Expert Systems with Applications 38 (2011) 4348–4360

Contents lists available at ScienceDirect

Expert Systems with Applications

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

A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem

Hongcheng Liu [a], Liang Gao [a,*], Quanke Pan [b]

[a] State Key Lab of Digital Manufacturing Equipment and Technology, Department of Industrial and Manufacturing System Engineering, Huazhong University of Science and Technology, 1037 Luoyu Road, 430074 Wuhan, China
[b] College of Computer Science, Liaocheng University, 252059 Liaocheng, China

ARTICLE INFO

Keywords:
PSO
EDA
Metaheuristics
Flowshop

ABSTRACT

In this paper we propose PSO–EDA, a hybrid particle swarm optimization (PSO) with estimation of distribution algorithm (EDA) to solve permutation flowshop scheduling problem (PFSP). PFSP is an NP-complete problem, for which PSO was recently applied. The social cognition in the metaphor of canonical PSO is incomplete, since information conveyed in the *non-gbest* particles is lost. Also, the intelligence of the particles is totally neglected by the canonical PSO and most of other literatures. To tackle such problems, we propose to enable the sharing of information from the collective experience of the swarm by hybridizing an EDA operator with PSO and to add the primitive intelligence to each particle by using a local search mechanism. To enhance the performance of the algorithm proposed, a new local search algorithm, *the minimization-of-waiting-time local search* (MWL), is applied. The computational experiment on different benchmark suites in PFSP, in which two new best known solutions have been found, shows a superiority of PSO–EDA over other counterpart algorithms in terms of accuracy.

1. Introduction

The Flowshop Scheduling Problem (FSP), sequencing n jobs for processing on m machines to meet certain criteria, is an NP-hard optimization problem (Kan, 1976), which involves devotion and efforts of researchers globally and has been discussed in literatures widely. For all the diverse objectives in scheduling varied types of flowshops, the initially stated problem—to minimize the makespan or total flow time in scheduling Permutation Flowshop Scheduling Problem (PFSP)—has not been completely solved. Formerly, exact algorithms, including branch and bound algorithm, dynamic programming, etc., were devised to solve PFSP. These algorithms can accurately work out the optimal solution but they are computationally costly. As the scale of the problem increases to multiple machines processing a comparatively large number of jobs, these exact algorithms could no longer afford the computation efficiently within reasonable time duration. To check such a problem, constructive heuristics were applied, including NEH proposed by Nawaz, Enscore, and Ham (1983), VNS by Hansen and Mladenovíc (1999) and other heuristics by Palmer (1965), Johnson (1954), Framinan, Leisten, and Ruiz-Usano (2002), etc. The constructive heuristic algorithms follow an illuminating methodology, which diminishes the pursuit of the ultimate optimal but seeks to find a good enough answer within a reasonable time. However, these heuristics have their drawbacks: they are not able to continue the search upon becoming trapped in a local optimum (Voß, 2001). To avoid such a defect, another type of methods, the metaheuristics, was devised, establishing a mechanism that guides known heuristics to escape local optimality (Voß, 2001). Simulated annealing (SA), tabu search (TS), ant colony optimization (ACO), genetic algorithms (GA), particle swarm optimization (PSO) are all metaheuristics widely discussed.

Among current popular topics concerning metaheuristics are natural computing paradigms (Banks, Vincent, & Anyakoha, 2007) simulations and animations of natural phenomena. PSO, which was first introduced in 1995 by Kennedy and Eberhart (1995), and which draws universal interests and attention recently, is one of such paradigms. It uses a metaphor of a natural flock or swarm: A swarm is constituted by a number of flying birds. Each bird is called a particle. These particles would adjust their flying direction based on their social cognition and self cognition. The canonical PSO simulates the swarm by calculating the flying velocity of each particle at each iteration according to the location of the global best particle (*gbest*) among the swarm and its personal best location (*pbest*) it *remembers* (Banks et al., 2007). However, this way of simulation is defective since the social cognition of a particle can be not only the ability of learning from the global best, but the ability of exploiting the information in the flying experience of

Abbreviations: MWL, the minimization-of-waiting-time local search; SInfo, swarm information; NEH_insertion, NEH with insertion as its neighborhood; NEH_interchange, NEH with interchange as its neighborhood.

* Corresponding author. Tel.: +86 27 87559419; fax: +86 27 87543074.
E-mail address: gaoliang@mail.hust.edu.cn (L. Gao).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.09.104

all particles in the swarm, one important part of the social cognition of the swarm. Though Wang (2007a) devised an EDA-based PSO to imitate such a part of social cognition, it uses binary encoding and PBIL probabilistic model (which is not what we propose in this paper). Apart from this, few other literatures discussed this part of cognition, nor has any its successful application on PFSP been made. Also, during the iterations of canonical PSO, and of most variants of the PSO in literatures, the intelligence of each particle within the swarm was almost totally neglected. According to Kaewkamnerdpong and Bentley (2005), each of the particles is allowed to observe the local area: If there is a better solution nearby it uses that as its individual best position. This mechanism of the particles, according to our perspective, can reflect the metaphor of the primitive intelligence of each particle. However, the notion of the particle intelligence to enhance the search has not been well discussed.

The canonical PSO, which was originally devised for continuous domain, cannot readily be applied to such discrete problems as FSP (Ucara & Tasgetiren, 2006). To fit into the specific characteristics of the FSP domain, different discrete PSO (DPSO) algorithms were proposed in literatures. A rough classification of the works would reveal a bifurcation with regard to whether the canonical formula of continuous PSO should or should not be maintained in the discrete PSO algorithms. Among the former are Tasgetiren, Sevkli, Liang, and Gencyilmaz (2004) Tasgetiren, Liang, Sevkli, and Gencyilmaz (2007). Tasgetiren et al. (2004) presented SPV (smallest position value) rule based on random key representation (Bean, 1994). The latter involves Rameshkumar et al. (2005), Pan, Wang, Tasgetiren, and Zhao (2008), Lian, Gu, and Jiao (2008), etc. Pan et al. (2008) proposed a discrete PSO by applying crossover and mutation operators of the genetic algorithm (GA), another famous metaheuristics. Similar method was applied in solving PFSP by Lian et al. (2008).

However, a crossover operator of traditional GA has its innate defect. Toussaint (2003) demonstrated that a crossover operator transforms, depending on the crossover mask, mutual information between loci into entropy. In total, it can only decrease such mutual information, and in contrast, as the author proposed, the objective of EDAs is to estimate the correlations between loci and exploit this information during exploration. By his perspective, EDA is a very promising algorithm which adds to the entropy after evaluation of the structure of the mutual information in the parent population, so that the entropy reduces in total. Estimation of distribution algorithm (EDA) was first introduced by PaaG (1996). Larraanaga and Lozano (2002) further discussed its characteristics. Lozano, Larranaga, Inza, and Bengoetxea (2006) recommended to combine local search algorithms with EDA. Santana, Larrañaga, and Lozano (2008) presented different approaches for tuning parameters of EDA adaptively. Jarboui, Eddaly, and Siarry (2009) initiated an EDA algorithm to minimize total flow time in flowshop scheduling.

Seeking to complete the social cognition of PSO, and at the same time, to benefit from the advantageous character of EDA, this paper proposes a new information sharing mechanism within the particle swarm that adds an EDA-based mechanism to the social cognition of each particle to learn from all the other particles in the swarm, defined as the *swarm information* (*SInfo*). Also, a primitive intelligence of the particles in a swarm is employed in this paper for the improvement of the swarm's collective performance.

Moreover, a local search mechanism is applied to further improve the performance of the algorithm. In literatures, a number of heuristics were proposed to be integrated with metaheuristics to enhance the performance of an algorithm. Among the most frequently used local search algorithms is NEH (Nawaz et al., 1983), an algorithm also applied in the hybrid algorithm that this paper proposes. In addition, another local search heuristic algorithm is devised in the paper, which seeks to minimize the waiting time of the machines in the flowshop.

Hybridizations of PSO with other approaches have been discussed by researchers including Liu, Wang, and Jin (2007), Chandrasekaran, Ponnambalam, Suresh, and Vijayakumar (2006), Pan et al. (2008), etc. Iqbal and Oca (2006) hybridized PSO and EDA for solving continuous problems. Wang (2007a) proposed a genetic PSO based on EDA in which PBIL and binary encoding were applied. Also by Wang (2007b) another variant of DPSO based on EDA was presented.

The rest of the paper is organized as follows: in Section 2, PFSP is stated. Section 3 introduces PSO, depicts EDA and presents the hybrid PSO with EDA (PSO–EDA) proposed. Also included in Section 3 are the detailed depiction of swarm information, clarification of the primitive intelligence and proposition on the local search mechanism with a new local search algorithm. The computational results are presented in Section 4 with two new best known solutions found to Watson's benchmark suite, and finally, the paper concludes in Section 5.

2. Permutation flowshop scheduling problem

A flowshop scheduling problem (FSP) can be stated as follows: there are N number of jobs to be scheduled with known processing time on M number of machines with no preemption allowed. The processing times are fixed and are not negative values and every job is available at time zero. At one time, each job can be processed on one machine at most and each machine can process one job at most. Once the procession starts, the sequence of the jobs may not be changed and the procession may not be interrupted. If the sequence of the jobs is identical and unidirectional, or namely, the jobs are processed in the same order on different machines, the problem becomes a permutation flowshop scheduling problem (PFSP). The objective of PFSP is to find a permutation (sequence of the jobs) to achieve certain objectives, such as to minimize total flow time, to minimize makespan, etc. In this paper, our objective is to minimize the makespan.

The following formulas are used to calculate the makespan of a certain permutation with N number of jobs on M number of machines:

$$C_{i,j} = \begin{cases} p_{i,j} & i = j = 1 \\ p_{i,j} + C_{i-1,j} & j = 1,\ i > 1 \\ p_{i,j} + C_{i,j-1} & j > 1,\ i = 1 \\ p_{i,j} + \max\{C_{i-1,j}, C_{i,j-1}\} & i > 1,\ j > 1 \end{cases} \tag{1}$$

$$C_{\max} = C_{M,N} \tag{2}$$

In the formulas, $P_{i,j}$ and $C_{i,j}$ denotes the processing time and completion time of the job assigned in the jth position to be processed on Machine i, respectively, and $C_{\max}$ is the makespan of the permutation.

To evaluate the algorithms devised for PFSP, different benchmark suites of instances are available online. In our computational experiments we use the instances of Watson, Barbulescu, Whitley, and Howe (2002), Reeves and Carlier benchmark suites, which we will present in detail in Section 4.

3. PSO, EDA and PSO–EDA

3.1. PSO

Particle swarm optimization (PSO) has been in existence for roughly a decade, during which it has gained widespread appeal amongst researchers and has been shown to offer good performance in a variety of application domains, with potential for hybridization and specialization (Banks et al., 2007). It is a simple model of social learning whose emergent behavior has found popularity in solving optimization problems (Banks et al., 2007).

The initial metaphor of PSO can be depicted as follows: each of the particles in a swarm has two cognitions—the individual cognition and the social cognition. To imitate the two kinds of cognitions, the following formulas for the update of the particles' positions were devised:

$$V_i^t = wV_i^{t-1} + c_1r_1(P_i^{t-1} - X_i^{t-1}) + c_2r_2(G^{t-1} - X_i^{t-1}) \tag{3}$$

$$X_i^t = X_i^{t-1} + V_i^t \tag{4}$$

where $X_i^t = \{x_{i,1}^{t-1}, x_{i,2}^t, \ldots x_{i,j}^t, \ldots, x_{i,N}^t\}$ denotes the position of the ith particle at the tth iteration and $x_{i,j}^t$ denotes the jth dimension of the position of the ith particle at the tth iteration; $V_i^t = \{v_{i,1}^{t-1}, v_{i,2}^t, \ldots v_{i,j}^t, \ldots, v_{i,N}^t\}$ denotes the velocity of the ith particle at the tth iteration and $v_{i,j}^t$ denotes the jth dimension of the velocity of ith particle at the tth iteration; w is the inertia weight, a parameter tuning the influence of the velocity of a particle at the last iteration; r_1 and r_2 are two random number uniformly distributed between [0,1]; and c_1 and $c_2$are two parameters determining the impact of individual cognition and the social cognition, respectively. With these two kinds of cognitions, every particle can adjust its flying direction by learning from its own experience and the performance of its peers (Banks, Vincent, & Anyakoha, 2008).

However, the social cognition in the canonical PSO is incomplete. The standard formula of canonical PSO enables the particles to learn only from the best of the peers by their social cognition. Considering the information that can be exploited from the other *non-gbest* peers, the social cognition in the canonical PSO is incomplete.

To apply the continuous PSO into the discrete domain of FSP, a recent method is to utilize the crossover and mutation operators of GA for the position update of the particles (Pan et al., 2008). However, a crossover operator is defective. Toussaint (2003) explained that exploration essentially means to add entropy to the search distribution but in the process all the entropy is added at the expense of mutual information from parental generation. The author also demonstrated that crossover destroys mutual information in the parent population by transforming it into entropy in the crossed population.

Seeking to complete social cognition and to avoid the defect of the crossover operator, we propose an application of EDA operator which would be presented in detail in Section 3.3. In addition, the intelligence of the particles, though being primitive and simple, can also adjust the flying direction. Such intelligence has not been well discussed in literatures. We propose to trigger the primitive intelligence of the particles when certain criterion is met, which we will go into details in Section 3.3.

3.2. EDA

Estimation of distribution algorithms (EDA) are evolutionary algorithms based on estimation of and sampling from probabilistic models (Santana, Larrañaga, & Lozano, 2009).

A canonical EDA follows the following iterative steps (Jarboui et al., 2009):

1. Randomly generate a population.
2. Select parent individuals according to the fitness values.
3. Estimate the probability of the distribution of the selected parent individuals according to the probabilistic model devised.
4. Generate new offspring according to the estimated probability.
5. Replace some of the individuals in the previous population by the new offspring.

EDAs are able to overcome some drawbacks exhibited by traditional GA (Santana et al., 2009). In contrast against a crossover operator of GA, EDA can add entropy during exploration while in total the entropy of the swarm will be reduced, because EDAs, except for those that do not estimate correlations, first estimate the structure of the mutual information in the parental generation and then add entropy while respecting that same structure (Toussaint, 2003). Seeking to employ such superiority of EDA, we hybridize PSO with EDA, using an EDA operator for population update in PSO at each iteration.

There are different variants of EDAs varying in probabilistic models, which results in different iterative formulas for updating individuals and which impacts the performance of the EDAs in different problem domains. The EDA operator proposed in our work follows the method discussed by Jarboui et al. (2009) where the author discussed an EDA for minimization of total flow time in flowshop scheduling.

In the following, we would discuss how to apply this method as an operator in our proposed algorithm.

3.3. PSO–EDA

3.3.1. Swarm Information of Proposed PSO–EDA

As discussed in Section 3.1, the metaphor in canonical PSO and in most of the variants of PSO in literatures imitate an incomplete social cognition of a swarm. To enhance the performance of PSO for solving FSP while avoiding the defects of the crossover operators formerly used in the discrete PSO variants, we propose to establish a new information sharing mechanism in the swarm which not only comprise the experience of the global best solution (*gbest*) and of personal best solutions (*pbest*), but of the information from the collective experience of all the particle, which we define as swarm information (*SInfo*) by applying an EDA operator converted from the EDA proposed by Jarboui et al. (2009). The new information sharing mechanism of PSO–EDA is presented as follows.

Following the encoding method of Pan et al. (2008), at tth iteration Particle i is denoted as

$$X(i,t) = \{x(1,t), x(2,t), x(3,t), \ldots, x(j,t), \ldots, x(N,t)\}$$

where $x(j,t)$ denotes that the $x(j,t)$th job is assigned to the jth position. Then at each iteration, each dimension of $X(i,t)$ is calculated as the following expression depicts:

$$x(j,t) = [a \otimes pbest(j,t-1)] \oplus [b \otimes gbest(j,t-1)] \oplus [c \otimes SInfo(j,t-1)] \oplus [d \otimes x(j,t-1)] \tag{5}$$

$$a + b + c + d = 1 \tag{6}$$

The explanation for Expression (5) is as follows:

$a \otimes pbest(j,t-1)$: a is the probability to learn from the best of the particle's individual experience (*pbest*). With a probability of a, the value (the job assigned) on the ith position of the particle is replaced by the value on the ith position of *pbest*.

$b \otimes gbest(j,t-1)$: b is the probability to learn from the best experience of the particles's peers among the swarm (*gbest*). With a probability of b, the value on the ith position of the particle is replaced by the value on the ith position of *gbest*.

$c \otimes SInfo(j,t-1)$: c is the probability to learn from the collective swarm information (*SInfo*). With a probability of c, the value on the ith position of the particle, following the method proposed by Jarboui et al. (2009), is decided as follows:

- First, calculate the fitness value $f(pbest)$ for each particle of the parental generation, update the personal best solutions. $f(pbest) = 1/C(pbest)$, where $C(pbest)$ refers to the makespan of a *pbest*. Sort (or rank) the *pbest* solutions in ascending order according to their fitness.

- Select the particles from all the *pbest* solutions according to the possibility calculated as:

$$prob(r) = \frac{2r}{pop(pop+1)} \tag{7}$$

where *pop* is the total number of *pbest* solutions, which equals to the number of the particles, and *r* is the ranking of the *pbest* solutions.

Denote $\vartheta(k,j)$ as the number of times of appearance of job *k* before or in the position *j* in the subset of the selected δ particles from the *pbest* solutions.

Denote $\mu(k,j-1)$ as the number of times of appearance of job *k* immediately after the job in the position $j-1$ in the subset of the selected ξ particles from *pbest* solutions.

N is the number of the jobs to be scheduled.

The new job on position *j* would be selected according to the probability of $\rho(k,j)$, i.e., $\rho(k,j)$ is the probability of the selection of Job *k* in the *j*th position for Particle *i*. This probability is calculated as follows:

$$\rho(k,j) = \frac{\vartheta(k,j) \times \mu(k,j-1)}{\sum_{k=1}^{N} \vartheta(k,j) \times \mu(k,j-1)} \tag{8}$$

$d \otimes x(j,t-1)$: *d* is similar with the inertia weight of the canonical PSO. With a probability of *d*, the value on *j*th position for Particle *i* will remain unchanged.

In Expression (5), $c \otimes SInfo(j,t-1)$ is the operator that employs the information from collective experience of the swarm (*SInfo*). As for the experiences of *gbest* and *pbest*, the algorithm still employs them using the operators $a \otimes pbest(j,t-1)$ and $b \otimes gbest(j,t-1)$. The tuning of the parameters *a*, *b*, *c* and *d* as well as δ and ξ balance the algorithm between exploration and exploitation.

The implementation of the Expression (5) is as follows:

- Generate a uniformly distributed random number between 0 and 1, denoted as *p*.
- If $1 > p > 1-a$, then the value on the *i*th position of the particle is replaced by the value on the *i*th position of *pbest*.
- If $1-a \geqslant p > 1-a-b$, then the value on the *i*th position of the particle is replaced by the value on the *i*th position of *gbest*.
- If $1-a-b \geqslant p > 1-a-b-c$, then the value on the *i*th position of the particle is decided by *SInfo*.
- If $1-a-b-c \geqslant p > 0$, or namely, $d \geqslant p > 0$, the value on the *i*th position of the particle remains unchanged.

Expression (6) ensures that, for the update of the value on each dimension of each particle, only one operator among the four operators, $a \otimes pbest(j,t-1)$, $b \otimes gbest(j,t-1)$, $c \otimes SInfo(j,t-1)$ and $d \otimes x(j,t-1)$ is performed at each iteration.

Note that we propose to determine the jobs at the positions one by one from the leftmost of the permutation to the rightmost, that is, the job assigned on the first position is decided firstly, and then the job on the second position, and so on. This method would bring infeasible solutions when two or more than two positions of a particle are assigned with the same job. To avoid such a kind of violation, we add two rules to Expression (5):

Additional Rule 1. For operator $c \otimes SInfo(j,t-1)$, jobs that have already been assigned previous to position *j* would not be considered for the assignment on Position *j*—the probability of the selection of these jobs would be set zero. For example, if Job *k* is assigned before Position*j*, then set both $\vartheta(k,j) = 0$ and $\mu(k,j-1) = 0$. And thereafter, $\rho\ (k,j) = 0$. Also, if $\forall i$, $\rho(i,j) = 0$, then set the value of the possibility for all jobs that have not been assigned to a position before Position *j* as $1/n$, where *n* is the number of the jobs that have not been assigned yet.

Additional Rule 2. If $1 > p > 1-a$ and $a \otimes pbest(j,t-1)$ would generate violation in Position *j* or, if $1-a \geqslant p > 1-a-b$ and $b \otimes gbest(j,t-1)$ would generate violation in Position *j*, then the job at Position *j* will be determined by the *SInfo*, in other word, *p* will be assigned a value so that $1-a-b \geqslant p > 1-a-b-c$.

3.4. Local search mechanisms and primitive intelligence

3.4.1. Framework of the local search mechanisms

To enhance the performance of PSO–EDA, we propose to apply three kinds of local search heuristics: NEH with insertion as its neighborhood (denoted as NEH_insertion), NEH with interchange as its neighborhood (denoted as NEH_interchange) and a new local search algorithm we have devised, which we call MWL (refer to Section 3.4.3). We apply the local search mechanism in four stages.

Stage 1: For the initialization of particles, three particles are generated with NEH_insertion, NEH_interchange and MWL, respectively, aiming to lead the swarm at the beginning of the search.

Stage 2: At each iteration, when the criterion, $bias(t) = bias(t-1)$, is met, the primitive intelligence (refer to Section 3.3.2) is triggered, enabling the search for better solution in the local area of *pbest*s. At iteration *t*, $bias(t) = F(gbest) - mean[F(pbest)]$, where $F(gbest)$ is the objective value of the *gbest* solution, and $mean[F(pbest)]$ is the mean objective value of *pbest* solutions.

Stage 3: At the end of each iteration, a local search mechanism with NEH_insertion is performed on a proportion of the particles randomly chosen. The probability for each particle to be chosen is determined by the parameter w_1. (Refer to Section 3.5 for detail.)

3.4.2. Primitive intelligence

The flocking behavior of a swarm has two kinds of intelligence: the collective swarm intelligence and the individual particle intelligence. (The individual particle intelligence discussed in this paper is different from what the canonical PSO seeks to imitate by using *pbests* to remember the best individual experience; we define the individual particle intelligence as the ability of the particles to process the experience remembered.) The former is best depicted by Johnson, "Leaderless individuals using low level rules to achieve higher levels of sophistication" (Banks et al., 2007), which is what different PSO variants try to imitate and simulate. However, as for the latter, most PSO algorithms presented in literatures overlooked it. The individual particle intelligence, though it might be too primitive and simple to direct the whole swarm, could play a role in at lease improving the performance. In the work of Kaewkamnerdpong and Bentley (2005) a mechanism that, in our view point, applies the primitive individual intelligence was proposed which allow each particle to observe the local area. If there is a better solution nearby, it uses that as its individual best position (Banks et al., 2007). However, instead of applying the intelligence to boost the search of the particles the author in this literature proposed to restrict the *flying* area of the particles to avoid being trapped in local optima. According to the best of our knowledge, apart from the work of Kaewkamnerdpong and Bentley (2005) few literatures have discussed the notion of the primitive intelligence of the particles, let alone its successful application to PFSP.

To employ the primitive intelligence of the individual particles, we apply a local search mechanism. At each iteration, if the preset criterion is met, a local search mechanism is triggered to for each *pbest* (the best experience of each particle) to search the local area.

If a better solution is found, the *pbest* will update itself by replacing the original solution with the better solution.

In order to accelerate the convergence of the swarm, we use the expression $bias(t) = bias(t-1)$, as the criterion to trigger the primitive intelligence. $bias(t)$ denotes the difference between the objective value of the *gbest* and the mean objective value of the *pbests* at iteration t. The flow chart for the local search mechanism as the implementation of the primitive intelligence is shown in Fig. 1.

As is shown in Fig. 1, P_1, and P_2 are parameters that determine the probability of the execution of NEH_interchange, and MWL, respectively, in this local search mechanism. The tuning of these parameters affects the performance of the search in exploitation.

3.4.3. The MWL, NEH_insertion and NEH_interchange

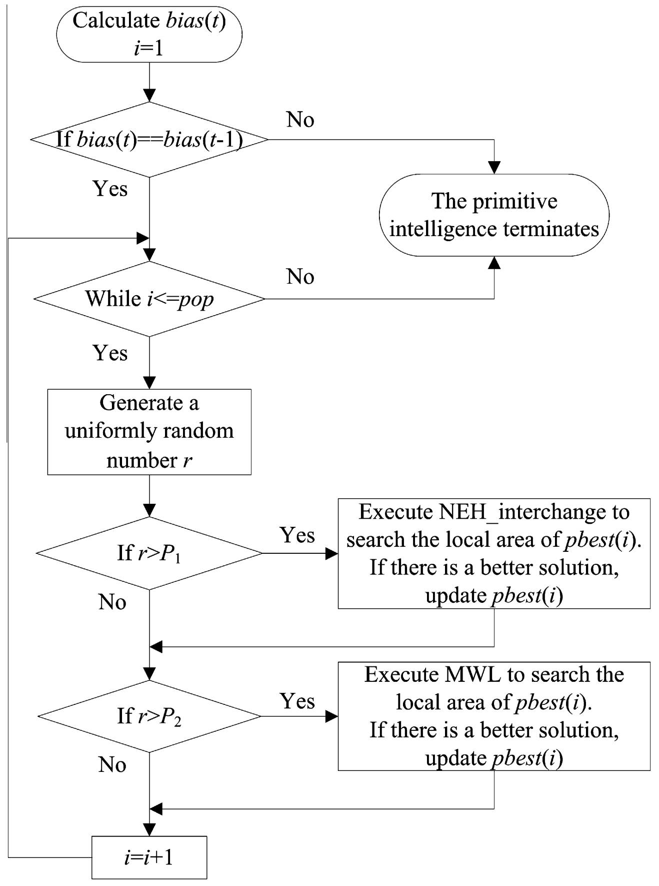
A well-devised local search algorithm can improve the efficiency of the search in exploitation. To enhance the performance of PSO–EDA, we propose a new local search algorithm which we call *the minimization-of-waiting-time local search* (MWL).

The MWL is a heuristics based on **Principle 1** and **Principle 2**:

Principle 1. The less the total waiting time of the machines is, the less the makespan becomes. By reducing the total waiting time of the machines, the makespan can be reduced.

Principle 2. Treat two or more than two subsequent jobs as one unit. The insertion of the unit into different positions in the sequence will not lead to longer total waiting time of the machines in processing the unit consisting the jobs (though the total waiting time of the machines processing all the jobs in the permutation might increase).

Based on these two principles, we devise MWL to search the local area of an initial solution. The MWL is presented as follows:



Calculate $bias(t)$ i=1
If $bias(t)$==$bias(t$-1)
No
Yes
The primitive intelligence terminates
While i<=pop
No
Yes
Generate a uniformly random number r
If r>P_1
Yes
Execute NEH_interchange to search the local area of $pbest(i)$. If there is a better solution, update $pbest(i)$
No
If r>P_2
Yes
Execute MWL to search the local area of $pbest(i)$. If there is a better solution, update $pbest(i)$
No
i=i+1

Fig. 1. The flow chart for the implementation of the primitive intelligence.

Step 1: Randomly generate a sequence $X = \{x_1, x_2, \ldots, x_i, \ldots, x_N\}$, in which N is the number of jobs and $x_i \in \{1, 2, \ldots, N\}$. Let $i = 1$. Then, go to *Step 2*.

Step 2: Choose the job on the x_ith position of the initial solution, and insert the chosen job into different positions. If the new sequence has a better objective value, replace the original sequence with the new one. Repeat this step until no improvement on the objective value can be made. Then, go to *Step 3*.

Step 3: Calculate the total waiting time between the chosen job and the nearest job prior to the chosen one, *WTP* (the sum of the time intervals that each machine waits after the one job prior to the chosen job is processed and before the chosen job to is be processed). And calculate the total waiting time between the chosen job and the nearest one job next to the chosen one, *WTN* (the sum of the time intervals that each machine waits after the chosen job is processed and before the one job next to the chosen one is to be processed).

- Case 1: If $WTP > WTN$, treat the chosen one and the one next to it as one unit;
- Case 2: If $WTP < WTN$, treat the chosen one and the one prior to it as one unit;
- Case 3: If $WTP = WTN$, treat the three as one unit.

Then go to *Step 4*.

Step 4: Insert the unit into different positions, output the sequence with the minimal makespan. Then, calculate the total waiting time between the unit and the one job prior to the unit,*WTP*, and the total waiting time between the unit and the one job next to the unit, *WTN*.

- Case 1: If $WTP > WTN$, treat the unit and the one next to it as a new unit;
- Case 2: If $WTP < WTN$, treat the unit and the one prior to it as a new unit;
- Case 3: If $WTP = WTN$, treat the three as a new unit.

Then go to *Step 5*.

Step 5: Repeat *Step 4* for several times, till the whole sequence becomes one unit. Then, go to *Step 6*.

Step 6: $i = i + 1$, choose the job on x_ith position of the initial solution and repeat all the steps from 2 to 6 until the termination criterion $i > N$ is met.

For example, the Gantt chart for the procession of a permutation consisting of 8 jobs on 3 machines—M1, M2 and M3—is shown in Fig. 2. Let the first chosen job to be Job 5. And Fig. 3 shows the sequence when no further improvement can be made by insertion. Then after the calculation and comparison of *WTP* and*WTN*, Case 3 in *Step 3* occurs. Therefore, as is shown in Fig. 3, Job 1, Job 5, and Job 2 are treated as one unit. Then insert the unit into different positions until no more improvement can be made by insertion and the permutation becomes the one shown in Fig. 4. Since $WTP > WTN$, Case 1 in *Step 4* occurs. Then as is shown in Fig. 5 the new unit consists of Job 1, Job 5, Job 2 and Job 4. Then the algorithm goes to *Step 5* and *Step 6*.

In the proposed PSO–EDA other two local search heuristics are applied—the NEH_insertion and NEH_interchange, which are frequently used in literatures. The pseudo-codes of NEH_insertion and NEH_interchange in this paper are as shown in Figs. 6 and 7.

3.5. Pseudo-code for PSO–EDA

The pseudo-code of the proposed PSO–EDA with swarm information and primitive intelligence is shown in Fig. 8.

As is shown in Fig. 8, the parameters w_1 and w_2 determine the probability for mutation, as well as the *Stage 3* of local search mechanism. The tuning of these two parameters affects the ability of the search to escape local optima. The mutation operator in our

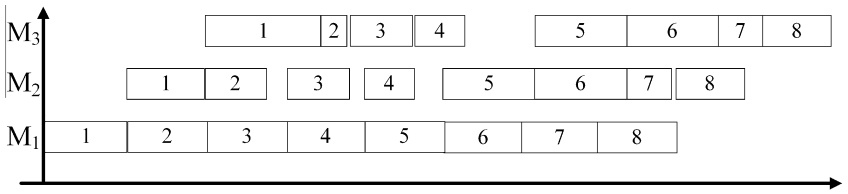


Fig. 2. The original permutation, Job 5 is selected.

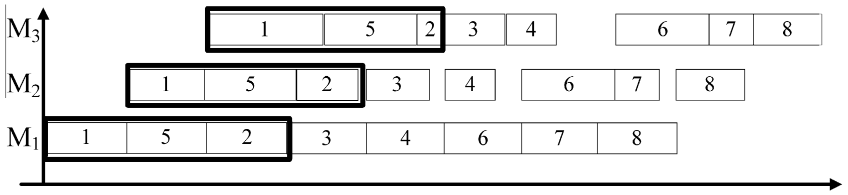


Fig. 3. Job 1, Job 5 and Job 2 together are treated as one unit.

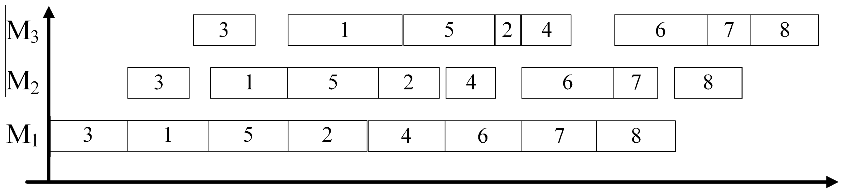


Fig. 4. The unit consisting of Job 1, Job 5 and Job 2 is inserted in the best position in neighborhood.

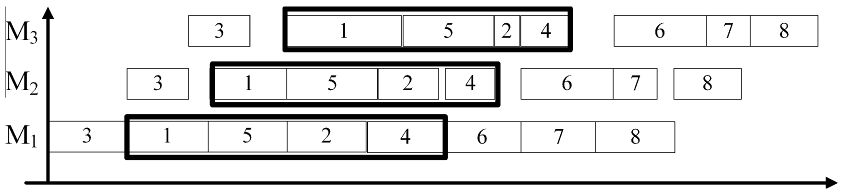


Fig. 5. A new unit consisting of Job 1, Job 5, Job2 and Job 4 is selected.

```
Function NEH_insertion (x)
$x_{best}$=x;
i=1;
For i=1:N
{
    j=1;
    For j=1:N
    {
        x'=insert the jobs in the Position i to Position j in $x_{best}$;
        If (makespan(x')<makespan($x_{best}$))
            { $x_{best}$= x';}
        j=j+1;
    }
}
Return $x_{best}$;
```

Fig. 6. The pseudo-code of NEH_insertion.

```
Function NEH_ interchange (x)
$x_{best}$=x;
i=1;
For i=1:N
{
    j=1;
    For j=1:N
    {
        x'=interchange the jobs in Position i and in Position j in $x_{best}$;
        If (makespan(x')<makespan($x_{best}$))
            { $x_{best}$= x';}
        j=j+1;
    }
}
Return $x_{best}$;
```

Fig. 7. The pseudo-code of NEH_interchange.

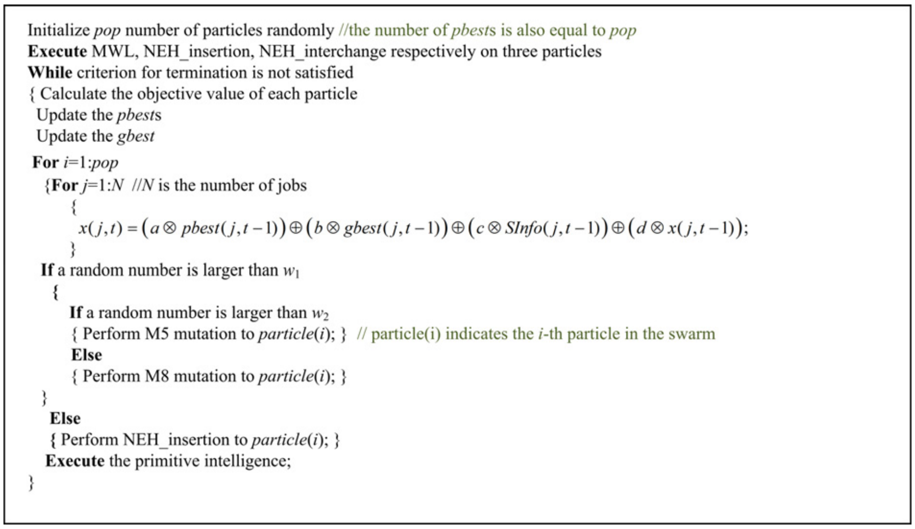
proposed algorithm follows the M5 and M8 mutation operators by 18 (Lian et al., 2008).

4. Experimental results

4.1. New best known solutions of Watson's benchmark suite

To test the efficiency and effectiveness of the proposed PSO–EDA for solving PFSP to minimize makespan, a computational experiment has been made in which PSO–EDA was implemented on Matlab 7.4 platform. We used 10 instances of Watson's benchmark suite (Tasgetiren et al., 2007; Watson et al., 2002) and two new best known solutions have been found during the experiment. We chose 7 job-correlated problems with $\alpha = 0.0$, two random problems and one narrow random problems. $\alpha \in \{0.0, 0.1, 0.2, \ldots, 1.0\}$ indicates the level of difficulty of the job-correlated problems, the smaller the value, the higher the difficulty level.

In the experiment, the parameters are set as follows:



```
Initialize pop number of particles randomly //the number of pbests is also equal to pop
Execute MWL, NEH_insertion, NEH_interchange respectively on three particles
While criterion for termination is not satisfied
{ Calculate the objective value of each particle
  Update the pbests
  Update the gbest
 For i=1:pop
   {For j=1:N  //N is the number of jobs
        {
         $x(j,t) = (a \otimes pbest(j,t-1)) \oplus (b \otimes gbest(j,t-1)) \oplus (c \otimes SInfo(j,t-1)) \oplus (d \otimes x(j,t-1))$;
        }
   If a random number is larger than $w_1$
    {
       If a random number is larger than $w_2$
       { Perform M5 mutation to particle(i); }  // particle(i) indicates the i-th particle in the swarm
       Else
       { Perform M8 mutation to particle(i); }
    }
    Else
    { Perform NEH_insertion to particle(i); }
   Execute the primitive intelligence;
}
```

Fig. 8. The Pseudo-code of the proposed PSO–EDA.

The number of the particles, *pop*: *pop* = 70.
w_1 and w_2: $w_1 = 0.5$, $w_2 = 0.5$.
a, b, c, d, P_1 and P_2: $a = 0.3$, $b = 0.3$, $c = 0.3$, $d = 0.1$ and $P_1 = 0.7$, $P_2 = 0.5$.
The maximum number of iterations, S: $S = 150$.
The termination criterion, T: $T = 2$.
δ and ξ: $\delta = [N/4]$, $\xi = [N/4]$.

N denotes the number of jobs. M is the number of machines. IR is the objective value of a randomly chosen particle among the randomly initialized particles. T is one termination criterion. When $bias(t) < T$, the algorithm will be terminated. The operator $[x]$ denotes the largest integer smaller than the value of x.

With 3 times of execution for each problem, the results are listed in Table 1. In the table, R, NR, and JC are the abbreviation of *narrow random*, *random* and *job-correlated*, respectively; Ins. is short for *Instance*; BN is the best known solutions published online, together with the instances (at http://www.cs.colostate.edu/sched/generator/); *C* is the best solution out of three times of execution on the same instance and RE is the relative error of the best solution. As is shown in Table 1, the proposed algorithm achieved three solutions better than the best known solutions published online provided by Watson et al. (2002). These three solutions are all in bold and marked with "*" or "**". Although the solution with "*" has been proposed by Tasgetiren et al. (2007) recently, the solutions with "**" are new best known solutions according to the best of our knowledge. (Refer to the Appendix for the start time and completion time of all the jobs of both solutions.)

Table 1
Computational results on 10 problems of Watson's suite. Numbers in bold are solutions better than upper bound published online by Watson et al. (2002). The solution with "*" has been published by Tasgetiren et al. (2007), while the solutions with "**" are new best known solutions according to the best of our knowledge.

Problem size	Ins.	α	BN	C	RE
20 × 20R	p.1	–	2213	**2211***	−0.09
20 × 20R	p.2	–	2155	2155	0.00
20 × 20NR	p.1	–	1969	1969	0.00
20 × 20JC	p.1	–	1113	1113	0.00
50 × 20JC	p.1	0.0	149	**148****	−0.67
50 × 20JC	p.2	0.0	4788	**4786****	−0.04
50 × 20JC	p.3	0.0	2635	2635	0.00
50 × 20JC	p.4	0.0	503	505	0.40
50 × 20JC	p.5	0.0	3069	3070	0.03
50 × 20JC	p.6	0.0	926	927	0.11

4.2. The comparison of PSO–EDA with PSO

To further present the performance of the proposed PSO–EDA, a comparison was made between the proposed algorithm with results of EM, and GA listed in the work of Yuan, Henequin, Wang, and Gao (2006) and PSOMA in the work of Liu et al. (2007) using instances from benchmark suites of Carlier and Reeves. Some of these instances are difficult, and they serve as good benchmark problems for testing algorithms (Liu et al., 2007). The results are listed in Table 2.

The electromagnetism-like mechanism (EM) is a comparatively new algorithm proposed by Yuan et al. (2006). Their experiments show that EM is more accurate than GA and NEH. PSOMA (PSO-based memetic algorithm) is an integrated hybridization of PSO with local search mechanism, which outperforms the canonical PSO and other simple hybrid PSO with local search heuristics, according to the numerical tests in the literature. The proposed PSO–EDA with primitive intelligence and the PSO–EDA without the primitive intelligence were implemented on Matlab 7.4 platform, and both were run for ten times for each instance. The parameters are set as proposed in Section 4.1.

In Table 2, EM is the electromagnetism-like mechanism; the results of EM and GA are both from the work of Yuan et al. (2006). PSOMA is the abbreviation of PSO-based memetic algorithm. PSO–EDA_PI is the proposed algorithm with primitive intelligence and PSO–EDA is the one without primitive intelligence. BRE, ARE and WRE are the best relative error, average relative error and the worst relative error, respectively.

As is shown in Table 2, PSO–EDA without primitive intelligence, with its ten executions, has less ARE than the BRE of EM and GA for most of the instances, especially some large scale instances, but it might not outperform PSOMA. However, the performance of PSO–EDA was enhanced when primitive intelligence was added to the swarm, as the ARE of PSO–EDA_PI for each instance is less than that of PSO–EDA without primitive intelligence. Also, PSO–EDA_PI has better (equal for some instances) BRE, ARE and WRE than PSOMA for each instance tested. According to the results of this test, the proposed PSO–EDA with primitive intelligence outperforms PSOMA in terms of accuracy.

4.3. The comparison of PSO–EDA with EDA

In the former subsection we have compared PSO–EDA with PSOMA which outperforms the canonical PSO and other simple

Table 2
The comparison of PSO–EDA with GA, EM, NEH and PSOMA.

Pro	$N \times M$	Cbest	EM		GA		PSOMA			PSO–EDA	PSO–EDA_PI		
			BRE	ARE	BRE	ARE	BRE	ARE	WRE	ARE	BRE	ARE	WRE
Car1	11×5	7038	0	0.21	0	0.27	0	0	0	0	0	0	0
Car2	13×4	7166	0	2.55	0	4.07	0	0	0	0	0	0	0
Car3	12×5	7312	0	2.19	1.19	2.95	0	0	0	0	0	0	0
Car4	14×4	8003	0	0.95	0	2.36	0	0	0	0	0	0	0
Car5	10×6	7720	0	1.27	0	1.46	0	0.018	0.375	0	0	0	0
Car6	8×9	8505	0	1.34	0	1.86	0	0.114	0.764	0	0	0	0
Car7	7×7	6590	0	1.12	0	1.57	0	0	0	0	0	0	0
Car8	8×8	8366	0	1.05	0	2.59	0	0	0	0	0	0	0
Rec01	20×5	1247	0.16	4.41	2.81	6.96	0	0.144	0.160	0.128	0	0.096	0.160
Rec03	20×5	1109	0.18	1.98	1.89	4.45	0	0.189	0.721	0.036	0	0.036	0.180
Rec05	20×5	1242	0.24	2.01	1.93	3.82	0.242	0.249	0.402	0.242	0.242	0.242	0.242
Rec07	20×10	1566	1.15	3.70	1.15	5.31	0	0.986	1.149	0	0	0	0
Rec09	20×10	1537	0.65	3.97	3.12	4.73	0	0.621	1.691	0.416	0	0.202	1.041
Rec11	20×10	1431	0.98	4.05	3.91	7.39	0	0.129	0.978	0.496	0	0.126	0.629
Rec13	20×15	1930	2.23	4.87	3.68	5.97	0.259	0.893	1.502	0.596	0.104	0.223	0.415
Rec15	20×15	1950	1.64	3.79	2.21	4.29	0.051	0.628	1.076	0.744	0	0.303	0.667
Rec17	20×15	1902	2.63	5.57	3.15	6.08	0	1.330	2.155	0.568	0	0.289	0.999
Rec19	30×10	2093	2.34	5.97	4.01	6.07	0.43	1.313	2.102	1.147	0.287	0.612	1.003
Rec21	30×10	2017	1.64	3.22	3.42	6.07	1.437	1.596	1.636	1.438	1.140	1.408	1.438
Rec23	30×10	2011	2.49	5.97	3.83	7.46	0.596	1.310	2.038	1.009	0.398	0.597	1.840
Rec25	30×15	2513	2.55	5.93	4.42	7.20	0.835	2.085	3.223	2.109	0.279	1.894	2.507
Rec27	30×15	2373	2.49	5.90	4.93	6.85	1.348	1.605	2.402	1.639	0.969	1.584	2.023
Rec29	30×15	2287	3.15	7.08	6.21	8.48	1.442	1.888	2.492	1.950	0.350	1.045	1.618
Rec31	50×10	3045	4.01	6.40	6.17	8.02	1.51	2.254	2.692	0.654	0.263	0.430	0.657
Rec33	50×10	3114	0.96	3.76	3.08	5.12	0	0.645	0.836	0.636	0	0.469	0.835
Rec35	50×10	3277	0	2.35	1.46	3.30	0	0	0	0	0	0	0
Rec37	75×20	4951	5.43	8.22	6.56	8.72	2.101	3.537	4.039	3.955	1.838	2.725	4.040
Rec39	75×20	5087	3.97	7.23	6.39	7.57	1.553	2.426	2.830	1.858	0.924	1.409	1.730
Rec41	75×20	4960	6.21	8.43	7.42	8.92	2.641	3.684	4.052	2.794	1.815	2.506	2.944

hybrid PSO with local search heuristics. In this subsection, we would compare PSO–EDA with EDA. The EDA operator we apply in our proposed algorithm follows the EDA proposed by Jarboui et al. (2009), which was applied for PFSP to minimize the total flow time, and according to the best of our knowledge, few literatures seeks to apply EDA to solve a PFSP with minimization of makespan as its objective.

In order to compare EDA with the proposed algorithm, we implement the EDA proposed by Jarboui et al. (2009) with all the parameters set as was presented in their work. Also, some parameters of the EDA operator of PSO–EDA were changed as follows:

pop of PSO–EDA: *pop* = 60
T of PSO–EDA: *T*=0, which means that the value of *T* is no longer a criterion for termination of the algorithm.

Maximum number of iterations of PSO–EDA, *S*: *S* = 100, which means that the algorithm would terminate after 100 iterations.

Note that the change of the parameters is to ensure that the EDA operator proposed would have the same parameters with EDA proposed by Jarboui et al. (2009).

The convergence property of PSO–EDA and EDA for instances of different scales of Reeves is shown in Figs. 9–14 in which the superiority of PSO–EDA over EDA is evident.

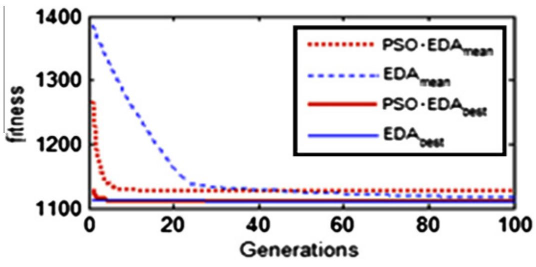


Fig. 9. The comparison on Rec03 (20×5).

In the figure, PSO–EDA$_{mean}$ and PSO–EDA$_{best}$ and are the mean objective value of the pbest solutions, and the objective value of global best solution of the iterations of PSO–EDA, while in the same time, EDA$_{mean}$ and EDA$_{best}$ are the mean objective value of all the population and the best objective value among the population. As is shown in the figure, the swarm of PSO–EDA converges faster than the population of EDA, and the latter tend to be trapped in areas where there are local optima.

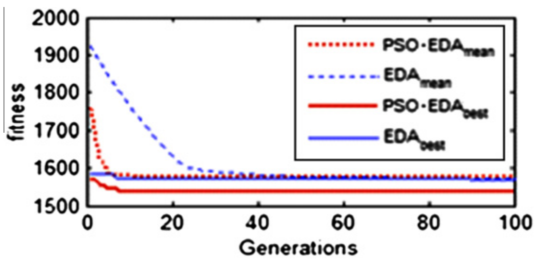


Fig. 10. The comparison on Rec09 (20×10).

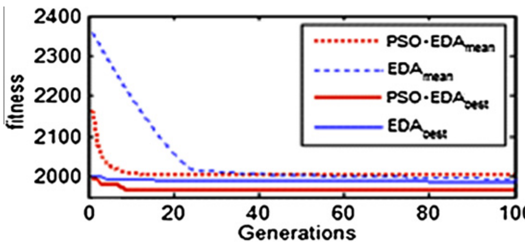


Fig. 11. The comparison on Rec15 (20×15).

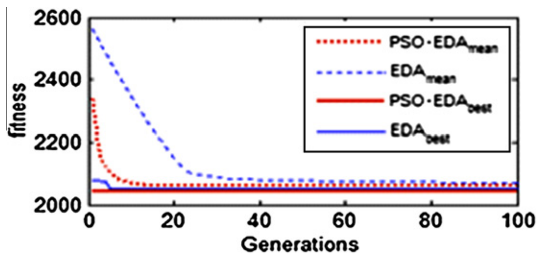


Fig. 12. The comparison on Rec21 (30×10).

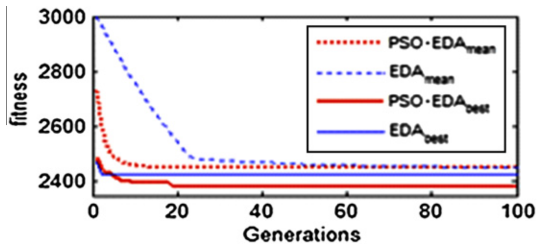


Fig. 13. The comparison on Rec27 (30×15).

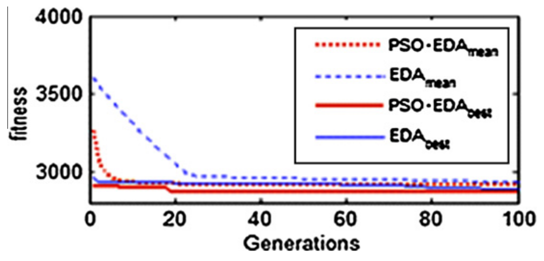


Fig. 14. The comparison on Rec33 (50×10).

5. Conclusions

This paper proposes a hybrid PSO with EDA (PSO–EDA) that enables the ability of the particles to learn from the information of the collective experience of the swarm and that utilizes the primitive intelligence to search the local area of *pbest* solutions to solve PFSP.

The performance of the proposed algorithm was tested on benchmark suites of Watson, Reeves and Carlier. Among the obtained results are new best known solutions to two Watson's instances. Also, the comparison of PSO–EDA with the counterpart algorithms in the literatures, including EM, GA, and PSOMA shows that the algorithm proposed in this paper outperforms other algorithms in terms of accuracy and quality of the solutions. Moreover, comparing PSO–EDA with primitive intelligence and PSO–EDA without primitive intelligence, the former has better performance.

The convergence property of PSO–EDA with primitive intelligence is compared with the EDA which the EDA operator of our proposed algorithm follows. The curves of the mean objective value and of the best objective value show the superiority of the proposed algorithm over the latter.

Future work would involve the mathematical demonstration of its convergence, the improvement on the probabilistic model of the EDA operator and the further refinement of the algorithm, including the adaptive tuning of parameters and the methods for better integration of the hybridized algorithms.

Acknowledgements

The authors thank Dr. Tiefang He and Dr. Xiaojuan Wang from HUST for their helpful comments and suggestions. The authors also would like to thank anonymous referees whose comments helped a lot to improve this paper. This paper is supported by 863 High Technology Plan Foundation of China under Grant No. 2006AA04Z131, Program or New Century Excellent Talents in University under Grant No. NCET-08-0232, the National Basic Research Program (973 Program) of China under Grant No. 2011CB706804, the Hi-Tech Research and Development Program.

Appendix A

Since the Gantt Charts for the new best known solutions are too complex to be clarified here, we present the start times and completion times of the jobs to be processed on different machines according to the schedules that the solutions denote instead.

A.1. The new best known solution for 50×20JC, $\alpha = 0.0$, p.1

The permutation, or namely the sequence of the jobs to be processed, is:

[27 43 29 7 17 14 36 24 1 37 20 18 50 21 23 30 33 39 38 49 45 10 6 16 35 11 31 2 19 22 4 5 13 34 12 15 44 47 42 25 40 48 3 46 8 26 41 9 32 28]

The start times and the completion times of each job's procession on the machines are:

Job 1: [10;11] [11;12] [14;15] [15;19] [19;20] [25;26] [27;29] [29;30] [33;36] [36;41] [41;47] [47;48] [48;52] [52;56] [56;57] [57;63] [63;68] [68;71] [71;72] [72;73]
Job 2: [43;44] [48;51] [51;54] [58;59] [61;64] [64;68] [68;70] [71;73] [74;76] [78;79] [83;84] [85;87] [88;90] [91;95] [99;101] [102;104] [104;108] [110;112] [114;115] [117;118]
Job 3: [77;78] [82;83] [84;85] [90;91] [93;99] [99;100] [103;105] [105;109] [113;115] [116;117] [118;121] [121;124] [124;125] [125;126] [129;130] [130;132] [132;133] [134;135] [138;139] [140;141]
Job 4: [49;50] [55;56] [60;61] [62;65] [69;72] [72;73] [73;74] [77;78] [82;83] [87;88] [88;91] [91;95] [96;97] [101;103] [108;109] [110;111] [111;112] [116;117] [117;120] [120;121]
Job 5: [50;53] [56;57] [61;63] [65;67] [72;73] [73;74] [74;77] [78;80] [83;84] [88;90] [91;94] [95;97] [97;100] [103;106] [109;110] [111;112] [112;113] [117;120] [120;121] [121;124]
Job 6: [33;34] [35;37] [41;45] [45;49] [50;53] [53;54] [54;59] [60;66] [66;67] [69;70] [75;77] [79;80] [83;84] [86;87] [90;91] [91;92] [96;97] [97;101] [101;103] [107;109]
Job 7: [3;4] [4;5] [6;7] [7;9] [10;12] [15;16] [17;18] [19;20] [21;22] [23;24] [27;28] [29;30] [34;36] [37;38] [40;42] [42;43] [43;45] [45;46] [46;47] [48;50]
Job 8: [81;82] [85;88] [88;90] [94;97] [100;102] [102;103] [106;107] [112;114] [116;119] [119;121] [122;123] [125;126] [126;128] [128;129] [131;133] [133;135] [135;137] [138;140] [140;141] [142;143]
Job 9: [84;85] [93;96] [97;98] [100;101] [105;107] [107;108] [111;114] [116;118] [124;126] [129;130] [130;131] [131;133] [133;136] [136;137] [137;138] [139;140] [140;142] [142;143] [144;145] [145;146]
Job 10: [32;33] [34;35] [38;41] [41;42] [49;50] [52;53] [53;54] [59;60] [64;65] [65;69] [71;75] [77;79] [82;83] [85;86] [89;90] [90;91] [94;96] [96;97] [100;101] [103;107]
Job 11: [39;42] [46;47] [47;48] [55;56] [58;59] [60;61] [61;63] [68;70] [72;73] [75;76] [80;82] [82;84] [86;87] [89;90] [97;98] [100;101] [102;103] [106;109] [111;112] [115;116]

Job 12: [55;61] [66;67] [67;68] [70;71] [77;80] [80;85] [85;86] [86;87] [90;93] [95;100] [100;103] [105;106] [106;112] [112;113] [113;115] [115;118] [118;119] [122;123] [124;125] [126;131]

Job 13: [53;54] [57;61] [63;66] [67;69] [73;76] [76;78] [78;80] [80;81] [84;85] [90;91] [94;95] [97;98] [100;103] [106;107] [110;112] [112;113] [113;117] [120;121] [121;123] [124;125]

Job 14: [5;6] [8;9] [9;10] [11;13] [15;16] [18;19] [19;23] [23;24] [26;27] [28;29] [32;33] [35;38] [40;41] [41;45] [45;46] [48;51] [51;52] [52;56] [56;57] [58;59]

Job 15: [61;64] [67;68] [68;69] [71;72] [80;81] [85;88] [88;89] [89;91] [93;94] [100;101] [103;106] [106;108] [112;113] [113;114] [115;116] [118;119] [119;122] [123;126] [126;127] [131;132]

Job 16: [34;38] [38;42] [45;46] [49;53] [53;57] [57;59] [59;60] [66;67] [67;71] [71;72] [77;79] [80;81] [84;85] [87;88] [91;96] [96;99] [99;101] [101;105] [105;107] [109;110]

Job 17: [4;5] [5;8] [8;9] [9;11] [12;15] [16;18] [18;19] [20;22] [22;26] [26;28] [28;32] [32;35] [36;40] [40;41] [42;44] [44;48] [48;50] [50;51] [51;54] [54;58]

Job 18: [13;15] [15;17] [17;18] [21;22] [24;26] [31;33] [33;34] [34;35] [38;40] [45;46] [50;51] [53;54] [55;57] [58;59] [59;60] [67;69] [71;72] [73;75] [76;78] [80;82]

Job 19: [44;45] [51;52] [54;56] [59;60] [64;66] [68;69] [70;71] [73;74] [76;77] [79;80] [84;86] [87;88] [90;91] [95;96] [101;103] [104;106] [108;109] [112;114] [115;116] [118;119]

Job 20: [12;13] [13;15] [16;17] [20;21] [23;24] [29;31] [31;33] [33;34] [37;38] [43;45] [49;50] [51;53] [53;55] [57;58] [58;59] [66;67] [69;71] [72;73] [75;76] [78;80]

Job 21: [16;17] [18;19] [22;24] [27;28] [28;30] [34;36] [36;37] [40;41] [41;42] [47;48] [52;53] [58;59] [61;63] [64;66] [70;71] [71;73] [76;78] [78;79] [79;80] [83;85]

Job 22: [45;49] [52;55] [56;60] [60;62] [66;69] [69;70] [71;73] [74;77] [77;82] [82;87] [87;88] [88;91] [91;96] [96;101] [103;108] [108;110] [110;111] [114;116] [116;117] [119;120]

Job 23: [17;18] [19;22] [24;27] [28;29] [30;36] [36;37] [37;43] [43;44] [44;45] [48;51] [53;57] [59;60] [63;64] [66;71] [71;72] [73;75] [78;79] [79;80] [80;86] [86;89]

Job 24: [9;10] [10;11] [13;14] [14;15] [17;18] [23;25] [25;27] [27;29] [32;33] [33;34] [38;40] [42;43] [47;48] [49;50] [54;55] [55;57] [60;61] [61;62] [65;67] [67;68]

Job 25: [73;74] [76;77] [77;78] [80;86] [89;90] [92;93] [96;97] [98;99] [104;109] [111;112] [114;115] [115;116] [118;119] [119;122] [122;127] [127;128] [128;130] [130;131] [132;133] [136;137]

Job 26: [82;83] [88;89] [90;91] [97;98] [102;104] [104;105] [107;109] [114;115] [119;120] [121;122] [123;124] [126;128] [128;129] [129;130] [133;134] [135;137] [137;138] [140;141] [141;142] [143;144]

Job 27: [0;1] [1;2] [2;3] [3;5] [5;6] [6;8] [8;9] [9;14] [14;15] [15;16] [16;17] [17;22] [22;27] [27;32] [32;33] [33;34] [34;35] [35;36] [36;41] [41;42]

Job 28: [87;90] [97;99] [100;101] [102;103] [108;110] [110;112] [115;117] [119;121] [128;131] [132;133] [134;135] [135;138] [138;139] [139;140] [141;142] [142;144] [144;145] [145;146] [146;147] [147;148]

Job 29: [2;3] [3;4] [4;6] [6;7] [8;10] [13;15] [16;17] [17;19] [19;21] [22;23] [26;27] [28;29] [32;34] [36;37] [39;40] [40;42] [42;43] [43;45] [45;46] [46;48]

Job 30: [18;19] [22;25] [27;28] [29;34] [36;41] [41;42] [43;44] [44;45] [45;49] [51;52] [57;63] [63;64] [64;70] [71;72] [72;75] [75;76] [79;80] [80;86] [86;89] [89;90]

Job 31: [42;43] [47;48] [48;49] [56;58] [59;61] [61;63] [63;64] [70;71] [73;74] [76;78] [82;83] [84;85] [87;88] [90;91] [98;99] [101;102] [103;104] [109;110] [112;114] [116;117]

Job 32: [85;87] [96;97] [98;100] [101;102] [107;108] [108;110] [114;115] [118;119] [126;128] [130;132] [132;134] [134;135] [136;138] [138;139] [139;141] [141;142] [142;144] [144;145] [145;146] [146;147]

Job 33: [19;20] [25;30] [30;31] [34;35] [41;42] [42;43] [44;45] [45;47] [49;50] [52;57] [63;65] [65;66] [70;72] [72;73] [75;78] [78;79] [80;81] [86;87] [89;92] [92;95]

Job 34: [54;55] [61;66] [66;67] [69;70] [76;77] [78;79] [80;81] [81;84] [85;90] [91;95] [95;100] [100;105] [105;106] [107;112] [112;113] [113;115] [117;118] [121;122] [123;124] [125;126]

Job 35: [38;39] [42;46] [46;47] [53;55] [57;58] [59;60] [60;61] [67;68] [71;72] [72;75] [79;80] [81;82] [85;86] [88;89] [96;97] [99;100] [101;102] [105;106] [107;111] [111;115]

Job 36: [6;9] [9;10] [10;13] [13;14] [16;17] [19;23] [23;24] [24;25] [27;32] [32;33] [33;38] [38;42] [42;47] [47;49] [49;54] [54;55] [55;60] [60;61] [61;65] [65;66]

Job 37: [11;12] [12;13] [15;16] [19;20] [20;23] [26;29] [29;30] [30;33] [36;37] [41;43] [47;49] [49;51] [52;53] [56;57] [57;58] [63;66] [68;69] [71;72] [72;75] [75;78]

Job 38: [21;26] [31;32] [32;36] [36;39] [43;44] [44;47] [47;51] [51;52] [52;57] [60;61] [66;67] [67;73] [73;79] [79;82] [82;83] [83;84] [84;85] [90;91] [93;97] [97;101]

Job 39: [20;21] [30;31] [31;32] [35;36] [42;43] [43;44] [45;46] [47;48] [50;51] [57;60] [65;66] [66;67] [72;73] [73;74] [78;79] [79;80] [81;82] [87;90] [92;93] [95;96]

Job 40: [74;75] [77;81] [81;82] [86;89] [90;91] [93;96] [97;101] [101;104] [109;111] [112;114] [115;116] [116;117] [119;123] [123;124] [127;128] [128;129] [130;131] [131;133] [133;137] [137;138]

Job 41: [83;84] [89;93] [93;97] [98;100] [104;105] [105;106] [109;111] [115;116] [120;124] [124;129] [129;130] [130;131] [131;133] [133;134] [134;137] [137;139] [139;140] [141;142] [142;144] [144;145]

Job 42: [71;73] [73;76] [76;77] [77;80] [87;89] [91;92] [92;96] [97;98] [103;104] [108;111] [111;114] [114;115] [117;118] [118;119] [120;121] [126;127] [127;128] [128;129] [129;132] [135;136]

Job 43: [1;2] [2;3] [3;4] [5;6] [6;8] [8;13] [13;16] [16;17] [17;19] [19;22] [22;26] [26;28] [28;32] [32;36] [36;39] [39;40] [40;41] [41;42] [42;44] [44;45]

Job 44: [64;70] [70;71] [71;73] [73;74] [81;86] [88;89] [89;91] [91;96] [96;101] [101;107] [107;108] [108;113] [113;116] [116;117] [117;118] [119;125] [125;126] [126;127] [127;128] [132;133]

Job 45: [30;32] [33;34] [37;38] [40;41] [45;49] [51;52] [52;53] [55;59] [61;64] [64;65] [69;71] [74;77] [80;82] [83;85] [85;89] [89;90] [90;94] [95;96] [99;100] [102;103]

Job 46: [78;81] [83;85] [85;88] [91;94] [99;100] [100;101] [105;106] [109;112] [115;116] [117;118] [121;122] [124;125] [125;126] [126;127] [130;131] [132;133] [133;134] [135;138] [139;140] [141;142]

Job 47: [70;71] [71;73] [73;75] [75;77] [86;87] [89;91] [91;92] [96;97] [101;103] [107;108] [108;109] [113;114] [116;117] [117;118] [118;120] [125;126] [126;127] [127;128] [128;129] [133;135]

Job 48: [75;77] [81;82] [82;84] [89;90] [91;93] [96;98] [101;103] [104;105] [111;113] [114;116] [116;118] [118;120] [123;124] [124;125] [128;129] [129;130] [131;132] [133;134] [137;138] [138;140]

Job 49: [26;30] [32;33] [36;37] [39;40] [44;45] [47;51] [51;52] [52;55] [57;61] [61;62] [67;69] [73;74] [79;80] [82;83] [83;85] [85;86] [86;90] [91;95] [97;99] [101;102]

Job 50: [15;16] [17;18] [18;22] [22;27] [27;28] [33;34] [34;35] [35;40] [40;41] [46;47] [51;52] [54;58] [58;61] [61;64] [64;70] [70;71] [72;76] [76;77] [78;79] [82;83]

A.2. The new best known solution for 50 × 20JC, α = 0.0, p.2

The permutation, or namely the sequence of the jobs to be processed, is:

[23 34 43 41 19 40 15 3 7 47 6 44 8 49 28 36 21 9 45 12 26 11 14 29 22 30 1 17 10 18 27 32 4 24 31 13 33 37 2 50 35 5 16 39 20 38 42 25 48 46].

The start time and the completion time of each job's procession on the machines are:

Job 1: [1781;1851] [1852;1917] [1927;1991] [1998;2070] [2070;2136] [2137;2206] [2210;2281] [2281;2354] [2358;2427] [2427;2494] [2495;2561] [2569;2636] [2639;2703] [2706;2775] [2775;2847] [2847;2919] [2922;2992] [2997;3071] [3073;3138] [3147;3213]
Job 2: [2611;2680] [2695;2764] [2767;2837] [2837;2906] [2909;2977] [2981;3051] [3051;3121] [3124;3193] [3194;3264] [3268;3337] [3337;3405] [3407;3477] [3479;3547] [3549;3617] [3617;3687] [3688;3758] [3760;3829] [3830;3900] [3900;3968] [3970;4039]
Job 3: [474;542] [542;611] [616;683] [685;756] [759;824] [828;895] [897;963] [969;1039] [1039;1110] [1110;1176] [1180;1249] [1250;1318] [1324;1392] [1392;1462] [1462;1534] [1535;1604] [1604;1675] [1675;1746] [1746;1814] [1817;1883]
Job 4: [2195;2267] [2267;2340] [2341;2411] [2415;2483] [2486;2551] [2556;2623] [2636;2701] [2702;2774] [2776;2848] [2848;2916] [2916;2982] [2985;3054] [3058;3127] [3127;3192] [3201;3272] [3272;3339] [3343;3416] [3417;3484] [3487;3553] [3555;3624]
Job 5: [2818;2888] [2904;2974] [2975;3043] [3046;3116] [3116;3184] [3188;3256] [3258;3327] [3331;3401] [3403;3471] [3474;3543] [3543;3612] [3616;3684] [3687;3755] [3756;3826] [3826;3895] [3895;3965] [3967;4036] [4037;4105] [4107;4175] [4177;4247]
Job 6: [677;741] [751;817] [823;891] [894;960] [961;1033] [1033;1103] [1103;1177] [1177;1250] [1250;1324] [1324;1388] [1388;1454] [1458;1529] [1529;1593] [1597;1661] [1674;1740] [1743;1807] [1814;1884] [1885;1959] [1959;2027] [2027;2100]
Job 7: [542;609] [611;680] [683;754] [756;823] [824;893] [895;962] [963;1030] [1039;1107] [1110;1178] [1178;1249] [1249;1318] [1318;1387] [1392;1461] [1462;1529] [1534;1605] [1605;1676] [1676;1745] [1746;1815] [1815;1884] [1884;1954]
Job 8: [813;884] [884;957] [959;1025] [1030;1103] [1103;1169] [1169;1236] [1248;1317] [1321;1386] [1389;1455] [1455;1525] [1525;1594] [1596;1663] [1663;1730] [1735;1805] [1813;1880] [1880;1953] [1953;2020] [2025;2094] [2100;2165] [2166;2238]
Job 9: [1161;1230] [1231;1304] [1304;1373] [1373;1440] [1446;1513] [1514;1580] [1588;1657] [1658;1729] [1730;1795] [1799;1866] [1868;1940] [1941;2014] [2014;2079] [2086;2157] [2157;2230] [2230;2303] [2303;2374] [2375;2447] [2447;2515] [2516;2585]
Job 10: [1918;1986] [1988;2058] [2061;2131] [2142;2212] [2212;2280] [2280;2349] [2354;2423] [2428;2496] [2498;2566] [2566;2634] [2634;2703] [2705;2773] [2776;2846] [2850;2918] [2918;2987] [2993;3061] [3065;3135] [3137;3205] [3205;3274] [3282;3351]
Job 11: [1436;1506] [1510;1579] [1580;1651] [1651;1720] [1722;1789] [1792;1860] [1860;1927] [1935;2003] [2007;2074] [2074;2144] [2144;2212] [2220;2287] [2290;2360] [2363;2432] [2435;2504] [2504;2575] [2577;2644] [2649;2718] [2718;2787] [2799;2870]
Job 12: [1300;1371] [1372;1438] [1441;1511] [1513;1584] [1584;1653] [1654;1721] [1722;1794] [1795;1864] [1864;1930] [1936;2003] [2011;2077] [2082;2154] [2154;2223] [2223;2295] [2295;2363] [2372;2439] [2442;2511] [2513;2581] [2584;2651] [2656;2728]
Job 13: [2402;2473] [2483;2556] [2558;2626] [2630;2695] [2700;2771] [2771;2843] [2843;2912] [2913;2986] [2986;3053] [3055;3128] [3128;3196] [3197;3269] [3270;3343] [3344;3414] [3414;3479] [3480;3547] [3548;3617] [3621;3690] [3694;3759] [3759;3830]
Job 14: [1506;1574] [1579;1646] [1651;1720] [1720;1789] [1789;1860] [1860;1925] [1927;2000] [2003;2073] [2074;2147] [2147;2220] [2220;2290] [2290;2362] [2362;2431] [2432;2498] [2504;2569] [2575;2643] [2644;2715] [2718;2791] [2791;2862] [2870;2938]
Job 15: [408;474] [476;542] [545;616] [618;685] [687;759] [759;828] [828;897] [897;969] [969;1037] [1039;1108] [1108;1180] [1180;1250] [1253;1324] [1324;1392] [1394;1460] [1465;1535] [1535;1602] [1606;1673] [1674;1745] [1746;1817]
Job 16: [2888;2957] [2974;3045] [3045;3113] [3116;3187] [3187;3256] [3256;3327] [3327;3398] [3401;3470] [3471;3541] [3543;3610] [3612;3681] [3684;3754] [3755;3823] [3826;3893] [3895;3965] [3965;4034] [4036;4103] [4105;4175] [4175;4245] [4247;4317]
Job 17: [1851;1918] [1918;1988] [1991;2061] [2070;2142] [2142;2211] [2211;2280] [2281;2354] [2354;2428] [2428;2498] [2498;2565] [2565;2630] [2636;2705] [2705;2776] [2776;2850] [2850;2916] [2919;2993] [2993;3065] [3071;3137] [3138;3203] [3213;3282]
Job 18: [1986;2054] [2058;2128] [2131;2201] [2212;2282] [2282;2350] [2350;2420] [2423;2493] [2496;2566] [2566;2634] [2634;2703] [2703;2772] [2773;2842] [2846;2916] [2918;2988] [2988;3058] [3061;3131] [3135;3205] [3205;3274] [3274;3344] [3351;3420]
Job 19: [270;340] [340;408] [408;476] [478;547] [547;615] [617;686] [688;757] [760;828] [829;899] [899;968] [970;1038] [1039;1108] [1114;1182] [1184;1254] [1256;1324] [1327;1397] [1397;1466] [1466;1536] [1536;1606] [1606;1676]
Job 20: [3026;3096] [3114;3183] [3184;3254] [3257;3329] [3329;3395] [3396;3464] [3467;3533] [3538;3609] [3609;3680] [3680;3752] [3752;3821] [3824;3893] [3893;3962] [3962;4031] [4035;4105] [4105;4173] [4174;4240] [4244;4311] [4315;4384] [4385;4451]
Job 21: [1091;1161] [1163;1231] [1234;1303] [1304;1372] [1377;1446] [1446;1514] [1519;1588] [1588;1658] [1660;1730] [1730;1799] [1799;1868] [1873;1941] [1944;2013] [2016;2086] [2086;2155] [2160;2230] [2232;2302] [2307;2375] [2377;2447] [2447;2516]
Job 22: [1642;1712] [1716;1784] [1789;1857] [1858;1927] [1928;1997] [1997;2066] [2069;2139] [2141;2211] [2217;2287] [2288;2356] [2360;2428] [2431;2500] [2500;2569] [2569;2637] [2638;2708] [2712;2780] [2784;2853] [2859;2927] [2932;3002] [3008;3078]
Job 23: [0;66] [66;135] [135;199] [199;271] [271;340] [340;408] [408;477] [477;544] [544;613] [613;684] [684;758] [758;832] [832;906] [906;979] [979;1048] [1048;1117] [1117;1183] [1183;1254] [1254;1320] [1320;1392]
Job 24: [2267;2332] [2340;2414] [2414;2485] [2485;2558] [2558;2625] [2625;2697] [2701;2765] [2774;2848] [2848;2915] [2916;2988] [2988;3058] [3058;3126] [3127;3198] [3198;3271] [3272;3340] [3340;3408] [3416;3481] [3484;3548] [3553;3623] [3624;3693]

Job 25: [3235;3304] [3320;3388] [3391;3460] [3469;3538] [3538;3608] [3608;3678] [3678;3746] [3750;3819] [3819;3889] [3889;3959] [3959;4027] [4028;4097] [4100;4169] [4170;4239] [4239;4309] [4309;4377] [4377;4447] [4448;4517] [4518;4587] [4587;4655]
Job 26: [1371;1436] [1438;1510] [1511;1580] [1584;1651] [1653;1722] [1722;1792] [1794;1859] [1864;1935] [1935;2007] [2007;2074] [2077;2142] [2154;2220] [2223;2290] [2295;2363] [2363;2435] [2439;2504] [2511;2577] [2581;2649] [2651;2716] [2728;2799]
Job 27: [2054;2124] [2128;2199] [2201;2271] [2282;2351] [2351;2419] [2420;2490] [2493;2563] [2566;2635] [2635;2705] [2705;2773] [2773;2840] [2842;2912] [2916;2984] [2988;3056] [3058;3128] [3131;3200] [3205;3274] [3274;3344] [3344;3412] [3420;3490]
Job 28: [952;1023] [1024;1093] [1095;1161] [1168;1237] [1240;1308] [1308;1377] [1384;1451] [1451;1521] [1526;1592] [1592;1662] [1662;1732] [1735;1804] [1804;1874] [1874;1945] [1945;2013] [2018;2089] [2092;2164] [2164;2236] [2238;2310] [2310;2377]
Job 29: [1574;1642] [1646;1716] [1720;1789] [1789;1858] [1860;1928] [1928;1997] [2000;2069] [2073;2141] [2147;2217] [2220;2288] [2290;2360] [2362;2431] [2431;2500] [2500;2568] [2569;2638] [2643;2712] [2715;2784] [2791;2859] [2862;2932] [2938;3008]
Job 30: [1712;1781] [1784;1852] [1857;1927] [1927;1998] [1998;2065] [2066;2137] [2139;2210] [2211;2281] [2287;2358] [2358;2427] [2428;2495] [2500;2569] [2569;2639] [2639;2706] [2708;2775] [2780;2847] [2853;2922] [2927;2997] [3002;3073] [3078;3147]
Job 31: [2332;2402] [2414;2483] [2485;2558] [2558;2630] [2630;2700] [2700;2768] [2768;2840] [2848;2913] [2915;2982] [2988;3055] [3058;3123] [3126;3197] [3198;3270] [3271;3344] [3344;3413] [3413;3480] [3481;3548] [3548;3621] [3623;3694] [3694;3759]
Job 32: [2124;2195] [2199;2267] [2271;2341] [2351;2415] [2419;2486] [2490;2556] [2563;2636] [2636;2702] [2705;2776] [2776;2840] [2840;2904] [2912;2985] [2985;3058] [3058;3125] [3128;3201] [3201;3272] [3274;3343] [3344;3417] [3417;3487] [3490;3555]
Job 33: [2473;2543] [2556;2625] [2626;2697] [2697;2768] [2771;2841] [2843;2913] [2913;2980] [2986;3057] [3057;3124] [3128;3199] [3199;3269] [3269;3337] [3343;3410] [3414;3481] [3481;3548] [3548;3616] [3617;3688] [3690;3761] [3761;3828] [3830;3899]
Job 34: [66;135] [135;201] [201;270] [271;338] [340;410] [410;475] [477;551] [551;617] [617;689] [689;760] [760;833] [833;902] [906;975] [979;1046] [1048;1121] [1121;1185] [1185;1255] [1255;1326] [1326;1393] [1393;1467]
Job 35: [2749;2818] [2834;2904] [2907;2975] [2976;3046] [3047;3116] [3119;3188] [3189;3258] [3263;3331] [3334;3403] [3405;3474] [3474;3542] [3547;3616] [3617;3687] [3687;3756] [3756;3826] [3826;3894] [3898;3967] [3968;4037] [4037;4107] [4108;4177]
Job 36: [1023;1091] [1093;1163] [1163;1234] [1237;1304] [1308;1377] [1377;1445] [1451;1519] [1521;1588] [1592;1660] [1662;1730] [1732;1799] [1804;1873] [1874;1944] [1945;2016] [2016;2085] [2089;2160] [2164;2232] [2236;2307] [2310;2377] [2377;2446]
Job 37: [2543;2611] [2625;2695] [2697;2767] [2768;2836] [2841;2909] [2913;2981] [2981;3050] [3057;3124] [3124;3194] [3199;3268] [3269;3335] [3337;3407] [3410;3479] [3481;3549] [3549;3617] [3617;3688] [3688;3760] [3761;3830] [3830;3899] [3899;3970]
Job 38: [3096;3167] [3183;3250] [3254;3322] [3329;3399] [3399;3467] [3467;3536] [3536;3607] [3609;3681] [3681;3750] [3752;3817] [3821;3893] [3893;3962] [3962;4027] [4031;4103] [4105;4172] [4173;4240] [4240;4307] [4311;4376] [4384;4451] [4451;4520]
Job 39: [2957;3026] [3045;3114] [3114;3184] [3187;3257] [3257;3326] [3327;3396] [3398;3467] [3470;3538] [3541;3609] [3610;3680] [3681;3751] [3754;3824] [3824;3893] [3893;3962] [3965;4035] [4035;4104] [4104;4174] [4175;4244] [4245;4315] [4317;4385]
Job 40: [340;408] [408;476] [476;545] [547;618] [618;687] [687;757] [757;827] [828;896] [899;967] [968;1039] [1039;1107] [1108;1178] [1182;1253] [1254;1324] [1324;1394] [1397;1465] [1466;1535] [1536;1606] [1606;1674] [1676;1746]
Job 41: [202;270] [270;339] [339;407] [409;478] [478;547] [549;617] [620;688] [690;760] [760;829] [829;899] [902;970] [971;1039] [1046;1114] [1114;1184] [1188;1256] [1258;1327] [1327;1397] [1397;1466] [1466;1536] [1536;1604]
Job 42: [3167;3235] [3250;3320] [3322;3391] [3399;3469] [3469;3536] [3536;3605] [3607;3674] [3681;3750] [3750;3818] [3818;3885] [3893;3959] [3962;4028] [4028;4100] [4103;4170] [4172;4239] [4240;4308] [4308;4376] [4376;4448] [4451;4518] [4520;4587]
Job 43: [135;202] [202;270] [270;339] [339;409] [410;478] [478;549] [551;620] [620;690] [690;759] [760;828] [833;902] [902;971] [975;1046] [1046;1114] [1121;1188] [1188;1258] [1258;1327] [1327;1397] [1397;1466] [1467;1536]
Job 44: [741;813] [817;883] [891;959] [960;1030] [1033;1101] [1103;1169] [1177;1248] [1250;1321] [1324;1389] [1389;1454] [1454;1524] [1529;1596] [1596;1663] [1663;1735] [1740;1813] [1813;1878] [1884;1952] [1959;2025] [2027;2100] [2100;2166]
Job 45: [1230;1300] [1304;1372] [1373;1441] [1441;1513] [1513;1583] [1583;1654] [1657;1722] [1729;1795] [1795;1864] [1866;1936] [1940;2011] [2014;2082] [2082;2154] [2157;2223] [2230;2295] [2303;2372] [2374;2442] [2447;2513] [2515;2584] [2585;2656]
Job 46: [3369;3440] [3459;3527] [3527;3599] [3611;3676] [3680;3752] [3752;3820] [3820;3887] [3887;3953] [3955;4021] [4032;4097] [4103;4174] [4174;4243] [4243;4316] [4316;4383] [4383;4449] [4449;4520] [4520;4586] [4586;4651] [4653;4720] [4720;4786]
Job 47: [609;677] [680;751] [754;823] [823;894] [894;961] [962;1031] [1031;1102] [1107;1177] [1178;1249] [1249;1320] [1320;1387] [1387;1458] [1461;1528] [1529;1597] [1605;1674] [1676;1743] [1745;1814] [1815;1885] [1885;1954] [1954;2023]
Job 48: [3304;3369] [3388;3459] [3460;3525] [3538;3611] [3611;3680] [3680;3751] [3751;3820] [3820;3885] [3889;3955] [3959;4032] [4032;4103] [4103;4170] [4170;4243] [4243;4310] [4310;4381] [4381;4447] [4447;4518] [4518;4586] [4587;4653] [4655;4720]
Job 49: [884;952] [957;1024] [1025;1095] [1103;1168] [1169;1240] [1240;1305] [1317;1384] [1386;1451] [1455;1526] [1526;1592] [1594;1662] [1663;1735] [1735;1803] [1805;1874] [1880;1945] [1953;2018] [2020;2092] [2094;2160] [2165;2238] [2238;2310]
Job 50: [2680;2749] [2764;2834] [2837;2907] [2907;2976] [2977;3047] [3051;3119] [3121;3189] [3193;3263] [3264;3334] [3337;3405] [3405;3473] [3477;3547] [3547;3617] [3617;3687] [3687;3756] [3758;3826] [3829;3898] [3900;3968] [3968;4037] [4039;4108]

References

Banks, A., Vincent, J., & Anyakoha, C. (2007). A review of particle swarm optimization. Part I: Background and development. *Natural Computing, 6*, 467–484.

Banks, A., Vincent, J., & Anyakoha, C. (2008). A review of particle swarm optimization. Part II: Hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing, 7*, 109–124.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal of Computing, 6*(2), 154–160.

Chandrasekaran, S., Ponnambalam, S. G., Suresh, R. K., & Vijayakumar, N. (2006). A hybrid discrete particle swarm optimization algorithm to solve flow shop scheduling problems. In *CIS, 2006*. IEEE.

Framinan, J. M., Leisten, R., & Ruiz-Usano, R. (2002). Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. *European Journal of Operational Research, 141*, 559–569.

Hansen, P., & Mladenovíc, N. (1999). An introduction to variable neighborhood search. *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*.

Iqbal, M., & Oca, M. A. M. D. (2006). An estimation of distribution particle swarm optimization algorithm. In *ANTS 2006. LNCS* (Vol. 4150, pp. 72–83).

Jarboui, B., Eddaly, M., & Siarry, P. (2009). An estimation of distribution algorithm for minimizing the total flow time in permutation flowshop scheduling problems. *Computers and Operations Research, 36*, 2638–2646.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly, 1*, 61–68.

Kaewkamnerdpong, B., & Bentley, P. (2005). Perceptive particle swarm optimization. In *The 7th international conference on adaptive and natural computing algorithms (ICCANGA 2005), 2005*.

Kan, A. H. G. R. (1976). *Machine scheduling problems: Classification, complexity and computations*. The Hague: Martinus Nijhoff.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks, Piscataway, NJ, 1995* (pp. 1942–1948).

Larraanaga, P., & Lozano, J. A. (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Boston/Dordrecht/London: Kluwer Academic Publishers.

Lian, Z., Gu, X., & Jiao, B. (2008). A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan. *Chaos, Solitons and Fractals, 35*, 851–861.

Liu, B., Wang, L., & Jin, Y. (2007). An effective PSO-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 37*(1).

Lozano, J. A., Larranaga, P., Inza, I., & Bengoetxea, E. (2006). *Towards a new evolutionary computation advances on estimation of distribution algorithms*. Berlin: Springer.

Nawaz, M., Enscore, E. E., & Ham, I. (1983). Heuristic algorithm for the m-machine, *n*-job flowshop sequencing problem. In *OMEGA, 1983*.

PaaG, M. H. (1996). From recombination of genes to the estimation of distribution. Binary parameters. In *Parallel problem solving from nature, PPSN, 1996, IV. Lecture notes in computer science* (Vol. 1411, pp.178–787).

Palmer, D. (1965). Sequencing jobs through a multi-stage process in the minimum total time-a quick method of obtaining near optimum. *Operational Research Quarterly, 16*(1), 101–107.

Pan, Q.-K., Wang, Ling, Tasgetiren, M. F., & Zhao, B-H. (2008). A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology, 38*, 337–347.

Rameshkumar, K., Suresh R. K., & Mohanasundaram, K. M. (2005). Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan. In *ICNC 2005. LNCS* (Vol. 3612, pp. 572–581).

Santana, R., Larrañaga, P., & Lozano, J. A. (2008). Adaptive estimation of distribution algorithms. In *Adaptive and multilevel metaheuristics, SCI* (Vol. 136, pp. 177–197).

Santana, R., Larrañaga, P., & Lozano, J. A. (2009). Research topics in discrete estimation of distribution algorithms based on factorizations. *Memetic Computing, 1*, 35–54.

Tasgetiren, M. F., Sevkli, M., Liang, Y.-C., & Gencyilmaz, G. (2004). Particle swarm optimization algorithm for single machine total weighted tardiness problem. In *Proceedings of the 2004 congress on evolutionary computation, 2004*.

Tasgetiren, M. F., Liang, Y-C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research, 177*, 1930–1947.

Toussaint, M. (2003). The structure of evolutionary exploration: On crossover, buildings blocks, and estimation-of-distribution algorithms. In *GECCO 2003. LNCS* (Vol. 2724, pp. 1444–1456).

Ucara, H., & Tasgetiren, M. F. (2006). A particle swarm optimization algorithm for permutation flow shop sequencing problem with the number of tardy jobs criterion. In *Proceedings of 5th international symposium on intelligent manufacturing systems, May 29–31, 2006* (pp. 1110–1120).

Voß, S. (2001). Meta-heuristics: The state of the art. Local search for planning and scheduling. *LNAI, 2148*, 1–23.

Wang, J. (2007a). A novel discrete particle swarm optimization based on estimation of distribution. In *ICIC 2007, LNAI* (Vol. 4682, pp. 791–802).

Wang, J. (2007b). Genetic particle swarm optimization based on estimation of distribution. In *LSMS 2007. LNCS* (Vol. 4688, pp. 287–296).

Watson, J. P., Barbulescu, L., Whitley, L. D., & Howe, A. E. (2002). structured and random permutation flowshop scheduling problems: Search space topology and algorithm performance. *ORSA Journal of Computing, 14*(2), 98–123.

Yuan, K, Henequin, S, Wang, X & Gao, L. (2006). A new heuristic-EM for permutation flowshop scheduling. In *Proceedings of the 12th IFAC symposium on information control problems in manufacturing (INCOM06), Saint-Etienne, France, May 17–19, 2006*.