International Journal of Production Research, 2013
http://dx.doi.org/10.1080/00207543.2013.848492

Taylor & Francis
Taylor & Francis Group

An improved NSGA-II algorithm for multi-objective lot-streaming flow shop scheduling problem

Yu-Yan Han[a], Dun-wei Gong[a]*, Xiao-Yan Sun[a] and Quan-Ke Pan[b]

[a]*School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou, China;* [b]*School of Computer Science, Liaocheng University, Liaocheng, China*

(*Received 7 May 2013; accepted 19 September 2013*)

Crossover and mutation operators in NSGA-II are random and aimless, and encounter difficulties in generating offspring with high quality. Aiming to overcoming these drawbacks, we proposed an improved NSGA-II algorithm (INSGA-II) and applied it to solve the lot-streaming flow shop scheduling problem with four criteria. We first presented four variants of NEH heuristic to generate the initial population, and then incorporated the estimation of distribution algorithm and a mutation operator based on insertion and swap into NSGA-II to replace traditional crossover and mutation operators. Last but not least, we performed a simple and efficient restarting strategy on the population when the diversity of the population is smaller than a given threshold. We conducted a serial of experiments, and the experimental results demonstrate that the proposed algorithm outperforms the comparative algorithms.

Keywords: NSGA-II; lot-streaming flow shop; multi-objective optimisation; heuristic rule; estimation of distribution algorithm; restarting strategy

1. Introduction

The lot-streaming flow shop (LSFS) scheduling problem, one representative branch of traditional flow shop schedule problems, is a simplified model of various real schedule problems, such as manufacturing systems, assembly lines, information service facilities, as well as chemical, textile, plastics and semiconductor industries (Pan, Pan, and Sang 2010). The main goal of the LSFS scheduling problem is to determine either the best allocation of sub-lots or the size of each sub-lot so as to minimise some given performance measures. Thus, developing effective and efficient methods to solve the above problems is of significance in theory and applications (Pan and Ruiz 2012). Different from traditional flow shops, LSFS splits a given job into several sub-lots, and each is transferred to the downstream machine after it is completed in the current one, thereby reducing the production cycle and waiting time, accelerating the manufacturing process and enhancing the production efficiency.

In the development of solving LSFS scheduling problem with single objective, efforts of some significant research of meta-heuristics have been made, such as genetic algorithm (GA), harmony search (HS), artificial bee colony (n) and estimation of distribution algorithm (EDA). Defersha and Chen (2012) proposed a method for solving the flexible LSFS scheduling problem with several jobs in multi-stage consisting of unrelated parallel machines. In this work, the authors constructed a mathematical model using integer programming and designed a parallel GA to solve the above model. The experimental results showed that the parallel implementation extremely improved the computational performance. Kim and Jeong (2009) presented an adaptive GA for the flexible LSFS scheduling with no-wait. In this study, the authors adopted four methods, that is, position-based crossover, local search-based mutation, iterative hill-climbing and adaptive regulation of GA parameters to improve the algorithm's performance, and the experimental results showed the proposed algorithm outperformed other GAs. Pan et al. (2010) designed a novel discrete HS algorithm for scheduling LSFS to minimise the makespan objective. To improve the algorithm's exploitation ability, the authors applied a local search into discrete HS algorithm, and the effectiveness and efficiency of the proposed algorithm were demonstrated by comparing with the existing composite ones, GA, ant colony optimisation (ACO) and the threshold accepting (TA) algorithm. Karaboga (2005) firstly proposed the ABC algorithm for continuous function optimisation. Given the above algorithm is of simple structure, strong convergence, more and more researchers have applied it to solve the flow shop scheduling problem. Lei and Guo (2013) adopted a modified ABC algorithm to solving the job shop with LSFS

*Corresponding author. Email: dwgong@vip.163.com

scheduling problem. In this work, employed and onlooker bees adopted the swap and insertion operators to produce new solutions, and the proposed algorithm adopted the elite solution to replace with the worst ones without considering the scouts. Recently, Pan and Ruiz (2012) developed an EDA for the LSFS problem with sequence-dependent set-up time. The experimental results showed that the proposed EDA was more effective than discrete ABC algorithm (Pan et al. 2011), ACO (Marimulthu, Ponnambalam, and Jawahar 2009), discrete particle swarm optimisation (Tseng and Liao 2008) and hybrid GA (Yoon and Ventura 2002).

However, production managers concern not only the scheduling problem with only one objective, but also the one with several objectives, such as makespan, tardiness time, earliness time, the total flow time and idle time of machines in various real-world applications. LSFS with single objective is insufficient; thus, multi-objective LSFS plays a key role in practical scheduling problems (Zhang et al. 2010). The well-known NSGA-II algorithm, firstly presented by Deb (2000), is one of effective evolutionary algorithms for multi-objective optimisation problems whose three main operators are fast non-dominated sorting, fast crowded distance estimation and simple crowded-based comparison Deb et al. (2002). In the light of its simplicity, good convergence and diversity, NSGA-II has caused much attention and a wide range of successful applications, such as customer churn prediction in telecommunications (Huang, Buckley, and Kechadi 2010), expansion planning (Murugan, Kannan, and Baskar 2009), economic and emission dispatch (Dhanalakshmi et al. 2011), estimation of prediction intervals of scale deposition rate in oil and gas equipments (Ronay et al. 2013), shape optimisation of axisymmetric cavitators in supercavitating flows (Shafaghat et al. 2011) and flow shop scheduling problems (Tseng and Liao 2008). The existing NSGA-II for solving the flow shop scheduling problems is commonly improved through modifying the non-dominated sorting or selection strategy. Zhang et al. (2010) designed an improved non-dominated sorting NSGA-II for the multi-objective flexible job shop scheduling problem, which is effective when both the rank and the crowed distance of the current two individuals are equal. Liu and Huang (2013) developed a new pre-selection method to obtain the minimal overall outsourcing cost and supply risk probability, which reduces the probability of design change at the stage of product production. However, the above algorithms did not consider the modification of the crossover and mutation operators, which guide the population towards the Pareto front, and thus decrease their effectiveness. In addition, to the best of our knowledge, there is few published work dealing with the multi-objective flow shop scheduling problem, especially the LSFS one, by using NSGA-II. So, it is considerably necessary to design an improved NSGA-II for the multi-objective LSFS problem.

EDA proposed by Muhlenbein and Paass (1996) has been widely studied by experts in evolutionary computation (Jarboui, Eddaly, and Siarry 2009). Because of its effectiveness, the EDA has been successfully applied to solve the permutation flow shop scheduling problem (Jarboui, Eddaly, and Siarry 2009; Zhang and Li 2011), the global continuous optimisation problem (Sun, Zhang, and Edward 2005), the nurse scheduling problem (Aickelin and Li 2007), LSFS problems (Pan and Ruiz 2012) and so on. The basic EDA mainly includes the following four steps (Larraaga and Lozano 2002). First, select PS promising solutions from the original population according to their fitness and then put them into a candidate population, $[\eta_{i,j}]_{\mathrm{PS}\times n}$. Second, build a probability distribution model, $[\xi_{i,j}]_{n\times n}$, based on the above population. Third, generate a new offspring, π_{new}, through learning and sampling from the constructed probabilistic model, and evaluate its fitness. Last, update the original population as follows: if π_{new} is better than the worst one of the original population, denoted as π_w, then replace π_w with π_{new}. In the multi-objective LSFS scheduling problem, EDA can take full advantage of valuable information of non-dominated solutions to construct a probabilistic model and then estimate the probability distribution of good chromosomes to generate promising offspring. So, we embedded EDA into NSGA-II instead of traditional crossover operator so as to lead the population towards the Pareto-optimal front.

Due to the LSFS problem has such characteristics as constraint conditions, dynamic nature, large scale, complex computation, studies on the LSFS problem with multiple objectives are relatively less than the one with single objective. Several objectives must be simultaneously considered in practical problems; therefore, it is necessary to design an algorithm for the LSFS scheduling problem with four objectives. Three main contributions of the proposed algorithm are drawn. First, the variants of NEH (Nawaz, Enscore, and Ham 1983) heuristics are adopted to construct four initial individuals with good performance Secondly, EDA is embedded in the NSGA-II algorithm to generate good offspring. Finally, the restart strategy is employed to avoid the proposed algorithm trapping in local optima.

The rest of this paper is organised as follows. After this brief introduction, in Section 2, the LSFS scheduling problem is stated. Section 3 describes the proposed algorithm. In Section 4, the evolutionary metrics are introduced, and the experimental results are provided in Section 5. Finally, the paper ends with some conclusions in Section 6.

2. LSFS scheduling problem

The LSFS scheduling problem can be formulated as follows (Pan and Ruiz 2012). There are $n\,(\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\})$ jobs and m machines. Each job $\pi(i) \in \pi$ is processed on each of m machines in the same series.

Table 1. Illustration of notations.

$\pi(i)$	The ith job of sequence π with jobs being indexed by $i = 1, 2, \ldots, n$
π_j	The jth job sequence or the jth individual with job sequences being indexed by $j = 1, 2, \ldots, PS$
m	The total number of machines with each being indexed by $t = 1, 2, \ldots, m$
$l_{\pi(i)}$	The total number of sub-lots of job $\pi(i)$
e	The eth sub-lot with each being indexed by $e = 1, 2, \ldots, l_{\pi(i)}$
$p_{\pi(i),t}$	The processing time of job i on machine t ($t \in \{1, 2, \ldots, m\}$)
$p'e, t$	The processing time of the eth sub-lot on machine t
$C_{\pi(i),t,e}$	The completion time of the eth sub-lot of job i on machines t
$S_{\pi(i),t,e}$	The start time of the eth sub-lot of job i on machines t
d_i	The due date of job i
f_1	Makespan
f_2	The total flow time
f_3	The idle time of all machines
f_4	The earliness time

Each job can be split into several sub-lots, and each sub-lot has different processing time on different machines. The constraints of the LSFS scheduling problem can be described below. (1) The job can be processed at the jth machine only when all sub-lots of the forgoing job are completed at the machine; (2) At any time, each machine can process at most one sub-lot, and each sub-lot can be processed on at most one machine at the same time; (3) All sub-lots of the same job $\pi(i) \in \pi$ should be continuously processed. Any two adjacent sub-lots of job j allow idle time at the same stage. Both the set-up time and the sub-lot transportation time are included in the processing time. In order to clearly describe the LSFS scheduling problem and well explain the mathematical model, Table 1 illustrates the meaning of all notations.

Figure 1 shows a Gantt comparison between traditional flow shop and LSFS with 2 jobs and 3 machines. Suppose that jobs 1 and 2 contain 3 and 2 sub-lots, respectively, and the processing time of each sub-lot is as follows.

$$\left[P'_{e,t}\right]_{3\times3} = \begin{bmatrix} p'_{1,1} & p'_{1,2} & p'_{1,3} \\ p'_{2,1} & p'_{2,2} & p'_{2,3} \\ p'_{3,1} & p'_{3,2} & p'_{3,3} \end{bmatrix} = \begin{bmatrix} 2 & 4 & 3 \\ 2 & 4 & 3 \\ 2 & 4 & 3 \end{bmatrix} \quad \left[P'_{e,t}\right]_{2\times3} = \begin{bmatrix} p'_{1,1} & p'_{1,2} & p'_{1,3} \\ p'_{2,1} & p'_{2,2} & p'_{2,3} \end{bmatrix} = \begin{bmatrix} 1 & 5 & 2 \\ 1 & 5 & 2 \end{bmatrix}$$

As shown in Figure 1, the process of traditional flow shop scheduling problem is illustrated by an instance with two jobs and three machines with their job processing time of {6, 12, 9} and {2, 10, 4}, respectively. If none of these jobs is split into sub-lots, the completion time will be 32. Whereas, when each of these jobs is split into some sub-lots of equal size, the completion time is reduced to 26. Thus, in this example, lot-streaming can reduce the manufacturing time by about 19%.

While evaluating the performance of the LSFS scheduling problem, a manager often considers the following four objectives, that is, the makespan, the total flow time, idle time of all machines and the earliness time. The makespan intuitively reflects the completion time produced by different schedulings under some constraints. The total flow time

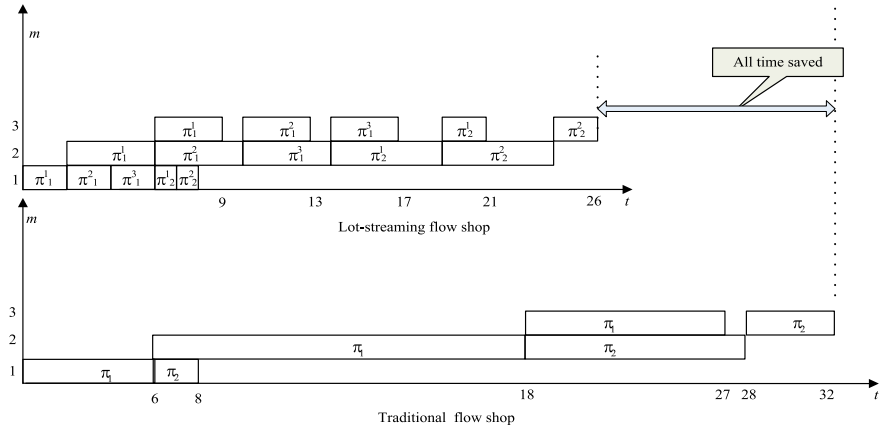

Figure 1. Gantt comparison between traditional flow shop and LSFS.

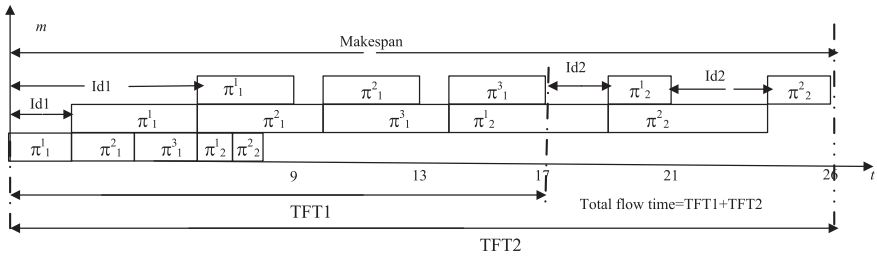

Figure 2. Computation of makespan, idle time and total flow time.

embodies the strict requirement for the whole flow time of all jobs, whereas the earliness time expresses the user requirement for ahead of schedule. The idle time reflects whether the machines can take full advantage of resource or not.

Hence, in view of practical production, this study formulates the LSFS scheduling problem as an optimisation one with the above four objectives. These objectives can be calculated as follows, and a Gantt for calculating the makespan, the total flow time and idle time is illustrated in Figure 2.

Let a job permutation $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ represent the sequence of jobs to be processed, and the start time of the first sub-lot of the first job on the first machine be equal to zero, that is, $S_{\pi(1),1,1} = 0$.

$$\begin{cases} S_{\pi(1),1,1} = 0 \\ C_{\pi(1),1,1} = S_{\pi(1),1,1} + p_{\pi(1),1} \end{cases} \tag{1}$$

$$\begin{cases} S_{\pi(1),t,1} = C_{\pi(1),t-1,1} \\ C_{\pi(1),t,1} = S_{\pi(1),t,1} + p_{\pi(1),t} \\ t = 2, 3, \ldots, m \end{cases} \tag{2}$$

$$\begin{cases} S_{\pi(1),1,e} = C_{\pi(1),1,e-1} \\ C_{\pi(1),1,e} = S_{\pi(1),1,e} + p_{\pi(1),1} \\ e = 2, 3, \ldots, l_{\pi(1)} \end{cases} \tag{3}$$

$$\begin{cases} S_{\pi(1),t,e} = \max\{C_{\pi(1),t-1,e}, C_{\pi(1),t,e-1}\} \\ C_{\pi(1),t,e} = S_{\pi(1),t,e} + p_{\pi(1),t} \\ e = 2, 3, \ldots, l_{\pi(1)};\ t = 2, 3, \ldots, m \end{cases} \tag{4}$$

$$\begin{cases} S_{\pi(i),1,1} = C_{\pi(i-1),1,l_{\pi(i-1)}} \\ C_{\pi(i),1,1} = S_{\pi(i),1,1} + p_{\pi(1),1} \\ i = 2,3,\ldots, n \end{cases} \tag{5}$$

$$\begin{cases} S_{\pi(i),t,1} = \max\{C_{\pi(i),t-1,1}, C_{\pi(i-1),t,l_{\pi(i-1)}}\} \\ C_{\pi(i),t,1} = S_{\pi(i),t,1} + p_{\pi(i),t} \\ i = 2, 3, \ldots, n;\ t = 2, 3, \ldots, m \end{cases} \tag{6}$$

$$\begin{cases} S_{\pi(i),1,e} = C_{\pi(i),1,e-1} \\ C_{\pi(i),1,e} = S_{\pi(i),1,e} + p_{\pi(i),1} \\ i = 2, 3, \ldots, n;\ e = 2, 3, \ldots, l_{\pi(i)} \end{cases} \tag{7}$$

$$\begin{cases} S_{\pi(i),t,e} = \max\{C_{\pi(i),t-1,e}, C_{\pi(i),t,e-1}\} \\ C_{\pi(i),t,e} = S_{\pi(i),t,e} + p_{\pi(i),t} \\ i = 2, 3, \ldots, n;\ e = 2, 3, \ldots, l_{\pi(i)};\ t = 2, 3, \ldots, m \end{cases} \tag{8}$$

$$\min f_1(\pi) = \min(C_{\max}(\pi)) = C_{\pi(n),m,l_{\pi(n)}} \tag{9}$$

$$\min f_2(\pi) = \min(\mathrm{Idle}_{\max}(\pi)) = \sum_{t=1}^{m} \left(C_{\pi(n),t,l_{\pi(n)}} - \sum_{i=1}^{n} \sum_{l-1}^{l_{\pi(i)}} p_{\pi(i),t} \right) \tag{10}$$

$$\min f_3(\pi) = \min(\mathrm{TFT}_{\max}(\pi)) = \mathrm{TFT1} + \mathrm{TFT2} = \sum_{i=1}^{n} C_{\pi(i),m,l_{\pi(i)}} \tag{11}$$

$$\min f_4(\pi) = \max\left(0,\ d_i - C_{\pi(i),m,l_{\pi(i)}}\right) \tag{12}$$

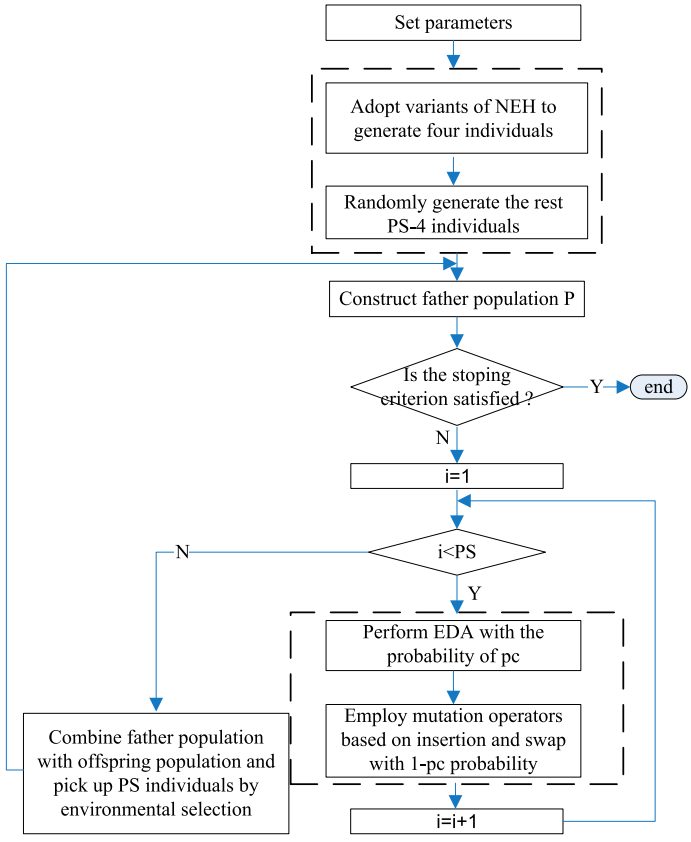

Figure 3. Flowchart of the proposed algorithm.

Procedure Initialization()
Input:
Parameters: k refers to the number of jobs in a subsequence. f_j refers to the jth objective, j={1,2,3,4}.
set: k=0, $\pi^*=\Phi$; j=1;$f_j(\pi^*)=\Phi$.
Output: initial population
Begin
Step 1. Computer the total time, $T_i = \sum_{i=1}^{m} C_{\pi(i),1,l_{\pi(i)}}$, i={1,2,⋯,$n$} on all machines for each job according to equations 1-8, and obtain a sequence, $\pi=\{\pi(1),\pi(2),\cdots,\pi(n)\}$, by sorting jobs according to the total processing time in a descendant order.
Step 2. Take the first two jobs from π , evaluate the values of f_j of two possible subsequences, and select the better as the current sequence π^*. Set k=2.
Step 3. Set k=k+1, take the kth job from π, and obtain k subsequences by inserting it into k possible positions of the current sequence π^*, then select the subsequence with the minimal $f_j(\pi^*)$ as the current sequence π^*.
Step 4. If k<n, go to Step 3; otherwise, let $\pi_i=\pi^*$, and go to Step 5.
Step 5. Set j=j+1. If j<4, go to Step 2; otherwise, obtain four sequences, π_1, π_2, π_3, π_4, and go to Step 6.
Step 6. the rest different PS-4 individuals are randomly generated in the search space.
End

Figure 4. Pseudo code of the initializing population.

```
for i = 1 to PS
    if( rand() < pc )
        Perform EDA to generate the offspring
    else
        Take out a parent individual and perform the mutation operation based on the insertion and
        swap operators
    end if
end for
```

Figure 5. Pseudo code of the generating offspring.

3. The proposed NSGA-II algorithm

Although individuals comparison, non-dominated sorting and selection operator of the exiting NSGA-II algorithms for multi-objective flow shop problem have been modified and showed the superiority to generate a good offspring, two critical operators, that is, crossover and mutation, have not been further researched. Traditional crossover and mutation operators are of randomness and aimlessness and cannot guarantee to generate offspring with a high quality, which reduces the convergence rate and influences the algorithm's efficiency. Therefore, an improved NSGA-II algorithm imbedding EDA is adopted in this paper to enhance the diversity, speed up the convergence of the population and to seek for a Pareto-optimal set U^* to minimise the above four objectives in the LSFS scheduling problem. The flowchart of the proposed algorithm is illustrated in Figure 3, and its detailed process is stated as follows.

3.1 *Initialising population*

Good seeds can generate candidates of the optimisation problem with high quality, which improves the efficiency of the whole algorithm. A rapid convergence can be obtained if a candidate makes one of the objectives minimal in an initial population for solving the multi-objective optimisation problem. Therefore, the above idea is employed in this study, that is, four variants of well-known NEH heuristics, called vNEH, are adopted to minimise makespan, the total flow time, the idle time of machines and the earliness time, respectively. It is worth noting that candidates in the proposed algorithm are represented as discrete job permutations. Let *PS* be the population size. Four solutions, $\pi_1, \pi_2, \pi_3, \pi_4$, can be yielded by performing the proposed vNEH, and the rest *PS-4* individuals in the search space are randomly generated. The details of vNEH are stated in Figure 4.

To simply illustrate the aforementioned steps, an example for generating good seeds is given here. Suppose that there are six jobs, denoted as $\pi(i) = i$, $i = \{1, 2, \ldots, 6\}$, whose numbers of associated sub-lots are $\{6, 5, 6, 3, 6, 6\}$, respectively. These jobs are processed on three machines, and their processing time is $\{12, 8, 9\}$, $\{29, 1, 26\}$, $\{23, 12, 28\}$, $\{14, 31, 24\}$, $\{16, 7, 10\}$ and $\{6, 29, 4\}$, respectively.

(1) The total processing time of each job can be calculated as $\{169, 318, 353, 238, 216, 364\}$ according to Equations (1)–(8); then, the sequence in the descendant order is $\pi = \{6, 3, 2, 4, 5, 1\}$.
(2) The first two jobs in π, that is, 6 and 3, are first taken; and their two possible permutations, that is, $\{3, 6\}$ and $\{6, 3\}$, are evaluated with the first objective, named makespan, as $f_1(\{3, 6\}) = 328$, $f_1(\{6, 3\}) = 360$. The subsequence with a smaller makespan is selected as the current sequence, π^*, and therefore, $\pi^* = \{3, 6\}$.
(3) Let $k = 3$, the third job in π, that is, 2, is taken and three possible permutations, that is, $\{2, 3, 6\}$, $\{3, 2, 6\}$ and $\{3, 6, 2\}$, are evaluated with objective f_1 as $f_1(\{2, 3, 6\}) = 473$, $f_1(\{3, 2, 6\}) = 467$ and $f_1(\{3, 6, 2\}) = 458$, and the best subsequence is chosen as the current one, π^*, and therefore, $\pi^* = \{3, 6, 2\}$.
(4) Similarly, the rest jobs are yielded as follows:

$k = 4$, $f_1(\{4, 3, 6, 2\}) = 500$, $f_1(\{3, 4, 6, 2\}) = 553$, $f_1(\{3, 6, 4, 2\}) = 571$, $f_1(\{3, 6, 2, 4\}) = 530$. $\pi^* = \{4, 3, 6, 2\}$;

$k = 5$, $f_1(\{5, 4, 3, 6, 2\}) = 596$, $f_1(\{4, 5, 3, 6, 2\}) = 596$, $f_1(\{4, 3, 5, 6, 2\}) = 591$, $f_1(\{4, 3, 6, 5, 2\}) = 563$, $f_1(\{4, 3, 6, 2, 5\}) = 560$, $\pi^* = \{4, 3, 6, 2, 5\}$;

$k = 6$, $f_1(\{1, 4, 3, 6, 2, 5\}) = 632$, $f_1(\{4, 1, 3, 6, 2, 5\}) = 632$, $f_1(\{4, 3, 1, 6, 2, 5\}) = 628$, $f_1(\{4, 3, 6, 1, 2, 5\}) = 618$, $f_1(\{4, 3, 6, 2, 1, 5\}) = 614$, $f_1(\{4, 3, 6, 2, 5, 1\}) = 614$, $\pi^* = \{4, 3, 6, 2, 1, 5\}$;

Lastly, the complete sequence with the smallest makespan is obtained, and therefore, $\pi_1 = \pi^* = \{4, 3, 6, 2, 1, 5\}$.

(5) Set j = 2, 3, 4, respectively. The rest good seeds with the smallest total flow time, the idle time and the earliness time are generated, respectively, that is, $\pi_2 = \{6,5,1,3,4,2\}$, $\pi_3 = \{4,1,5,3,6,2\}$ and $\pi_4 = \{4,1,5,2,3,6\}$.

3.2 *Generate the offspring*

Crossover and mutation operators play an important role, whose contribution is that the offspring inherits from the excellent farther chromosomes. Due to the multi-objective optimisation problem has many non-dominated solutions with a high quality, taking full advantage of the valuable information of non-dominated solutions will lead the population towards the Pareto-optimal front. The merit of EDA lies in that it can utilise the information of non-dominated solutions to construct a probabilistic model and then estimate the probability distribution of good chromosomes to build M offspring. To this end, EDA is adopted to generate offspring instead of the traditional crossover operator in this paper.

In addition, Insertion, swap and inverse operators are commonly used to produce a promising neighbouring solution, which can enhance the exploitation of an algorithm by slightly disturbing the current solution. Given the insertion and swap operators are superior to the inverse one (Wang and Zheng 2003), the former two are considered as the mutation operators in this paper. The detailed process is illustrated in Section 3.2.2.

To sum up, in the light of the performance of EDA in exploration and that of the insertion and swap operators in exploitation, EDA with the probability of *pc* and the insert and swap operators with 1 – *pc* probability are employed to generate the offspring, with the purpose of leading the population towards the Pareto-optimal front during the evolution. The general framework of generating the offspring is given as follows (Figure 5).

3.2.1 *EDA*

The detailed description of EDA for LSFS scheduling problem is given as follows (Pan and Ruiz 2012):

Select *PS* promising solutions to put into the candidate population $[\eta_{i,j}]_{PS\times n}$. All non-dominated solutions obtained at each iteration are considered as the candidate individuals. If the number of non-dominated solutions, *Q,* is equal to *PS*, then put all these non-dominated solutions into $[\eta_{i,j}]_{PS\times n}$; if Q is less than *PS*, then the rest $PS - Q$ individuals are selected by using tournament selection with size 2 to conduct the candidate population as well as Q non-dominated solutions.

Build a probabilistic model $[\xi_{i,j}]_{n\times n}$ whose aim is to improve the algorithm's efficiency and effectiveness for optimising the LSFS scheduling problem (Pan and Ruiz 2012). The probabilistic model obtained by Han et al. (2012) is

```
Initialize [ρ_{i,j}]_{n×n} = 0 ;  [β_{j',j}]_{n×n} = 0

  for i=1 to n
    for j=1 to PS
      ρ[i][η[j][i]] = ρ[i][η[j][i]] + 1 ;
    endfor
  endfor
  for j=1 to PS
    for i=1 to job-1
      if(j!=i)
        β[η[j][i]][η[j][i+1] = β[η[j][i]][η[j][i+1] + 1 ;
      endif
    endfor
  endfor
```

Figure 6. Pseudo code of the generating elements of two matrixes.

adopted in this paper. According to information of the candidate population $[\eta_{i,j}]_{PS\times n}$, two matrixes, named $[\rho_{i,j}]_{n\times n}$ and $[\beta_{i,j}]_{n\times n}$, are established based on the order of jobs in the permutation and the similar blocks of jobs, respectively.

$$[\rho_{i,j}]_{n\times n}=\begin{bmatrix}\rho_{1,1} & \rho_{1,2} & \cdots & \rho_{1,n}\\ \rho_{2,1} & \rho_{2,2} & \cdots & \rho_{2,n}\\ \cdots & \cdots & \cdots & \cdots\\ \rho_{n,1} & \rho_{n,2} & \cdots & \rho_{n,n}\end{bmatrix}\quad [\beta_{j',j}]_{n\times n}=\begin{bmatrix}\beta_{1,1} & \beta_{1,2} & \cdots & \beta_{1,n}\\ \beta_{2,1} & \beta_{2,2} & \cdots & \beta_{2,n}\\ \cdots & \cdots & \cdots & \cdots\\ \beta_{n,1} & \beta_{n,2} & \cdots & \beta_{n,n}\end{bmatrix}$$

where $\rho_{i,j}$ is the number of times that job j appears in position i in the selected sequences, and $\beta_{j',j}$ the number of times that job j appears immediately after job j'. The elements of $[\rho_{i,j}]_{n\times n}$ and $[\beta_{i,j}]_{n\times n}$ can be obtained according to Figure 6.

The probability of each job, $\xi_{i,j}$, in sequence π is calculated according to the following equation:

$$\xi_{i,j}=\begin{cases}\dfrac{\rho_{i,j}}{\sum_{t\in\mu(i)}\rho_{i,t}} & i=1\\ \dfrac{\dfrac{\rho_{i,j}}{\sum_{t\in\mu(i)}\rho_{i,t}}+\dfrac{\beta_{j',j}}{\sum_{t\in\mu(i)}\beta_{j',t}}}{2} & i=2,3,\ldots,n\end{cases}\tag{13}$$

where $\mu(i)$ is the unscheduled sequence set, and i the position that job j appears in the sequence.

Generate new offspring. Let the new solution be empty, that is, $\pi_{\text{new}}=\phi$. First, regard the ith sequence of the original population as the unscheduled sequence $\mu(i)$, and then, the first job of the new sequence, π_{new}, is obtained as follows. Randomly take 5 jobs from the unscheduled sequence and compute their probabilities according to Equation (13). Followed that the job with the largest probability among the 5 jobs is picked up and viewed as the first job of the new sequence. Second, set $i=i+1$, and a new subsequence $\mu(i)$ is constructed by deleting the selected job from $\mu(i-1)$. Calculate the probability of each job in $\mu(i)$ according to Equation (13), and put the job with the largest probability into the ith position of π_{new}. Repeat the second step until π_{new} is a legal and complete sequence.

In this section, an example for constructing the new offspring is given as follows. Suppose that there are 7 selected sequences in the candidate population, $[\eta_{i,j}]_{PS\times n}$ ($PS=7$), and each has 7 jobs. The candidate population, $[\rho_{i,j}]_{n\times n}$ and $[\beta_{i,j}]_{n\times n}$, are given as follows:

$$[\eta_{i,j}]_{7\times 7}=\begin{bmatrix}2&6&1&4&7&5&3\\7&2&5&1&4&4&6\\3&2&7&6&5&4&1\\1&4&7&2&6&5&3\\2&3&4&7&6&5&1\\5&3&7&6&4&1&2\\4&1&7&5&3&2&6\end{bmatrix}\quad [\rho_{i,j}]_{7\times 7}=\begin{bmatrix}1&2&1&1&1&0&1\\1&2&2&1&0&1&0\\1&0&0&1&1&0&4\\1&1&0&1&1&2&1\\0&0&1&2&1&2&1\\1&1&1&1&3&0&0\\2&1&2&0&0&2&0\end{bmatrix}\quad [\beta_{j',j}]_{7\times 7}=\begin{bmatrix}-&1&0&3&0&0&1\\0&-&1&0&1&3&1\\0&2&-&1&0&1&1\\3&0&1&-&0&0&1\\2&0&4&1&-&0&0\\1&0&0&1&3&-&0\\0&2&0&0&2&3&-\end{bmatrix}$$

Let $\mu(1)=\{1,2,3,4,5,6,7\}$, $\pi_{\text{new}}=\phi$

(a) Let $i=1$, randomly select 5 jobs from $\mu(1)$, for example, 2, 3, 5, 6, 7, and compute their probabilities.

$$\xi_{1,2}=2/(1+2+1+1+1+0+1)=0.286;$$

$$\xi_{1,3}=1/(1+2+1+1+1+0+1)=0.143;$$

$$\xi_{1,5}=1/(1+2+1+1+1+0+1)=0.143;$$

$$\xi_{1,6}=0/(1+2+1+1+1+0+1)=0;$$

$$\xi_{1,7}=1/(1+2+1+1+1+0+1)=0.143;$$

The job with the largest probability among the 5 jobs is picked up, that is, $\pi_{\text{new}}=\{2\}$.

(b) Let $i=i+1$, the unscheduled subsequence $\mu(i)$ consists of the remaining of $\mu(i-1)$, that is, $\mu(i)=\{1,3,4,5,6,7\}$. Next, calculate the probability of each job in $\mu(i)$:

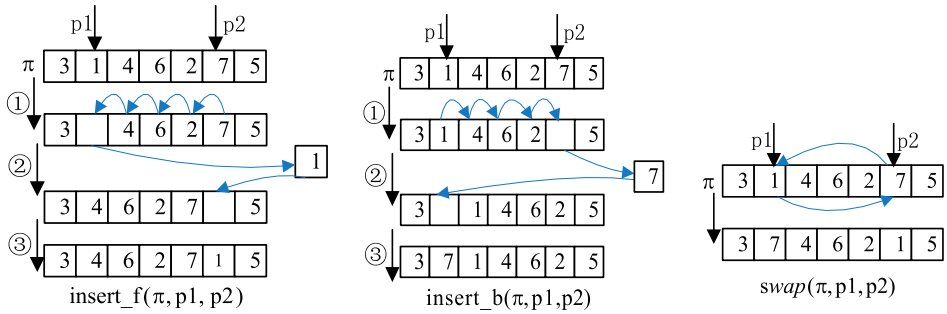

Figure 7. *Insert_f*$(\pi, p1, p2)$, *insert_b*$(\pi, p1, p2)$ and *swap*$(\pi, p1, p2)$.

$$\xi_{2,1} = (1/(1+2+1+0+1+0)+0/(0+1+0+1+3+1)) = 2 = 0.100;$$

$$\xi_{2,3} = (2/(1+2+1+0+1+0)+1/(0+1+0+1+3+1)) = 2 = 0.283;$$

$$\xi_{2,4} = (1/(1+2+1+0+1+0)+0/(0+1+0+1+3+1)) = 2 = 0.100;$$

$$\xi_{2,5} = (0/(1+2+1+0+1+0)+1/(0+1+0+1+3+1)) = 2 = 0.083;$$

$$\xi_{2,6} = (1/(1+2+1+0+1+0)+3/(0+1+0+1+3+1)) = 2 = 0.375;$$

$$\xi_{2,7} = (0/(1+2+1+0+1+0)+1/(0+1+0+1+3+1)) = 2 = 0.083;$$

Put the job with the largest probability into π_{new}, that is, $\pi = \{2, 6\}$. If $i < n$ ($n = 7$), go to step (b); otherwise, stop the process.

3.2.2 *Mutation*

The *insert*operator has two ways, named forward insertion, *insert_f*, and backward insertion, *insert_b*. As shown in Figure 7, *insert_f* randomly selects two different positions, $p1$ and $p2$. If $p1 < p2$, all jobs between position $p1 + 1$ and $p2$ move forward a position in turn. At last, put the original job in position $p1$ into position $p2$. As for *insert_b*, randomly select two different positions, $p1$ and $p2$. When $p1 < p2$, all jobs between position $p1 + 1$ and $p2$ move backward a position in turn. With respect to the swap operator, randomly select two different positions from the sequence, and interchange their corresponding jobs.

Insert and swap operators are commonly used to generate neighbouring solutions for flow shop scheduling problems (Wang 2003) and have been demonstrated as superior to the inverse neighbourhood (Ruiz and Thomas 2008). In this section, six strategies based on insert and swap operators are proposed on this account: (1) perform *insert_f* once; (2) apply *insert_b* one time; (3) conduct the swap operator once; (4) employ *insert_f* twice; (5) use *insert_b* two times; and (6) execute the swap operator twice.

Generally speaking, more strategies generate different solutions with a larger probability than a single strategy and avoid the population trapping in local optima, we randomly chose one of the above six strategies to generate a new offspring.

3.3 *Environmental selection*

The environmental selection of NSGA-II mainly includes constructing a non-dominated set and picking up PS individuals. To speed up the convergence and maintain the diversity of the population, the selection operator based on the crowed distance developed by Deb et al. (2002) is utilised, and its detailed steps are given in Figure 8.

3.4 *Restart strategy*

The diversity of the population may diminish at certain generations, and many job sequences in the population may become very similar, which leads to the stagnation of the population's evolution. To overcome the above problem, when

Begin

Step 1. Initialize variables

num[r]=0, *r*=1, f=false, F={F_1,F_2,…} % The array *num* denotes the number of non-dominated solutions, and *r* the rank of each solutions; variable *f* flags whether the solution has been selected or not. F is a bounded set to save the solutions of each rank.

Step 2. Combine the father with offspring populations and sort non-dominated solutions

while(the flags of all solutions are equal to false) **do**

$F_r = \phi$;

for *i*=1 **to** PS+PS

if (π_i.f == false and $\pi_i \prec \forall \pi_j$) $j \in \{1,2,...,PS+PS\} \cap j \neq i$ **then**

The rank of π_i is equal to *r,* that is, π_i.*rank*=*r*;

$F_r = F_r \cup \pi_i$

num[r]=num[r]+1;

π_i.f = true;

end if

end for

r=r+1;

end do

Step 3. Set $P = \phi$,r=1;

Step 4. Select PS individual as new farther population

while($|P| \neq PS$) **do**

if(num[r]+|P|<PS) **then**

$P = P \cup F_r$ r=r+1;

else if(num[r]+|P|>*PS*)

The bounded sets F_r are taken and sorted according to makespan in a descendent order. Compute the crowded distance of each solution in F_r according to following equation:

$$distance(\pi_i) = \sum_{i=2}^{N-1}\sum_{k=1}^{4}(f_{i+1,k} - f_{i-1,k}) \quad (14)$$

The $|PS - num[r]|$ solutions with the largest crowding distance will be picked up and put in P; **break**;

else

$P = P \cup F_r$; **break**;

end if

end do

End

Figure 8. Pseudo code of environmental selection.

the diversity falls below a given threshold, γ, adopting a restart strategy based on the ideas adopted by the authors in (Pan and Ruiz 2012 and Ruiz et al. 2006) is much more necessary. The detailed steps are stated as follows:

Step 1: Two matrixes, $[\rho_{i,j}]_{n\times n}$ and $[\beta_{i,j}]_{n\times n}$, are constructed using the same method as in Section 3.2.1 with the exception that $[\rho_{i,j}]_{n\times n}$ and $[\beta_{i,j}]_{n\times n}$ record information of the current population.

Step 2: Count the number of elements larger than zero in $[\rho_{i,j}]_{n\times n}$ and denote it as θ.

Step 3: Count the number of elements larger than zero in $[\beta_{i,j}]_{n\times n}$ and denote it as η.

Step 4: Compute the diversity of the current population according to the following formula presented by the authors in (Pan and Ruiz 2012).

$$\text{diversity } (P) = \left(\frac{\theta - n}{n \times \min(PS, n)} + \frac{\eta - (n-1)}{(n-1) \times \min(n-1, PS-1)}\right)/2 \tag{15}$$

Step 5: If diversity $(P) < \gamma$, put the 40% non-dominated solutions chosen from the current non-dominated set into the current population, and the rest are randomly generated.

In this section, two simple examples about the diversity are given. Suppose that there are four sequences in the current population, P. The population and the two matrixes are shown as follows:

Example 1:

$$P = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix} \quad [\rho]_{4\times4} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \quad [\beta]_{4\times4} \begin{bmatrix} 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\text{diversity } (P) = \left(\frac{4-4}{4 \times \min(4,4)} + \frac{3-(4-1)}{(4-1) \times \min(4-1, 4-1)}\right)/2 = 0$$

Example 2:

$$P = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 2 & 1 \\ 4 & 3 & 1 & 2 \end{bmatrix} \quad [\rho]_{4\times4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad [\beta]_{4\times4} = \begin{bmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

$$\text{diversity } (P) = \left(\frac{16-4}{4 \times \min(4,4)} + \frac{8-(4-1)}{(4-1) \times \min(4-1, 4-1)}\right)/2 = 0.65$$

It can be seen from the above examples that the diversity is in the range of zero to one. When the sequences of the population are every similar or identical, the diversity trends or equals to zero, as shown in Example 1. However, as shown in Example 2, all sequences of the population are different, that is, different jobs occupy different positions, and no similar job blocks exist among these sequences, so the diversity trends to one. It can be observed that the larger the diversity, the more diverse the population. Another mentionable thing is that the diversity is computed when the number of iterations is larger than or equal to 100, which can reduce the computation overhead.

3.5 *Summarising the proposed algorithm*

With the above approaches, the proposed algorithm can be summarised as follows:

Step 1: Set the values of such parameters as *PS* (the population size), *pc* (the crossover probability), γ (the diversity threshold) and *time* (the maximal computation time), and let $t = 0$ (the number of iterations).

Step 2: Initialise the population using the method in Section 3.1, and evaluate each solution in the population according to formula 1 to 8.

Step 3: Repeat the following steps, until the termination condition is satisfied.

- Step 3.1: Generate the new offspring by adopting EDA with the probability of *pc* and the insert and swap operators with $1 - pc$ probability presented in Section 3.2.
- Step 3.2: Perform the environment selection operator based on non-dominated sorting and the crowed distance by using the method in Section 3.3.
- Step 3.3: Update the archive and father population, and save the limited non-dominated solutions obtained so far.
- Step 3.4: Let $t = t + 1$. If the number of iterations is greater than or equal to 100, compute the diversity of the current population. If the diversity is smaller than the given threshold, apply the restart strategy to initialise the current population according to Section 3.4, and set $t = 0$.

4. Evolution metric

In general, non-dominated solutions have a close relationship with the Pareto-optimal front; thus, the quality of these non-dominated solutions is very important to evaluate the performance of an algorithm. In this paper, the following four

metrics are employed to evaluate the obtained non-dominated solutions: the number of obtained non-dominated solutions, the ratio of non-dominated solutions, the average distance between the non-dominated solutions to the reference solutions, and the spread and the distribution of the obtained non-dominated solutions. Denote the non-dominated solutions set obtained by jth$(j \in \{1,2,3,4\})$ algorithm as S_j, where $S_j \in NDS\{INSGA-II, NSGA-II, DHS, TA\}$. Let S be the union of all non-dominated solution sets. It is worth mentioning that the true Pareto front is often not known for a multi-objective optimisation problem, so a reference set is used to evaluate the performance of an algorithm in the literature (Pan et al. 2009). In this work, the Pareto-optimal front of the best-so-far solutions obtained by all these algorithms is regarded as the reference set S^*.

4.1 *The number of non-dominated solutions,* $N(S_j)$

The non-dominated solutions obtained by the jth algorithm are compared with the union of all non-dominated solutions S, and the number of solutions in S_j that are not dominated by any solution in S is counted. Its definition is as follows (Pan et al. 2009):

$$N(S_j) = |x' \in S_j| \; \exists x' \prec \forall y, \quad y \in S, \; x \neq y| \quad S = \bigcup_{j=1,2,3,4} S_j \tag{16}$$

It can be seen from Equation (16) that the larger the metric $N(S_j)$, the better the jth algorithm.

4.2 *The ratio of non-dominated solutions,* $R(S_j)$

The value of $R(S_j)$, equal to the ratio of $N(S_j)$ to $|S_j|$, is used to evaluate the quality of solutions in S_j (Pan et al. 2009). The value of $R(S_j)$ is in the range of zero to one, and if $R(S_j) = 0$, it means that any solution in S_j is dominated by some solution(s) in S; when $R(S_j) = 1$, it shows that any solution in S_j is not dominated by solutions in S; with respect to $0 < R(S_j) < 1$, it refers that there exist some solutions in S_j not dominated by any solution in S. Thus, the larger the metric $R(S_j)$, the better the quality of S_j.

4.3 *The average distance between the Pareto-optimal front and solution set* S_j

The metric D(S_j) shows the shortest normalised average distances between reference solutions and the set S_j, which reflects the convergence of the solutions in the algorithm S_j towards to the Pareto-optimal front (Pan et al. 2009). The smaller the $D(S_j)$, the better approximation to the reference set. The equation is shown as follows.

$$D(S_j) = \frac{1}{|S^*|} \sum_{y \in S^*} d_y(S_j) \tag{17}$$

$$d_y(S_j) = \min_{x \in S_j} \left\{ \sqrt{\sum_{k=1}^{4} \left(\frac{f_k(x) - f_k(y)}{f_k^{\max}(\bullet) - f_k^{\min}(\bullet)} \right)^2} \right\}$$

where $f_k(\bullet)$ denotes the kth objective value; and $f_k^{\max}(\bullet)$ and $f_k^{\min}(\bullet)$ are the minimum and maximum of the kth objective value in the reference set S^*, respectively. The $|S^*|$ is the number of reference solutions in S^*.

4.4 *The spread of non-dominated solutions*

The $SC(S_j)$ is utilised to measure the performance of non-dominated solutions in distribution in the objective space (Veldhuizen and Lamont 2000). Its definition is stated as follows:

$$SC(S_j) = \sqrt{\frac{1}{|S_j| - 1} \sum_{i=1}^{|S_j|} (\bar{d} - d_i)^2} \tag{18}$$

$$d_i = \min_{j \in \{1,2,\ldots,|S_j|\}} \left(\sum_{k=1}^{4} |f_k^i(x) - f_k^j(x)| \right) \quad x \in S_j \quad i \in \{1,2,\ldots,|S_j|\} \cap i \neq j. \tag{19}$$

$$\bar{d} = \frac{1}{|S_j|}\sum_{i=1}^{|S_j|} d_i \tag{20}$$

where d_i is the Euclidean distance between x and its nearest neighbour. As can be observed from Equation (18), the variance value $SC(S_j) = 0$ reflects the uniform distribution of solutions in S_j. On the contrary, $SC(S_j)$ explains the irregular distribution of solutions in S_j. So, the smaller the value of $SC(S_j)$, the more uniform the non-dominated solutions.

5. Experiments

The experiments are conducted to evaluate the performance of the proposed algorithm in this section. All algorithms are written with C++ and implemented in a PC with Pentium(R) Dual 2.8 GHZ and 2G memory. Each instance is independently run 5 times, and its average computational time is also reported. Following Veldhuizen and Lamont (2000) and Tseng and Liao (2008), the data for the instances of the LSFS scheduling problem are given by discrete uniform distributions.

5.1 *Experimental setting*

- The population size, PS, the crossover probability, pc, and the size of the external archive are set to 20, 0.6 and 100, respectively.
- The numbers of jobs and machines for each instance are randomly chosen from the following sets $n \in \{10, 30, 50, 70, 90, 110\}$ and $m \in \{5, 10, 15, 20\}$, respectively.
- For each instance, the maximal computation time is set to $m \times n$ milliseconds.
- Set the due date of each job as $d_i = rand()\%(15 \times m + 1) + 15 \times n$.
- Let the number of sub-lots of each job $\pi(i)$ be $l_{\pi(i)} = rand()\%6 + 1$.
- The processing time of job i on machine t is set as $P_{\pi(i),t} = rand()\%31 + 1$.

Table 2. Performance of proposed vNEH strategy and random one.

Instance $n \times m$	vNEH			Random		
	$R(S_j)$ (%)	$D(S_j)$	Time (s)	$R(S_j)$ (%)	$D(S_j)$	Time (s)
10 × 5	98	0.00	0.00	33	0.72	0.00
10 × 10	60	0.28	0.00	20	0.78	0.00
10 × 15	100	0.00	0.00	25	0.62	0.00
10 × 20	0	0.20	0.00	100	0.00	0.00
30 × 5	100	0.00	0.02	25	0.13	0.00
30 × 10	100	0.00	0.01	33	0.54	0.00
30 × 15	100	0.00	0.02	60	0.12	0.00
30 × 20	100	0.00	0.02	0	1.25	0.00
50 × 5	0	0.83	0.03	100	0.00	0.00
50 × 10	100	0.00	0.05	0	1.75	0.00
50 × 15	33	0.19	0.08	33	0.00	0.00
50 × 20	0	0.66	0.09	95	0.00	0.00
70 × 5	100	0.00	0.06	0	1.12	0.00
70 × 10	100	0.00	0.14	0	0.51	0.00
70 × 15	100	0.00	0.19	81	0.95	0.00
70 × 20	100	0.00	0.30	91	0.10	0.00
90 × 5	100	0.00	0.11	50	0.34	0.00
90 × 10	80	0.04	0.28	85	0.04	0.00
90 × 15	55	0.26	0.42	55	0.06	0.00
90 × 20	100	0.00	0.58	0	1.20	0.02
110 × 5	100	0.00	0.22	0	0.12	0.00
110 × 10	100	0.00	0.48	0	0.21	0.00
110 × 15	100	0.00	0.77	0	0.07	0.00
110 × 20	91	0.01	1.05	80	0.30	0.02
Mean	80	0.10	0.21	40	0.46	0.00

Table 3. Performance of six strategies of mutation operator in $D(S_j)$ and $R(S_j)$.

Instance $n \times m$	$D(S_j)$						$R(S_j)$ (%)					
	$r1$	$r2$	$r3$	$r4$	$r5$	$r6$	$r1$	$r2$	$r3$	$r4$	$r5$	$r6$
10 × 5	0.02	0.02	0.01	0.01	0.68	0.21	69	65	87	74	87	85
10 × 10	0.00	0.00	0.01	0.01	0.34	0.04	98	98	98	100	98	98
10 × 15	0.00	0.00	0.00	0.00	0.00	0.00	88	88	87	100	100	88
10 × 20	0.00	0.00	0.00	0.00	0.07	0.03	96	96	91	96	96	96
30 × 5	0.01	0.01	0.01	0.03	1.65	2.24	63	41	50	19	25	43
30 × 10	0.01	0.01	0.01	0.01	0.97	0.92	21	17	22	8	26	27
30 × 15	0.01	0.01	0.01	0.01	2.48	7.60	49	0	62	47	16	0
30 × 20	0.32	0.32	0.25	0.06	6.09	6.20	2	30	0	15	2	100
50 × 5	0.01	0.01	0.01	0.03	1.08	3.29	63	19	11	29	6	2
50 × 10	0.04	0.04	0.02	0.02	1.69	2.84	04	11	33	35	25	65
50 × 15	0.03	0.03	0.03	0.12	1.59	1.6	49	59	37	18	87	0
50 × 20	0.05	0.05	0.43	0.02	7.63	4.07	64	46	26	69	0	48
70 × 5	0.02	0.02	0.02	0.03	2.12	0.83	86	40	7	16	43	36
70 × 10	0.17	0.17	0.11	0.06	9.07	8.16	14	4	15	64	29	23
70 × 15	0.08	0.08	0.03	0.03	2.36	7.98	0	17	100	96	10	46
70 × 20	0.09	0.09	0.03	0.07	2.00	4.81	14	11	38	10	100	61
90 × 5	0.01	0.01	0.01	0.01	2.71	0.49	99	43	16	21	92	56
90 × 10	0.02	0.02	0.03	0.01	1.62	2.33	5	22	24	4	17	75
90 × 15	0.12	0.12	0.01	0.06	4.17	11.21	56	16	80	25	40	55
90 × 20	0.07	0.07	0.08	0.13	7.51	3.10	3	7	84	42	79	9
110 × 5	0.01	0.01	0.05	0.02	2.60	3.46	50	90	19	70	51	14
110 × 10	0.01	0.01	0.03	0.01	0.79	6.14	12	1	0	25	12	87
110 × 15	0.09	0.09	0.05	0.13	0.30	4.57	53	35	42	21	20	23
110 × 20	0.00	0.00	0.00	0.00	0.00	0.00	48	69	61	2	52	72
Mean	0.05	0.05	0.06	0.04	2.48	3.42	46	39	45	42	46	50

5.2 *Experimental results*

Han (2012) developed the discrete HS (DHS) algorithm for solving the multi-objective LSFS scheduling problem, with providing better results than NEH algorithm. Marimulthu et al. (2009) presented the TA algorithm for the same problem. In this paper, the proposed algorithm (INSGA-II, for short) is compared with these existing algorithms in the literature, that is, DHS, TA and the basic NSGA-II. Meanwhile, all parameters commonly used in these algorithms are set the same values as those in their original paper. The experimental results are shown in the following tables and figures.

5.2.1 *Performance of initialisation strategies*

To evaluate the performance of the proposed vNEH, Table 2 reports the performance of vNEH and the random strategy on the premise of the same parameters, where the column of 'vNEH' represents the data corresponding to the proposed vNEH strategies and that of 'Random' refers to the data corresponding to randomly initialising the population.

As can be observed from Table 2, with respect to the ratio of the non-dominated solutions and the distance between the reference set and the solution set, there are 80% non-dominated solutions yielded by the proposed vNEH strategies, and the distance is 0.1, whereas those obtained by the random strategy are 40% and 0.46, respectively, on the premise of the same computational time. So, the proposed initialisation strategies can generate solutions with high quality and accelerate the convergence of the population.

5.2.2 *Performance of proposed mutation operator*

To evaluate the performance of the six strategies, Table 3 lists the data of such indicators as the ratio of non-dominated solutions and the distance between the reference set and the solution set, where 'rm' ($m = 1, 2, 3, 4, 5, 6$) represents the data corresponding to the mth strategy of the proposed mutation operator.

From Table 3, (1) for the indicator of $D(S_j)$, the value obtained by using the fourth strategy is smaller than those of the rest strategies, followed by the value of $r1$ and $r2$. The sixth strategy is the worst, suggesting that the insert operator has a good performance in convergence; (2) with respect to the indicator of $R(S_j)$, the value obtained by using the *six*th strategy is larger than those of the rest strategies, followed by the value of $r1$ and $r5$, indicating that the swap operator

Table 4. The number of solutions in S_j obtained by INSGA-II, NSGA-II, DHS and TA algorithms.

Instance	INSGA-II			NSGA-II			DHS			TA		
$n \times m$	MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG
10 × 5	29.0	40.0	34.5	9.0	19.0	14.0	61.0	65.0	63.0	27.0	27.0	27.0
10 × 10	23.0	27.0	25.0	11.0	16.0	13.5	21.0	25.0	23.0	23.0	23.0	23.0
10 × 15	34.0	34.0	34.0	6.0	15.0	10.5	26.0	28.0	27.0	13.0	13.0	13.0
10 × 20	31.0	31.0	31.0	6.0	13.0	9.5	32.0	35.0	33.5	14.0	14.0	14.0
30 × 5	100	100	100	10.0	18.0	14.0	23.0	26.0	24.5	34.0	34.0	34.0
30 × 10	95.0	100	97.5	7.0	17.0	12.0	13.0	18.0	15.5	22.0	22.0	22.0
30 × 15	96.0	100	98.0	11.0	16.0	13.5	18.0	19.0	18.5	55.0	55.0	55.0
30 × 20	92.0	100	96.0	11.0	16.0	13.5	12.0	13.0	12.5	23.0	23.0	23.0
50 × 5	100	100	100	9.0	15.0	12.0	12.0	13.0	12.5	41.0	41.0	41.0
50 × 10	99.00	100	99.5	14.0	16.0	15.0	8.0	8.0	8.0	35.0	35.0	35.0
50 × 15	100	100	100	13.0	17.0	15.0	10.0	10.0	10.0	32.0	32.0	32.0
50 × 20	93.0	100	96.5	13.0	18.0	15.5	11.0	12.0	11.5	24.0	24.0	24.0
70 × 5	95.0	100	97.5	12.0	16.0	14.0	4.0	4.0	4.0	43.0	43.0	43.0
70 × 10	99.0	100	99.5	13.0	18.0	15.5	7.0	7.0	7.0	38.0	38.0	38.0
70 × 15	99.0	100	99.5	14.0	18.0	16.0	7.0	7.0	7.0	45.0	45.0	45.0
70 × 20	100	100	100	12.0	18.0	15.0	7.0	7.0	7.0	25.0	25.0	25.0
90 × 5	100	100	100	11.0	16.0	13.5	4.0	4.0	4.0	39.0	39.0	39.0
90 × 10	100	100	100	15.0	17.0	16.0	4.0	4.0	4.0	39.0	39.0	39.0
90 × 15	100	100	100	15.0	17.0	16.0	4.0	4.0	4.0	69.0	69.0	69.0
90 × 20	88.0	100	100	9.0	15.0	12.0	5.0	5.0	5.0	30.0	30.0	30.0
110 × 5	100	100	94.0	11.0	16.0	13.5	4.0	4.0	4.0	28.0	28.0	28.0
110 × 10	100	100	100	13.0	17.0	15.0	5.0	5.0	5.0	40.0	40.0	40.0
110 × 15	100	100	100	12.0	18.0	15.0	4.0	4.0	4.0	56.0	56.0	56.0
110 × 20	100	100	100	10.0	14.0	12.0	5.0	5.0	5.0	42.0	42.0	42.0
Mean	86.4	88.8	87.6	11.2	16.5	13.8	12.8	13.8	13.3	34.9	34.9	34.9

Table 5. Number of non-dominated solutions $N(S_j)$, $R(S_j)$ and $D(S_j)$ of the INSGA-II, NSGA-II, DHS and TA algorithms.

Instance	$N(S_j)$				$R(S_j)$ (%)				$D(S_j)$			
$n \times m$	INSGA-II	NSGA-II	DHS	TA	INSGA-II	NSGA-II	DHS	TA	INSGA-II	NSGA-II	DHS	TA
10 × 5	25	0	36	0	72	0	57	0	0.01	0.18	0.05	0.27
10 × 10	15	0	17	0	60	0	74	0	0.02	0.48	0.09	0.09
10 × 15	34	0	20	0	100	0	74	0	0.05	2.37	0.01	0.45
10 × 20	26	0	22	0	84	0	66	0	0.00	0.73	0.04	0.49
30 × 5	90	0	12	0	90	0	49	0	0.05	1.16	0.03	0.58
30 × 10	100	0	8	0	100	0	52	0	0.02	2.54	0.8	3.51
30 × 15	92	0	14	0	94	0	76	0	0.03	4.07	0.48	3.56
30 × 20	100	0	5	0	100	0	40	0	0.03	5.89	0.91	4.82
50 × 5	100	0	9	2	100	0	72	5	0.03	1.13	0.05	0.79
50 × 10	79	0	6	0	79	0	75	0	0.00	3.14	0.18	3.99
50 × 15	95	0	3	0	95	0	30	0	0.02	1.69	0.04	1.40
50 × 20	100	0	1	0	100	0	9	0	0.01	4.09	0.86	4.16
70 × 5	100	2	3	0	100	14	75	0	0.02	0.52	0.12	0.94
70 × 10	98	0	6	0	98	0	86	0	0.00	4.96	0.08	7.06
70 × 15	100	0	1	0	100	0	14	0	0.00	4.35	0.12	4.68
70 × 20	100	0	6	0	100	0	86	0	0.01	3.72	0.14	3.61
90 × 5	100	0	4	0	100	0	100	0	0.01	3.56	0.10	5.40
90 × 10	100	0	4	0	100	0	100	0	0.02	1.78	0.08	1.78
90 × 15	99	0	4	1	99	0	100	1	0.02	1.42	0.09	1.55
90 × 20	100	0	4	4	100	0	80	13	0.06	0.95	0.13	0.80
110 × 5	73	0	1	0	78	0	25	0	0.01	3.70	0.46	4.40
110 × 10	45	0	4	1	45	0	80	3	0.08	1.43	0.07	1.30
110 × 15	66	0	4	0	66	0	100	0	0.03	2.03	0.59	3.89
110 × 20	100	0	5	1	100	0	100	2	0.03	1.02	0.17	1.07
Mean	80.7	0.1	8.3	0.4	91	1	67	1	0.02	2.37	0.24	2.52

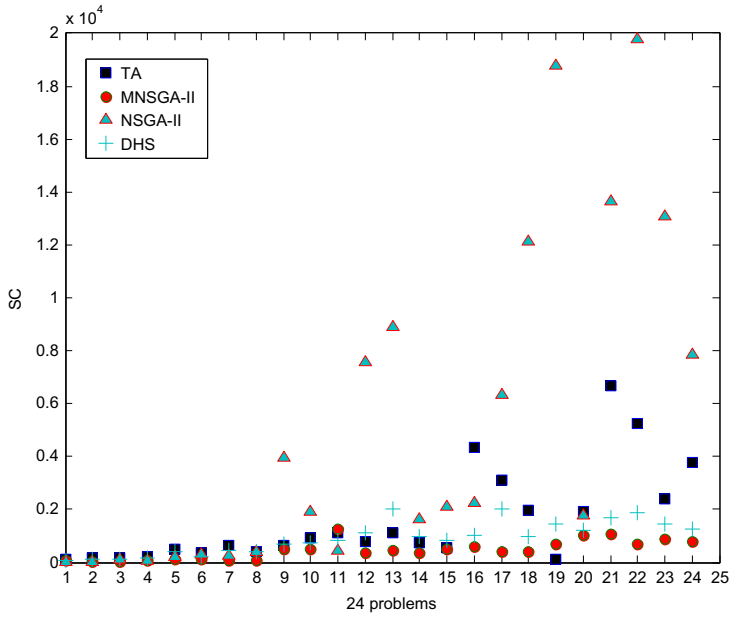

Figure 9. Scatter $SC(S_j)$ of INSGA-II, NSGA-II, DHS and TA algorithms.

can generate more non-dominated solutions than the insert operator. Since insert and swap operators show good performances from various indicators, we randomly chose one of these six strategies to generate a promising offspring.

5.2.3 *Performance of INSGA-II, NSGA-II, DHS and TA algorithms*

The number of solutions in S_j, $N(S_j)$, $R(S_j)$, $D(S_j)$ and $SC(S_j)$ produced by INSGA-II, NSGA-II, DHS and TA algorithms is reported in Tables 4 and 5, respectively. The bottom rows of these tables give the mean values of the corresponding metrics.

(1) Table 4 presents the average (AVG), the minimal (MIN) and the maximal (MAX) numbers of solutions in S_j obtained by INSGA-II, NSGA-II, DHS and TA, respectively, in 5 independent runs. It can be clearly found from Table 4 that the overall AVG (86.4), MAX (88.8) and MIN (87.6) yielded by INSGA-II are much better than those generated by NSGA-II, DHS and TA algorithms in the same computation time. The reason why INSGA-II is better than the other algorithms is that the proposed algorithm can take full of the non-dominated solutions to generate excellent offspring and the insertion and swap operators to disturb old individuals.

(2) The superiority of INSGA-II over NSGA-II, DHS and TA algorithms is further demonstrated by Table 5 in terms of $N(S_j)$ and $R(S_j)$. Combining Tables 4 and 5, in the average, 91% non-dominated solutions yielded by INSGA-II are not dominated by any other solution in S, significantly more than those obtained by NSGA-II, DHS and TA algorithms. Especially, for large size instances, such as 70×15, 90×5, 90×10, 110×20, the value of $R(S_j)$ generated by INSGA-II is equal to 100%. Thus, it can be concluded that the proposed algorithm yields more non-dominated solutions with high quality than the comparative ones.

(3) As mentioned before, due to the true Pareto front of an optimisation problem is difficult to find, a reference solution set is necessary to compute the performance measure $D(S_j)$. Table 5 reports the shortest distance, $D(S_j)$, between the achieved non-dominated solutions and the reference set. Within the same computational time, the average distance produced by INSGA-II is 0.02, whereas those obtained by NSGA-II, DHS and TA algorithms are 2.37, 0.24 and 2.52, respectively. Therefore, with respect to the average distance, it is evident that INSGA-II performs better than the comparative ones for all instances.

(4) Figure 9 gives a more intuitive illustration on the spread of neighbourhood vectors of achieved non-dominated solutions. As Figure 8 shows, for almost all instances, the non-dominated solutions generated by INSGA-II are uniformly distributed. It is worth noting that the spread of NSGA-II is highly not uniform. The reason is that NSGA-II is suitable for solving continuous optimisation problems, but hardly for generating discrete values. So, NSGA-II is less effective and efficient for the LSFS scheduling problem. Based on the data of

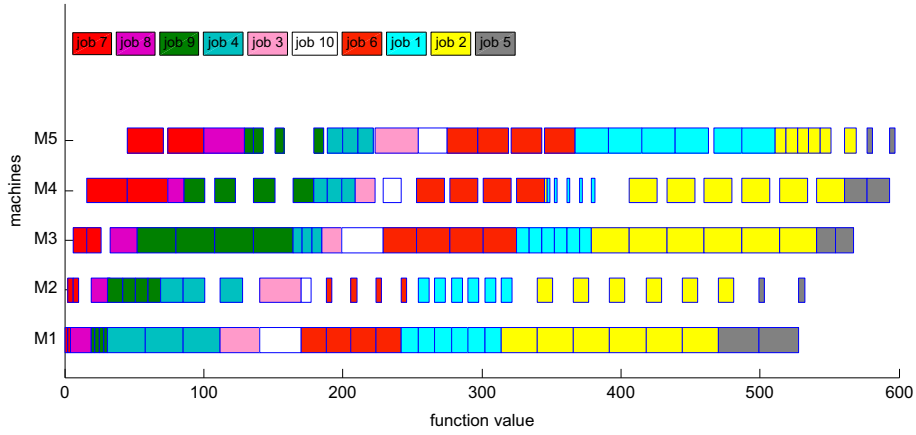

Figure 10. Optimal solution of the 10 × 5 instance ($f_1 = 597$, $f_2 = 541$, $f_3 = 3210$, $f_4 = 138$) obtained by INSGA-II.

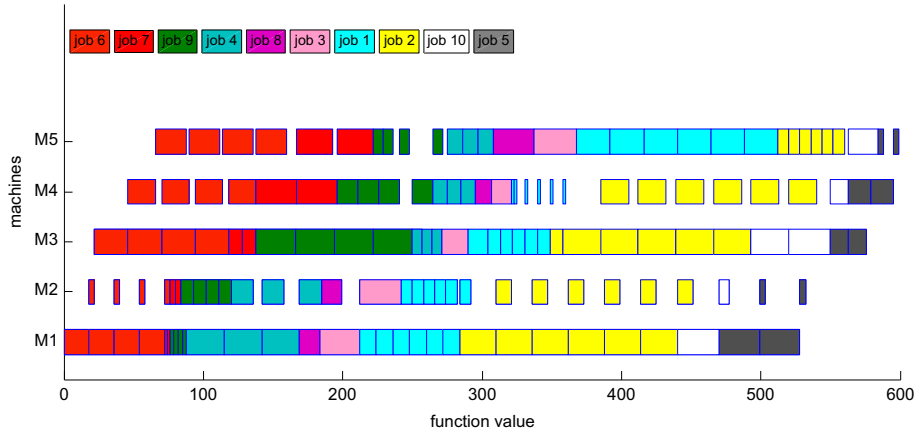

Figure 11. Optimal solution of the 10 × 5 instance ($f_1 = 599$, $f_2 = 554$, $f_3 = 3922$, $f_4 = 18$) obtained by INSGA-II.

Table 5 and Figure 9, the superior performance of the proposed algorithm in convergence and distribution can be justified.

(5) In addition, Gantt charts of the two test instances, 10 × 5 and 10 × 20, are showed in Figures 10, 11 and 12 in the light of the non-dominated solutions obtained by INSGA-II, respectively. Figures 9 and 10 show two scheduling results of the 10 × 5 instance, corresponding to the following two optimal solutions $\{7, 8, 9, 4, 3, 10, 6, 1, 2, 5\}$ and $\{6, 7, 9, 4, 8, 3, 1, 2, 10, 5\}$. The due dates of job i, $i \in \{1, 2, \ldots, 10\}$, are $\{191, 218, 189, 170, 205, 178, 199, 168, 156, 157\}$, and the numbers of sub-lots of these jobs are $\{2, 1, 4, 3, 1, 1, 4, 6, 6, 2\}$. Figure 12 illustrates the Gantt chart of the 10 × 20 instance, corresponding to the optimal solution $\{1, 6, 8, 10, 5, 9, 4, 3, 2, 7\}$. The due dates of job i, $i \in \{1, 2, ..., 10\}$, are $\{191, 442, 410, 388, 315, 357, 346, 441, 417, 335\}$, and the numbers of sub-lots of these jobs are $\{6, 5, 5, 4, 1, 1, 6, 3, 2, 2\}$.

In summary, Tables 4 and 5 give us a clear illustration that the non-dominated solutions obtained by INSGA-II have better quality. The superiority of the proposed algorithm attributes to EDA and the insertion and swap operators, by which its capabilities in exploration and exploitation are improved.

5.2.4 *Wilcoxon two-sided rank sum test*

Table 6 reports the two-side Wilcoxon rank sum tests of null hypothesis of INSGA-II, NSGA-II, DHS and TA algorithms with significant level of 5%. The test is equivalent to a MANN–Whitney U-test. In Table 6, there are two values, that is, p and h values, where p is the probability of observing the given result if the null hypothesis is true. When h

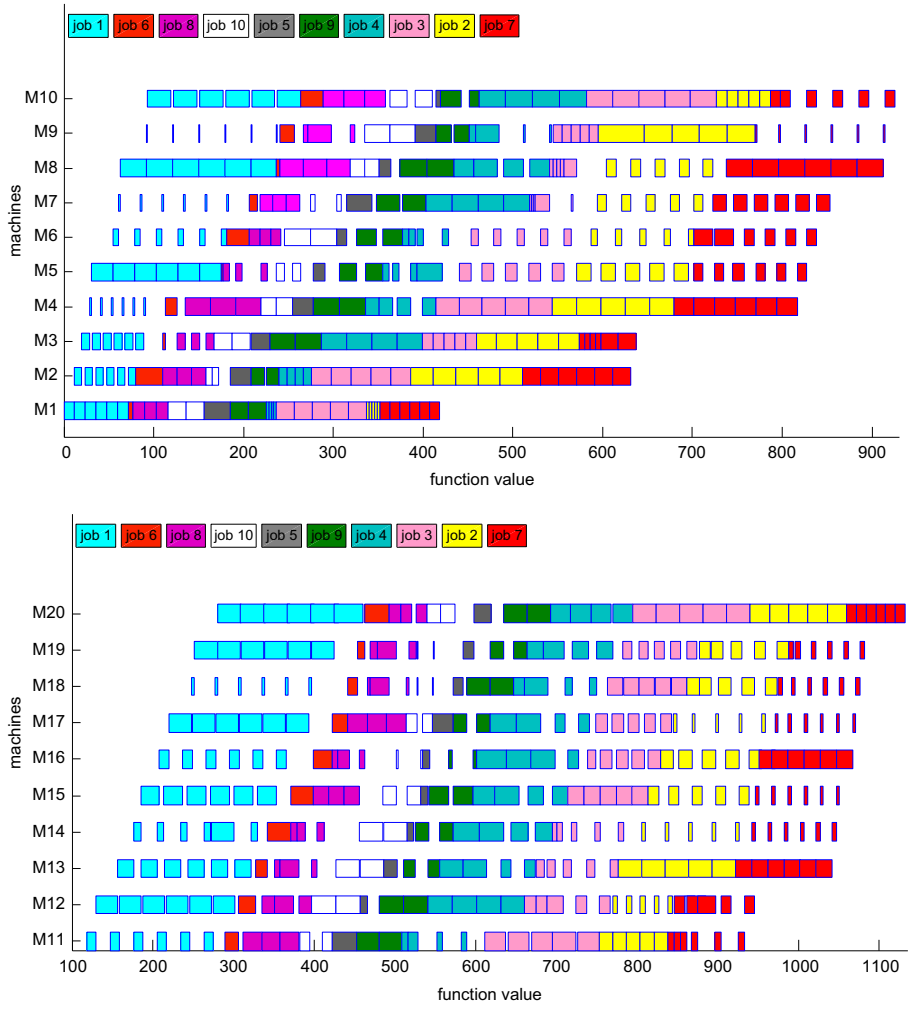

Figure 12. Optimal solution of the 10 × 20 instance (f_1 = 1126, f_2 = 7643, f_3 = 7452, f_4 = 21) obtained by INSGA-II.

Table 6. The Wilcoxon two-sided rank sum test results of INSGA-II, NSGA-II, DHS and TA algorithms.

(INSGA-II, NSGA-II)		(INSGA-II, DHS)		(INSGA-II, TA)	
p	*h*	*p*	*h*	*p*	*h*
2.8151E–009	1	3.0272E–007	1	2.8151E–009	1

equals 1, it indicates that the results obtained by the two compared algorithms are significantly different. If *h* is equal to 0, the difference between the two algorithms is not significant at 5% significant level. As shown in Table 6, the *p* value is approximately equal to 0 and the *h* value is exactly equal to 1, suggesting that INSGA-II is significantly different from the other comparative algorithms in terms of $D(S_j)$.

5.2.5 *Sensitivity study on parameter pc*

Hereinafter, there is a parameter, *pc*, which controls whether the offspring individual undergoes EDA or not. Obviously, pc is an important parameter for INSGA-II. Therefore, investigating the effect of *pc* by doing experiments is much necessary. Without loss of generality, the value of pc changes from 0 to 1.0 with the step size of 0.1. The benchmark instances and the parameter settings are kept the same as in Section 5.1. The comparative results in terms of the quality and the distribution of non-dominated solutions are shown in Figures 12 and 13.

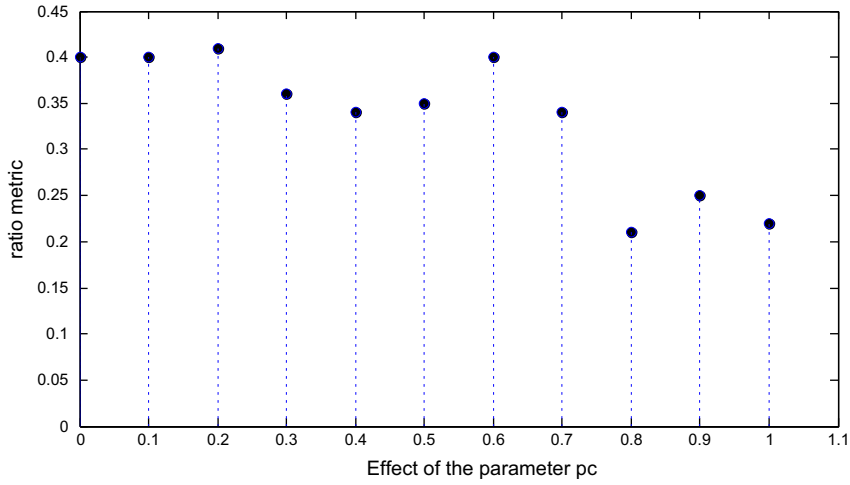

Figure 13. Sensitivity study of pc with respect to the ratio of non-dominated solutions.

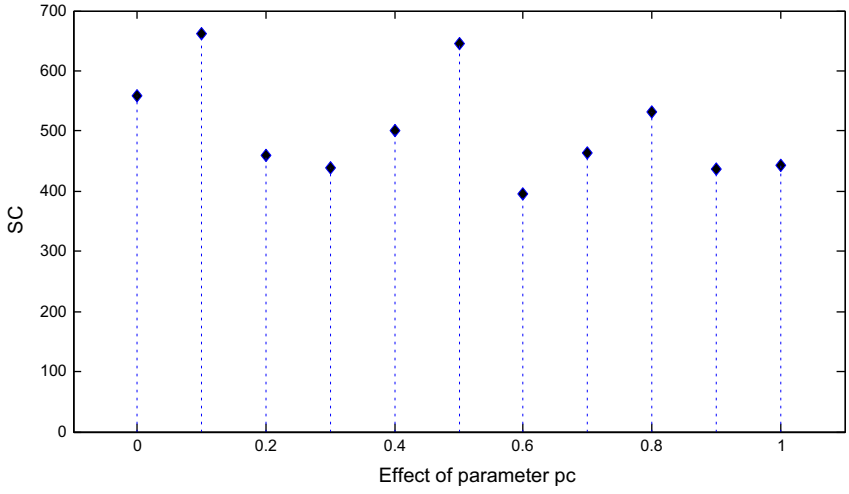

Figure 14. Sensitivity study of pc from the viewpoint of the distribution of non-dominated solutions.

Figure 13 reports that the trajectory tendency is relatively stable when the value of *pc* is in the range of [0, 0.2], and then, the ratio of non-dominated solutions gradually decreases to a low level. When *pc* is equal to 0.6, the above ratio is greater than the others, suggesting that EDA can take full advantage of valuable information of non-dominated solutions to estimate the probability distribution of good chromosomes to generate promising offspring. When the value of *pc* is in the range of [0.7, 1], the value of $R(S_j)$ becomes small along with the increase of *pc*. Specifically, $pc = 1$ represents that there is no mutation operator and the whole algorithm only adopts EDA to generate offspring, which will result in obtaining local optimal of the optimisation problem. Thus, we set the value of pc as 0.6 in this study. Similar experiments are done from the viewpoint of the distribution of non-dominated solutions. Figure 14 shows that when pc is equal to 0.6, the SC value is smaller than the others. The above experimental results demonstrate that the performance of INSGA-II is sensitive to its parameter.

6. Conclusions

LSFS scheduling problems with single objective are not sufficient in modern manufacturing environments, and the multi-objective LSFS scheduling problems play a key role in real-world applications. Therefore, developing effective strategies for the multi-objective ones is necessary and significant.

In recent years, NSGA-II has attracted much attention in the community of evolutionary optimisation. However, the applications of NSGA-II in practical problems are not fully investigated. Thus, an improved NSGA-II is proposed to

solve the multi-objective LSFS scheduling problem in this study. The characteristics of the proposed algorithm lie mainly in the following three aspects: (1) four variants of NEH heuristics are designed to form initial individuals with a good performance in the initialisation phase; (2) EDA taking full advantage of valuable information of non-dominated solutions and the mutation operators based on insertion and swap are embedded in the proposed algorithm to generate good offspring; (3) an efficient restarting strategy is employed to maintain the diversity of the population and avoids the population trapping in local optima.

The performance of INSGA-II proposed in this paper is evaluated on a set of 24 instances and compared with NSGA-II, DHS and TA algorithms. The experimental results demonstrated the superiority of the proposed algorithm in terms of the quality and distribution of non-dominated solutions. It is worth mentioning that the outperformance of the proposed algorithm attributes to EDA and mutation operators, by which its capabilities in exploration and exploitation are improved.

There are also some limitations in our work. The exploitation capability of multi-objective evolutionary algorithms should be further considered. The future work is to apply INSGA-II to solve other scheduling problems with different objectives.

Funding

This research is partially supported by Natural Science Foundation of China under Grant Nos. 61105063, 61075061, 61104179 and 61174187, the Fundamental Research Funds for the Central Universities and Research, Innovation Project for College Graduates of Jiangsu Province with Granted No. CXZZ13 0932, Basic scientific research foundation of Northeast University under Grant N110208001, starting foundation of Northeast University under Grant 29321006 and Science Foundation of Liaoning Province in China (2013020016).

References

Aickelin, U., and J. P. Li. 2007. “An Estimation of Distribution Algorithm for Nurse Scheduling.” *Annals of Operations Research* 155: 283–309.

Deb, K. 2000. “Multi-objective Evolutionary Optimization: Past, Present and Future.” In *Proceeding of the Fourth International Conference on Adaptive Computing in Design and Manufacture*, edited by I. C. Parmee, 225–236. London.

Deb, K., A. Pratap, S. Agarwal, and T. Meyarivant. 2002. “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II.” *Evolutionary Computation. IEEE Transactions* 6: 182–197.

Defersha, F. M., and M. Y. Chen. 2012. “Mathematical Model and Parallel Genetic Algorithm for Hybrid Flexible Flowshop Lot Streaming Problem.” *International Journal of Advanced Manufacturing Technology* 62: 249–265.

Dhanalakshmi, S., S. Kannan, K. Mahadevan, and S. Baskar. 2011. “Application of Modified NSGA-II Algorithm to Combined Economic and Emission Dispatch Problem.” *Electrical Power and Energy Systems* 33: 992–1002.

Han, H. Y. 2012. “A Multi-objective Hybrid Discrete Harmony Search Algorithm for Lot-streaming Flow Shop Scheduling.” 7th International Conference ICIC, 66–73. Zhengzhou.

Han, Y. Y., J. J. Liang, Q. K. Pan, and J. Q. Li. 2012. Effective Hybrid Discrete Artificial Bee Colony Algorithms for the Total Flowtime Minimization in the Blocking Flowshop Problem. *International Journal of Advanced Manufacturing Technology* 67: 397–414.

Huang, B. Q., B. Buckley, and T. M. Kechadi. 2010. “Multi-objective Feature Selection by Using NSGA-II for Customer Churn Prediction in Telecommunications.” *Expert Systems with Applications* 37: 3638–3646.

Jarboui, B., M. Eddaly, and P. Siarry. 2009. “An Estimation of Distribution Algorithm for Minimizing the Total Flowtime in Permutation Flowshop Scheduling Problems.” *Computers Operations Research* 35: 2638–2646.

Karaboga, D. 2005. *An Idea Based on Honey Bee Swarm for Numerical Optimization*. Technical Report TR06. Turkey: Computer Engineering Department, Erciyes University.

Kim, K., and I. J. Jeong. 2009. “Flow Shop Scheduling with No-wait Flexible Lot Streaming Using an Adaptive Genetic Algorithm.” *International Journal of Advanced Manufacturing Technology* 44: 1181–1190.

Larraaga, P., and J. Lozano. 2002. *A Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Boston: Kluwer Academic.

Lei, D. M., and X. P. Guo. 2013. “Scheduling Job Shop with Lot Streaming and Transportation Through a Modified Artificial Bee Colony.” *International Journal of Production Research* 51: 4930–4941.

Liu, C. H., and D. H. Huang. 2013. “Reduction of Power Consumption and Carbon Footprints by Applying Multi-objective Optimisation via Genetic Algorithms.” *International Journal of Production Research*. doi:10.1080/00207543.2013.825740.

Marimulthu, S., S. G. Ponnambalam, and N. Jawahar. 2009. “Threshold Accepting and Ant Colony Optimization Algorithm for Scheduling M-machine Flow Shop with Lot Streaming.” *Journal of Material Processing Technology* 209: 1026–1041.

Muhlenbein, H., and G. Paass. 1996. “From Recombination of Genes to the Estimation of Distributions I Binary Parameters.” In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature 1141*, 178–187. Berlin: Springer.

Murugan, P., S. Kannan, and S. Baskar. 2009. "NSGA-II Algorithm for Multi-objective Generation Expansion Planning Problem." *Electric Power Systems Research* 79: 622–628.

Nawaz, M., E. E. J. Enscore, and I. Ham. 1983. "A Heuristic Algorithm for Them-machine, N-job Flow Shop Sequencing Problem." *International Journal of Management Science* 11: 91–95.

Pan, Q. K., J. H. Duan, J. J. Liang, K. Z. Gao, and J. Q. Li. 2010. "A Novel Discrete Harmony Search Algorithm for Scheduling Lot-streaming Flow Shops." Chinese Control and Decision Conference, 1531–1536. Xuzhou.

Pan, Y. X., Q. K. Pan, and H. Y. Sang. 2010. "Discrete Particle Swarm Optimization Algorithm for Lot-streaming Flow Shop Problem." *Computer Engineering and Applications* 46: 52–55.

Pan, Q. K., and R. Ruiz. 2012. "An Estimation of Distribution Algorithm for Lot-streaming Flow Shop Problems with Setup times." *Omega* 40: 166–180.

Pan, Q. K., P. N. Suganhan, M. F. Tasgetiren, and T. J. Chua. 2011. "A Novel Artificial Bee Colony Algorithm for a Lot-streaming Flow Shop Scheduling Problem." *Information Sciences* 181: 2455–2468.

Pan, Q. K., L. Wang, and B. Qian. 2009. "A Novel Differential Evolution Algorithm for Bi-criteria No-wait Flow Shop Scheduling Problems." *Computers and Operations Research* 36: 2498–2511.

Ronay, A. K., Y. F. Li, Valeria Vitelli, Enrico Zio, Enrique Lopez Droguett, and Carlos Magno Couto Jacinto. 2013. "NSGA-II-Trained Neural Network Approach to the Estimation of Prediction Intervals of Scale Deposition Rate in Oil and Gas Equipment." *Expert Systems with Applications* 40: 1205–1212.

Ruiz, R., C. Maroto, and J. Alcaraz. 2006. "Two New Robust Genetic Algorithms for the Flowshop Scheduling Problem." *Omega* 34: 461–476.

Ruiz, R., and S. Thomas. 2008. "An Iterated Greedy Heuristic for the Sequence Dependent Setup times Flowshop Problem with Makespan and Weighted Tardiness Objectives." *European Journal of Operational Research* 187: 1143–1159.

Shafaghat, R., S. M. Hosseinalipour, I. Lashgari, and A. Vahedgermi. 2011. "Shape Optimization of Axisymmetric Cavitators in Supercavitating Flows, Using the NSGA II Algorithm." *Applied Ocean Research* 33: 193–198.

Sun, J. Y., Q. F. Zhang, and P. K. Edward. 2005. "DE/EDA: A New Evolutionary Algorithm for Global Optimization." *Information Sciences* 169: 249–262.

Tseng, C. T., and C. Liao. 2008. "A Discrete Particle Swarm Optimization for Lot-streaming Flowshop Scheduling Problem." *European Journal of Operational Research* 191: 360–373.

Veldhuizen, D. V., and G. Lamont. 2000. "On Measuring Multi-objective Evolutionary Algorithm Performance." In *Paper Presented at the Annual Meeting of the IEEE Congress on Evolutionary Computation*: 204–211. La Jolla, CA.

Wang, L. 2003. *Shop Scheduling with Genetic Algorithms*. Beijing: Tsinghua University Press.

Wang, L., and D. Z. Zheng. 2003. "An Effective Hybrid Heuristic for Flow Shop Scheduling." *International Journal of Advanced Manufacturing Technology* 21: 38–44.

Yoon, S. H., and J. A. Ventura. 2002. "An Application of Genetic Algorithms to Lot-streaming Flow Shop Scheduling." *IIE Transactions* 34: 779–787.

Zhang, C. Y., X. Dong, X. J. Wang, X. Y. Li, and Q. Liu. 2010. "Improved NSGA-II for the Multi-objective Flexible Job-shop Scheduling Problem." *Journal of Mechanical Engineering* 46: 156–164.

Zhang, Y., and X. P. Li. 2011. "Estimation of Distribution Algorithm for Permutation Flow Shops with Total Flowtime Minimization." *Computers and Industrial Engineering* 60: 706–718.