statistical results of the compared algorithms ($\rho = 60$).

Instance	HDDE	hmgHS	IABC	DE-ABC	MFFO	IG	TPA	RAIS	SVNS	MA	B-IG	P-EDA
20 × 5	**0.000**	0.118	0.020	0.013	0.002	0.039	0.147	0.155	0.050	**0.000**	0.030	**0.000**
20 × 10	0.005	0.034	0.019	0.008	0.003	0.092	0.123	0.071	0.007	0.007	0.002	**0.000**
20 × 20	0.001	0.010	0.010	0.006	**0.000**	0.057	0.023	0.015	0.008	**0.000**	**0.000**	**0.000**
50 × 5	0.288	0.398	1.290	1.652	0.563	1.026	0.994	0.526	0.147	**−0.018**	0.169	0.050
50 × 10	0.113	0.385	1.057	1.323	0.284	1.170	0.578	0.285	0.120	**−0.102**	−0.018	−0.037
50 × 20	0.134	0.329	0.896	1.032	0.232	1.234	0.459	0.368	0.497	**−0.061**	0.023	0.136
100 × 5	1.124	0.586	2.151	2.140	0.682	0.830	0.631	0.051	−0.733	−0.553	0.021	**−0.813**
100 × 10	0.457	0.096	1.544	1.401	−0.025	0.894	0.161	−0.045	−0.638	−0.921	−0.792	**−0.951**
100 × 20	0.293	−0.019	1.192	1.067	−0.161	1.011	−0.139	0.660	−0.134	−0.681	**−1.043**	−0.695
200 × 5	2.262	1.763	3.325	2.422	1.434	0.486	0.423	0.030	−0.545	−0.960	−0.258	**−1.421**
200 × 10	1.786	1.479	2.674	2.019	0.810	1.708	0.414	0.980	−0.850	−0.831	−0.635	**−1.070**
200 × 20	0.715	0.440	1.506	0.897	−0.262	1.009	−0.538	1.304	−0.986	−1.448	**−1.732**	−1.442
500 × 5	3.293	2.987	2.378	2.789	2.273	0.729	0.613	1.215	−0.560	−1.892	−0.303	**−2.193**
500 × 10	1.612	1.508	2.134	1.218	0.434	0.924	−0.599	2.049	−0.568	−2.851	−1.939	**−3.118**
500 × 20	0.951	0.901	1.555	0.622	−0.243	1.106	−0.796	2.285	−2.483	−2.589	−2.204	**−2.669**
Average	0.869	0.734	1.450	1.240	0.402	0.821	0.166	0.663	−0.444	−0.873	−0.581	**−0.948**

Note: For full names of algorithms, see footnote to Table 5.

elapsed CPU time t and *algorithm* are considered as factors. Table 8 summarizes the significance of these factors and their interactions. It can be seen from this table that these considered factors and their interactions are highly significant (p value < 0.05). The ANOVA hypotheses are tested by a residual analysis, which shows small departures from normality. Therefore, the results obtained in the ANOVA validate the above conclusions. Figure 4 depicts a 95% confidence interval plot of the interaction between t and all the compared algorithms. It can be seen from this figure that the total ARPD of almost all algorithms declines with increasing elapsed CPU time, except for TPA. The proposed P-EDA statistically outperforms the other algorithms under the corresponding termination criterion. The total ARPD of TPA when $\rho = 90$ is slightly larger than when $\rho = 60$, which indicates that TPA has been convergent and that additional CPU time does not translate into much better results. RAIS is more influenced by t; this may be because if contains no effective local search, so it requires more time to obtain better solutions.

Table 7. Statistical results of the compared algorithms ($\rho = 90$).

Instance	HDDE	hmgHS	IABC	DE-ABC	MFFO	IG	TPA	RAIS	SVNS	MA	B-IG	P-EDA
20 × 5	**0.000**	0.047	0.003	0.002	**0.000**	0.018	0.175	0.191	**0.000**	**0.000**	**0.000**	**0.000**
20 × 10	0.008	0.023	0.012	0.006	0.002	0.060	0.120	0.073	**0.000**	**0.000**	**0.000**	**0.000**
20 × 20	**0.000**	0.008	0.007	0.001	0.001	0.045	0.036	0.012	**0.000**	**0.000**	**0.000**	**0.000**
50 × 5	0.229	0.416	1.205	1.587	0.447	0.966	1.026	0.588	0.097	**−0.095**	0.046	0.054
50 × 10	0.059	0.287	0.993	1.226	0.218	1.091	0.671	0.258	0.126	**−0.144**	−0.090	**−0.067**
50 × 20	0.106	0.296	0.852	0.981	0.222	1.172	0.433	0.263	0.441	**−0.125**	−0.049	0.092
100 × 5	0.944	0.437	2.084	2.031	0.393	0.706	0.689	0.027	−0.817	−0.613	−0.082	**−0.858**
100 × 10	0.321	0.043	1.514	1.286	−0.173	0.759	0.115	−0.186	−0.758	−1.043	−0.917	**−1.107**
100 × 20	0.207	0.042	1.147	0.987	−0.230	0.856	−0.154	0.002	−0.247	−1.014	**−1.221**	−0.767
200 × 5	2.092	1.722	3.396	2.360	1.147	0.358	0.422	−0.102	−0.736	−1.022	−0.480	**−1.494**
200 × 10	1.649	1.368	2.620	1.945	0.630	1.399	0.427	0.514	−0.904	−0.822	−0.936	**−1.169**
200 × 20	0.574	0.376	1.420	0.843	−0.407	0.876	−0.522	0.361	−1.110	−1.416	**−1.904**	−1.543
500 × 5	3.147	2.865	2.386	2.704	1.972	0.436	0.575	0.610	−0.701	−1.902	−0.208	**−2.227**
500 × 10	1.458	1.396	2.130	1.128	0.214	0.611	−0.632	1.187	−0.822	−2.711	−2.012	**−3.165**
500 × 20	0.869	0.833	1.526	0.528	−0.414	0.980	−0.826	1.356	−2.529	−2.650	−2.445	**−2.710**
Average	0.778	0.677	1.420	1.174	0.268	0.689	0.170	0.344	−0.528	−0.903	−0.686	−0.997

Note: For full names of algorithms, see footnote to Table 5.

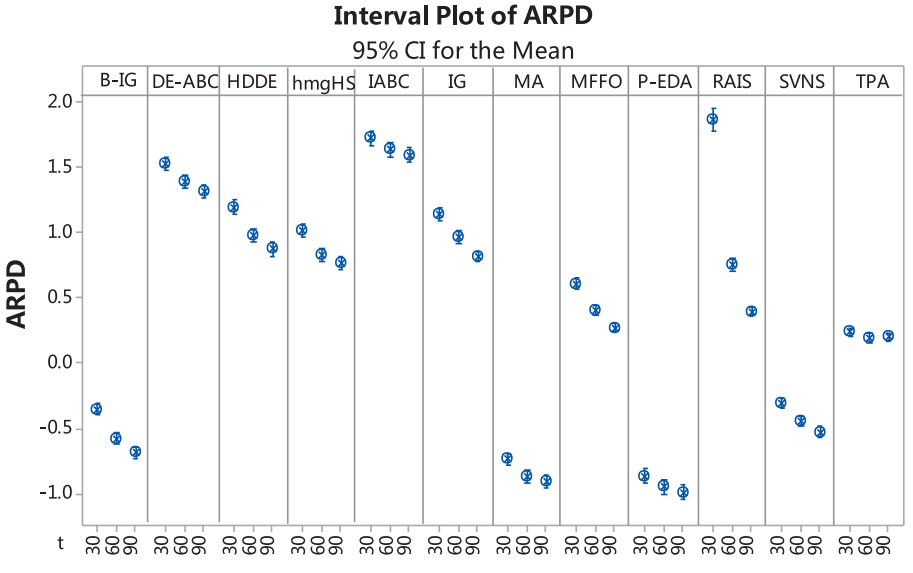

Figure 4. 95% confidence interval (CI) plot of average relative percentage deviation (ARPD) by all compared algorithms and *t*.

Figures 5 and 6 investigate the interactions between *m* and *n* with the compared algorithms. As can be seen from Figure 5, SVNS, MA, RAIS and P-EDA are less influenced by *m*, while the other algorithms have different performances in respect to the *m* values. It can be observed from Figure 6 that most of algorithms have a progressive reduction in the performance with increasing *n* value, except for B-IG, MA, SVNS and P-EDA. As *n* increases, the performances of these four algorithms grow better and better. In particular, P-EDA and MA have good performance on the large-scale instances. The convergence ability is also very important for the proposed algorithm, and some convergence curves and relative analyses are shown in the online supplementary material.

Although all the explanations and details given in the original articles were strictly followed to implement these compared algorithms and closely reproduce the published results, the results obtained here still deviate from the original results since some implemented details were not mentioned in the original articles. Hence, to make the results of the comparison more convincing, the results of P-EDA are also compared with the results reported in the literature. These compared

Table 8. ANOVA results for the comparison of algorithms.

Source	Degrees of freedom	Sum of squares	Mean square	*F* ratio	*p* Value
Mean effects					
n	4	2,297.4	574.34	2,468.73	0
m	2	2,774.6	1,387.31	5,963.14	0
algorithm	8	39,474.2	3,588.56	15,424.9	0
t	2	1,012.0	506.01	2,175.00	0
Interactions					
*n*m*	8	2,618.7	327.33	1,406.99	0
*n*algorithm*	32	25,509.9	579.77	2,492.06	0
*n*t*	8	302.8	37.85	162.69	0
m algorithm*	16	3,456.2	157.10	675.27	0
*m*t*	4	12.5	3.13	13.45	0
*t*algorithm*	22	1,257.0	57.14	245.60	0
Error	53,869	12,532.5	0.23		
Total	53,996	91,257.2			

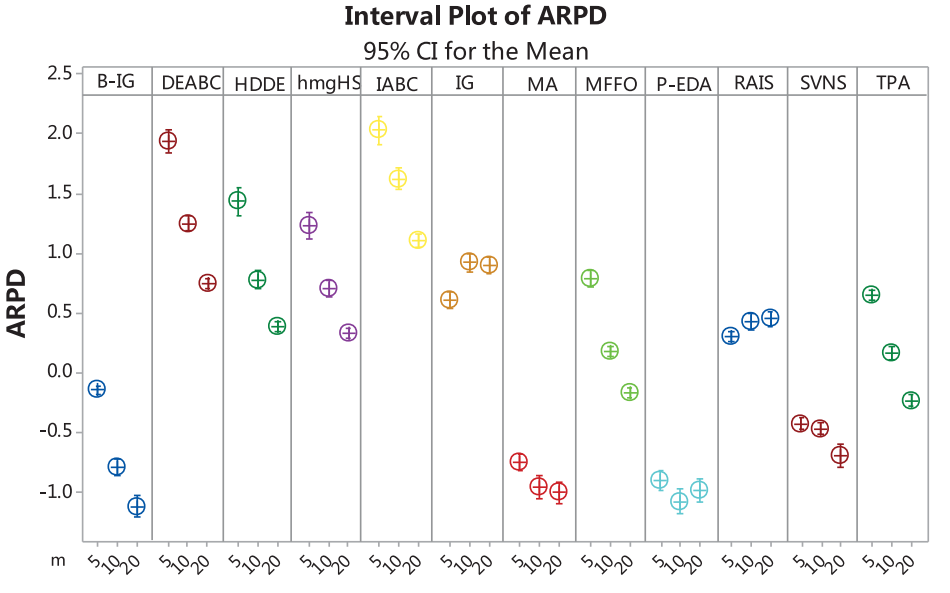

Figure 5. 95% confidence interval (CI) plot of average relative percentage deviation (ARPD) by all compared algorithms and *m*.

algorithms adopt the same performance evaluation method, *i.e.* Equation (14) and test instances. Test instances come from standard Taillard's benchmarks, which exclude the instance sizes 200 × 5, 500 × 10 and 500 × 5. These compared algorithms are hmgHS, IG, RAIS, TPA, HDDE, SVNS_D, MA and B-IG. The results of these algorithms were obtained from the relevant literature. The termination criterion of P-EDA is set to $\rho = 100$. Each instance is independently executed in 10 replications. The comparison results are reported in Table 9.

It can be seen from Table 9 that P-EDA achieves the best ARPD among all the compared algorithms. To be specific, for small-scale instances ($n = 20$), HDDE, RAIS, B-IG and P-EDA obtain the best solutions. For the medium-scale instances ($n = 50, 100$), B-IG obtains the best performance, except for 100 × 5. P-EDA outperforms the other algorithms at the considered margin for large-scale instances ($n = 200, 500$). There are three reasons for this. The first is that the PF-NEH heuristic provides several excellent initial solutions. The second is that the mRLS fully exploits the area around a solution by frequently transforming the search path. The third is that EDA quickly explores the promising area in the solution space. This comparison with the results in the literature confirms that

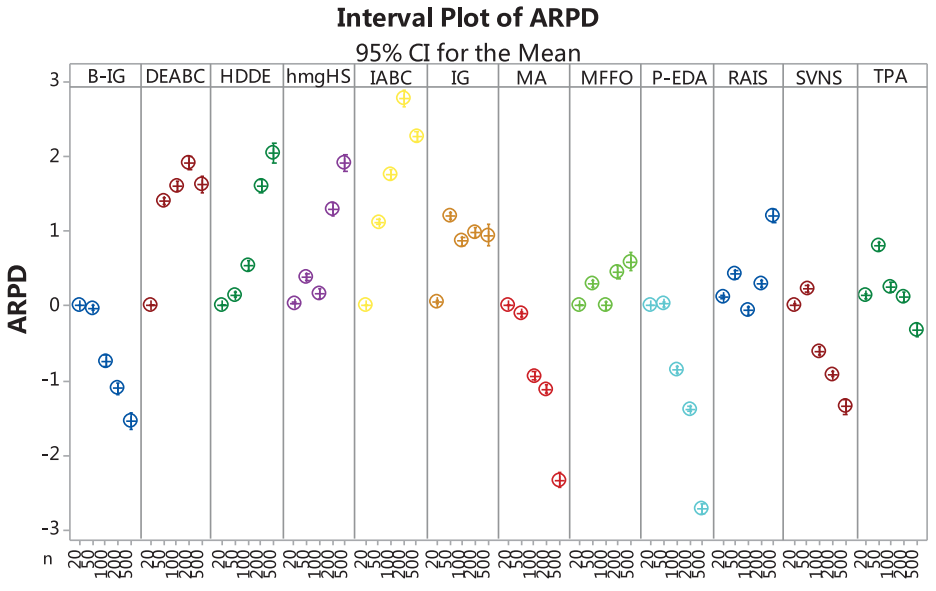

Figure 6. 95% confidence interval (CI) plot of average relative percentage deviation (ARPD) by all compared algorithms and *n*.

Table 9. Comparison results of the algorithms.

Instance	HDDE	hmgHS	IG	RAIS	TPA	SVNS	MA	B-IG	P-EDA
20 × 5	**0.00**	0.09	0.39	**0.00**	0.40	0.16	0.03	**0.00**	**0.00**
20 × 10	**0.00**	0.03	0.48	**0.00**	0.21	0.13	0.01	**0.00**	**0.00**
20 × 20	**0.00**	0.03	0.31	**0.00**	0.03	0.07	0.01	**0.00**	**0.00**
50 × 5	0.11	0.37	2.71	−0.19	1.56	0.77	0.25	**−0.19**	−0.11
50 × 10	0.21	0.49	3.24	−0.20	1.07	0.75	0.18	**−0.42**	−0.24
50 × 20	0.16	0.29	2.88	0.00	0.76	0.70	0.16	**−0.39**	−0.12
100 × 5	0.19	0.80	3.82	−0.82	1.71	−0.14	−0.06	−0.72	**−0.96**
100 × 10	0.65	0.46	3.34	−0.88	0.59	0.14	−0.42	**−1.35**	−1.21
100 × 20	0.56	0.31	0.03	−0.71	0.07	0.44	−0.35	**−1.36**	−1.02
200 × 10	−1.10	1.26	3.85	−0.33	1.22	0.04	−0.38	−1.15	**−1.41**
200 × 20	−0.63	0.37	2.31	−0.63	0.03	−0.27	−0.45	−1.76	**−1.82**
500 × 20	−0.53	0.17	1.32	−0.02	−0.05	−1.74	−2.30	−2.20	**−2.94**
Average	−0.03	0.39	2.31	−0.32	0.63	0.09	−0.28	−0.80	**0.82**

Note: For full names of algorithms, see footnote to Table 5.

the proposed P-EDA is an effective and efficient approach for the BFSP with makespan criterion, especially for the large-scale BFSP.

Since the proposed P-EDA is competitive with other well-performing algorithms, the best solutions obtained by the proposed P-EDA are compared with the results reported in the literature. Some new upper bound solutions have been discovered during the experiments. Table 10 summarizes the comparison results, where the new upper bound solutions obtained by P-EDA are highlighted in bold. Note that the instance sizes of 200 × 20, 500 × 5 and 500 × 10 are not included in Table 10, since the algorithms in the literature do not consider these instances. In Table 10, the 'source' column represents the methods adopted to obtain the best solution. In the source column, 1 indicates HDDE, 2 indicates IABC, 3 indicates DE-ABC, 4 indicates MFFO, 5 indicates IG, 6 indicates TPA, 7 indicates RAIS, 8 indicates SVNS, 9 indicates MA, 10 indicates B-IG and 11 indicates the proposed P-EDA. As shown in Table 10, all of the compared algorithms achieve 30 best values for the small-scale instances, *i.e.* 20 × 5, 20 × 10 and 20 × 20. For the remaining 90 instances, P-EDA provides 59 new unique upper bound solutions, while others come from four of MA, one of MFFO, 14 of SVNS and 12 of B-IG.

Table 10. Upper bound solutions for Tailliard's benchmarks.

Instance	Best	Source	Instance	Best	Source	Instance	Best	Source	Instance	Best	Source
20 × 5			50 × 5			100 × 5			200 × 10		
Ta1	1374	1–11	Ta31	**2984**	11	Ta61	**6073**	11	Ta91	**13,214**	11
Ta2	1408	1–11	Ta32	**3187**	11	Ta62	**5954**	11	Ta92	13,100	8
Ta3	1280	1–11	Ta33	**3000**	11	Ta63	5851	8	Ta93	**13,264**	11
Ta4	1448	1–11	Ta34	3120	4	Ta64	5656	9	Ta94	**13,175**	11
Ta5	1341	1–11	Ta35	3139	8	Ta65	**5909**	11	Ta95	**13,188**	11
Ta6	1363	1–11	Ta36	**3158**	11	Ta66	**5759**	11	Ta96	**12,922**	11
Ta7	1381	1–11	Ta37	3012	10	Ta67	5920	8	Ta97	**13,431**	11
Ta8	1379	1–11	Ta38	**3048**	11	Ta68	5817	8	Ta98	**13,306**	11
Ta9	1373	1–11	Ta39	**2889**	11	Ta69	**6048**	11	Ta99	**13,127**	11
Ta10	1283	1–11	Ta40	3102	10	Ta70	**6071**	11	Ta100	13,202	10
20 × 10			50 × 10			100 × 10			200 × 20		
Ta11	1698	1–11	Ta41	**3614**	11	Ta71	**6926**	11	Ta101	14,192	9
Ta12	1833	1–11	Ta42	3482	8	Ta72	6673	10	Ta102	**14,770**	11
Ta13	1659	1–11	Ta43	3469	8	Ta73	**6813**	11	Ta103	**14,874**	11
Ta14	1535	1–11	Ta44	3650	8	Ta74	**7064**	11	Ta104	**14,808**	11
Ta15	1617	1–11	Ta45	3582	8	Ta75	**6740**	11	Ta105	**14,682**	11
Ta16	1590	1–11	Ta46	**3575**	11	Ta76	**6548**	11	Ta106	**14,765**	11
Ta17	1622	1–11	Ta47	**3669**	11	Ta77	**6715**	11	Ta107	**14,832**	11
Ta18	1731	1–11	Ta48	3555	8	Ta78	**6764**	11	Ta108	**14,836**	11
Ta19	1747	1–11	Ta49	3508	8	Ta79	**6940**	11	Ta109	**14,752**	11
Ta20	1782	1–11	Ta50	**3610**	11	Ta80	6844	10	Ta110	**14,750**	11
20 × 20			50 × 20			100 × 20			500 × 20		
Ta21	2436	1–11	Ta51	4479	9,11	Ta81	7717	10	Ta111	**35,513**	11
Ta22	2234	1–11	Ta52	4262	10,11	Ta82	**7764**	11	Ta112	**35,893**	11
Ta23	2479	1–11	Ta53	4261	9,10,11	Ta83	7734	10	Ta113	**35,566**	11
Ta24	2348	1–11	Ta54	4338	10,11	Ta84	7750	10	Ta114	**35,030**	11
Ta25	2435	1–11	Ta55	**4253**	11	Ta85	**7730**	11	Ta115	**35,574**	11
Ta26	2383	1–11	Ta56	4276	8,10,11	Ta86	**7799**	11	Ta116	**35,877**	11
Ta27	2390	1–11	Ta57	**4291**	11	Ta87	7879	10	Ta117	**35,541**	11
Ta28	2328	1–11	Ta58	4298	8,11	Ta88	7919	10	Ta118	**35,671**	11
Ta29	2363	1–11	Ta59	4304	8	Ta89	**7826**	11	Ta119	**35,421**	11
Ta30	2323	1–11	Ta60	**4399**	11	Ta90	**7863**	11	Ta120	**35,761**	11

These experimental results demonstrate that the proposed P-EDA can generate better results than all of the compared algorithms. Some Gantt charts of the upper bound solutions obtained by P-EDA are shown in the online supplementary material.

5. Conclusion

In this article, an effective EDA, named P-EDA, is proposed for solving the BFSP with the objective of minimizing makespan. In P-EDA, a problem-specific constructive heuristic (PF-NEH) and the random method were combined to initialize the population. A modified linear rank selection was adopted to select superior individuals for EDA. An effective probabilistic model was used to explore the solution space. To improve the convergence property and the blindness of EDA, the path relinking technique was incorporated to guide the search process. A local search method was also introduced to exploit the solution space, and provided an appropriate balance between exploration of the global search and exploitation of the local search. Finally, a diversity-maintaining scheme was used to prevent the deterioration of the population. The influence of parameter settings was investigated using a design of experiments approach. The significant role of path relinking for promoting the performance of EDA was also demonstrated. An extensive comparison of P-EDA against other well-performing algorithms was carried out to validate the performance of P-EDA. The experimental results and statistical analyses showed that P-EDA outperformed the recently published methods by a wide statistical margin. In future research, the P-EDA will be applied to solve the BFSP with the total flow-time criterion, the BFSP with the total tardiness criterion or multi-objective BFSP. With regard

to multi-objective BFSP in particular, to the authors' knowledge, no one has solved this problem so far.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This research was supported by the National Natural Science Foundation of China under [grant number U1433116]; Fundamental Research Funds for the Central Universities under [grant number NP2017208]; Funding of Jiangsu Innovation Program for Graduate Education under [grant number KYLX16_0382]; and Postgraduate Research & Practice Innovation Program of Jiangsu Province [grant number KYCX17_0287].

References

Companys, Ramon, Imma Ribas, and Manel Mateo. 2010. "Note on the Behaviour of an Improvement Heuristic on Permutation and Blocking Flow-Shop Scheduling." *International Journal of Manufacturing Technology & Management* 20 (1-2): 331–357.

Davendra, Donald, and Magdalena Bialic-Davendra. 2013. "Scheduling Flow Shops with Blocking Using a Discrete Self-Organising Migrating Algorithm." *International Journal of Production Research* 51 (8): 2200–2218.

Ding, Jian-Ya, Shiji Song, Jatinder N. D. Gupta, Cheng Wang, Rui Zhang, and Cheng Wu. 2016. "New Block Properties for Flowshop Scheduling with Blocking and their Application in an Iterated Greedy Algorithm." *International Journal of Production Research* 54 (16): 4759–4772.

Glover, Fred, and Manuel Laguna. 2007. *Tabu Search*. Boston: Kluwer Academic Publisher.

Gong, Hua, Lixin Tang, and C. W. Duin. 2010. "A Two-Stage Flow Shop Scheduling Problem on a Batching Machine and a Discrete Machine with Blocking and Shared Setup Times." *Computers & Operations Research* 37 (5): 960–969.

Grabowski, Józef, and Jarosław Pempera. 2007. "The Permutation Flow Shop Problem with Blocking. A Tabu Search Approach." *Omega* 35 (3): 302–311.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. 1979. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey." *Annals of Discrete Mathematics* 5: 287–326.

Hall, Nicholas G., and Chelliah Sriskandarajah. 1996. "A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process." *Operations Research* 44 (3): 510–525.

Han, Yuyan, Dunwei Gong, Yaochu Jin, and Quan-Ke Pan. 2016. "Evolutionary Multi-objective Blocking Lot-Streaming Flow Shop Scheduling with Interval Processing Time." *Applied Soft Computing* 42 (C): 229–245.

Han, Yuyan, Dunwei Gong, Junqing Li, and Yong Zhang. 2016. "Solving the Blocking Flow Shop Scheduling Problem with Makespan Using a Modified Fruit Fly Optimisation Algorithm." *International Journal of Production Research* 54 (22): 1–16.

Han, Yu Yan, Dunwei Gong, and Xiaoyan Sun. 2014. "A Discrete Artificial Bee Colony Algorithm Incorporating Differential Evolution for the Flow-Shop Scheduling Problem with Blocking." *Engineering Optimization* 47 (7): 1–20.

Han, Yu Yan, Quan-Ke Pan, Jun Qing Li, and Hong Yan Sang. 2012. "An Improved Artificial Bee Colony Algorithm for the Blocking Flowshop Scheduling Problem." *The International Journal of Advanced Manufacturing Technology* 60 (9-12): 1149–1159.

Hauschild, Mark, and Martin Pelikan. 2011. "An Introduction and Survey of Estimation of Distribution Algorithms." *Swarm & Evolutionary Computation* 1 (3): 111–128.

Jarboui, Bassem, Mansour Eddaly, Patrick Siarry, and Abdelwaheb Rebaï. 2009. *An Estimation of Distribution Algorithm for Minimizing the Makespan in Blocking Flowshop Scheduling Problems*. Berlin: Springer.

Karthikeyan, S., P. Asokan, S. Nickolas, and Tom Page. 2015. "A Hybrid Discrete Firefly Algorithm for Solving Multi-objective Flexible Job Shop Scheduling Problems." *International Journal of Bio-Inspired Computation* 7 (6): 386–401.

Koren, Yoram, Wencai Wang, and Xi Gu. 2017. "Value Creation through Design for Scalability of Reconfigurable Manufacturing Systems." *International Journal of Production Research* 54 (6): 4759–4772.

Liang, J. J., Quan-Ke Pan, Tiejun Chen, and Ling Wang. 2011. "Solving the Blocking Flow Shop Scheduling Problem by a Dynamic Multi-swarm Particle Swarm Optimizer." *The International Journal of Advanced Manufacturing Technology* 55 (5): 755–762.

Lin, Shih Wei, and Kuo Ching Ying. 2013. "Minimizing Makespan in a Blocking Flowshop Using a Revised Artificial Immune System Algorithm." *Omega* 41 (2): 383–389.

Merchan, Andres F., and Christos T. Maravelias. 2016. "Preprocessing and Tightening Methods for Time-Indexed MIP Chemical Production Scheduling Models." *Computers & Chemical Engineering* 84: 516–535.

Montgomery, Douglas C. 2008. *Design and Analysis of Experiments*. Hoboken, NJ: John Wiley & Sons.

Pan, Quan-Ke, and Rubén Ruiz. 2012a. "An Estimation of Distribution Algorithm for Lot-Streaming Flow Shop Problems with Setup Times." *Omega* 40 (2): 166–180.
Pan, Quan-Ke, and Rubén Ruiz. 2012b. "Local Search Methods for the Flowshop Scheduling Problem with Flowtime Minimization." *European Journal of Operational Research* 222 (1): 31–43.
Pan, Quan-Ke, and Ling Wang. 2012. "Effective Heuristics for the Blocking Flowshop Scheduling Problem with Makespan Minimization." *Omega* 40 (2): 218–229.
Pan, Quan-Ke, Ling Wang, Jun Qing Li, and Jun Hua Duan. 2014. "A Novel Discrete Artificial bee Colony Algorithm for the Hybrid Flowshop Scheduling Problem with Makespan Minimisation." *Omega* 45 (2): 42–56.
Pan, Quan-Ke, Ling Wang, Hong Yan Sang, Jun Qing Li, and Min Liu. 2013. "A High Performing Memetic Algorithm for the Flowshop Scheduling Problem With Blocking." *IEEE Transactions on Automation Science & Engineering* 10 (3): 741–756.
Perez-Rodriguez, Ricardo, and Arturo Hernandez-Aguirre. 2016. "Simulation Optimization for the Vehicle Routing Problem with Time Windows Using a Bayesian Network as a Probability Model." *International Journal of Advanced Manufacturing Technology* 85 (9): 2505–2523.
Qi, Xuewei, Ke Li, and Walter D. Potter. 2016. "Estimation of Distribution Algorithm Enhanced Particle Swarm Optimization for water Distribution Network Optimization." *Frontiers of Environmental Science & Engineering* 10 (2): 341–351.
Ribas, Imma, Ramon Companys, and Xavier Tort-Martorell. 2011. "An Iterated Greedy Algorithm for the Flowshop Scheduling Problem with Blocking." *Omega* 39 (3): 293–301.
Ribas, Imma, Ramon Companys, and Xavier Tort-Martorell. 2013. "A Competitive Variable Neighbourhood Search Algorithm for the Blocking Flow Shop Problem." *European Journal of Industrial Engineering* 7 (6): 729–754.
Ribas, Imma, Ramon Companys, and Xavier Tort-Martorell. 2015. "An Efficient Discrete Artificial Bee Colony Algorithm for the Blocking Flow Shop Problem with Total Flowtime Minimization." *Expert Systems with Applications* 42 (15–16): 6155–6167.
Ronconi, Débora P. 2004. "A Note on Constructive Heuristics for the Flowshop Problem with Blocking." *International Journal of Production Economics* 87 (1): 39–48.
Ronconi, Débora P., and Luís R. S. Henriques. 2009. "Some Heuristic Algorithms for Total Tardiness Minimization in a Flowshop with Blocking." *Omega* 37 (2): 272–281.
Shao, Weishi, and Dechang Pi. 2016. "A Self-Guided Differential Evolution with Neighborhood Search for Permutation Flow Shop Scheduling." *Expert Systems with Applications* 51: 161–176.
Shao, Zhongshi, Dechang Pi, and Weishi Shao. 2017. "An Extended Continuous Estimation of Distribution Algorithm for Solving the Permutation Flow-Shop Scheduling Problem." *Engineering Optimization* 5: 1–22.
Taillard, E. 1993. "Benchmarks for Basic Scheduling Problems." *European Journal of Operational Research* 64 (2): 278–285.
Vallada, Eva, and Rubén Ruiz. 2010. "Genetic Algorithms with Path Relinking for the minimum Tardiness Permutation Flowshop Problem." *Omega* 38 (1-2): 57–67.
Wang, Ling, Quan-Ke Pan, and M. Fatih Tasgetiren. 2010. "Minimizing the Total Flow Time in a Flow Shop with Blocking by Using Hybrid Harmony Search Algorithms." *Expert Systems with Applications* 37 (12): 7929–7936.
Wang, Ling, Quan-Ke Pan, and M. Fatih Tasgetiren. 2011. "A Hybrid Harmony Search Algorithm for the Blocking Permutation Flow Shop Scheduling Problem." *Computers & Industrial Engineering* 61 (1): 76–83.
Wang, Ling, Quan-Ke Pan, P. N. Suganthan, Wen Hong Wang, and Ya Min Wang. 2010. "A Novel Hybrid Discrete Differential Evolution Algorithm for Blocking Flow Shop Scheduling Problems." *Computers & Operations Research* 37 (3): 509–520.
Wang, Cheng, Shiji Song, Jatinder N. D. Gupta, and Cheng Wu. 2012. "A Three-Phase Algorithm for Flowshop Scheduling with Blocking to Minimize Makespan." *Computers & Operations Research* 39 (11): 2880–2887.
Wang, Xianpeng, and Lixin Tang. 2012. "A Discrete Particle Swarm Optimization Algorithm with Self-Adaptive Diversity Control for the Permutation Flowshop Problem with Blocking." *Applied Soft Computing* 12 (2): 652–662.
Wang, J., K. Tang, J. A. Lozano, and X. Yao. 2015. "Estimation of Distribution Algorithm with Stochastic Local Search for Uncertain Capacitated Arc Routing Problems." *IEEE Transactions on Evolutionary Computation* 8 (2): 96–109.
Wang, Hui, Wenjun Wang, Hui Sun, Zhihua Cui, Shahryar Rahnamayan, and Sanyou Zeng. 2017. "A New Cuckoo Search Algorithm with Hybrid Strategies for Flow Shop Scheduling Problems." *Soft Computing* 21 (15): 4297–4307.
Xu, Jianyou, Yunqiang Yin, T. C. E. Cheng, Chin Chia Wu, and Shusheng Gu. 2014. "An Improved Memetic Algorithm Based on a Dynamic Neighbourhood for the Permutation Flowshop Scheduling Problem." *International Journal of Production Research* 52 (4): 1188–1199.
Zhang, Qingfu, and Heinz Mühlenbein. 2004. "On the Convergence of a Class of Estimation of Distribution Algorithms." *IEEE Transactions on Evolutionary Computation* 8 (2): 127–136.