Soft Comput (2009) 13:847–869
DOI 10.1007/s00500-008-0350-8

FOCUS

Multi-objective no-wait flow-shop scheduling with a memetic algorithm based on differential evolution

Bin Qian · Ling Wang · De-Xian Huang · Xiong Wang

Published online: 9 August 2008

Abstract In this paper, a memetic algorithm (MA) based on differential evolution (DE), namely MADE, is proposed for the multi-objective no-wait flow-shop scheduling problems (MNFSSPs). Firstly, a largest-order-value rule is presented to convert individuals in DE from real vectors to job permutations so that the DE can be applied for solving flow-shop scheduling problems (FSSPs). Secondly, the DE-based parallel evolution mechanism is applied to perform effective exploration, and several local searchers developed according to the landscape of multi-objective FSSPs are applied to emphasize local exploitation. Thirdly, a speed-up computing method is developed based on the property of the no-wait FSSPs. In addition, the concept of *Pareto dominance* is used to handle the updating of solutions in sense of multi-objective optimization. Due to the well balance between DE-based global search and problem-dependent local search as well as the utilization of the speed-up evaluation, the MNFSSPs can be solved effectively and efficiently. Simulation results and comparisons demonstrate the effectiveness and efficiency of the proposed MADE.

B. Qian (✉) · L. Wang · D.-X. Huang · X. Wang
Department of Automation, Tsinghua University,
Beijing 100084,
People's Republic of China
e-mail: qianb04@mails.tsinghua.edu.cn

L. Wang
e-mail: wangling@mail.tsinghua.edu.cn

B. Qian
Department of Automation,
Kunming University of Science and Technology,
Kunming 650051,
People's Republic of China

Keywords Multi-objective no-wait flow-shop scheduling · Differential evolution · Memetic algorithm · Local search · Exploration and exploitation

1 Introduction

Production scheduling is an important issue faced daily both in the manufacturing systems and the service industries. So, it is necessary to develop effective and efficient advanced manufacturing and scheduling technologies and approaches (Stadtler 2005; Dimopoulos and Zalzala 2000; Wang 2003). The flow shop scheduling problem (FSSP) is a class of widely studied scheduling problems. It represents nearly a quarter of manufacturing systems, assembly lines and information service facilities nowadays (Pinedo 2002) and is considered as being difficult to solve (Garey and Johnson 1979; Baker 1974). In many FSSPs, there exists a constraint that once the processing of a job begins, subsequent processing must be performed without waiting between or on consecutive machines. Such a FSSP is termed as no-wait FSSP (NFSSP). Some typical situations are encountered in metal, plastic, chemical and pharmaceutical industries. For example, a plastic product requires to be processed through a continuous sequence of operations to prevent degradation. According to the research work by Garey and Johnson (1979), the NFSSP is NP-hard.

Due to its significance both in theory and industrial applications, the NFSSP has been studied by many researchers. Historically, NFSSP has been primarily treated by the branch-and-bound method (Van Deman and Baker 1974), constructive methods (Bonney and Gundry 1976; King and Spachis 1980; Gangadharan and Rajendran 1993; Rajendran 1994). Recently, metaheuristic methods have attracted wide research attention, including such topics as genetic algorithm

(GA) (Chen et al. 1996; Kumar et al. 2000; Aldowaisan and Allahverdi 2003), simulated annealing (SA) algorithm (Aldowaisan and Allahverdi 2003), tabu search (TS) algorithm (Grabowski and Pempera 2005), descending search (DS) algorithm (Grabowski and Pempera 2005), hybrid particle swarm optimization (HPSO) algorithm (Liu et al. 2007). An early comprehensive survey of the NFSSP can be found in Hall and Sriskandarajah (1996). As we know, most real scheduling problems naturally involve the optimization of multiple objectives. Up to now, few researchers have studied multi-objective NFSSPs (MNFSSPs). Allahverdi and Aldowaisan (2004) presented two hybrid algorithms based on SA and GA to minimize a weighted sum of makespan and maximum lateness. Tavakkoli-Moghaddam et al. (2007) addressed the MNFSSP that minimizes both the weighted mean completion time and weighted mean tardiness, and developed a hybrid multi-objective immune algorithm (IA) to obtain *Optimal Pareto solution* for such problem.

Over the past 15 years, memetic algorithms (MAs) have been a hot topic in the fields of both computer science and operational research (Reeves and Yamada 1998; Murata et al. 1996; Ong et al. 2006; Hart et al. 2004). It assumes that combining the features of different methods in a complementary fashion may result in more robust and effective optimization tools. MAs belong to the class of evolutionary algorithms (EAs) that combine the global and local search by using an EA to execute exploration while the local search method executes exploitation, which are inspired by Darwinian principles of natural evolution and Dawkins' notion of a meme defined as a unit of cultural evolution that is capable of local refinements. Some recent work demonstrates that MAs (sometimes called hybrid EAs) can yield promising results for solving combinatorial and nonlinear optimization problems (Tang et al. 2007; Zhou et al. 2007; Ong and Keane 2004) and engineering problems (Caponio et al. 2007; Ong and Keane 2004). In MAs, studies (Hart et al. 2004; Zhu et al. 2007; Ishibuchi et al. 2003) have been focused on how to achieve a reasonable combination of global search and local search, and how to make a good balance between exploration and exploitation. As for multi-objective FSSPs, Ishibuchi and Murata (1998) firstly devised a multi-objective memetic algorithm (IMMOGLS) by combining GA with the local search method, which used a scalar fitness function with random weights to guide the evolution process of GA-based search and local search to find *optimal Pareto front*. Jaszkiewicz (2002) implemented another genetic local search algorithm (JMOGLS), which was similar to IMMOGLS. The main difference lies in the selection mechanism of parents. Ishibuchi et al. (2003) proposed a modified IMMOGLS (IMMOGLS2) by adopting a more reasonable local searcher in multi-objective sense. Moreover, the impact of balance between global search and local search was analyzed. Arroyo and Armentano (2005) conceived a multi-objective GA (AAMOGLS), in which the concept of *Pareto* dominance is used to rank the population and assign suitable fitness values to all the individuals. Subsequently, a local searcher based on *Pareto* dominance was applied to perform the exploitation. However, the research work about MAs for NFSSPs and MNFSSPs is very scarce. Liu et al. (2007) proposed an effective memetic algorithm based on particle swarm optimization (PSO) algorithm to minimize makespan, where several local searchers or memes with adaptive learning strategy were incorporated into PSO. Tavakkoli-Moghaddam et al. (2007) implemented a hybrid multi-objective IA to minimize both the weighted mean completion time and weighted mean tardiness, where two local searchers, namely *bacterial mutation* and *gene transfer*, were applied to improve the quality of some selected individuals (i.e., antibodies).

Differential evolution (DE) is a novel population-based evolutionary mechanism recently proposed for global optimization over continuous spaces (Storn and Price 1997). Despite DE's structure simplicity, the key evolutionary operators of *mutation* and *crossover* are very efficient, which allows the search behavior of each individual to self-adapting. Due to its ease of use, fast convergence and robustness, DE has gained wide application in a variety of fields (Ilonen et al. 2003; Chang and Wu 2005; Feoktistov 2006; Price and Storn 2007). Because DE's individual is a real vector, it is difficult to directly present feasible solutions to combinatorial optimization problems (COPs). Thus, the research work of DE for scheduling problems is quite limited. Recently, Tasgetiren et al. (2004) implemented an encoding scheme to convert the continuous values of individuals in DE to job permutations and incorporated *Interchange*-based local searcher into DE to minimize the makespan criterion of FSSPs. Onwubolu and Davendra (2006) developed a DE-based heuristic to solve FSSPs with the objectives of makespan, mean flowtime, and total tardiness. Nearchou and Omirou (2006) designed a stochastic method based on DE to deal with three classic NP-hard scheduling problems: the flow-shop scheduling problem, the single-machine total weighted tardiness problem, and the single machine common due date scheduling problem. Nearchou (2008) presented a DE-based algorithm to address the common due date early/tardy job scheduling problem on a single machine, which could find new upper bounds to nearly 60% of the testing benchmark problems. As for multi-objective scheduling problems, Qian et al. (2008) proposed a memetic algorithm based on DE for multi-objective job shop scheduling problems (MJSSPs), which used several memes and adopted an adaptive Meta-Lamarckian strategy to dynamically decide which meme to be selected to emphasize exploitation in each generation. To the best of our knowledge, there are few other published papers on DE for shop scheduling, and especially there is no published paper on DE for the multi-objective no-wait FSSPs.

In this paper, we will devise a memetic algorithm based on DE (MADE) by combining DE with several problem dependent memes for the no-wait multi-objective FSSPs. The motivation of MADE is based on the ‘no free lunch theory’ (Wolpert and Macready 1997). That is, any algorithm without adopting the domain knowledge of problems is only equal to a kind of random search, and it is impossible to obtain excellent performance on special problems. Thus, when designing MADE, we fully considered some structure information of MNFSSPs. For FSSPs, the solution space landscape induced by some specific neighborhoods (i.e., *Insert*, *Interchange*, *Swap*, etc.) has a “big valley,” where local optima tend to be relatively close to each other and to the global optima at the big valley’s bottom part (Reeves and Yamada 1998; Reeves 1999). And the size of the bottom region of the big valley containing global or satisfactory local optima is considerably small with respect to the whole valley. However, the number of solutions in the bottom region is also so large that it is unlikely to apply a total search (Nowicki and Smutnicki 2006). For MNFSSPs, different objective induces different shape of big valley. Since the objectives are usually not positively correlated, it is difficult for a solution to simultaneously reach the bottom regions of all big valleys. That is to say, the effectiveness of searching in the different big valleys directly determines the performance of MADE. Fortunately, the DE’s *mutation* operation is quite unique, which is driven by the differences between contemporary population members. This allows the search behavior of each individual to self-tune during the search process, and gives an appropriate search direction to guide the search to find the global optima. So, in our MADE for MNFSSPs, DE is applied to find the promising solutions or regions over the solution space, and three efficient memes based on the landscape of FSSP are conceived to exploit the solution space from those regions and to guide the population to the bottom regions of big valleys, where *Optimal Pareto solutions*or satisfactory *Pareto solutions*are contained. In short, firstly, a largest-order-value (LOV) rule is proposed to map the continuous values of individuals in DE to job permutations so as to make DE suitable for solving FSSPs; secondly, the DE-based search is applied for exploration in a parallel framework, while several problem-dependent memes are utilized to emphasize exploitation; thirdly, a speed-up computing method base on the property of the NFSSPs is developed to calculate the objective functions efficiently. Simulation results and comparisons based on the testing benchmark instances validate the effectiveness and efficiency of the proposed MADE.

The remaining contents of this paper are partitioned into five sections. Section 2 briefly introduces no-wait FSSP and MNFSSP. Section 3 provides a brief review of DE. Section 4 presents MADE in detail, while Sect. 5 presents and discusses simulation results and comparisons. Finally, Sect. 6 gives some conclusions and states future research directions.

2 NFSSP and MNFSSP

2.1 NFSSP

The no-wait FSSP with n jobs and m machines can be described as follows: given the processing time $p(i, j)$ of job i on machine j, each of n jobs will be sequentially processed on machine $1, 2, \ldots, m$. At any time, each machine can process at most one job and each job can be processed on at most one machine. The sequence in which the jobs are to be processed is the same for each machine. To follow the no-wait restrictions, the difference between the completion time of the last operation of a job and the start time of its first operation is equal to the sum of the processing times of its operations. In other words, the operation of each job must be processed without interruptions between consecutive machines. The problem is to find a sequence or permutation for processing all jobs on all machines so that one or more given objectives are optimized. The objective widely used is the minimization of the maximum completion time, i.e., makespan ($C_{\max}$).

Let $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ denote the permutation of jobs to be processed, $P_{sum}(\pi_j)$ the total processing time of job π_j on all machines, $MD(\pi_{j-1}, \pi_j)$ the minimum delay on the first machine between the start of job π_j and π_{j-1} restricted by the no-wait constraint. Then *MD* can be calculated as follows:

$$MD(\pi_{j-1}, \pi_j) = p(\pi_{j-1}, 1) + \max\left[0, \max_{2\le k\le m}\left\{\sum_{h=2}^{k} p(\pi_{j-1}, h) - \sum_{h=1}^{k-1} p(\pi_j, h)\right\}\right]. \tag{1}$$

Thus, the makespan can be defined as

$$C_{\max}(\pi) = \sum_{j=2}^{n} MD(\pi_{j-1}, \pi_j) + P_{sum}(\pi_n). \tag{2}$$

where $P_{sum}(\pi_n) = \sum_{k=1}^{m} p(\pi_n, k)$.

The aim of the no-wait FSSP with the makespan criterion is to find a permutation π^* in the set of all permutations Π such that

$$C_{\max}(\pi^*) = \min_{\pi\in\Pi} C_{\max}(\pi). \tag{3}$$

Figure 1 shows an example of a no-wait FSSP with $n = 4$ and $m = 4$.

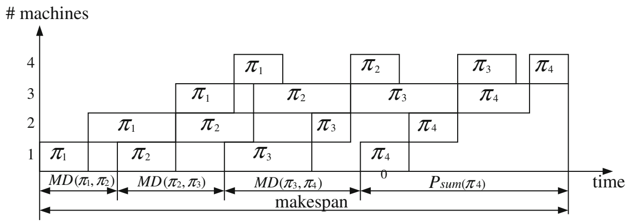

Fig. 1 No-wait FSSP example with $n = 4$ and $m = 4$

2.2 MNFSSP

2.2.1 Objective functions

For a FSSP, let $C(\pi_j, k)$ denote the completion time of job π_j on machine k. If there exists a due date d_j for job j, we can define $L(\pi_j) = C(\pi_j, m) - d_j$ as the lateness of job j. Then, the tardiness and earliness of job π_j can be defined as $T(\pi_j) = \max\{L(\pi_j), 0\}$ and $E(\pi_j) = \max\{-L(\pi_j), 0\}$, respectively. In some real application, machine idleness may be considered. Let $I_k(\pi) = C(\pi_n, k) - \sum_{j=1}^{n} p(\pi_j, k)$ denote the idleness time on machine k. Besides, we can use the indicator function $U(\pi_j)$ to denote whether job π_j is tardy ($U(\pi_j) = 1$) or not ($U(\pi_j) = 0$). Assuming λ_j is a possible weight associated to job j, the following objective functions are frequently used (Pinedo 2002):

(1) Maximum completion time or makespan $C_{\max}(\pi) = C(\pi_n, m)$;
(2) Total weighted completion time $C_w(\pi) = \sum_{j=1}^{n} \lambda_j C(\pi_j, m)$;
(3) Maximum tardiness $T_{\max}(\pi) = \max\limits_{j} T(\pi_j)$;
(4) Total weighted tardiness $T_w(\pi) = \sum_{j=1}^{n} \lambda_j T(\pi_j)$;
(5) Total machine idleness $I_{sum}(\pi) = \sum_{k=1}^{m} I_k(\pi)$;
(6) Maximum earliness $E_{\max}(\pi) = \max_j E(\pi_j)$;
(7) Total weighted earliness $E_w = \sum_{j=1}^{n} \lambda_j E(\pi_j)$;
(8) Total number of tardy jobs $N_T(\pi) = \sum_{j=1}^{n} U(\pi_j)$; and so on.

2.2.2 MNFSSP

For the MNFSSP, some of objectives mentioned in Sect. 2.2.1 can be considered simultaneously. However, these objectives often conflict with themselves, that is, an improvement in one objective may worsen another. Since there is usually no such a solution that is the best to all objectives, the multi-objective optimization algorithms are required to find a set of optimal solutions (called non-dominated solutions) (Ishibuchi et al. 2003; Arroyo and Armentano 2005). Without loss of generality, a general multi-objective optimization problem (MOP) with w objectives can be described as follows:

$$\begin{aligned} &\text{Minimize } f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_w(\mathbf{x}) \\ &\text{subject to } \mathbf{x} \in \mathbf{X}, \end{aligned} \tag{4}$$

where $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_w(\mathbf{x})$ are w objectives to be minimized, $\mathbf{x}$ is a vector of decision variables, and $\mathbf{X}$ is the set of feasible solution.

To multi-objective optimization, the following basic definitions are of importance:

(1) Pareto dominance: A solution $\mathbf{x}_1$ is said to (Pareto) *dominate* another solution $\mathbf{x}_2$ (denoted $\mathbf{x_1} \succ \mathbf{x_2}$) if and only if

$$\begin{aligned} &(\forall i \in \{1, 2, \ldots, w\} : f_i(\mathbf{x_1}) < f_i(\mathbf{x_2})) \\ &\wedge(\exists j \in \{1, 2, \ldots, w\} : f_j(\mathbf{x_1}) < f_j(\mathbf{x_2})). \end{aligned} \tag{5}$$

(2) Optimal Pareto solution: A solution $\mathbf{x}_1$ is said to be an *optimal Pareto solution* if and only if there is no any solution $\mathbf{x}_2$ that satisfies $\mathbf{x}_2 \succ \mathbf{x}_1$.
(3) Optimal Pareto set: The set containing all *optimal Pareto solutions* is said as *optimal Pareto set*.
(4) Optimal Pareto front: The set of all objective function values corresponding to the solutions in the *optimal Pareto set* is said as *optimal Pareto front*.

According to (Deb et al. 2002), basically two main goals should be considered to evaluate the obtained non-dominated solutions: (1) convergence to the *optimal Pareto front*; (2) maintenance of diversity (i.e., spread and distribution) of the obtained non-dominated solutions.

3 Introduction to differential evolution

DE is a class of population-based evolutionary algorithms, which absorbs the concepts of "population" from GA and "self-adapting mutation" from evolution strategy (ES). The procedure of DE is almost the same as that of GA whose main operations are *mutation*, *crossover*, and *selection*. The main difference between DE and GA lies in the *mutation* operation. In DE, it starts with the random initialization of a population of individuals in the search space and works on the cooperative behaviors of the individuals in the population. At each generation, the *mutation* and *crossover* operators are applied to individuals to generate a new population. Then, the one-to-one greedy *selection* takes place and the population is updated.

The basic scheme of DE, which is denoted as *DE/rand/1/bin*, can be summarized as follow.

Let the ith individual in the N-dimensional search space at generation t be $X_i(t) = [x_{i,1}, x_{i,2}, \ldots, x_{i,N}]$

($i = 1, 2, \ldots, PS$), where *PS* denotes the size of the population, and $X_i(t)$ is a real vector.

Step 1: DE's *initialization*. Randomly initialize the population with *PS* individuals and determine the best individual *bestit* with the best objective value.

Step 2: DE's *mutation*. In order to obtain each individual's corresponding *mutant vector* $V_i(t+1) = [v_{i,1}(t+1), \ldots, v_{i,N}(t+1)]$, *mutation* operation is performed for each individual according to the following equation:

$$V_i(t+1) = X_{r1}(t) + F * (X_{r2}(t) - X_{r3}(t)), \quad (6)$$

where $r1, r2, r3 \in \{1, 2, \ldots, PS\}$ are randomly chosen and mutually different and also different from the current index i, $F \in (0, 2)$ is a constant called scaling factor which controls amplification of the differential variation $X_{r2}(t) - X_{r3}(t)$.$X_{r1}(t)$ is the base vector to be perturbed.

Step 3: DE's *crossover*. To get each individual's *trial vector* $U_i(t+1) = [u_{i,1}(t+1), \ldots, u_{i,N}(t+1)]$, *crossover* operation is performed between each individual and its corresponding *mutant vector* by the following equation:

$$u_{i,j}(t+1) = \begin{cases} v_{i,j}(t+1), & \text{if } (rand(j) < CR) \text{ or } j = randn(i), \\ x_{i,j}(t), & \text{otherwise.} \end{cases} \quad (7)$$

$$j = 1, \ldots, N,$$

where $rand(j)$ is the jth evaluation of a random number uniformly distributed in the range of [0,1], $randn(i)$ is a randomly chosen index from the set $\{1, 2, \ldots, N\}$, $CR \in [0, 1]$ is a constant crossover parameter that controls the diversity of the population.

Step 4: DE's *selection*. To generate the new individual for the next generation, *selection* operation is performed between each individual and its corresponding *trial vector* by the following greedy selection criterion:

$$X_i(t+1) = \begin{cases} U_i(t+1), & \text{if } f(U_i(t+1)) < f(X_i(t)), \\ X_i(t), & \text{otherwise.} \end{cases} \quad (8)$$

where f is the objective function and $X_i(t+1)$ is the individual of the new population.

Step 5: Update *bestit*. If a stopping criterion is met, then output *bestit* and its objective value; otherwise go back to Step 2.

The key parameters in DE are the size of population (PS), the scaling factor (F), and the crossover parameter (CR). Proper configuration of these parameters can increase the convergence velocity and robustness of the search process. In (Storn and Price 1997) some suggestions have been given for selecting suitable parameters for DE.

There are also some other DE variants (Price and Storn 2007), only differ in the type of *mutation* operation and *crossover* operation. The general format of DE is *DE/x/y/z*, where x represents a base vector to be mutated/perturbed, y is the number of difference vectors used for perturbation of x, and z stands for the type of crossover being used (bin: binomial; exp: exponential).

4 MADE for MNFSSP

In this section, we will present the memetic DE for no-wait FSSP after explaining the solution representation, speed-up computing method, DE-based search, problem-dependent local search and multi-objective handling techniques.

4.1 Solution representation

Due to the continuous nature of DE, the standard encoding scheme of DE cannot be directly adopted for FSSPs. So, it is crucial to develop a suitable mapping scheme that converts the individuals (continuous vectors) to the job sequence. In this paper, we design a largest-order-value (LOV) rule based on random key representation (Bean 1994) to convert individual $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ in DE to the job solution or permutation vector $\pi_i = [\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,n}]$. Some other conversion techniques can be found in (Tasgetiren et al. 2004; Price et al. 2005). The conversion procedure is as follows:

Step 1: Rank all elements in $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$ by descending order to obtain a sequence $\varphi_i = [\varphi_{i,1}, \varphi_{i,2}, \ldots, \varphi_{i,n}]$.

Step 2: Calculate the job permutation π_i by the following formula:

$$\pi_{i,\varphi_{i,l}} = l, \quad (9)$$

where the dimension l varies from 1 to n.

To better understand the LOV rule, a simple example ($n = 6$) is illustrated in Table 1, where the individual is $X_i = [1.36, 3.85, 2.55, 0.63, 2.68, 0.82]$. The details that correspond with the steps of the conversion procedure are given as follows:

Table 1 Solution representation

Dimension l	1	2	3	4	5	6
$x_{i,l}$	1.36	3.85	2.55	0.63	2.68	0.82
$\varphi_{i,l}$	4	1	3	6	2	5
$\pi_{i,l}$	2	5	3	1	6	4

Step 1: Since $x_{i,2}$ is the largest value of X_i, $x_{i,2}$ is selected first and assigned the rank value 1. Then $x_{i,5}$ is selected second and assigned the rank value 2. In the same way, $x_{i,3}, x_{i,1}, x_{i,6}$ and $x_{i,4}$ are assigned the rank values 3, 4, 5, and 6, respectively. Thus, the sequence is $\varphi_i = [4, 1, 3, 6, 2, 5]$.

Step 2: According to (9), if $l = 1$, then $\varphi_{i,1} = 4$ and $\pi_{i,\varphi_{i,1}} = \pi_{i,4} = 1$; if $l = 2$, then $\varphi_{i,2} = 1$ and $\pi_{i,\varphi_{i,2}} = \pi_{i,1} = 2$; if $l = 3$, then $\varphi_{i,3} = 3$ and $\pi_{i,\varphi_{i,3}} = \pi_{i,3} = 3$; and so on. Thus, we obtain the job permutation $\pi_i = [2, 5, 3, 1, 6, 4]$.

Obviously, such a conversion process is very simple, and it makes DE suitable for solving FSSPs.

In our MADE, memes or local searchers are not directly applied to individual $X_i(t)$ with real values, but to the job permutation π_i. Thus, after the whole local search completes, $X_i(t)$ should be repaired because its corresponding job permutation should match the permutation resulted by the local search. Based on the mechanism of LOV rule, the repair process is easy to implement, which is given as follows:

Step 1: Calculate the sequence φ_i by the following formula:

$$\varphi_{i,\pi_{i,l}} = l. \tag{10}$$

where l varies from 1 to n.

Step 2: Values in $X_i(t)$ are rearranged to keep consistent with φ_i. That is, $X_{i,l}$ should be set to the $\varphi_{i,l}$th largest value of the old $X_i(t)$.

A simple instance on the repair is shown in Tables 2 and 3, where $\pi_{i,3} = 3$ and $\pi_{i,4} = 1$ are interchanged. As seen in Table 2, the LOV rule is violated because the new job permutation π_i does not match the old individual $X_i(t)$. Thus, $X_i(t)$ and φ_i should be repaired, as the results in Table 3. The details that match with the steps of the repair process are given as follows:

Table 2 Job solution resulted by local search (before repairing)

Dimension l	1	2	3	4	5	6
$x_{i,l}$	1.36	3.85	2.55	0.63	2.68	0.82
$\varphi_{i,l}$	4	1	3	6	2	5
$\pi_{i,l}$	2	5	**1**	**3**	6	4

Table 3 Job solution resulted by local search (after repairing)

Dimension l	1	2	3	4	5	6
$x_{i,l}$	**2.55**	3.85	**1.36**	0.63	2.68	0.82
$\varphi_{i,l}$	**3**	1	**4**	6	2	5
$\pi_{i,l}$	2	5	**1**	**3**	6	4

Step 1: Rearrange φ_i by equation (10). That is, if $l = 1$, then $\pi_{i,1} = 2$ and $\varphi_{i,\pi_{i,1}} = \varphi_{i,2} = 1$; if $l = 2$, then $\pi_{i,2} = 5$ and $\varphi_{i,\pi_{i,2}} = \varphi_{i,5} = 2$; and so on. Then we can get the new sequence $\varphi_i = [3, 1, 4, 6, 2, 5]$ in Table 3.

Step 2: Rearrange $X_i(t)$ to keep consistent with the new φ_i . If $l = 1$, then $\varphi_{i,1} = 3$. That is to say, $X_{i,1}$ should be set to the third largest value of the old $X_i(t)$ (i.e., 2.55). If $l = 2$, then $\varphi_{i,2} = 1$ and $X_{i,2}$ should be set to the largest value of the old $X_i(t)$ (i.e., 3.85). Similarly, the new individual $X_i(t) = [2.55, 3.85, 1.36, 0.63, 2.68, 0.82]$ can be obtained in Table 3.

4.2 Speed-up computing method

In FSSPs, $MD(\pi_{j-1}, \pi_j)$ is not only determined by the job π_{j-1} and π_j but also affected by the jobs before π_{j-1}. But in no-wait FSSPs (NFSSPs), $MD(\pi_{j-1}, \pi_j)$ is only decided by the job π_{j-1} and π_j. Based on this property of NFSSPs, one method can be adopted to reduce the computing complexities of the objective functions in Section 2.2. That is, $MD(\pi_{j-1}, \pi_j)$ and $P_{sum}(\pi_j)$ can be calculated and saved in the initial phase of MADE and then can be used as constant values in the evolution phase of MADE.

According to equation (1), $MD(\pi_{j-1}, \pi_j)$ can be calculated by the algorithm in Appendix 1. This algorithm has a computing complexity (CC) of O(m). In the initial phase of MADE, $MD(\pi_{j-1}, \pi_j)$ and $P_{sum}(\pi_j)(\pi_{j-1}, \pi_j \in \{1, \ldots, n\})$ need to be calculated $n(n-1)$ times and n times respectively. Since n is usually much larger than m, the total CC of $MD(\pi_{j-1}, \pi_j)$ and $P_{sum}(\pi_j)(\pi_{j-1}, \pi_j \in \{1, \ldots, n\})$ is $O(n^2 m)$.

For the sake of simplicity, let MADE_nospeedup denote MADE without speed-up method, TCC_MADE_nospeedup (*a certain objective function*) denote the total CC of calculating *a certain objective function* in MADE_nospeedup and TCC_MADE(*a certain objective function*) denote the total CC of calculating *a certain objective function* in MADE. Suppose the total solution evaluation times (*TET*) of any objective function in MADE_nospeedup and MADE are K.

4.2.1 Analysis of the CC of calculating $C_{\max}(\pi)$

In MADE_nospeedup, it can be seen from Eq. (2) that the CC of $C_{\max}(\pi)$ is $O(nm)$. Then, TCC_MADE_nospeedup

$(C_{\max}(\pi))$ is $O(Knm)$. In MADE, $L(\pi_{j-1}, \pi_j)$ and $P_{sum}(\pi_j)$ can be regarded as constant values in the evolution phase. Then, the CC of $C_{\max}$ is reduced to $O(n)$. Thus TCC_MADE $(C_{\max}(\pi))$ is $O(n^2m+Kn)$. Because K is usually much larger than nm, TCC_MADE($C_{\max}(\pi)$) is reduced from $O(Knm)$ of TCC_MADE_nospeedup($C_{\max}(\pi)$) to $O(Kn)$.

4.2.2 Analysis of the CC of calculating $I_{sum}(\pi)$

The total machine idleness $I_{sum}(\pi)$ can be defined as follow:

$$\begin{aligned} I_{sum}(\pi) &= \sum_{k=1}^{m} I_k(\pi) = m\sum_{x=2}^{n} MD(\pi_{x-1}, \pi_x) \\ &\quad + \sum_{k=1}^{m}\sum_{y=1}^{k} p(\pi_n, y) - \sum_{k=1}^{m}\sum_{j=1}^{n} p(\pi_j, k) \\ &= m\sum_{x=2}^{n} MD(\pi_{x-1}, \pi_x) + \sum_{k=1}^{m}\sum_{y=1}^{k} p(\pi_n, y) \\ &\quad - \sum_{j=1}^{n} P_{sum}(\pi_j). \end{aligned} \tag{11}$$

where $\sum_{k=1}^{m}\sum_{y=1}^{k} p(\pi_n, y)$ can be computed in $O(m)$ by the algorithm in Appendix 2.

In MADE_nospeedup, the CCs of $m\sum_{x=2}^{n} MD(\pi_{x-1}, \pi_x)$ and $\sum_{k=1}^{m}\sum_{y=1}^{k} p(\pi_n, y)$ and $\sum_{j=1}^{n} P_{sum}(\pi_j)$ are $O(nm)$ and O(m) and $O(nm)$ respectively. And n is larger than m. So, the CC of $I_{sum}(\pi)$ in MADE_nospeedup is $O(nm)$. In MADE, the CCs of $m\sum_{x=2}^{n} MD(\pi_{x-1}, \pi_x)$ and $\sum_{k=1}^{m}\sum_{y=1}^{k} p(\pi_n, y)$ and $\sum_{j=1}^{n} P_{sum}(\pi_j)$ are $O(n)$ and $O(m)$ and $O(n)$ respectively. That is, the CC of $I_{sum}(\pi)$ in MADE is $O(n)$. Like the analysis in Sect. 4.2.1, TCC_MADE($I_{sum}(\pi)$) is decreased from $O(Knm)$ of TCC_MADE_nospeedup($I_{sum}(\pi)$) to $O(Kn)$.

4.2.3 Analysis of the CCs of calculating other objective functions in Sect. 2.2

Let *OF* denote any objective function in Section 2.2 except $C_{\max}(\pi)$ and $I_{sum}(\pi)$. When calculating *OF*, $C(\pi_j, m)(\pi_j \in \{1, \ldots, n\})$ is required to be computed first, which can be obtained by the algorithm in Appendix 3. The CC of this algorithm is $O(nm)$, which can be decreased from $O(nm)$ to $O(n)$ by using the speed-up method. Obviously, the CC of computing *OF* is equal to that of computing $C(\pi_j, m)(\pi_j \in \{1, \ldots, n\})$. This means TCC_MADE(*OF*) can be decreased from $O(Knm)$ of TCC_MADE_nospeedup(*OF*) to $O(Kn)$.

4.3 DE-based search

In MADE, DE-based search is designed based on *DE/rand-to-best/1/exp* scheme to perform parallel exploration, in which "*exp*" of "*DE/rand-to-best/1/exp*" means the exponential crossover is adopted and "*rand-to-best*" means the base vector is the best individual of the current population (Price and Storn 2007). Thus, those individuals performing DE-based operation will share the information of the best individual of the population. Note that, DE-based evolution is not performed on permutation-based solution space but continuous space. So, DE is used to stress exploration in a continuous searching space. Furthermore, DE-based search can be regarded as a kind of scatter search. In the *mutation* and *crossover* phase, each individual can transform probabilistically to any other individual in the solution space. Therefore, a wide range of solution space can be searched. In the *selection* phase, only the better individual can be accepted. This means that the DE-based search has the ability to reach enough promising regions over the solution space.

Because of the parallel evolutionary framework of DE, it is easy to incorporate local search into DE to design effective memetic algorithms. Next, we will present the problem-dependent local search.

4.4 Problem-dependent Local Search

For the MNFSSPs, we designed local searchers or memes based on the following three neighborhoods which are often employed in the literature. (i) Remove the job at the uth dimension and insert it in the vth dimension of the job solution $\pi(insert(\pi, u, v))$, (ii) interchange the job at the uth dimension and the job at the vth dimension of the job solution $\pi(interchange(\pi, u, v))$, (iii) swap the two neighboring jobs at the dimension u and $(u+1)$ of the job solution $\pi(swap(\pi, u, u+1))$.

According to (Schiavinotto and Stützle 2007), the diameter of *Insert* is $n-1$. That is, using *Insert* at most $n-1$ times, one solution π can transit to any other solution. The diameters of *Interchange* and *Swap* are $n-1$ and $n(n-1)/2$, respectively. Therefore, the solutions in the big valley caused by *Insert* or *Interchange* are closer to each other than those in the big valley caused by *Swap*. This means *Insert* and *Interchange* can perform a more efficient and thorough search than *Swap* with the same computational efforts. So we select one neighborhood $N_{exploitation}$ from *Insert* and *Interchange* to perform exploitation in local search. However, it is easy to fall into local optima only with a single neighborhood. Inspired by the observation that a local optimum within one neighborhood is not necessary one within another neighborhood (Mladenovic and Hansen 1997), we choose a neighborhood $N_{perturbation}$, which is different from

$N_{exploitation}$, to execute a perturbation operation before perform exploitation.

As for MNFSSPs, different objectives, which are usually conflicting or not positively correlated, cause different shapes of big valleys. Thus, it is very difficult for a solution to simultaneously reach the bottoms of all big valleys. This indicates that only one type of meme is unlikely to efficiently exploit the solution space. Geiger (2007) also showed that not a single neighborhood performs best for all multi-objective FSSPs and combining different neighborhoods in a random fashion can significantly improve the solutions quality. In addition, Neri et al. (2007) presented an adaptive multimeme algorithm for designing multidrug therapies, which utilized different memes to exploit the solution space from complementary perspectives and can obtain very satisfactory solution. Therefore, we design three types of meme to enrich the search behavior and enhance the search ability. These memes are Meme(*Interchange*, *Insert*, *LS_Len*), Meme(*Insert, Interchange*, *LS_Len*) and Meme(*Swap*, *Insert*, *LS_Len*), in which *LS_Len* denotes the exploitation depth of a meme. The general form of these memes, namely Meme ($N_{perturbation}$, $N_{exploitation}$, *LS_Len*), is given as follows:

Step 1: Convert individual $X_i(t)$ to a job permutation π_{i_0} according to the LOV rule.
Step 2: Perturbation phase.
Randomly select u and v, where $u \neq v$;

$$\pi_i = N_{pertubation}(\pi_{i\_0}, u, v); // \quad N_{perturbation}$$

Step 3: Exploitation phase.
Set *kcount* = 1;
Do
Randomly select u and v, where $u \neq v$;

$$\pi_{i\_1} = N_{\text{exp}loitation}(\pi_i, u, v); \quad // \quad N_{exploitation}$$

if $f(\pi_{i\_1})$ dominates $f(\pi_i)$, then $\pi_i = \pi_{i\_1}$;
kcount = *kcount*+1;
While *kcount* < *LS_Len*
Step 4: If $f(\pi_i)$ dominates $f(\pi_{i\_0})$, then $\pi_{i\_0} = \pi_i$.
Step 5: Convert π_{i_0} back to $X_i(t)$.

The characteristic of the above algorithm lies in two aspects. (1) In Step 2, u and v performing perturbation are randomly chosen and the new solution is always accepted, so the meme can avoid falling into local optima and also can reach different regions. (2) In Step 3, the new solution π_{i_1} is accepted only if it dominates the old solution π_i. Thus, the meme can guide the search to the promising regions nearby the bottoms of different valleys in a relatively short time and spend more time to perform $N_{exploitation}$ -based thorough search in these regions.

If $N_{pertubation} = Interchange$ and $N_{\text{exp}loitation} = Insert$, Meme($N_{perturbation}$, $N_{exploitation}$, *LS_Len*) is transformed to Meme(*Interchange*, *Insert*, *LS_Len*). Similarly, the other two memes can also be obtained.

4.5 Multi-objective handling techniques

In order to handle multi-objective no-wait FSSPs (MNFSSPs), several techniques are adopted as follows.

(1) For any big valley of MNFSSPs, the *optimal Pareto solutions* and good solutions lie not only in the bottom part but also in the sub-regions near the bottom part. Therefore, in order to exploit enough sub-regions to find all *optimal Pareto solutions*, both global search and local search should be stressed and balanced. In the former (Qian et al. 2009), we investigated how to reasonably fuse DE-based global search and problem-based local search for solving multi-objective FSSPs with limited buffers and found that applying local search to 1/4–1/5 individuals in population can achieve better results. Based on our experiments, similar conclusion can also be drawn for MNFSSPs. In MADE, 1/5 individuals are selected to apply local search.
(2) In MADE, a tentative non-dominated solutions set S is used to store the obtained non-dominated job permutations and the corresponding individuals. This elitism can improve the optimization speed of multi-objective GA (Zitzler et al. 2000). According to (Jonathan et al. 2003), restricting the number of solutions in S can induce retreating and shrinking estimated *Pareto fronts*. Fortunately, the number of solutions in S is usually less than 30. Thus, it is not necessary to define an upper bound for S. At every generation, S is updated by the new population. In particular, if a solution in the new population is dominated by any solution in S, it will be discard; otherwise, it will be added to S, and all the solutions dominated by the added one are deleted from S.
(3) To enrich the searching direction and to speed up the total convergence process, in MADE all non-dominated solutions in S are treated equally, and one solution randomly selected from S is used as the best individual *bestit* (base vector) of the current population.
(4) To obtain non-dominated solutions with reasonable diversity and good proximity, two measures, whose effectiveness has been validated in (Qian et al. 2009), are adopted at the DE's *selection* step in MADE. The first one is that the *trail vector* is compared with the individual $r1$ rather than individual i. The second one

is that the dominated solution can be accepted with a small probability.

4.6 Procedure of MADE

Based on the above solution conversion, speed-up computing method, DE-based search, problem-dependent local search, solution repair mechanism and multi-objective handling techniques, the procedure of MADE is proposed as follows:

Based on the above sub-sections, the procedure of MADE is given as follows:

Step 0: Let G denote a generation, $Pop(t)$ a population with size PS in generation t, $X_i(t)$ the ith individual with dimension $N(N = n)$ in $Pop(t)$, $x_{i,j}(t)$ the jth variable of individual $X_i(t)$, tmp_j the jth variable of tmp, CR the crossover probability, and $random(0, 1)$ the random value in the interval [0,1]. The values of the objectives of each individual are calculated by speed-up method.

Step 1: Calculate and save $MD(\pi_i, \pi_j)$ and $P_{sum}(\pi_j)(\pi_i, \pi_j \in \{1, \ldots, n\})$.

Step 2: Input N, PS, $CR \in [0, 1]$. Set $S = \phi$, and let initial bounds be $lower(x_{i,j}) = 0$, $upper(x_{i,j}) = 4$, $j = 1, \ldots, N$. As for *DE/rand-to-best/1/exp* scheme, the mutation needs to randomly choose different $r1$, $r2$ and index i from $\{1, 2, \ldots, PS\}$. So *PS* must be greater or equal to 3.

Step 3: Population initialization.
Generate $x_{i,j}(0) = lower(x_{i,j}) + random(0, 1) * (upper(x_{i,j}) - lower(x_{i,j}))$, $j = 1, \ldots, N$ for $i = 1, \ldots, PS$.

Step 4: Update S and let $t = 1$.

Step 5: Evolution phase (between Step 5 and Step 12). Let $i = 1$.

Step 6: Set *the trial vector* $tmp = X_i(t - 1)$ and $L = 0$. Randomly select an individual from S as $bestit$, randomly select $j \in (1, \ldots, N)$, and randomly select $r1, r2 \in (1, \ldots, PS)$, where $r1 \neq r2 \neq i$.

Step 7: Perform DE's *mutation* and *crossover*.

Step 7.1: Let $tmp_j = tmp_j + F * (bestit_j - tmp_j) + F * (x_{r1,j}(t - 1) - x_{r2,j}(t - 1))$.

If $tmp_j < lower(x_{i,j})$, then let $tmp_j = 2 * lower(x_{i,j}) - tmp_j$.
If $tmp_j > upper(x_{i,j})$, then let $tmp_j = 2 * upper(x_{i,j}) - tmp_j$.

Step 7.2: Set $j = (j + 1) \bmod N$ and $L = L + 1$.

Step 7.3: If $j = 0$, then $j = N$.

Step 7.4: If $(random(0, 1) < CR)$ and $(L < N)$, then go to Step 7.1.

Step 8: Perform DE's *selection*.
If $((tmp$ dominates $X_{r1}(t-1))$ or $(random(0, 1) < 0.05))$, then let $X_i(t) = tmp$; else, let $X_i(t) = X_i(t - 1)$.

Step 9: If (i mod 5) = 1, then
Randomly select $LS \in (1, 2, 3)$;
If *LS*= 1 then apply Meme(*Interchange*, *Insert*, *LS_Len*) to $X_i(t)$;
If *LS*= 2 then apply Meme(*Insert, Interchange*, *LS_Len*) to $X_i(t)$;
If *LS*= 3 then apply Meme(*Swap*, *Insert*, *LS_Len*) to $X_i(t)$;

Step 10: Let $i = i + 1$. If $i < PS$, then go to Step 6.

Step 11: Update S.

Step 12: Let $t = t + 1$. If $t < t\_max$, then go to Step 5.

Step 13: Output S.

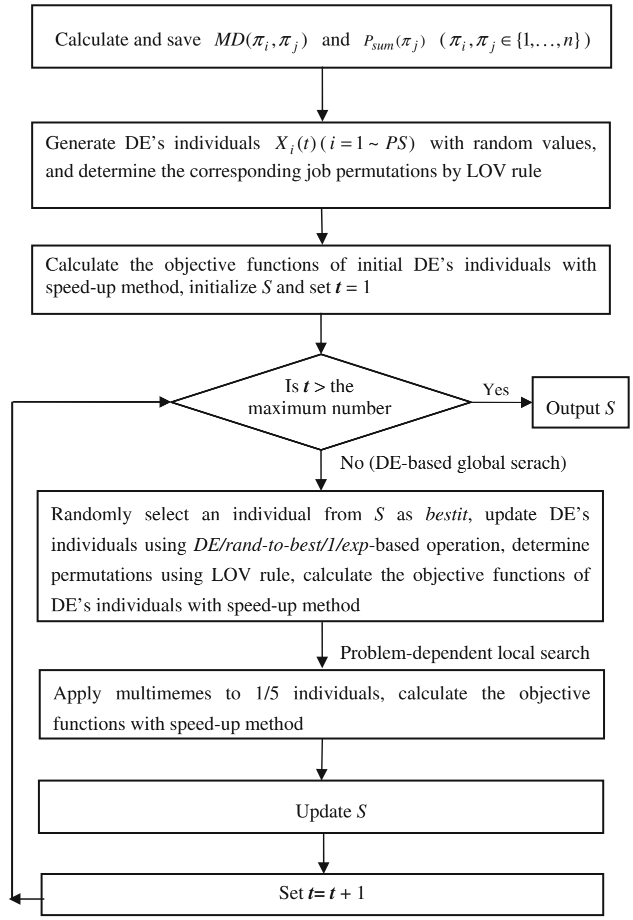

Fig. 2 The framework of the MADE

To be more straightforward, a MADE framework is illustrated in Fig. 2. It can be seen that DE and memes are hybridized. On one aspect, the inherent random and scatter search mechanism of DE and special designed DE's *selection* are utilized to find enough promising sub-regions over the whole solution space. On the other aspect, three problem-specific memes are applied to perform thorough exploitation in certain sub-regions. Due to the reasonable fusion of DE and memes, searching behavior can be enriched, and global exploration and local exploitation are stressed and well balanced.

5 Simulation results and comparisons

5.1 Experimental setup

To test the performances of the proposed MADE for MNFSSPs, numerical simulations are carried out with 28 well-studied benchmarks (i.e., Car1, Car5, Car8, Rec01, Rec05, Rec09, Rec11, Rec15, Rec19, Rec21, Rec25, Rec29, Rec31, Rec35, Rec39, Ta061, Ta065, Hel1, Ta071, Ta075, Ta081, Ta085, Ta091, Ta095, Ta101, Ta105, Ta111, Ta115) with different scales (Carlier 1978; Reeves 1995; Taillard; Heller 1960). In proposed MADE, parameters are set as follows: the population size $PS = 60$, the scaling factor $F = 0.7$, the crossover parameter $CR = 0.1$, the meme's exploitation depth $LS\_Len = 12n$ when $n < 75$, $LS\_Len = n^2/6$ when $n = 100$, $LS\_Len = n^2/15$ when $n = 200$, $LS\_Len = n^2/30$ when $n = 500$.

To analyze the effectiveness of MADE, two variants of MADE are compared, whose abbreviations are as follows:

(1): MADE_nospeedup: MADE without speed-up method.
(2): MADE_noSL: MADE without speed-up method and local searchers (i.e., memes).

Moreover, a famous multi-objective optimization algorithm, namely IMMOGLS2 (Ishibuchi et al. 2003), is also adopted for comparison. In IMMOGLS2, the population size is also set as $PS = 60$, and other parameters are set the same as those in (Ishibuchi et al. 2003). That is, crossover probability=0.9, mutation probability=0.6, number of elite solutions $(N_{elite}) = 10$, number of neighbors to be examined $(k) = 2$, tournament size=5, and local search probability $(p_{LS}) = 0.8$.

All algorithms are coded in Delphi 6.0 and experiments are executed on the same PC with Pentium IV 3.0 GHz CPU and 1 GB memory. Each benchmark is independently run 10 times with every algorithm for comparison.

In Sects. 5.3 and 5.4, makespan $C_{\max}(\pi)$ and maximum tardiness $T_{\max}(\pi)$ are considered as criteria. We set the two objectives as follows:

$$\text{Minimize } f_1(\pi) = C_{\max}(\pi) \text{ and } f_2(\pi) = T_{\max}(\pi), \quad (12)$$

where π is a job permutation.

In the real application, the total machine idleness I_{sum} sometimes plays a key role and has to be regarded as unutilized resources (Geiger 2007), and the total number of tardy jobs $N_T(\pi)$ is often monitored and relative to which managers are measured (Pinedo 2002). In order to examine the performance of MADE under these criteria, $I_{sum}(\pi)$ and total $N_T(\pi)$ are also used as criteria. Because IMMOGLS2 adopts a scalar fitness function with random weights to handle multi-objectives and the value of $N_T(\pi)$ is much smaller $I_{sum}(\pi)$, we set the two objectives as follows:

$$\text{Minimize } f_1(\pi) = I_{sum}(\pi) \text{ and } f_2(\pi) = 1000^* N_T(\pi), \quad (13)$$

where constant multipliers are used in equation (13) to handle the two criteria.

Since MADE uses the concept of *Pareto dominance* to deal with multiple objectives, the performance of MADE is independent from constant multipliers. During the search process, both MADE and IMMOGLS2 use the above constant multipliers. But when evaluating the obtained non-dominated solutions by the performance metrics in Sect. 5.2, we set all constant multipliers as 1.

The due date of each job is specified as follows:

(1) For each problem p, randomly generate a permutation π^r of the jobs.
(2) Calculate the completion time of each job in the permutation π^r.
(3) Specify the due data of each job by

$$d_{p,j} = c_{p,j} + random[-C_p(\pi^r)/50, C_p(\pi^r)/50], \quad (14)$$

where $d_{p,j}$ is the due date of job j to problem p, $c_{p,j}$ is the completion time of job π_j^r to problem p, $C_p(\pi^r)$ is the makespan of π^r to problem p, and $random[-C_p(\pi^r)/50, C_p(\pi^r)/50]$ is a random value in the interval $[-C_p(\pi^r)/50, C_p(\pi^r)/50]$.

5.2 Performance metrics

Unlike single-objective problems, proper comparison of two multi-objective algorithms itself is a multi-objective problem. That is, the two goals mentioned in Sect. 2.2.2 should be considered. Here, four performance metrics are used to evaluate the searching quality of the algorithm.

(1) Ratio of non-dominated solution (*RNDS*)
Let S denote the union of the K non-dominated solution sets (i.e., $S = S_1 \cup \cdots \cup S_K$). A straightforward performance metric (Ishibuchi et al. 2003) of the non-dominated solution set S_j with respect to the K non-dominated solution sets is the ratio of solutions in S_j that are not dominated by any other solutions in S. This metric is written as:

$$RNDS(S_j) = \frac{|S_j - \{\mathbf{x} \in S_j | \exists \mathbf{y} \in S : \mathbf{y} \succ \mathbf{x}\}|}{|S_j|}, \quad (15)$$

where $y \prec x$ means that the solution x is dominated by the solution y. In the numerator of (15), dominated

Table 4 Comparisons of MADE_noSL with IMMOGLS2 when considering $f = (C_{\max}, T_{\max})$ (same running time)

Problem	n, m	*Tavg*	MADE_noSL				IMMOGLS2			
			ONVG	*ONSN*	DI_R	*AQ*	*ONVG*	*ONSN*	DI_R	*AQ*
Car1	11,5	2.201	7.800	7.000	4.894	4221.411	9.500	9.200	1.994	4202.432
Car5	10,6	2.005	8.400	6.900	4.634	4883.763	9.000	9.000	0.050	4830.118
Car8	8,8	1.532	6.000	6.000	10.587	4782.080	6.900	6.900	0.473	4761.811
Rec01	20,5	4.925	9.300	2.500	25.769	859.575	9.300	9.100	8.126	807.743
Rec05	20,5	5.130	9.400	3.000	22.483	830.390	12.700	12.400	5.497	794.026
Rec09	20,10	6.181	8.200	1.800	32.450	1179.760	12.600	12.400	4.897	1091.125
Rec11	20,10	6.386	7.400	1.300	38.893	1043.839	9.400	9.300	6.704	966.840
Rec15	20,15	7.046	10.400	0.900	30.217	1397.505	12.600	12.500	2.940	1306.010
Rec19	30,10	10.984	8.800	0.900	50.721	1812.499	12.400	12.400	4.867	1608.427
Rec21	30,10	10.974	9.100	0.600	65.397	1795.125	10.900	10.900	3.270	1542.850
Rec25	30,15	13.400	8.900	0.700	57.744	2205.373	8.400	8.400	8.539	1952.038
Rec29	30,15	13.373	7.900	1.900	58.365	2095.286	11.700	11.700	9.005	1805.912
Rec31	50,10	26.791	9.000	0.000	87.161	3169.323	10.000	10.000	0.000	2483.894
Rec35	50,10	26.622	9.300	0.100	85.648	3337.035	16.100	16.100	0.209	2600.327
Rec39	75,20	82.537	8.400	0.000	97.130	6841.359	16.500	16.500	0.000	5003.819
Ta061	100,5	93.945	10.700	0.000	101.894	5339.674	15.200	15.200	0.000	3956.172
Ta065	100,5	93.514	9.500	0.000	103.903	5305.148	13.300	13.300	0.000	3816.982
Hel1	100,10	138.139	8.500	0.000	107.285	624.380	7.400	7.400	0.000	427.535
Ta071	100,10	133.516	11.000	0.000	99.681	7065.041	19.400	19.400	0.000	5005.006
Ta075	100,10	133.952	10.600	0.000	103.521	7047.056	17.400	17.400	0.000	4880.311
Ta081	100,20	201.217	10.200	0.000	103.130	9512.533	18.400	18.400	0.000	6483.817
Ta085	100,20	202.622	10.900	0.000	98.762	9174.423	20.300	20.300	0.000	6312.293
Ta091	200,10	437.112	12.100	0.000	112.416	15435.132	15.000	15.000	0.000	9998.205
Ta095	200,10	436.875	7.300	0.000	115.981	15615.812	12.300	12.300	0.000	9879.640
Ta101	200,20	658.878	8.800	0.000	116.670	20047.310	16.500	16.500	0.000	12768.669
Ta105	200,20	657.628	8.600	0.000	118.055	20647.405	17.500	17.500	0.000	12654.251
Ta111	500,20	5156.385	9.100	0.000	128.427	56798.486	10.100	10.100	0.000	31407.120
Ta115	500,20	5172.271	12.700	0.000	126.851	57069.386	14.200	14.200	0.000	33055.223
Average			9.225	1.200	75.310	9647.718	13.036	12.993	2.020	6300.093

solutions x by other solutions y in S are removed from S_j. $|S_j|$ is the number of solutions in S_j. $RNDS(S_j) = 1$ means that all solutions in S_j are not dominated by any solutions in S. On the contrary, $RNDS(S_j) = 0$ represents that each solution in S_j is dominated by some solutions in S. The higher the ratio $RNDS(S_j)$ is, the better the solution set S_j is.

(2) Overall non-dominated vector generation (*ONVG*)
For an obtained non-dominated solution set S_j, the metric *ONVG* is defined as $|S_j|$, which is the number of distinct non-dominated solutions.

(3) Overall non-dominated solutions number (*ONSN*)
The metric *ONSN* is the number of those solutions in S_j not dominated by any other solutions in S:

$$ONSN(S_j) = |S_j - \{\mathbf{x} \in S_j | \exists \mathbf{y} \in S : \mathbf{y} \succ \mathbf{x}\}|. \quad (16)$$

The larger the value of $ONSN(S_j)$ is, the better the solution set S_j is.

(4) Average distance (DI_R)
Let $d_{\mathbf{yx}}(S_j)$ denote the shortest normalized distance from a reference solution $\mathbf{y}$ to a solution set S_j, which is given as follows:

$$d_{\mathbf{yx}}(S_j) = \min_{\mathbf{x} \in S_j} \left\{ \sqrt{\sum_{i=1}^{w} \left(\frac{f_i(\mathbf{y}) - f_i(\mathbf{x})}{f_i^{\max}(\cdot) - f_i^{\min}(\cdot)} \right)^2} \right\}, \quad (17)$$

where $\mathbf{y}$ belongs to the reference solution set S^*, and $f_i(\cdot)$ is the ith objective value, and $f_i^{\max}(\cdot)$ and $f_i^{\min}(\cdot)$ are the maximum and minimum value of the ith objective in S, respectively. If the *optimal Pareto front* is not

Table 5 Comparisons of MADE_noSL with IMMOGLS2 when considering $f = (I_{\text{sum}}, N_T)$ (same running time)

Problem	n, m	$Tavg$	MADE_noSL				IMMOGLS2			
			$ONVG$	$ONSN$	DI_R	AQ	$ONVG$	$ONSN$	DI_R	AQ
Car1	11,5	1.962	2.200	0.800	29.513	6448.268	3.000	2.800	0.467	6290.819
Car5	10,6	1.966	1.700	1.300	37.646	11102.719	3.100	3.000	1.264	10954.321
Car8	8,8	1.510	2.800	1.100	5.511	17006.262	2.000	2.000	0.000	16890.764
Rec01	20,5	4.351	3.000	0.100	53.612	1271.101	3.500	3.400	1.228	1072.022
Rec05	20,5	4.859	3.300	0.100	50.277	1319.162	4.300	4.300	0.613	1150.131
Rec09	20,10	5.424	3.000	0.100	52.631	5485.688	4.300	4.200	0.250	4849.659
Rec11	20,10	6.183	3.000	0.100	58.111	4811.464	3.900	3.900	1.359	4249.716
Rec15	20,15	6.445	2.300	0.200	61.855	10200.901	3.700	3.600	2.784	9262.229
Rec19	30,10	10.118	4.100	0.000	76.427	8432.260	4.200	4.200	0.000	6502.370
Rec21	30,10	10.069	3.700	0.100	69.759	8313.107	5.100	5.100	0.907	6335.484
Rec25	30,15	12.623	3.900	0.100	74.005	17884.648	5.300	5.300	0.538	13973.767
Rec29	30,15	12.748	3.800	0.200	72.547	17526.069	4.400	4.400	2.352	13065.492
Rec31	50,10	24.644	4.700	0.000	96.090	15127.245	5.400	5.400	0.000	9469.133
Rec35	50,10	24.452	4.700	0.000	93.376	15792.367	5.100	5.100	0.000	9987.971
Rec39	75,20	79.368	5.700	0.000	102.626	79810.039	6.400	6.400	0.000	50073.733
Ta061	100,5	89.339	6.200	0.000	108.461	9232.315	6.600	6.600	0.000	4250.400
Ta065	100,5	88.289	5.700	0.000	109.956	9345.865	7.200	7.200	0.000	3969.264
Hel1	100,10	132.930	6.400	0.000	106.158	2858.441	7.100	7.100	0.000	1487.329
Ta071	100,10	128.686	6.000	0.000	108.378	32002.972	7.300	7.300	0.000	16932.977
Ta075	100,10	127.942	5.700	0.000	108.395	33267.009	7.300	7.300	0.000	17244.356
Ta081	100,20	201.776	5.800	0.000	109.854	108575.143	7.900	7.900	0.000	63098.259
Ta085	100,20	205.230	5.700	0.000	107.509	103677.036	6.100	6.100	0.000	59754.862
Ta091	200,10	435.360	6.300	0.000	119.863	69583.990	9.100	9.100	0.000	33434.113
Ta095	200,10	436.854	8.500	0.000	119.924	70300.940	10.300	10.300	0.000	33399.257
Ta101	200,20	660.289	8.000	0.000	119.132	214568.344	8.600	8.600	0.000	115728.581
Ta105	200,20	657.980	8.100	0.000	118.778	217651.066	8.300	8.300	0.000	116083.820
Ta111	500,20	4976.331	12.300	0.000	127.879	575892.948	9.700	9.700	0.000	277332.576
Ta115	500,20	5013.203	9.100	0.000	127.430	573640.086	8.200	8.200	0.000	280857.271
Average			5.204	0.150	86.632	80040.266	5.979	5.957	0.420	42417.881

known, we will combine these K non-dominated solution sets and select all the non-dominated solutions to form the set S^*.
The average distance $DI_R(S_j)$ is the average of those shortest normalized distances from all the reference solutions to S_j (Czyzak and Jaszkiewicz 1998; Knowles and Corne 2002), that is,

$$DI_R(S_j) = \frac{1}{|S^*|} \sum_{y \in S*} d_{\mathbf{yx}}(S_j). \tag{18}$$

According to (Ishibuchi et al. 2003), $DI_R(S_j)$ can be used to evaluate the spread of S_j as well as the proximity of S_j to the reference set S^*. Obviously, smaller $DI_R(S_j)$ values correspond to a better convergence performance to S^*. If $DI_R(S_j) = 0$, all the reference solutions in S^* are included in the solution set S_j.

(5) Average Quality (AQ)
In (Jaszkiewicz 2003), a metric was designed to measure the quality of the solution set, which was originally expressed in the form of weighted Tchebycheff function. But that function may hide certain aspects about the quality of solution set because poor performance with respect to convergence could be compensated by good performance in distribution of solutions. So, diversity indicators of spread and space are added to the formulation to overcome the limitation, and a metric is given by the average value of a scalarized function over a representative sample of weight vectors as follow:

$$AQ = \sum_{\lambda \in \Lambda} s_a(\mathbf{f}, \mathbf{z^0}, \lambda, \rho)/|\Lambda| \tag{19}$$

Table 6 Comparisons of MADE_noSL with MADE_nospeedup when considering $f = (C_{\max}, T_{\max})$ (same running time)

Problem	n, m	*Tavg*	MADE_noSL				MADE_nospeedup			
			ONVG	*ONSN*	DI_R	*AQ*	*ONVG*	*ONSN*	DI_R	*AQ*
Car1	11,5	2.201	7.800	6.900	6.851	4221.411	10.500	10.500	0.516	4177.247
Car5	10,6	2.005	8.400	6.500	7.228	4883.763	9.400	9.300	1.671	4761.508
Car8	8,8	1.532	6.000	6.000	10.587	4782.080	6.900	6.900	0.473	4761.811
Rec01	20,5	4.925	9.300	0.100	35.784	859.575	12.900	12.900	0.237	780.550
Rec05	20,5	5.130	9.400	0.200	30.339	830.390	13.300	13.300	0.223	775.708
Rec09	20,10	6.181	8.200	0.000	40.376	1179.760	13.100	13.100	0.000	1059.114
Rec11	20,10	6.386	7.400	0.000	48.547	1043.839	6.200	6.200	0.000	967.713
Rec15	20,15	7.046	10.400	0.000	32.445	1397.505	16.800	16.800	0.000	1287.958
Rec19	30,10	10.984	8.800	0.000	55.610	1812.499	10.700	10.700	0.000	1520.520
Rec21	30,10	10.974	9.100	0.000	71.544	1795.125	11.500	11.500	0.000	1482.138
Rec25	30,15	13.400	8.900	0.000	64.226	2205.373	10.600	10.600	0.000	1905.917
Rec29	30,15	13.373	7.900	0.000	65.484	2095.286	12.100	12.100	0.000	1761.073
Rec31	50,10	26.791	9.000	0.000	85.853	3169.323	12.700	12.700	0.000	2343.021
Rec35	50,10	26.622	9.300	0.000	83.835	3337.035	13.700	13.700	0.000	2418.707
Rec39	75,20	82.537	8.400	0.000	98.716	6841.359	9.900	9.900	0.000	4638.389
Ta061	100,5	93.945	10.700	0.000	103.620	5339.674	12.400	12.400	0.000	3533.217
Ta065	100,5	93.514	9.500	0.000	101.229	5305.148	13.200	13.200	0.000	3385.204
Hel1	100,10	138.139	8.500	0.000	105.861	624.380	9.700	9.700	0.000	390.130
Ta071	100,10	133.516	11.000	0.000	100.326	7065.041	11.300	11.300	0.000	4475.233
Ta075	100,10	133.952	10.600	0.000	103.556	7047.056	13.100	13.100	0.000	4441.916
Ta081	100,20	201.217	10.200	0.000	101.339	9512.533	12.800	12.800	0.000	5863.456
Ta085	100,20	202.622	10.900	0.000	100.241	9174.423	13.200	13.200	0.000	5787.260
Ta091	200,10	437.112	12.100	0.000	110.936	15435.132	13.200	13.200	0.000	8730.636
Ta095	200,10	436.875	7.300	0.000	112.335	15615.812	12.700	12.700	0.000	8766.039
Ta101	200,20	658.878	8.800	0.000	112.893	20047.310	11.100	11.100	0.000	11370.351
Ta105	200,20	657.628	8.600	0.000	114.678	20647.405	10.300	10.300	0.000	11331.889
Ta111	500,20	5156.385	9.100	0.000	123.361	56798.486	12.600	12.600	0.000	28300.375
Ta115	500,20	5172.271	12.700	0.000	116.829	57069.386	13.100	13.100	0.000	28372.362
Average			9.225	0.704	76.594	9647.718	11.750	11.746	0.111	5692.480

where $s_a(\mathbf{f}, \mathbf{z^0}, \lambda, \rho) = \min_i\{\max_j\{\lambda_j(f_j(\mathbf{x}_i) - z_j^0)\} + \rho\sum_{j=1}^{w}\lambda_j(f_j(\mathbf{x}_i) - z_j^0)\}$, and $f_j(\cdot)$ is the jth objective, and $\Lambda = \{\lambda = (\lambda_1, \ldots, \lambda_w)|\lambda_j \in \{0, 1/r, 2/r, \ldots, 1\}, \sum_{j=1}^{w}\lambda_j = 1\}$, and $\mathbf{z}^0$ is a reference point in the objective space that is set to (0, 0) for two-objective problems, and ρ is a sufficiently small number which is set to 0.01 in this paper. Besides, r is a parameter changed as the number of objectives set as 50. *AQ* can evaluate both the convergence performance and diversity of the solution set. Lower metric value represents better solution set.

5.3 Comparisons of MADE_noSL, IMMOGLS2 and MADE_nospeedup

In this subsection, we set the maximum generation of MADE_nospeedup as *t_max*=300 and let MADE_noSL and IMMOGLS2 run at the same time as MADE_nospeedup. The average running time (second) of each instance, namely *Tavg*, is given in the corresponding tables.

5.3.1 Comparisons of MADE_noSL with IMMOGLS2

In order to test DE's global search ability, we compare MADE_noSL with IMMOGLS2. Simulation results on $f = (C_{\max}, T_{\max})$ and $f = (I_{sum}, N_T)$ can be found in Tables 4 and 5, respectively.

In Tables 4 and 5, it can be seen from *ONVG* metric and *ONSN* metric that IMMOGLS2 can obtain obviously more non-dominated solutions with better convergence performance than MADE_noSL. From DI_R metric, it can be seen that the DI_R values of IMMOGLS2 are much smaller than those of MADE_noSL. This means that IMMOGLS2 can obtain solutions closer to the *optimal Pareto front*

Table 7 Comparisons of MADE_noSL with MADE_nospeedup when considering $f = (I_{\text{sum}}, N_T)$ (same running time)

Problem	n, m	*Tavg*	MADE_noSL				MADE_nospeedup			
			ONVG	*ONSN*	DI_R	*AQ*	*ONVG*	*ONSN*	DI_R	*AQ*
Car1	11,5	1.962	2.200	0.600	29.681	6448.268	2.500	2.500	5.119	6266.588
Car5	10,6	1.966	1.700	1.300	37.089	11102.719	3.100	2.900	1.660	10947.810
Car8	8,8	1.510	2.800	1.100	5.511	17006.262	2.000	2.000	0.000	16890.764
Rec01	20,5	4.351	3.000	0.100	69.779	1271.101	3.400	3.400	0.944	1013.571
Rec05	20,5	4.859	3.300	0.000	61.036	1319.162	4.300	4.300	0.000	1095.335
Rec09	20,10	5.424	3.000	0.000	62.541	5485.688	4.500	4.500	0.000	4671.140
Rec11	20,10	6.183	3.000	0.000	60.727	4811.464	4.400	4.400	0.000	4092.068
Rec15	20,15	6.445	2.300	0.000	68.257	10200.901	3.300	3.300	0.000	9046.000
Rec19	30,10	10.118	4.100	0.000	84.596	8432.260	4.200	4.200	0.000	6193.537
Rec21	30,10	10.069	3.700	0.000	78.235	8313.107	4.400	4.400	0.000	6128.207
Rec25	30,15	12.623	3.900	0.000	79.948	17884.648	4.500	4.500	0.000	13753.284
Rec29	30,15	12.748	3.800	0.000	80.605	17526.069	4.100	4.100	0.000	12636.022
Rec31	50,10	24.644	4.700	0.000	97.303	15127.245	4.800	4.800	0.000	9232.426
Rec35	50,10	24.452	4.700	0.000	94.346	15792.367	4.700	4.700	0.000	9787.135
Rec39	75,20	79.368	5.700	0.000	105.474	79810.039	5.300	5.300	0.000	49885.688
Ta061	100,5	89.339	6.200	0.000	110.828	9232.315	5.300	5.300	0.000	4012.488
Ta065	100,5	88.289	5.700	0.000	114.042	9345.865	6.400	6.400	0.000	3823.921
Hel1	100,10	132.930	6.400	0.000	109.835	2858.441	4.900	4.900	0.000	1432.022
Ta071	100,10	128.686	6.000	0.000	111.971	32002.972	5.600	5.600	0.000	16601.863
Ta075	100,10	127.942	5.700	0.000	111.445	33267.009	6.100	6.100	0.000	16983.272
Ta081	100,20	201.776	5.800	0.000	114.387	108575.143	4.900	4.900	0.000	62198.099
Ta085	100,20	205.230	5.700	0.000	109.491	103677.036	5.800	5.800	0.000	59750.541
Ta091	200,10	435.360	6.300	0.000	124.324	69583.990	4.600	4.600	0.000	33021.617
Ta095	200,10	436.854	8.500	0.000	123.447	70300.940	5.800	5.800	0.000	33146.905
Ta101	200,20	660.289	8.000	0.000	121.001	214568.344	5.500	5.500	0.000	115171.246
Ta105	200,20	657.980	8.100	0.000	119.916	217651.066	5.600	5.600	0.000	116028.471
Ta111	500,20	4976.331	12.300	0.000	128.798	575892.948	5.700	5.700	0.000	269275.363
Ta115	500,20	4997.203	9.100	0.000	127.558	573640.086	5.200	5.200	0.000	270432.812
Average			5.204	0.111	90.792	80040.266	4.675	4.668	0.276	41554.221

than MADE_noSL. However, the values of *RNDS*, *ONSN* and DI_R metrics can not directly reflect the diversity of the solution set. So, we use *AQ* metric that considers both the convergence performance and diversity for comprehensive comparisons. From Tables 4 and 5, it can be found that the *AQ* values of IMMOGLS2 are less than those of MADE_noSL. That is to say, the solutions obtained by IMMOGLS2 are closer to the *optimal Pareto front* than those obtained by MADE_noSL and the solutions obtained by IMMOGLS2 distribute more uniformly and cover more area of the *optimal Pareto front*.

So it is concluded that DE's global search is not suitable to perform thorough exploitation in the promising regions and has no enough ability to obtain non-dominated solutions with good performance.

5.3.2 *Comparisons of MADE_noSL with MADE_nospeedup*

Next we compare MADE_noSL with MADE_nospeedup to examine the effect of memes on the performance of our MADE_nospeedup. The statistics of performance metrics are given in Tables 6 and 7, respectively.

Tables 6 and 7 show the statistical results produced by MADE_nospeedup are much better than those by MADE_noSL for every instance. As the size of the instance increases, the superiority of MADE_nospeedup over MADE_noSL also increases. This indicates that by embedding the proposed memes into MADE_noSL to enhance exploitation it becomes more efficient and effective to obtain an approximate *optimal Pareto set* with high quality.

Table 8 Comparisons of MADE_nospeedup with IMMOGLS2 when considering $f = (C_{\max}, T_{\max})$ (same running time)

Problem	*Tavg*	MADE_nospeedup					IMMOGLS2				
		RNDS	*ONSN*	DI_R	*TGen*	*TET*	*RNDS*	*ONSN*	DI_R	*TGen*	*TET*
Car1	2.201	0.981	10.300	0.936	300	500460	0.926	8.800	4.055	1117	213050
Car5	2.005	0.989	9.300	1.856	300	457260	0.944	8.500	3.698	1048	204457
Car8	1.532	1.000	6.900	0.000	300	370860	1.000	6.900	0.000	803	157865
Rec01	4.925	0.992	12.800	0.112	300	889260	0.043	0.400	18.132	2452	431492
Rec05	5.130	0.925	12.300	0.961	300	889260	0.157	2.000	15.562	2442	432015
Rec09	6.181	0.977	12.800	0.935	300	889260	0.079	1.000	17.984	2714	476850
Rec11	6.386	0.887	5.500	7.594	300	889260	0.436	4.100	12.037	2704	474582
Rec15	7.046	0.881	14.800	0.802	300	889260	0.302	3.800	10.194	2962	517233
Rec19	10.984	0.953	10.200	1.004	300	1321260	0.097	1.200	24.439	4588	777589
Rec21	10.974	0.991	11.400	1.108	300	1321260	0.083	0.900	30.157	4594	778664
Rec25	13.400	0.953	10.100	0.986	300	1321260	0.155	1.300	29.019	4933	836617
Rec29	13.373	0.917	11.100	3.768	300	1321260	0.291	3.400	26.349	4969	836932
Rec31	26.791	0.945	12.000	1.293	300	2185260	0.220	2.200	34.254	8934	1465672
Rec35	26.622	0.942	12.900	1.852	300	2185260	0.217	3.500	25.536	8939	1464873
Rec39	82.537	0.949	9.400	4.794	300	3265260	0.218	3.600	41.084	16389	2644907
Ta061	93.945	0.992	12.300	0.358	300	6026460	0.039	0.600	48.857	25031	4012639
Ta065	93.514	1.000	13.200	0.011	300	6026460	0.008	0.100	44.404	24971	4000829
Hel1	138.139	0.990	9.600	0.765	300	6026460	0.068	0.500	50.339	27951	4488400
Ta071	133.516	0.991	11.200	0.612	300	6026460	0.015	0.300	51.567	28021	4477938
Ta075	133.952	1.000	13.100	0.000	300	6026460	0.000	0.000	51.771	27991	4476658
Ta081	201.217	0.938	12.000	1.179	300	6026460	0.196	3.600	41.062	30651	4899760
Ta085	202.622	0.970	12.800	2.348	300	6026460	0.148	3.000	43.256	30611	4899094
Ta091	437.112	0.970	12.800	0.685	300	9626460	0.087	1.300	53.196	48001	7608995
Ta095	436.875	0.992	12.600	0.073	300	9626460	0.016	0.200	44.984	48068	7617100
Ta101	658.878	0.982	10.900	0.321	300	9626460	0.018	0.300	46.706	51801	8202919
Ta105	657.628	1.000	10.300	0.140	300	9626460	0.074	1.300	48.894	51901	8217402
Ta111	5156.385	1.000	12.600	0.000	300	30024060	0.000	0.000	68.139	137301	21792864
Ta115	5172.271	1.000	13.100	0.000	300	30024060	0.000	0.000	64.011	136801	21667364
Average		0.968	11.368	1.232	300	5694103	0.208	2.243	33.917	26382	4216956

5.3.3 Comparisons of MADE_nospeedup and IMMOGLS2

As for multi-objective FSSPs, IMMOGLS2 (Ishibuchi et al. 2003) is famous for its abilities to efficiently find uniformly distributed non-dominated solutions and it outperforms two famous multi-objective evolutionary algorithms, the strength Pareto evolutionary algorithm (SPEA) (Zitzler and Thiele 1999) and the revised non-dominated sorting genetic algorithm (NSGAII) (Deb et al. 2002). Thus, to investigate the search ability of MADE, we compare MADE_nospeedup with IMMOGLS2. The experiment results are illustrated in Tables 8 and 9, where *TGen* denotes the average generations.

From Tables 8 and 9, it can be found that the values of *RNDS*, *ONSN* and DI_R produced by MADE_nospeedup are much better than those by IMMOGLS2 for every problem. The *TGen* values of MADE_nospeedup are equal to 300, which are much less than those of IMMOGLS2, but the *TET* value of MADE_nospeedup is obviously larger than that of IMMOGLS2 for each instance. That is, when running at the same time, MADE_nospeedup can explore and exploit more promising regions in the whole solution space than IMMOGLS2. This is the main reason that MADE_nospeedup has relatively good performance.

To better understand the performance of MADE_nospeedup on the objectives $f = (C_{\max}, T_{\max})$, we plotted the non-dominated solutions of $S_{MADE\_nospeedup}$ (circle point) and $S_{IMMOGLS2}$ (star point) and S_{MADE_noSL} (triangle point) for Rec25 in one typical run in Fig. 3. The figure shows that the solutions obtained by MADE_nospeedup (circle point) and IMMOGLS2 (star point) are much closer to the *optimal Pareto front* than those of MADE_noSL (triangle point) and dominate all the MADE_noSL's

Table 9 Comparisons of MADE_nospeedup with IMMOGLS2 when considering $f = (I_{\text{sum}}, N_T)$ (same running time)

Problem	*Tavg*	MADE_nospeedup					IMMOGLS2				
		RNDS	*ONSN*	DI_R	*TGen*	*TET*	*RNDS*	*ONSN*	DI_R	*TGen*	*TET*
Car1	1.962	0.960	2.400	16.555	300	500460	0.700	2.100	2.583	946	204288
Car5	1.966	0.903	2.800	11.728	300	457260	0.935	2.900	7.827	915	198171
Car8	1.510	1.000	2.000	0.000	300	370860	1.000	2.000	0.000	734	161790
Rec01	4.351	1.000	3.400	2.610	300	889260	0.057	0.200	46.845	2006	411119
Rec05	4.859	0.977	4.200	0.520	300	889260	0.023	0.100	33.145	2000	404462
Rec09	5.424	0.911	4.100	1.949	300	889260	0.116	0.500	24.932	2299	457297
Rec11	6.183	0.909	4.000	1.828	300	889260	0.128	0.500	22.973	2257	456910
Rec15	6.445	0.939	3.100	3.067	300	889260	0.162	0.600	25.160	2509	493028
Rec19	10.118	0.976	4.100	0.133	300	1321260	0.048	0.200	43.740	3963	735307
Rec21	10.069	0.955	4.200	1.782	300	1321260	0.059	0.300	30.721	3934	735933
Rec25	12.623	0.911	4.100	2.852	300	1321260	0.132	0.700	24.013	4348	797826
Rec29	12.748	0.976	4.000	1.348	300	1321260	0.045	0.200	31.889	4357	803212
Rec31	24.644	0.813	3.900	4.260	300	2185260	0.204	1.100	26.574	7879	1372958
Rec35	24.452	0.702	3.300	7.252	300	2185260	0.333	1.700	21.725	7841	1365446
Rec39	79.368	0.887	4.700	6.583	300	3265260	0.281	1.800	25.932	15339	2544527
Ta061	89.339	1.000	5.300	0.000	300	6026460	0.000	0.000	45.715	23581	3932148
Ta065	88.289	0.984	6.300	0.807	300	6026460	0.056	0.400	39.465	23411	3908370
Hel1	132.930	1.000	4.900	0.000	300	6026460	0.000	0.000	49.604	26721	4465173
Ta071	128.686	0.982	5.500	1.853	300	6026460	0.082	0.600	32.793	26781	4417008
Ta075	127.942	0.984	6.000	2.812	300	6026460	0.151	1.100	34.735	26721	4407456
Ta081	201.776	0.918	4.500	3.951	300	6026460	0.165	1.300	40.698	29801	4854814
Ta085	205.230	0.793	4.600	8.402	300	6026460	0.426	2.600	29.525	29841	4866382
Ta091	435.360	1.000	4.600	3.085	300	9626460	0.033	0.300	50.849	46868	7560615
Ta095	436.854	0.914	5.300	10.268	300	9626460	0.223	2.300	40.336	46834	7555082
Ta101	660.289	0.927	5.100	19.363	300	9626460	0.349	3.000	31.199	50668	8109910
Ta105	657.980	0.893	5.000	14.450	300	9626460	0.422	3.500	28.167	50868	8140486
Ta111	4976.331	1.000	5.700	0.000	300	30024060	0.000	0.000	80.053	136601	21606733
Ta115	4997.203	1.000	5.200	0.000	300	30024060	0.000	0.000	88.523	135951	21537702
Average		0.936	4.368	4.552	300	5694103	0.219	1.071	34.276	25571	4160862

solutions. Furthermore, it can be easily seen that MADE_no speedup performs better than IMMOGLS2. The corresponding values of performance metrics are reported in Table 10. Experiment results for other problems are similar.

Moreover, typical results of a replication for the three algorithms based on the objectives $f = (I_{sum}, N_T)$ and the benchmark Rec11 are shown in Fig. 4. And the corresponding values of the performance metrics are given in Table 11. As can be seen from Fig. 4 and Table 11, MADE_nospeedup's solutions (circle point) dominate all the IMMOGLS2's solutions (star point) and all the MADE_noSL's solutions (triangle point). As for other problems, conclusions are similar.

To sum up, MADE_nospeedup is a more effective and efficient multi-objective optimization algorithm than IMMOGLS2 and MADE_noSL.

5.4 Comparisons of MADE with MADE_nospeedup

To further show the effectiveness of MADE by incorporating speed-up computing method into MADE_nospeedup, we compare MADE with MADE_nospeedup. In MADE_no speedup, both $(C_{\max}, T_{\max})$ and (I_{sum}, N_T) can be computed in $O(nm)$. And in MADE, where speed-up computing method is adopted, both the CC of computing $(C_{\max}, T_{\max})$ and that of computing (I_{sum}, N_T) are reduced to $O(n)$. Let TCC_MADE_nospeedup(*OF1, OF2*) denote the total CC of calculating two objective functions, i.e., *OF1* and *OF2*, in MADE_nospeedup, TCC_MADE (*OF1, OF2*) denote the total CC of calculating *OF1* and *OF2* in MADE. When MADE and MADE_nospeedup run at the same *TET* (i.e., K_1, times), TCC_MADE$(C_{\max}, T_{\max})$ can be reduced

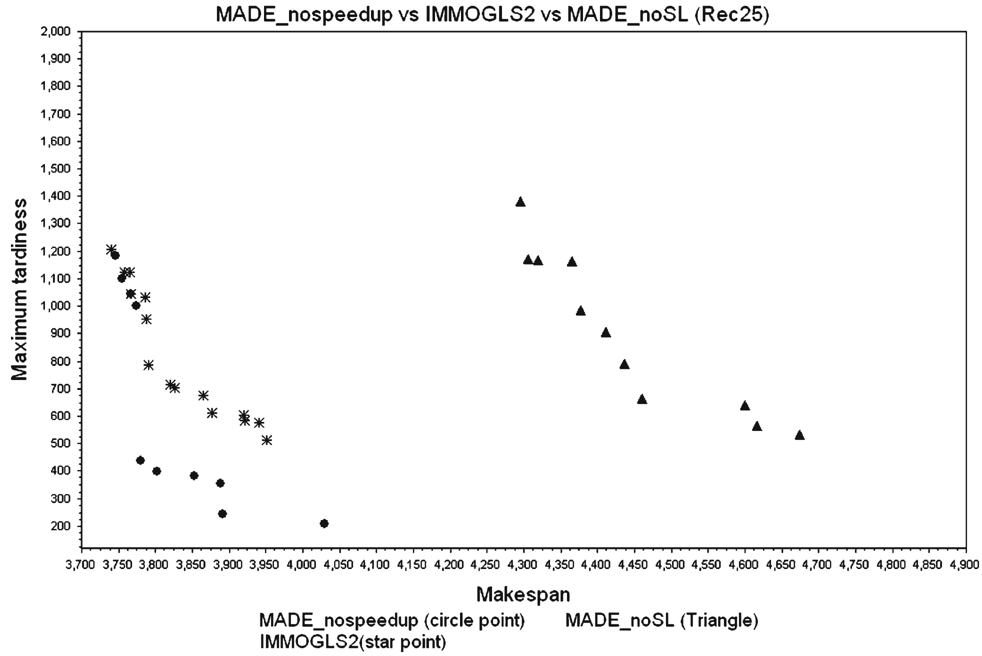

Fig. 3 Non-dominated solutions of MADE_nospeedup and IMMOGLS2 and MADE_noSL ($C_{\max}$, $T_{\max}$)

Table 10 The values of performance metrics corresponding to Fig. 3

($C_{\max}$, $T_{\max}$)	MADE_noSL vs IMMOGLS2		MADE_noSL vs. MADE_nospeedup		MADE_nospeedup vs. IMMOGLS2	
	MADE_noSL	IMMOGLS2	MADE_noSL	MADE _nospeedup	MADE _nospeedup	IMMOGLS2
RNDS	0.000	1.000	0.000	1.000	1.000	0.133
ONVG	11.000	15.000	11.000	10.000	10.000	15.000
ONSN	0.000	15.000	0.000	10.000	10.000	2.000
DI_R	58.963	0.000	66.553	0.000	0.257	22.003
AQ	2211.807	1930.320	2211.807	1904.625	1904.625	1930.320

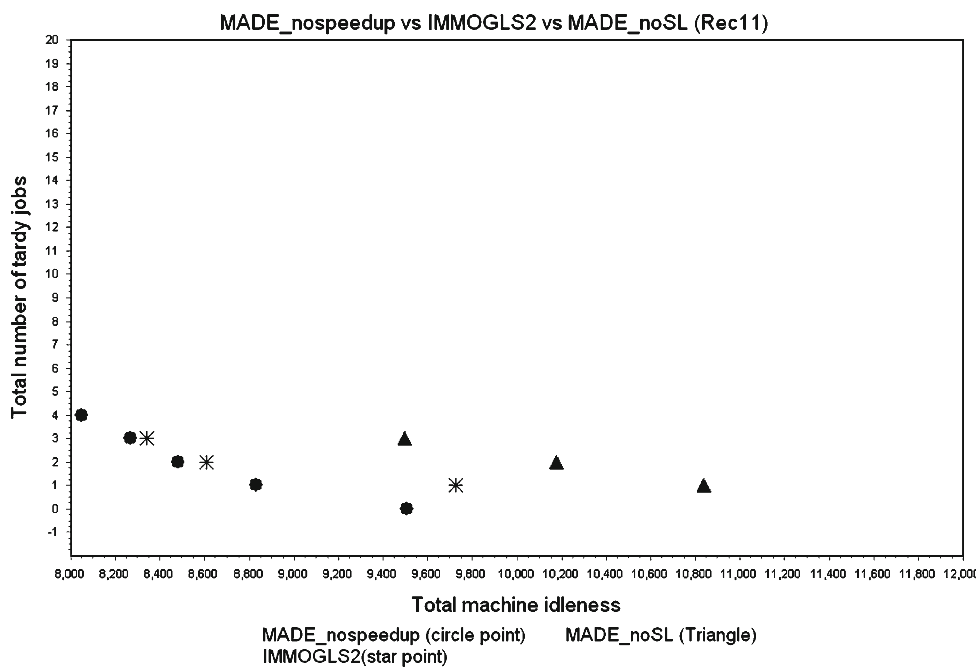

Fig. 4 Non-dominated solutions of MADE_nospeedup and IMMOGLS2 and MADE_noSL (I_{sum}, N_{T})

Table 11 The values of performance metrics corresponding to Fig. 4

(I_{sum}, N_T)	MADE_noSL vs IMMOGLS2		MADE_noSL vs. MADE_nospeedup		MADE_nospeedup vs. IMMOGLS2	
	MADE_noSL	IMMOGLS2	MADE_noSL	MADE _nospeedup	MADE _nospeedup	IMMOGLS2
RNDS	0.000	1.000	0.000	1.000	1.000	0.000
ONVG	3.000	3.000	3.000	5.000	5.000	3.000
ONSN	0.000	3.000	0.000	5.000	5.000	0.000
DI_R	53.344	0.000	50.918	0.000	0.000	19.518
AQ	4806.019	4221.229	4806.019	4078.764	4078.764	4221.229

Table 12 Comparisons of MADE with MADE_nospeedup when considering $f = (C_{\max}, T_{\max})$ (same running time)

Problem	MADE_nospeedup					MADE				
	RNDS	*ONSN*	DI_R	*AQ*	*TET*	*RNDS*	*ONSN*	DI_R	*AQ*	*TET*
Car1	1.000	10.900	1.417	4170.667	500460	0.991	11.200	0.597	4170.453	999526
Car5	0.968	9.200	1.973	4749.634	457260	0.980	9.600	0.577	4750.289	932900
Car8	1.000	7.000	0.000	4761.173	370860	1.000	7.000	0.000	4761.173	810011
Rec01	0.528	6.700	4.362	782.824	889260	0.836	11.700	0.954	779.693	1768382
Rec05	0.419	5.400	5.020	778.069	889260	0.800	10.400	1.860	776.328	1705842
Rec09	0.310	4.000	5.008	1062.080	889260	0.866	13.600	0.846	1056.934	2164076
Rec11	0.719	4.600	9.258	968.124	889260	0.975	7.900	1.323	963.206	2192827
Rec15	0.439	6.800	3.009	1286.751	889260	0.880	16.900	0.369	1284.291	2697004
Rec19	0.274	3.100	6.886	1519.210	1321260	0.795	8.900	1.702	1511.279	3030012
Rec21	0.364	3.900	7.574	1483.129	1321260	0.738	10.400	2.333	1475.876	3106201
Rec25	0.240	2.300	10.286	1906.478	1321260	0.845	9.800	1.948	1893.397	3865451
Rec29	0.391	4.500	5.262	1757.966	1321260	0.690	10.900	1.588	1760.117	3912574
Rec31	0.330	3.800	6.619	2343.239	2185260	0.779	11.300	1.765	2336.331	5401146
Rec35	0.496	6.500	4.111	2409.761	2185260	0.698	11.100	1.771	2400.081	4875970
Rec39	0.144	1.700	10.528	4646.397	3265260	0.932	12.400	0.857	4618.432	12509041
Ta061	0.397	4.800	7.216	3536.685	6026460	0.780	9.200	4.646	3527.528	13748287
Ta065	0.220	2.700	8.257	3393.402	6026460	0.844	11.900	2.092	3378.404	13947158
Hel1	0.152	1.400	11.088	390.267	6026460	0.890	9.700	0.947	387.593	18515170
Ta071	0.246	3.300	7.482	4450.146	6026460	0.835	11.600	1.920	4430.940	19728485
Ta075	0.235	2.300	10.354	4438.651	6026460	0.831	10.800	1.453	4410.699	17121062
Ta081	0.167	2.100	9.028	5873.568	6026460	0.922	14.100	0.992	5809.913	29485226
Ta085	0.153	1.700	11.638	5800.144	6026460	0.929	14.500	1.457	5734.586	29774494
Ta091	0.141	2.000	8.680	8752.709	9626460	0.989	17.300	1.542	8638.003	35355688
Ta095	0.219	2.800	10.126	8727.668	9626460	0.920	16.100	1.548	8606.164	35179204
Ta101	0.165	2.000	12.579	11207.666	9626460	0.936	13.200	0.685	11024.192	48821952
Ta105	0.091	0.900	12.432	11398.121	9626460	0.938	15.100	0.276	11256.865	51249944
Ta111	0.112	1.600	31.764	28276.972	30024060	0.910	15.100	26.069	27733.851	154273380
Ta115	0.095	1.100	14.956	28277.364	30024060	1.000	13.400	1.533	27627.245	150170100
Average	0.358	3.896	8.461	5683.888	5694103	0.876	11.968	2.273	5610.852	23833611

from $O(K_1 nm)$ of TCC_MADE_nospeedup($C_{\max}$, $T_{\max}$) to $O(K_1 n)$ and TCC_MADE(I_{sum}, N_T) can be reduced from $O(K_1 nm)$ of TCC_MADE_nospeedup(I_{sum}, N_T) to $O(K_1 n)$.

5.4.1 Comparisons based on a fixed computational budget

Firstly, we set the maximum generation of MADE_nospeedup as *t_max*=300 and let MADE run at the same time as

Table 13 Comparisons of MADE with MADE_nospeedup when considering $f = (I_{\text{sum}}, N_T)$ (same running time)

Problem	MADE_nospeedup					MADE				
	RNDS	*ONSN*	DI_R	*AQ*	*TET*	*RNDS*	*ONSN*	DI_R	*AQ*	*TET*
Car1	0.913	2.100	8.356	6266.591	500460	1.000	2.500	3.490	6266.589	711295
Car5	0.909	3.000	1.809	10945.839	457260	0.941	3.200	0.250	10941.900	601583
Car8	1.000	2.000	0.000	16890.764	370860	1.000	2.000	0.000	16890.764	547114
Rec01	0.568	2.100	13.687	1009.662	889260	0.854	3.500	5.945	1003.196	1423373
Rec05	0.286	1.200	9.499	1095.613	889260	0.864	3.800	2.122	1086.271	1267170
Rec09	0.317	1.300	8.305	4668.916	889260	0.837	4.100	1.420	4650.637	1938220
Rec11	0.553	2.100	7.588	4106.237	889260	0.756	3.100	3.642	4078.965	1566238
Rec15	0.667	2.000	6.783	9033.474	889260	0.903	2.800	1.575	9033.122	2234323
Rec19	0.147	0.500	21.763	6203.915	1321260	0.919	3.400	4.493	6155.382	3511369
Rec21	0.333	1.400	12.939	6132.472	1321260	0.750	3.600	4.877	6081.009	3292931
Rec25	0.289	1.300	17.157	13729.679	1321260	0.784	4.000	1.787	13600.439	4099744
Rec29	0.300	1.200	14.968	12661.646	1321260	0.778	3.500	4.321	12478.479	4130572
Rec31	0.268	1.100	15.538	9193.316	2185260	0.783	4.700	2.385	9107.949	6704254
Rec35	0.298	1.400	15.193	9830.845	2185260	0.750	4.500	3.489	9700.398	6662006
Rec39	0.091	0.300	23.802	49939.014	3265260	0.902	5.500	0.850	49022.620	19307188
Ta061	0.415	2.200	14.964	4045.913	6026460	0.815	5.300	4.521	3981.624	19917312
Ta065	0.278	1.000	19.572	3800.234	6026460	0.786	4.400	4.447	3758.555	20475758
Hel1	0.213	1.000	24.682	1437.772	6026460	0.796	4.300	2.923	1419.663	25879430
Ta071	0.118	0.600	20.911	16602.528	6026460	0.902	5.500	0.693	16342.738	29083466
Ta075	0.109	0.700	18.269	17009.129	6026460	0.873	6.200	0.484	16814.526	23350351
Ta081	0.125	0.500	20.528	62180.878	6026460	0.955	6.400	0.784	61269.480	40545679
Ta085	0.143	0.700	19.700	59635.511	6026460	0.935	7.200	0.414	58806.785	40891193
Ta091	0.156	1.000	17.354	32946.528	9626460	0.947	7.100	1.356	32470.454	44078276
Ta095	0.000	0.000	22.980	33211.048	9626460	1.000	7.200	0.000	32627.303	39976360
Ta101	0.090	0.700	22.972	115092.964	9626460	0.973	7.100	1.906	113810.132	62357740
Ta105	0.094	0.500	24.442	115944.076	9626460	0.958	6.800	0.895	114933.432	67352772
Ta111	0.000	0.000	39.002	268005.032	30024060	1.000	7.300	0.000	265567.733	189251340
Ta115	0.000	0.000	35.874	270647.158	30024060	1.000	8.400	0.000	266637.278	189301380
Average	0.310	1.139	17.094	41509.527	5694103	0.884	4.907	2.110	41019.194	30373516

MADE_nospeedup. Simulation results on $f = (C_{\max}, T_{\max})$ and $f = (I_{sum}, N_T)$ are shown in Tables 12 and 13, respectively.

From Table 12, it can be seen that the average *TET* of MADE is 4.2 times larger than that of MADE_nospeedup, which testifies that the evaluation time of solution can be significantly reduced by the speed-up computing method. Furthermore, all performance metrics values of MADE are obviously better than those of MADE_nospeedup. This means that, under the same running time, MADE can spend more time in executing search operation in the solution space, which is helpful to obtain the *optimal Pareto front*. Similar conclusion can be drawn from Table 13.

5.4.2 Comparisons with the same running generation

Secondly, we let MADE and MADE_nospeedup run the same generation (i.e., *t_max*=300). Test results on $f = (C_{\max}, T_{\max})$ and $f = (I_{sum}, N_T)$ are illustrated in Tables 14 and 15 respectively, where *Tavg* denotes the average CPU time (second) of ten runs.

It is shown from Table 14 that the differences between performance metrics values of MADE and those of MADE_nospeedup are quite minor. That is to say, when running same generation, MADE and MADE_nospeedup can obtain similar non-dominated solutions. However, especially for the large problems, the *Tavg* of MADE is obviously

Table 14 Comparisons of MADE with MADE_nospeedup when considering $f = (C_{\max}, T_{\max})$ (same running generation)

Problem	MADE_nospeedup					MADE				
	RNDS	*ONSN*	DI_R	*AQ*	*Tavg*	*RNDS*	*ONSN*	DI_R	*AQ*	*Tavg*
Car1	0.972	10.400	1.243	4177.153	2.423	1.000	10.900	0.308	4168.629	1.586
Car5	0.979	9.200	1.107	4766.167	1.930	0.989	9.200	1.605	4786.309	1.275
Car8	1.000	6.800	2.005	4764.539	1.494	1.000	6.900	1.059	4763.264	0.936
Rec01	0.634	8.500	3.055	781.260	4.561	0.629	9.000	2.779	783.343	2.773
Rec05	0.602	8.000	2.744	776.329	5.253	0.606	7.700	3.718	777.120	3.431
Rec09	0.645	8.900	2.974	1060.886	5.805	0.624	8.300	2.876	1061.066	2.922
Rec11	0.933	5.600	5.827	966.583	6.659	0.788	5.200	5.337	966.331	2.803
Rec15	0.573	9.800	2.027	1287.307	6.627	0.614	10.200	2.153	1286.758	2.704
Rec19	0.648	6.800	4.428	1520.098	10.300	0.524	6.500	3.905	1524.042	4.139
Rec21	0.544	6.200	4.770	1484.482	10.585	0.560	7.000	4.892	1485.308	4.453
Rec25	0.613	6.500	4.362	1902.208	12.775	0.557	5.400	5.176	1906.549	4.409
Rec29	0.545	6.100	4.851	1769.807	12.951	0.570	6.900	4.021	1766.794	4.481
Rec31	0.614	7.000	3.511	2346.677	25.820	0.481	6.300	4.295	2338.303	9.719
Rec35	0.626	7.700	3.522	2418.355	26.020	0.526	7.000	3.217	2414.257	10.059
Rec39	0.512	6.300	4.175	4645.296	80.105	0.576	7.600	5.106	4649.754	17.308
Ta061	0.464	5.200	5.776	3533.974	94.277	0.648	7.000	4.913	3531.098	52.614
Ta065	0.521	6.200	5.993	3395.795	93.497	0.635	8.000	3.750	3381.977	52.511
Hel1	0.793	6.900	4.921	389.757	137.488	0.387	4.100	8.140	391.299	52.589
Ta071	0.652	7.500	5.035	4455.661	132.941	0.475	5.800	5.292	4466.872	52.614
Ta075	0.569	6.600	6.409	4446.235	132.933	0.543	6.300	5.075	4426.617	52.760
Ta081	0.433	5.200	6.084	5882.187	199.245	0.679	8.900	3.690	5868.730	54.208
Ta085	0.535	6.100	5.386	5809.396	201.292	0.577	7.100	4.779	5813.518	57.378
Ta091	0.565	7.000	4.319	8730.569	438.911	0.483	6.900	4.245	8739.799	119.537
Ta095	0.652	7.500	4.821	8705.978	437.271	0.523	5.800	6.589	8736.021	119.490
Ta101	0.521	5.000	8.143	11375.480	658.766	0.639	7.600	3.184	11254.455	128.914
Ta105	0.505	5.300	7.768	11446.594	649.971	0.562	6.800	6.091	11419.405	122.112
Ta111	0.644	8.700	7.521	28718.063	5195.164	0.525	6.400	10.709	28962.025	1026.930
Ta115	0.551	7.600	4.825	28418.446	5209.621	0.607	8.200	3.243	28242.137	1057.640
Average	0.637	7.093	4.557	5713.403	492.667	0.619	7.250	4.291	5711.135	107.939

smaller than that of MADE_nospeedup. The average *Tavg* of MADE is 4.6 times smaller than that of MADE_nospeedup. This is very meaningful for real-time applications and dynamic scheduling, in which achieving satisfied solutions within a short time is critical. Similar conclusion also can be drawn from Table 15.

In conclusion, MADE is the most effective and efficient multi-objective optimization algorithm among the four compared algorithms.

6 Conclusions

In this article, a memetic algorithm based on DE (MADE) is presented for solving multi-objective no-waiting flow-shop scheduling problems (MNFSSPs). In order to apply DE for FSSPs problems, we proposed a LOV rule to map the continuous values of an individual into a job permutation of FSSPs. We also adopted the concept of *Pareto dominance* to handle the updating of solutions in multi-objective sense. In our proposed algorithm, not only did DE-based wide scatter search be utilized to find enough promising regions, but also several problem-specific memes were designed to perform a thorough and deep search in these promising regions. Moreover, a speed-up computing method was developed to reduce the computing complexity of solution evaluation, that is, we considered the improvements of both the effectiveness of searching solutions and the efficiency of evaluating solutions. Simulation results and comparisons based on a set of benchmarks demonstrated the effectiveness and efficiency of MADE.

Table 15 Comparisons of MADE with MADE_nospeedup when considering $f = (I_{\text{sum}}, N_T)$ (same running generation)

Problem	MADE_nospeedup					MADE				
	RNDS	*ONSN*	DI_R	*AQ*	*Tavg*	*RNDS*	*ONSN*	DI_R	*AQ*	*Tavg*
Car1	0.960	2.400	7.185	6266.589	2.291	0.958	2.300	11.850	6266.591	1.470
Car5	0.935	2.900	10.472	10949.779	2.191	0.917	3.300	1.800	10937.960	1.498
Car8	1.000	2.000	0.000	16890.764	1.430	1.000	2.000	0.000	16890.764	0.884
Rec01	0.800	2.800	7.809	1011.980	4.442	0.667	2.600	7.243	1005.925	2.592
Rec05	0.548	2.300	4.843	1096.169	5.500	0.675	2.700	5.118	1097.177	3.639
Rec09	0.643	2.700	7.992	4671.443	5.658	0.683	2.800	6.155	4671.642	2.625
Rec11	0.639	2.300	11.080	4098.513	6.620	0.561	2.300	5.100	4113.106	2.511
Rec15	0.606	2.000	6.117	9036.557	6.673	0.933	2.800	8.205	9036.561	2.525
Rec19	0.538	2.100	9.718	6194.875	10.233	0.516	1.600	7.939	6203.557	3.678
Rec21	0.366	1.500	10.803	6151.922	10.239	0.632	2.400	7.518	6122.275	3.822
Rec25	0.614	2.700	7.330	13695.844	12.958	0.455	2.000	8.641	13733.615	3.981
Rec29	0.558	2.400	7.006	12627.610	13.077	0.500	2.100	9.687	12638.520	4.028
Rec31	0.500	2.300	11.894	9267.815	25.277	0.630	3.400	8.807	9254.565	7.748
Rec35	0.327	1.800	9.283	9775.352	25.221	0.744	3.200	7.308	9797.619	7.665
Rec39	0.554	3.100	13.729	50041.328	81.525	0.564	3.100	8.614	50009.214	13.617
Ta061	0.589	3.300	11.598	4072.539	89.651	0.600	3.300	7.624	4026.334	29.163
Ta065	0.611	3.300	7.880	3787.457	88.714	0.540	2.700	10.576	3826.838	29.123
Hel1	0.688	3.300	7.388	1432.918	133.989	0.403	2.500	12.623	1438.734	29.052
Ta071	0.633	3.800	7.489	16539.856	129.081	0.421	2.400	11.177	16537.319	29.192
Ta075	0.571	3.200	9.345	16903.369	128.408	0.490	2.500	11.180	16986.848	29.494
Ta081	0.333	2.200	11.707	62325.617	203.800	0.786	4.400	5.873	61905.444	35.497
Ta085	0.509	2.800	7.525	59573.442	211.613	0.541	3.300	8.696	59500.372	40.523
Ta091	0.640	3.200	10.347	32932.648	433.299	0.525	3.100	16.526	33085.582	93.883
Ta095	0.672	4.300	6.338	33023.507	444.964	0.408	2.900	15.918	32944.996	106.859
Ta101	0.433	2.900	7.993	114888.203	664.591	0.644	4.700	5.738	115302.791	102.898
Ta105	0.667	3.800	12.361	116246.703	653.510	0.492	3.200	7.842	115722.781	93.365
Ta111	0.451	3.200	13.215	268461.025	4942.188	0.346	2.700	10.363	268620.583	661.523
Ta115	0.539	4.100	3.438	271268.743	4961.109	0.444	3.200	10.066	270881.992	691.407
Average	0.604	2.811	8.639	41544.020	474.938	0.610	2.839	8.507	41519.989	72.652

To the best of our knowledge, this is the first paper to apply the standard DE for multi-objective no-wait flow-shop scheduling problems. In our future research, we will propose some adaptive strategies to improve the efficiency of MADE, and extend MADE to solve other kinds of scheduling problems, such as stochastic scheduling.

Acknowledgments This research is partially supported by National Science Foundation of China (60774082, 60574072), National 863 Hi-Tech R&D Plan (2007AA04Z155, 2007AA04Z193) and the Project-sponsored by SRF for ROCS, SEM. The authors would like to acknowledge the guest editors and three anonymous referees for their helpful comments and suggestions on the earlier manuscript of this paper.

Appendix 1 : The algorithm to calculate $MD(\pi_{j-1}, \pi_j)$

Step 1: Set $p_1 = 0$, $p_2 = 0$ and $k = 2$;
Step 2: Do
$p_1 = p_1 + p(\pi_{j-1}, k)$;
$p_2 = p_2 + p(\pi_j, k-1)$;
If $k = 2$ then
$max\_p = p_1 - p_2$
Else
$max\_p = \max\{max\_p, p_1 - p_2\}$;
$k = k + 1$;
While $k \leq m$;
Step 3: $MD(\pi_{j-1}, \pi_j) = p(\pi_{j-1}, 1) + \max\{0, max\_p\}$;

Appendix 2 : The algorithm to calculate $\sum_{k=1}^{m} \sum_{y=1}^{k} p(\pi_n, y)$

Step 1: Set $p_{sum} = 0$, $p = 0$ and $k = 1$;
Step 2: Do

$p = p + p(\pi_n, k)$; // $p = \sum_{y=1}^{k} p(\pi_n, y)$

$p_{sum} = p_{sum} + p$;

$k = k + 1$;
While $k \le m$;

Remark In Step 2, p is used to save the current value of $\sum_{y=1}^{k} p(\pi_n, y)$ in each loop. If the equation $p = p + p(\pi_n, k)$ is replaced with the equation $p = \sum_{y=1}^{k} p(\pi_n, y)$, the CC of $\sum_{k=1}^{m} \sum_{y=1}^{k} p(\pi_n, y)$ will rise from $O(m)$ to $O(m^2)$. For the purpose of reducing the computing complexity, the equation $p = p + p(\pi_n, k)$ is adopted in Step 2.

Appendix 3 : The algorithm to calculate $C(\pi_j, m)$ $(\pi_j \in \{1, \ldots, n\})$

Step 1: Calculate $C(\pi_1, m) = \sum_{y=1}^{m} p(\pi_1, y)$ and set $MD\_sum = 0$;
Step 2: Set $j = 2$ and $p = 0$;
Do
$MD\_sum = MD\_sum + MD(\pi_{j-1}, \pi_j)$;
$C(\pi_j, m) = MD\_sum + P_{sum}(\pi_j)$;
$j = j + 1$;
While $j \le n$;

References

Aldowaisan T, Allahverdi A (2003) New heuristics for no-wait flowshops to minimize makespan. Comput Oper Res 30:1219–1231

Allahverdi A, Aldowaisan T (2004) No-wait flowshops with bicriteria of makespan and maximum lateness. Eur J Oper Res 152:132–147

Arroyo JEC, Armentano VA (2005) Genetic local search for multi-objective flowshop scheduling problems. Eur J Oper Res 167: 717–738

Baker KR (1974) Introduction to sequencing and scheduling. Wiley, New York

Bean JC (1994) Genetic algorithm and random keys for sequencing and optimization. ORSA J Comput 6(2):154–160

Bonney MC, Gundry SW (1976) Solutions to the constrained flowshop sequencing problem. Oper Res Quart 24:869–883

Caponio A, Cascella GL, Neri F, Salvatore N, Sumner M (2007) A fast adaptive memetic algorithm for online and offline control design of PMSM drives. IEEE T Syst Man Cybern B 37(1):28–41

Carlier J (1978) Ordonnancements a contraintes disjonctives. R.A.I.R.O. Recherche operationelle/Oper Res 12:333–351

Chang YP, Wu CJ (2005) Optimal multiobjective planning of large-scale passive harmonic filters using hybrid differential evolution method considering parameter and loading uncertainty. IEEE T Power Deliver 20(1):408–416

Chen CL, Neppalli RV, Aljaber N (1996) Genetic algorithms applied to the continuous flow-shop problem. Comput Ind Eng 30:919–929

Czyzak P, Jaszkiewicz A (1998) Pareto-simulated annealing—a metaheuristic technique for multi-objective combinatorial optimization. J Multi-Crit Decis Anal 7(1):34–47

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE T Evol Comput 6(2):182–197

Dimopoulos C, Zalzala AMS (2000) Recent development in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons. IEEE T Evolut Comput 4:93–113

Feoktistov V (2006) Differential evolution: in search of solutions. Springer, Heidelberg

Gangadharan R, Rajendran C (1993) Heuristic algorithms for scheduling in the no-wait flowshop. Int J Prod Econ 32(3):285–290

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of np-completeness. Freeman, San Francisco

Geiger MJ (2007) On operators and search space topology in multi-objective flow-shop scheduling. Eur J Oper Res 181(1):195–206

Grabowski J, Pempera J (2005) Some local search algorithms for no-wait flow-shop problem with makespan criterion. Comput Oper Res 32:2197–2212

Hart WE, Krasnogor N, Smith JE (2004) Recent advances in memetic algorithms. Springer, Heidelberg

Hall NG, Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. Oper Res 44:510–525

Heller J (1960) Some numerical experiments for an M×J flow-shop and its decision-theoretical aspects. Oper Res 8:178–184

Ilonen J, Kamarainen JK, Lampinen J (2003) Differential evolution training algorithm for feed-forward neural networks. Neural Process Lett 17(1):93–105

Ishibuchi H, Murata T (1998) A multiobjective genetic local search algorithm and its application to flowshop scheduling. IEEE T Syst Man Cybern C 28(3):392–403

Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE T Evol Comput 7: 204–223

Jaszkiewicz A (2002) Genetic local search for multi-objective combinatorial optimization. Eur J Oper Res 137(1):50–71

Jaszkiewicz A (2003) Do multiple-objective metaheuristcs deliver on their promises? A computational experiment on the set-covering problem. IEEE T Evolut Comput 7(2):133–143

Jonathan EF, Richard ME, Sameer S (2003) Using unconstrained elite archives for multiobjective optimization. IEEE T Evolut Comput 7(3):305–323

King JR, Spachis AS (1980) Heuristics for flowshop scheduling. Int J Prod Res 18:343–357

Knowles JD, Corne DW (2002) On metrics for comparing nondominated sets. In: 2002 Congress on Evolutionary Computation, Honolulu, HI, USA, pp 711–716

Kumar S, Bagchi TP, Sriskandarajah C (2000) Lot streaming and scheduling heuristics for m-machine no-wait flowshops. Comput Ind Eng 38:149–172

Liu B, Wang L, Jin YH (2007) An effective hybrid particle swarm optimization for no-wait flow-shop scheduling. Int J Adv Manuf Tech 31(9–10):1001–1011

Murata T, Ishibuchi H, Tanaka H (1996) Genetic algorithms for flowshop scheduling problems. Comput Ind Eng 30:1061–1071

Nearchou AC (2008) A differential evolution approach for the common due date early/tardy job scheduling problem. Comput Oper Res 35:1329–1343

Mladenovic N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100

Nearchou AC, Omirou SL (2006) Differential evolution for sequencing and scheduling optimization. J Heuristics 12(6):395–411

Neri F, Toivanen J, Cascella GL, Ong YS (2007) An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE ACM T Comput BI 4(2):264–278

Nowicki E, Smutnicki C (2006) Some aspects of scatter search in the flow-shop problem. Eur J Oper Res 169:654–666

Ong YS, Keane AJ (2004) Meta-Lamarckian learning in memetic algorithms. IEEE T Evol Comput 8:99–110

Ong YS, Lim MH, Zhu N, Wong KW (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE T Syst Man Cy B 36(1):141–152

Onwubolu G, Davendra D (2006) Scheduling flow-shops using differential evolution algorithm. Eur J Oper Res 171(2):674–692

Pinedo M (2002) Scheduling: theory, algorithms and systems, 2nd edn. Prentice-Hall, NJ

Price K, Storn R (2007) Differential evolution (DE) for continuous function optimization. http://www.icsi.berkeley.edu/%7Estorn/code.html. Accessed 13 July 2007

Price K, Storn R, Lampinen J (2005) Differential evolution: a practical approach to global optimization. Springer, Berlin, pp 227–238

Qian B, Wang L, Huang DX, Wang X (2008) Scheduling multi-objective job shops using a memetic algorithm based on differential evolution. Int J Adv Manuf Technol 35:1014–1027

Qian B, Wang L, Huang DX, Wang WL, Wang X (2009) An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers. Comput Oper Res 36(1):209–233

Rajendran C (1994) A no-wait flowshop scheduling heuristic to minimize makespan. J Oper Res Soc 45(4):472–478

Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22:5–13

Reeves CR (1999) Landscapes, operations and heuristic search. Ann Oper Res 86:473–490

Reeves CR, Yamada T (1998) Genetic algorithms, path relinking and the flowshop sequencing problem. Evol Comput 6:45–60

Schiavinotto T, Stützle T (2007) A review of metrics on permutations for search landscape analysis. Comput Oper Res 34(10): 3143–3153

Stadtler H (2005) Supply chain management and advanced planning-basics, overview and challenges. Eur J Oper Res 163:575–588

Storn R, Price K (1997) Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359

Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G (2004) Differential evolution algorithm for permutation flowshop sequencing problem with makespan criterion. In: Proceedings of 4th international symposium on intelligent manufacturing systems, Sakarya, Turkey, pp 442–452

Tang J, Lim MH, Ong YS (2007) Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems. Soft Comput 11(9):873–888

Tavakkoli-Moghaddam R, Rahimi-Vahed A, Hossein Mirzaei A (2007) A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives: weighed mean completion time and weighted mean tardiness. Inf Sci. doi:10.1016/j.ins.2007.06.001

Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64: 278–285

Van Deman JM, Baker KR (1974) Minimizing mean flow time in the flowshop with no intermediate queues. AIIE Trans 6:28–34

Wang L (2003) Shop scheduling with genetic algorithms. Tsinghua Univ. Press & Springer, Beijing

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE T Evol Comput 1:67–82

Zhou Z, Ong YS, Lim MH, Lee BS (2007) Memetic algorithm using multi-surrogates for computationally expensive optimization problems. Soft Comput 11(10):957–971

Zhu Z, Ong YS, Dash M (2007) Wrapper-filter feature selection algorithm using a memetic framework. IEEE T Syst Man Cybern B 37(1):70–76

Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE T Evolut Comput 3(4):257–271

Zitzler E, Deb K, Thiele L (2000) Comparison of multiobjective evolutionary algorithms: Empirical results. IEEE T Evol Comput 8(2):173–195