Arab J Sci Eng (2013) 38:2233–2244
DOI 10.1007/s13369-013-0611-4

RESEARCH ARTICLE - SYSTEMS ENGINEERING

Scheduling a Bi-Objective Hybrid Flow Shop with Sequence-Dependent Family Setup Times Using Metaheuristics

M. Fadaei · M. Zandieh

Received: 13 January 2011 / Accepted: 6 October 2011 / Published online: 4 April 2013

Abstract This paper deals with the problem of hybrid flow shop scheduling. In this investigation, we considered group scheduling within the area of sequence-dependent family setup times and two objectives of minimizing makespan and total tardiness are taken into consideration simultaneously. Due to the computational complexity in solving these set of problems with multiple objectives, metaheuristics has a high priority, because these algorithms are capable of solving combinatorial problems in a reasonable time. This study focuses on three multi-objective algorithms, multi-objective genetic algorithm, sub-population genetic algorithm-II and non-dominated sorting genetic algorithm-II, to solve the mentioned problem. In order to investigate the effectiveness and efficiency of applying the noted metaheuristics for such an NP-hard problem, we evaluate non-dominated solution sets obtained via each algorithm through some evaluation metrics.

Keywords Multi-objective optimization · Hybrid flow shop · Multi-objective evolutionary algorithms · Sequence-dependent family setup times · Group scheduling · Makespan · Total tardiness

الخلاصة

تتناول هذه الورقة العلمية مشكلة جدولة تدفق المحل الهجين. وفي هذا التحقيق أخذنا في الاعتبار--وفي وقت واحد جدولة المجموعة داخل منطقة أوقات إعداد الأسرة التي تعتمد على التسلسل وتحقيق هدفين بتقليل وقت التجهيز والتأخر الكلي إلى أدنى حد. وبسبب التعقيد الحسابي في حل هذه المشاكل ذات المجموعة من الأهداف المتعددة، فإن لخوارزمية metaheuristics أولوية عالية، وذلك لأن هذه الخوارزميات هي قادرة على حل المشاكل الاندماجية في فترة زمنية معقولة.

تركز هذه الدراسة على ثلاث خوارزميات متعددة الهدف، الأولى: الخوارزمية الجينية (MOGA)، والثانية: خوارزمية II- شبه السكان الجينية (II-SPGA)، والثالثة: خوارزمية الفرز الجينية--II (II NSGA) غير المهيمنة لحل المشكلة المذكورة. وقد قمنا --من أجل بحث فعالية وكفاءة تطبيق الخوارزمية المشار إليها لمشكلة NP الصعبة هذه بتقييم الحل غير المهيمن وحل المجموعات التي تم الحصول عليها عن طريق كل خوارزمية من خلال بعض مقاييس التقييم.

1 Introduction

There is a variety of manufacturing environment species in industries, and "hybrid flow shop" (HFS) is a type of them. There has been lots of research articles on HFS scheduling problems, but most of them employed only one objective. It is apparent that the majority of real-world industrial systems are expected to optimize more than one objective. Hence, it is explicitly needed to consider more objectives simultaneously in solving scheduling problems.

The HFS scheduling problem is a broadened form of the conventional flow shop model. Flow shops are formed by a set of n jobs which must be processed on m successive stages, and all the jobs should be processed through all the stages in the same order. An HFS comprised of more than one stage, in which, at least one stage has got two or more parallel machines. Reduction in completion time is the main reason for installing multiple machines in one/some stage(s).

M. Fadaei
Department of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran

M. Zandieh (✉)
Department of Industrial Management, Management and Accounting Faculty, Shahid Beheshti University, G. C., Tehran, Iran
e-mail: m_zandieh@sbu.ac.ir

Most of the researches on HFS scheduling problem has done on two-stage HFS [1–4] and there have been few papers containing more than two stages. Based on the complexity of HFS scheduling problems, the current algorithms for solving HFS can be divided into two categories: exact methods and heuristic algorithms. Branch and bound approach is a kind of exact methods which has widely employed to optimize HFS scheduling problems [5–7], but because of huge amount of computational applications, this method cannot be applicable to a wide variety of real-world systems, so practitioners have tried to extend heuristic algorithms.

Botta-Genoulaz [8] proposed six new heuristics to schedule a k-stage HFS with the objective of minimizing maximum lateness. An artificial immune system was introduced as a new approach to solve HFS scheduling problem by Engin and Doyen [9], they considered makespan minimization as the objective of their research. Janiak et al. [10] studied an HFS scheduling problem considering criterion of three parts: the total weighted earliness, the total weighted tardiness and the total weighted waiting time. They proposed three constructive algorithms and developed three metaheuristics to solve the problem. Gupta et al. [11] proposed a heuristic constructive algorithm for an HFS scheduling problem considering a criteria based on due dates. Allaoui and Artiba [12] dealt with five objectives separately for an HFS. Jungwattanakit et al. [13] formulated an integer program and proposed some constructive heuristics for a bi-objective flexible flow shop scheduling problem with unrelated parallel machines. The study considered positively weighted convex sum of makespan and the number of tardy jobs as two different objectives.

One way to increase the productivity of manufacturing systems is to use of group technology and 'cellular manufacturing'. By the advent of manufacturing cell scheduling problems (MCSPs), there have done some studies on MCSPs [14]. In cellular manufacturing, jobs (parts) and machines are grouped based on their similarities, for instance, jobs which need to be processed on the same machine can be gathered in one group, it causes to have a 'part family' which included jobs with the same setup times. Therefore, the machine that is going to process these jobs needs to be prepared once and then it can process all the jobs of this family, so it results in decreasing of setup time as well as completion time. To simplify scheduling problems, setup times have been rarely considered, while the most real-world manufacturing environments have separated setup time and processing time. Ham et al. [15] proposed a polynomial-time algorithm to solve the problem of two-machine group scheduling with sequence-independent setups, with the objective of minimizing makespan.

Gupta and Tunc [2] proposed four new heuristic algorithms to minimize completion time for an HFS. They considered setup times and removal times separated from processing times.

Kurz and Askin [16] used random keys genetic algorithm (RKGA) to minimize makespan for a sequence-dependent setup time flexible flowline. Zandieh et al. [17] proposed an immune algorithm to solve the sequence-dependent setup time HFS, which resulted to a more efficiency in comparing with RKGA proposed by Kurz and Askin [16].

There are two stages in custom group scheduling problems. At first, the sequence of jobs belonging to each family must be identified and then the sequence of families must be determined, which commonly known as major setups. Since, in grouping the jobs, resembling between jobs must be considered, so the required setup time to switch from one job to another one, of the same family, for processing on a machine is very low, and can be ignored. On the other hand, because of the dissimilarity of jobs in different families, required setup time for changing families on a stage is noticeable and should be accounted separately from processing time. In this study, we consider an HFS with SDFST which results in a set of more complicated problems. Even considering two machines for a flow-line with SDFSTs causes to encounter with NP-hard problems [18]. Genetic algorithms [19], tabu searches [20], and hybrid metaheuristics [21] are some metaheuristics which are used for solving MCSPs with SDFSTs.

Considering multiple conflicting objectives causes to encounter with more complicated problems and due to the difficulties there are few studies in literature for this class of problems. To schedule multi-objective problems we deal with the concept of Pareto-set. Among the techniques of generating Pareto-sets, evolutionary algorithms are more efficient because of using a population of solutions [22]. Because of the complexity of these class of optimization problems, multi-objective evolutionary algorithms (MOEAs) are developed to solve them. The first implementation of MOEAs was vector evaluation genetic algorithm (VEGA), which was extended by Schaffer [23]. After that researchers developed many MOEAs such as strength Pareto evolutionary algorithm (SPEA), sub-population genetic algorithm (SPGA), proposed non-dominated sorting genetic algorithm (NSGA), NSGA-II and so on. Murata et al. [24] introduced multi-objective genetic algorithm (MOGA) assigning variable weights to objective functions, they employed MOGA to the flow shop scheduling problem. Zitzler et al. [25,26] introduced SPEA and SPEA-II. Chang et al. [27] proposed two-phase SPGA for a parallel machine scheduling problem. Recently, Chang and Chen [28] developed a sub-population genetic algorithm-II (SPGA-II) for multi-objective combinatorial problems. Deb et al. [29] proposed non-dominated sorting genetic algorithm II (NSGA-II).

As reviewed above, there has been almost no study on HFS considering SDFST and minimizing makespan and total tardiness simultaneously.

In this paper, we consider a HFS scheduling problem with sequence-dependent family setup time and considering two

objectives: makespan and total tardiness. We adapted three MOEAs to solve this problem.

The remainder of the paper is structured as follows: Sect. 2 gives the problem description. Section 3 introduces three adapted MOEAs for this study and describes the characteristics of these evolutionary algorithms (EAs). Parameters tuning, evaluation metrics and the experimental results are explained in Sect. 4. Finally, Sect. 5 presents the conclusions of the paper and further research directions.

2 Problem Definition and Notations

This study is fond of scheduling the problem of HFS. The HFS scheduling problem with the following assumptions is considered in this study:

- All n jobs are available at the beginning of scheduling.
- Preemption is not allowed.
- There is no breakdown for machines and all the machines are always available.
- Parallel machines in each stage are identical.
- Infinite buffers are installed between stages.
- Traveling time between stages is zero.
- Each job can be processed on one machine at a time.
- A machine cannot process more than one job at the same time.
- Families (groups) have setup times which are sequence-dependent.

It should be noted that in this study we have just family setups and there is no separate setup time for jobs. Since the part setups are negligible so it is assumed that setup time of each job has been considered with the processing time of that job.

We considered the following definitions too:

- n_j = total number of jobs;
- n_g = number of groups;
- n_s = number of stages;
- C_j = the completion time of jth job.
- d_j = due date for job j;
- pr_j^s = processing time for job j at stage s;
- t_{ij}^s = sequence-dependent setup time for switching from group i to group j at stage s;
- m^s = the vector contains number of machines for stage s.

2.1 Two *Objectives* Employed in this Study

In this study, we investigate minimizing two objectives: makespan and total tardiness, which can be briefly described as follows:

Makespan ($C_{\max}$): the required time to fulfill processing all jobs;

Total tardiness (TT): the sum of tardiness of all jobs $\left(\sum_{j=1}^{n} T_j\right)$

where T_j is equal to $\max\{C_j - d_j, 0\}$ and d_j is the due date of jth job.

To prevail over trap of dealing with objective values of different units, we normalize the value of each objective function by the following equations:

To prevail over the trap of dealing with different measurement sizes of objective values, we normalize the value of each objective function by the following equations:

$$f_1(x) = \frac{C_{\max}(x) - \text{best}(C_{\max})}{\text{worst}(C_{\max}) - \text{best}(C_{\max})} \quad \text{and}$$

$$f_2(x) = \frac{\text{TT}(x) - \text{best(TT)}}{\text{worst(TT)} - \text{best(TT)}}$$

where for solution x, $C_{\max}(x)$ and $\text{TT}(x)$ denote the value of two objectives: makespan and total tardiness, respectively, $\text{best}(C_{\max})$ and $\text{worst}(C_{\max})$ indicate the best and worst obtained value for objective function $C_{\max}$, best(TT) and worst(TT) indicate the best and worst obtained value for total tardiness, and $f_i(x)$ is the normalized value of ith objective function for solution x.

2.2 Pareto-Optimal Solutions

Single objective optimization acquires one single solution as final optimum solution. Whereas multi-objective optimization is accustomed to deal with a set of best solutions, which none of these solutions are dominated by another one. This set of non-dominated solutions, which commonly known as Pareto-optimal solutions, is proposed by Tamaki et al. [30].

If x is a solution for a multi-objective problem and there is no solution y which dominates x, thus x is a Pareto-optimal solution. In order to clarify that Pareto-optimal solutions are non-dominated, assume that S is a Pareto-optimal set for a bi-objective minimization problem and $x \in S$. In addition y is a feasible solution for this problem, and not a member of Pareto-optimal set, so solution x dominates solution y.

Totally, it can be said that solution x dominates solution y if:

$$f_i(x) \leq f_i(y), \forall i \in \{1, 2\} \quad \text{and} \quad f_i(x) < f_i(y), \exists i \in \{1, 2\}$$

3 Multi-Objective Evolutionary Algorithms

Hybrid flow shop cannot be solved with exact algorithms in a reasonable time because it is an NP-hard problem. Furthermore with considering multiple objectives, the problem becomes more complicated. As discussed above, employing

MOEAs is effective for solving this class of problems. In the following, we describe three MOEAs which are employed in this study.

3.1 Multi-Objective Genetic Algorithm

Murata et al. [24] proposed the MOGA with the aim of searching Pareto-optimal solutions for bi-objective and tri-objective scheduling problem. Thereafter, some developments have been implemented on this new multi-objective algorithm.

3.1.1 Weighting Method

Murata et al. [24] calculated sum of weighted objective values for each solution and used it for evaluating that solution. In this way, they changed a multiple objective problem into a single objective problem as follows:

$$f(x) = \sum_{i=1}^{n} w_i f_i(x)$$

where $f_i(x)$ is the value of ith objective function for solution x, and w_i is the weight assigned to ith objective function. A magnificent characteristic of MOGA is using of variable weights which are assigned to different objectives. These weights should satisfy the following relationships:

$$w_i \geq 0, \quad i = 1, \ldots, n$$
$$\sum_{i=1}^{n} w_i = 1$$

Using variable weights leads to switch searching area into different directions. There are several weighting approaches. For instance, 'constant weight' method is a kind of weighting methods, in which each objective is assigned a constant weight value, but the shortcoming of this method is that these constant weights must be determined at first. One way for weighting is using of random numbers. For instance, we can use the following equation to make weights:

$$w_i = \frac{u_i}{\sum_{i=1}^{n} u_i}, \quad i = 1, 2, \ldots, n$$

u_i is a random number in the interval (0, 1), and w_i indicates the weight corresponding to ith objective. This method is applied in this survey.

3.1.2 Solution Representation

Bean [31] proposed RKGA. The major difference between RKGA and conventional genetic algorithm is in the solution representation. Solution representation in RKGA is based on an encoding method employing random numbers. Norman and Bean [32] suggested a new solution representation for an identical multiple-machine problem. Their offered solution representation is described in the following. We employ this approach to assign and determine the sequence of groups and jobs assigned to each group at the first stage. For each group we generate a random number in the interval $(1, m^1 + 1)$ where the value of m^1 is the number of machines for the first stage. The integer part of created real number shows the machine number to which the group is assigned and the fractional part is used for sorting groups assigned to each machine. For identifying the sequence of jobs related to each group, we create a random number in the interval (0, 1) for each job, the sequence of processing jobs is based on increasing the created numbers. In other words, the job related to the smallest number is processed first and so on.

	Machine	J_1	J_2	J_3	J_4
G_1	2.28	0.28	0.76	0.18	-
G_2	1.84	0.86	0.19	-	-
G_3	1.35	0.45	0.86	0.34	0.09
G_4	3.19	0.72	0.39	0.12	-
G_5	2.48	0.67	0.94	0.14	-

Fig. 1 Solution representation

Assume there are five groups and number of jobs of group 1 to group 5 are 3, 2, 4, 3 and 3, respectively, and it is supposed that we have three identical machines at the first stage. An example of encoding above problem can be seen in Fig. 1.

According to the solution representation in Fig. 1, the sequence of jobs and groups on machines of first stage is concluded as follows:

Machine 1: G_3 (J_4, J_3, J_1, J_2), G_2 (J_2, J_1)
Machine 2: G_1(J_3, J_1, J_2), G_5(J_3, J_1, J_2)
Machine 3: G_4(J_3, J_2, J_1).

3.1.3 The Step-by-Step MOGA

In the following, explanation of MOGA employed in this study is given:

3.1.3.1. *Initialization*. First, we determine the value of algorithm parameters such as: population size (N), selection strategy, rate of crossover (r_c), rate of mutation (r_m) and stopping criteria. Then a random population of N chromosome is generated. To represent solutions, we employ random number approach described in Sect. 3.1.2.

3.1.3.2. *Evaluation.* Thereafter the value of each objective function is calculated for each solution. As stated in Sect. 2.1, to avoid trapping of different units for different objectives, the normalized objective functions are applied. As it was explained, we consider the aggregate of weighted objective functions for evaluation. We employ random numbers for weighting as described in Sect. 3.1.1. Afterwards we can evaluate solution x as follows:

3.11	0.48	0.71	0.24	-
2.14	0.36	0.28	-	-
2.35	0.92	0.34	0.82	0.34
1.94	0.85	0.84	0.27	-
1.25	0.31	0.47	0.81	-

Parent 1

2.18	0.18	0.76	0.31	-
3.52	0.34	0.19	-	-
1.96	0.89	0.64	0.74	0.85
1.12	0.75	0.85	0.39	-
2.54	0.29	0.34	0.28	-

Parent 2

0.18	0.69	0.27	0.93	-
0.96	0.73	0.26	-	-
0.36	0.29	0.84	0.64	0.39
0.75	0.39	0.59	0.47	-
0.45	0.85	0.43	0.71	-

Random numbers

3.11	0.48	0.71	0.31	-
3.52	0.34	0.28	-	-
2.35	0.92	0.64	0.82	0.34
1.12	0.85	0.84	0.27	-
1.25	0.29	0.47	0.28	-

offspring

Fig. 2 Crossover example

$$f(x) = w_1 \cdot f_1(x) + w_2 \cdot f_2(x)$$

where $f(x)$ is the fitness function for solution x, $f_1(x)$ and $f_2(x)$ are normalized makespan and normalized total tardiness, respectively.

3.1.3.3. *Elitism.* We apply elitism for reproduction. In this way, the solutions with best fitness function values are copied to the next generation.

3.1.3.4. *Selection strategy.* we use roulette wheel selection proposed by Goldberg [33] as selection strategy for MOGA employed in this study.

3.1.3.5. *Crossover scheme.* After selecting two parents by the use of selection strategy, crossover operator makes new offsprings by exchanging corresponding parts of parents. We apply crossover with the probability of r_c. we set the chromosome with better fitness function value as parent 1, and the other one is parent 2. Random numbers in the interval (0, 1) is generated for each gene. For a specific gene, if this created number is <0.7, the related value of parent 1 is used for that gene of offspring, otherwise the corresponding measure of parent 2 is copied for offspring (see Fig. 2).

3.1.3.6. *Mutation scheme.* We apply mutation operator with the probability of r_m; at first we choose two groups randomly. The corresponding values of these two groups are swapped, then for each group, two jobs are selected randomly, and the related values of them are swapped.

3.1.3.7. *Updating Pareto-archive.* A Pareto-optimal archive is identified. We calculate the Pareto-optimal solutions for this generation and update the Pareto-archive.

3.1.3.8. *Stopping criterion.* Finally, we check whether the stopping criterion is met or not. There are alternative options for stopping criteria, but we set a specific number of iteration as the termination of algorithm.

The pseudo-code of MOGA is explained in Fig. 3.

3.2 Non-Dominated Sorting Genetic Algorithm-II

Sirnivas and Deb [34] proposed NSGA, which is a class of MOEAs. Because of weaknesses of this approach such as high computational complexity, lack of elitism and need for specifying the sharing parameter, Deb et al. [29] modified the NSGA approach and proposed NSGA-II. The new proposed EA eliminated the criticisms of the former NSGA, to some extent.

In using NSGA-II, we encounter with two other procedures too, 'crowding distance assignment' and 'fast non-dominated sorting algorithm'. First, we explain these two algorithms and then we describe the basic structure of NSGA-II.

3.2.1 Fast Non-Dominated Sorting

A noticeable subject in dealing with a population of solutions in NSGA-II approach is to identify the different non-dominated fronts. To divide a population of solutions into different non-dominated levels, we employ the approach of fast non-dominated sorting.

To move step-by-step, the first front should be clarified at first, then the second front and so on. For separating the first front of solutions from a population comprising of N different solutions, each solution should be compared with all of the other solutions to determine whether it is non-dominated or not.

After examining all solutions, the first front will be identified. The solutions of the first front should be eliminated temporarily from the population of solutions and above procedure will be repeated for the remainder of solutions to find

Fig. 3 The pseudo-code of MOGA

Step 1: Encode solutions by employing random numbers
Step 2: Initialize a random population of N chromosomes.
Step 3: Calculate the value of makespan and total tardiness and normalize their values for each solution.
Step 5: Evaluate fitness function for each solution
Step 6: Calculate Pareto-optimal solutions of this iteration and update the Pareto-archive
Step 7: Apply reproduction using elitist strategy.
Step 8: Apply roulette wheel selection strategy.
Step 9: Apply crossover operator based on random numbers with the probability of r_c.
Step 10: Apply mutation operator based on random numbers with the probability of r_m.
Step 11: If the stopping criteria is met, it is the termination of algorithm, otherwise go to Step 3.

Fig. 4 Fast non-dominated sorting (POP)

Step 1: Let *fr*=1 (Set front counter equal to 1)
Step 2: Calculate makespan and total tardiness of jobs for each $x \in POP$.
Step 3: For each solution x belonging to *POP*:
3-1: Find n_x (the number of solutions which solution x is dominated by them)
3-2: Find S_x (set of all solutions which are dominated by solution x)
3-3: If solution x is not dominated by any other solution (n_x=0), assign it to first non-dominated front (front 1)
Step 4: Eliminate solutions belonging front *fr* from *POP*.
Step 5: Set *fr*=*fr*+1, start identifying solutions of next front.
Step 6: For each solution x belonging to last determined front (front (*fr*-1)):
6-1: For each y belonging to S_x, set n_y=n_y-1 and if n_y=0, y belongs to *fr*th front.
Step 7: If all solutions are assigned to fronts go to Step 8, otherwise go to Step 4
Step 8: Termination of algorithm

Fig. 5 Crowding distance (S)

Step 1: Set k=1 (k is objective counter)
Step 2: Calculate kth objective function for each solution
Step 3: Sort S in an ascending order according to the value of kth objective function.
Step 4: For the first and last solution of sorted set, distance values are equal to infinite, and for the others, use the normalized difference in the function values of two adjacent solutions as crowding distance.
Step 5: If k equals to 2 (the number of objectives in this study) go to Step 6, otherwise set k=k+1 and go to Step 2.
Step 6: The value of crowding distance for each solution is the summation of crowding distance of that specific solution for different objectives

the next front. This approach will continue till assigning all the solutions to different fronts.

It is assumed that n_x shows the number of solutions, where solution x is dominated by them, and all the solutions which are dominated by solution x are saved in S_x. n_x and S_x should be calculated for all solutions. It can be easily seen that each solution x with $n_x = 0$ belongs to the first non-dominated front because no other solution dominates it. Now, the first front solutions should be eliminated from the population and identifying the second front should be started. Since for every solution x belonging to the first front we have set S_x, for each member y of set S_x we set $n_y = n_y - 1$, because the solution x should be discounted. Now each solution with $n_y = 0$ belongs to the second front, and the second front is identified too. The above process should be repeated for the solutions belonging to the second non-dominated front. This procedure continues until determining all fronts.

The procedure of fast non-*dominated* sorting for population *POP* is briefly described in Fig. 4.

3.2.2 Crowding Distance

At first, we describe the crowded-comparison operator which is used for doing selection process in NSGA-II. Assume that we deal with two solutions, solution x and solution y. solution x is preferred if Solution x and solution y are positioned into different fronts and solution x belongs to the front with lower front number, or, Solution x and solution y belong to the same front but solution x has a higher crowding distance.

The crowded-comparison operator '$\prec$' can be briefly described as follows:

$$x \prec y \text{if} : \text{rank}(x) \prec \text{rank}(y), \text{or, rank}(x) = \text{rank}(y) \quad \text{and} \quad \text{distance}(x) \succ \text{distance}(y).$$

It means that between two solutions of the same level, the solution that is placed in an area which is less crowded is preferable to the other solution.

Crowding distance of solution x gives us an estimation of the density of solutions which are located around the specific solution x. To calculate the crowding distance for all solutions of a non-dominated set, we do following procedure for each objective function separately: at first all solutions of the non-dominated set should be sorted in an intensifying order according to specific objective, thereafter we assign an infinite value to the solutions with smallest and largest function measures which are known as boundary solutions. The other solutions which are located between boundary solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions [29]. The above process will be repeated for all objectives, and sum of crowding distance of different objective functions for each solution is used as final crowding distance of that solution.

The brief description of crowding distance algorithm for set S can be seen in Fig. 5.

3.2.3 Main Procedure of NSGA-II

By the help of two algorithms illustrated above, detailed procedure of NSGA-II for a bi-objective HFS with SDFST is presented in this section. At first, we generate random population P_1 of N chromosomes. Random number approach

which is described in Sect. 3.1.2. is employed for solution representation. All the solutions should be ranked with the help of fast non-dominated sorting algorithm. Each solution is assigned a fitness value according to its non-domination level. We use the following equation as fitness function for the solution x:

$$\text{fit}(x) = \frac{1}{\text{level}(x)}$$

It means that the fitness value for the solutions which are in the first non-dominated level is equal to 1, fitness value for solutions of second level is 0.5 and so on.

We use binary tournament selection, crossover and mutation operator for creating the offspring population Q_1. The crossover operator is done as described in Sect. 3.1.3.5, in addition for applying mutation operator we follow to the letter as explained in Sect. 3.1.3.6.

Combination of the parent population P_1 and the offspring population Q_1 results a new population R_1 with $2N$ chromosomes. Mixing parent population and offspring population continues for all the iterations.

In ith iteration, we have set R_i which is obtained by combination of Q_i and P_i. we identify the ranking of all solutions of R_i. Then all solutions of R_i are assigned to non-domination levels. Thereafter the new population P_{i+1} should be made for next iteration. For making P_{i+1} initially we use the first non-domination level (level 1) because solutions belonging to this level are the best individuals. It should be considered that we need N chromosomes to construct the population P_{i+1}, so if the number of solutions assigned to level'1' is less than N, we need to employ solutions of level'2', if sum of individuals assigned to level'1' and level '2' is less than N, solutions of third level should be applied and so on. Suppose that level'm' is the last non-domination set that we can use for making P_{i+1}, we define variable S as an entity which satisfies following relations:

$$S = |\text{level'1'}| + \cdots + |\text{level'}m-1'| \quad \text{and}$$
$$S + |\text{level'}m'| > N$$

We use crowding distance assignment and crowded-comparison operator to sort the solutions of level'm' in descending order, afterwards we use number of $N - S$ of the best solutions of *level*'m' to form all N needed chromosomes of P_{i+1}. Now we have new population P_{i+1} with N solutions which are the best solutions of the last generation. Again we use a binary tournament selection, crossover and mutation operator on population P_{i+1} to create population Q_{i+1}.

The above procedure continues until the stopping criterion is satisfied. A specific number of iterations determined as stopping criterion in this study. Figure 6 presents the Pseudo-code of NSGA-II.

3.3 Sub-Population Genetic Algorithm II

Chang et al. [27] proposed two-phase SPGA. The basic idea of SPGA is to divide the original population into some sub-populations. Afterwards we assign different weights into these sub-populations. All solutions of one sub-population make use of the same weight assigned to that specific sub-population. The main reason why we assign different weights into different sub-population is to expand the search area in different directions as well as result a better convergence. SPGA-II proposed by Chang and Chen [28] is the developed SPGA. Some characteristics differentiate SPGA-II from SPGA: SPGA-II employs a global Pareto-archive which immediately after finding a new Pareto-set should be updated. In addition SPGA-II applies a two phase approach, and the best solutions of first phase participate in generating initial solutions of second phase.

SPGA-II is implemented as follows: initially we generate a population containing N chromosomes. Such as two other MOEAs explained previously in this paper, we apply random number approach, described in Sect. 3.1.2. for solution representation. All solutions of this population should be split into several sub-populations. Afterwards the value of objective functions should be calculated for all individuals. As explained in Sect. 2.1 for avoiding the difficulties of dealing with different units for value of objective functions we employ the normalized objective functions. Thereafter, we assign weights to each objective, these weights are different for each sub-population. We compute the fitness value by the use of determined weights and normalized objective functions of each solution. Since this study considers two objectives the fitness function value for the solution x is calculated as follows:

$$\text{fit}(x) = w_t \cdot f_1(x) + (1 - w_t) \cdot f_2(x)$$

where $f_1(x)$ and $f_2(x)$ are, respectively, normalized value of makespan and total tardiness which are the objective functions of this study, these normalized values are obtained as explained in Sect. 2.1 w_t is the identified weight for the sub-population t which contains solution x. The value of w_t is acquired by:

$$w_t = \left|\sin\left(\frac{2\pi t}{C}\right)\right|$$

where C is determined equal to 40 and t is the number of sub-population containing the solution x.

In SPGA-II approach, the first phase execute the sub-populations with some weight vectors near to boundaries. For instance, if we construct 20 sub-population, the sub-population 1–5 and 16–20 are implemented in first phase and the other intermediate sub-populations are fulfilled in phase 2.

Fig. 6 The pseudo-code of NSGA-II

Step 1: Encode solutions by employing random numbers.
Step 2: Let I=1 (set iteration counter equal to 1)
Step 3: Create random population P_I of N chromosomes.
Step 4: Rank solutions of P_I by the help of 'fast non-dominated sorting' and considering minimization of two objectives.
Step 5: Calculate fitness value of each solution.
Step 6: Create offspring population Q_I of N chromosomes:
6-1: Use binary tournament selection.
6-2: Do the crossover based on random numbers with the probability of r_c.
6-3: Do the mutation based on random numbers with the probability of r_m.
Step 7: Make new population R_I by combining parent population P_I and offspring population Q_I, *i.e.* $R_I = P_I \cup Q_I$.
Step 8: If it is the maximum number of iteration (which is the stopping criteria in this study) go to Step 11. Otherwise set I=I+1.
Step 9: By the use of new population R_I, create population P_I with N chromosomes based on crowded-comparison operator.
Step 10: Go to Step 4.
Step 11: Rank solutions of population R_I by applying 'fast non-dominated sorting algorithm' and set the first non-dominated level as the final Pareto-set.
Step 12: Termination of algorithm.

Fig. 7 The pseudo-code of SPGA-II

Step 1: Encode solution by employing random numbers
Step 2: Generate a new population of N chromosomes.
Step 3: Decompose the original population into several sub-populations.
Step 4: Assign different weights to each objective for each sub-population
Step 5: Start phase 1.
Step 6: Apply SPGA for each sub-population of this phase:
6-1: Calculate value of two objective functions: makespan and total tardiness, and normalize their value.
6-2: Evaluate fitness value for each solution.
6-3: Find Pareto-set of each sub-population and update the global Pareto-archive
6-4: Apply binary tournament selection, reproduction, crossover and mutation operator
6-5: Replace the current sub-population by acquired solutions
Step 7: If it is the end of phase 1 go to Step 10.
Step 8: If it is the end of phase 2 go to Step 12.
Step 9: Go to Step 6.
Step 10: Start phase 2; Generate initial solutions for sub-populations of second phase by applying tournament selection for current solutions.
Step 11: Go to Step 6.
Step 12: Termination of algorithm.

After calculating fitness value, we determine the non-domination Pareto-set of each sub-population and update the global Pareto-archive. Then we apply binary tournament as selection strategy. for generating new solutions to replace the current sub-population with them we use reproduction, crossover and mutation operators, as described in Sects. 3.1.3.4–3.1.3.6, respectively.

After implementing phase 1, we should generate initial solutions for sub-populations of second phase. We apply tournament selection for current solutions to create sub-populations of phase 2. This way accelerates the convergence process.

The above procedure continues for second phase till meeting the stopping criteria.

In this study, we determine specific number of iterations as stopping criterion for each phase of SPGA-II.

The illustrated pseudo-code of SPGA-II can be seen in Fig. 7.

4 Experimental Results

In this section, we evaluate three MOEAs employed in this study and compare Pareto-optimal sets obtained by each algorithm. In order to code these algorithms for the considered problem, we used MATLAB 7.6 and run coded algorithms on a PC under Windows XP with an Intel core 2 Due, 2 GHz processor and using 2 GB of RAM.

4.1 Data Generation

Two different problem sizes are ranged: small and large. For each size of problem, we set required data consisting of total number of jobs (n_j), number of groups (n_g), number of stages (n_s), number of machines for stage s (m^s), range of processing times for each job, and range of sequence-dependent setup times for each group. In this study, it is assumed that number of jobs for all groups are identical, for example

Table 1 Factors and their levels

Factor	Level	
	Small	Large
Number of stages (n_s)	3–4–5	6–7–8
Number of groups (n_g)	4–5–7	8–11–12
Number of jobs (n_j)	12–15–49	64–110–144
Processing times (Pr_i^s)	$U(5, 75)$	$U(5, 75)$
	$U(5, 100)$	$U(5, 150)$
	$U(5, 150)$	$U(5, 300)$
	$U(5, 200)$	$U(5, 400)$
	$U(5, 250)$	$U(5, 450)$
Setup times (t_{ij}^s)	$U(5, 25)$	$U(5, 25)$
	$U(5, 50)$	$U(5, 100)$
	$U(5, 75)$	$U(5, 150)$
	$U(5, 100)$	$U(5, 200)$
	$U(5, 150)$	$U(5, 250)$
Maximum number of machines in each stage	3	4

in a small group with 12 jobs and 4 groups, each group have 3 jobs. Number of stages alters in the interval [3,5,6,8] for small and large problems, respectively. Number of groups is equal to 4, 5 and 7 for small problems and 8, 11 and 12 for large problems. Number of jobs is determined 12, 15, and 49 for small problems, and 64, 110 and 144 for large problems. We determine an entity which indicates the maximum number of machines for parallel stages, this measure is 3 and 4 for small and large problem, respectively, and by considering this measure as the higher bound we create a random vector of number of machines for each stage.

Processing times and setup times are uniformly distributed as shown in Table 1.

As stated above, each factor of problem data can take different levels. The factors and their levels are shown in Table 1.

We made ten instances of small and large problem size instances by combination of these factors. Each instance is run ten times to have more reliable data.

Due date of each job is an important subject which should be determined to compute due date for each job, at first we calculate sum of processing time of that job on all stages:

$$\mathrm{Pr}_i = \sum_{s=1}^{n_s} \mathrm{Pr}_i^s \quad \forall i = 1, \ldots, n_j$$

where Pr_i is the total processing time of job i on all. Thereafter, we calculate the mean of setup times for all possible subsequent of groups at each stage and aggregate them:

$$T_j = \sum_{s=1}^{n_s} \frac{\sum_{i=1, i\neq j}^{n_g} t_{ij}^s}{n_g - 1}, \quad j = 1, 2, \ldots, n_g$$

where T_j is the average of setup times for group j. We set setup time for each job equals to setup time of the group which job is assigned to.

$$st_i = T_j, \quad \forall i \in j\text{th group}$$

where st_i is the setup time for job i.

The due date for each job is generated as follows:

$$d_i = \frac{(1 + \mathrm{random} \times 2) \times n_s \times (st_i + \mathrm{Pr}_i)}{(n_g \times \min(m^s))}, \quad \forall i = 1, 2, \ldots, n_j$$

where d_i is the due date for job i, random is a random number over interval (0, 1).

4.2 Evaluation Metrics

The quality of non-dominated solutions obtained by each algorithm should be evaluated to study the performance of these algorithms. For this reason, we consider four metrics as follows:

(1) Mean ideal distance (MID): we set (0, 0) as ideal point and calculate the distance between Pareto-solution and this ideal point. The average of these distance is called MID. The lower value of MID, the better non-dominated set we have.

The value of MID can be computed as follows:

$$\mathrm{MID} = \frac{\sum_{i=1}^{n} c_i}{n}$$

where n is the number of non-dominated solutions, c_i is the distance between ith non-dominated solution and ideal point and this measure is acquired by the following equation:

$$c_i = \sqrt{f_{1i}^2 + f_{2i}^2}$$

where f_{1i} and f_{2i} are, respectively, the value of first and second objective function for ith non-dominated solution.

(2) Spread of non-dominated solution (SNS): this metric indicates the measure of diversity of Pareto-solutions, and more diversity of solutions is desirable. The value of SNS is measured as follows:

$$\mathrm{SNS} = \sqrt{\frac{\sum_{i=1}^{n} (\mathrm{MID} - c_i)^2}{n - 1}}$$

(3) Number of Pareto-solutions (NPS): the number of solutions in a non-dominated set is shown by this metric.

(4) Computational time to obtain the Pareto-optimal set via each algorithm.

4.3 Parameter Setting

The value of different parameters is one of the factors which influence the performance of each algorithm. Assigning a

Table 2 The suggested parameters for MOGA, SPGA-II, and NSGA-II

Algorithm	Problem size	Population size	Maximum number of iteration	Crossover rate (r_c)	Mutation rate (r_m)	Number of sub-population
MOGA	Small	80	300	0.8	0.1	–
	Large	250	400	0.8	0.1	–
SPGA-II	Small	80	300	0.8	0.1	10
	Large	250	400	0.85	0.05	20
NSGA-II	Small	80	300	0.8	0.2	–
	Large	250	400	0.9	0.1	–

suitable value for each parameter leads to achieve better outcomes. Moreover, various combinations of parameters may result in different sets of non-dominated solutions with different efficiency and effectiveness. For tuning three MOEAs proposed in this paper, we conduct extensive experiments with alternative sets of parameters to determine more effective values. To evaluate parameters measures, we applied two problems with different sizes of small and large (as explained in Sect. 4.1) for implementing an empirical study to identify the best level of each parameter.

Considering three different algorithms causes to have different factors for each of them. Various alternatives sets of parameter examined and based on the examined parameters, the suggested parameters values corresponding to three proposed algorithms for each size of problem are shown in Table 2.

It should be noted that *population size*, *maximum number of iteration*, Crossover rate (r_c), and mutation rate (r_m) are joint factors for all three algorithms, but *number of sub-population* is just related to SPGA-II.

4.4 Computational Results

In this section, we present a comprehensive experimental comparison of three MOEAs proposed in this study. For this sake, four evaluation metrics described in Sect. 4.1 are implemented for each non-dominated set corresponding to every test problem obtained by each employed algorithm. More precisely, we conducted an analysis of variance (ANOVA) technique to analyze the implemented experiments.

For each algorithm, we run each test problem ten times and four performance metrics are computed for each of them. The averages of these experiments are shown in Table 3.

As it can be observed, three metrics (MID, NPS and computational time) have better values for NSGA-II in comparing with two other algorithms.

Non-dominated solutions for a single run by MOGA, SPGA-II, and NSGA-II are presented in Figs. 8 and 9 for problems of small and large size, respectively.

It can be seen, NSGA-II outperforms the other two employed MOEAs in lower computational time. Tables 4 and 5 show the results of ANOVA technique which has carried out for computational time of each algorithm for small and large problems, respectively.

5 Conclusions and Future Studies

This paper deals with three MOEAs which are adapted to solve a bi-objective HFS scheduling problem with sequence-dependent family setup times. In this study, minimization of makespan and total tardiness of jobs are two conflicting objectives considered simultaneously. Taking into account two objectives results in dealing with a class of much more complex problems, where exact methods are unable to solve them in a reasonable time, for this reason, MOEAs are employed. MOGA, SPGA-II, and NSGA-II are three EAs applied in this investigation. Two different problem sizes of small and large are constructed and required data are set for each size of problem. Ten test problems of small and large sizes are made. We run the three noted algorithms for these test problems to achieve the Pareto solutions for each of them and calculate minimization of makespan and total tardiness for jobs. Thereafter, some evaluation metrics (MID, SNS, NPS, and CPU time) are employed to evaluate the Pareto solutions obtained by each algorithm. We run each test problem ten times for each algorithm and four performance metrics are computed for each of them.

Three metrics of MID, NPS and CPU time show better function of NSGA-II in comparison with two other algorithms for both sizes of small and large, and what distincts NSGA-II from MOGA and SPGA-II is CPU time, because NSGA-II performs in less computational time in comparing with two other algorithms which are used in this investiga-

Table 3 Evaluation of non-dominated solutions obtained by each algorithm

Problem	Size	MID			SNS			NPS			Computational time		
		MOGA	SPGA-II	NSGA-II	MOGA	SPGA-II	NSGA-II	MOGA	SPGA-II	NSGA-II	MOGA	SPGA-II	NSGA-II
1	Small	1,239.8	1,227.5	1,224.6	62.25	61.3	58.85	5.8	8	7.8	135.83	25.76	13.83
2		1,523.7	1,498.8	1,492.2	47.09	59.73	52.11	1.8	3	3	134.64	10.22	5.67
3		3,547.9	3,558.3	1,492.2	43.69	66.99	69.99	2.2	4	3.8	134.67	14.91	4
4		11,391.6	11,383.4	11,347.7	198.57	219.05	206.19	6	7	6.2	1,150.59	655.29	230.50
5		11,459.9	11,385.6	11,383.9	265.35	205.84	200.41	6.4	7.6	8.4	1,121.45	874.76	300.55
6	Large	16,782.7	16,764.2	16,632.7	161.58	294.05	227.39	2.8	2.6	2.8	1,079.16	858.69	481.67
7		18,996.4	18,952.9	18,920.4	206.07	49.91	52.48	2.4	3.8	3.2	1,167.75	717.3	260.33
8		13,145.2	13,079.4	13,030.4	1.41	66.75	49.12	2	3.2	5.6	4,645.51	3,716.51	1,445.1
9		26,505.1	26,556.6	26,469.3	128.99	133.6	142.4	5.8	7.4	9.4	4,610.88	4,075.17	2,372.2
10		32,671	32,673	32,571.1	127.7	142.5	170.77	2.8	5.2	4.6	6,977.83	5,690.12	2,031.59
Average		13,726.3	13,708.03	13,456.45	124.27	129.972	122.971	3.8	5.18	5.48	2,115.83	1,663.87	714.54

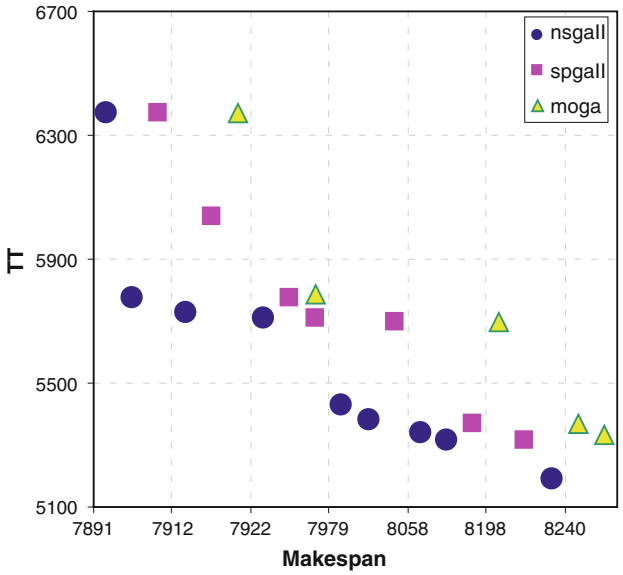

Fig. 8 Non-dominated solutions for small problem

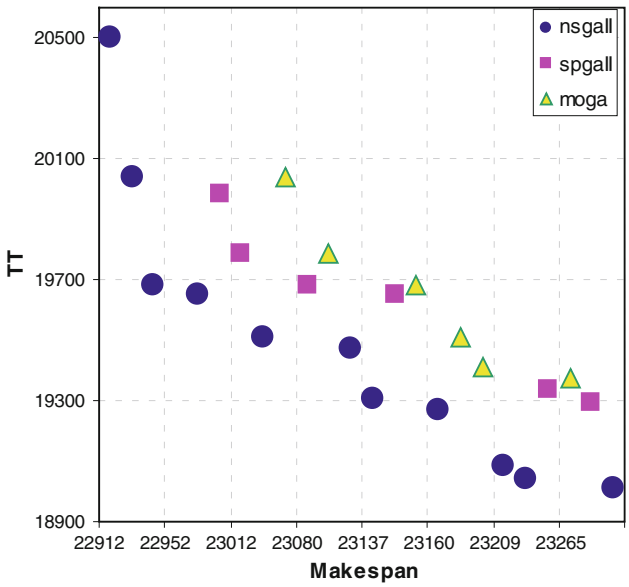

Fig. 9 Non-dominated solutions for large problem

tion. Also the result of ANOVA technique which has carried out for each metrics show NSGA-II outperforms the other two MOEAs implemented in this investigation.

Table 4 Analysis of variance for computation time (for small problem)

Source	DF	SS	MS	*F*	*P*
Factor	2	2,253,615	1,126,807	7.88	0.001
Error	79	10,296,990	143,014		
Total	74	12,550,605			

Table 5 Analysis of variance for computation time (for large problem)

Source	DF	SS	MS	*F*	*P*
Factor	2	74,928,593	37,464,296	10.91	0.000
Error	72	247,215,246	3,433,545		
Total	74	322,143,839			

For future researches, developing more effective metaheuristics, for solving this class of problems, is recommended. Moreover, the proposed metaheuristics can be extended for this problem with considering other objectives or constraints.

References

1. Gupta, J.N.D.; Tunc, E.A.: Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. Int. J. Prod. Res. **29**, 1489–1502 (1991)
2. Gupta, J.N.D.; Tunc, E.A.: Scheduling a two-stage hybrid flowshop with separable setup and removal times. Eur. J. Oper. Res. **77**, 415–428 (1994)
3. Gupta, J.N.D.; Hariri, A.M.A.; Potss, C.N.: Scheduling a two-stage hybrid flow shop with parallel machines at first stage. Ann. Oper. Res. **69**, 171–191 (1997)
4. Gupta, J.N.D.; Tunc, E.A.: Minimizing tardy jobs in a two-stage hybrid flow shop. Int. J. Prod. Res. **36**(9), 2397–2417 (1998)
5. Brah, S.A.; Hunsucker, J.L.: Branch and bound algorithm for the flow shop with multiple processors. Eur. J. Oper. Res. **51**, 88–99 (1991)
6. Portmann, M.C.; Vignier, A.; Dardilhac, D.; Dezalay, D.: Branch and bound crossed with GA to solve hybrid flowshops. Eur. J. Oper. Res. **107**(2), 389–400 (1998)
7. Moursli, O.; Pochet, Y.: A branch-and-bound algorithm for the hybrid flowshop. Int. J. Prod. Econ. **64**, 113–125 (2000)
8. Botta-Genoulaz, V.: Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. Int. J. Prod. Econ. **64**, 101–111 (2000)
9. Engin, O.; Doyen, A.: A new approach to solve hybrid flow shop scheduling problem by artificial immune system. Future Gen. Comput. Syst. **20**, 1083–1095 (2004)
10. Janiak, A.; Kozan, E.; Lichtensein, M.; Oguz, C.: Metaheuristics approaches to the hybrid flow shop scheduling problem with a cost-related criterion. Int. J. Prod. Econ. **105**(2), 407–424 (2007)
11. Gupta, J.N.D.; Kruger, K.; Lauff, V.; Warner, F.; Sotskov, Y.N.: Heuristics for hybrid flow shop with controllable processing times and assignable due dates. Comput. Oper. Res. **29**(10), 1417–1439 (2002)
12. Allaoui, H.; Artiba, A.: Integrating simulation and optimization to schedule a hybrid flow shop with maintenance constraints. Comput. Ind. Eng. **47**, 431–450 (2004)
13. Jungwattanakit, J.; Reodecha, M.; Chaovalitwongse, P.; Werner, F.: An evaluation of sequencing heuristics for flexible flowshop scheduling problem with unrelated parallel machines and dual criteria. Otto-von-Guericke-Universitat Magdeburg, Preprint 28/05, pp. 1–23 (2005)
14. Mitrofanov, S.P.: The Scientific Principles of Group Technology. National Lending Library, Boston Spa(1966)
15. Ham, I.; Hitmoi, K.; Yoshida, T.: Group technology: applications to production management. Hinghama: Kluwer (1985)
16. Kurz, M.E.; Askin, R.G.: Scheduling flexible flow lines with sequence-dependent setup times. Eur. J. Oper. Res. **159**, 66–82 (2004)
17. Zandieh, M.; Fatemi Ghomi, S.M.T.; Moattar Husseini, S.M.: An immune algorithm approach to hybrid flowshops scheduling with sequence-dependent setup times. Appl. Math. Comput. **180**, 111–127 (2006)
18. Hendizadeh, S.H.; Faramarzi, H.; Mansouri, S.A.; Gupta, J.N.D.; Elmekkawy, T.Y.: Meta-heuristics for scheduling a flowline manufacturing cell with sequence dependent family setup times. Int. J. Prod. Econ. **111**, 593–605 (2008)
19. Franca, P.M.; Gupta, J.N.D.; Mendes, A.S.; Moscato, P.; Veltink, K.: Evolutionary algorithm for scheduling a flowshop manufacturing cell with sequence dependent family setups. Comput. Ind. Eng. **48**, 491–506 (2005)
20. Logendran, R.; Salmasi, N.; Sriskandarajah, C.: Two-machine group scheduling problems in discrete parts manufacturing with sequence-dependent setups. Comput. Oper. Res. **33**, 158–180 (2006)
21. Zolfaghari, S.; Liang, M.: Jointly solving the group scheduling and machining speed selection problems: a hybrid tabu search and simulated annealing approach. Int. J. Prod. Res. **37**, 2377–2397 (1999)
22. Mukerjee, A.; Biswas, R.; Deb, K.; Mathur, A.P.: Multi-objective evolutionary algorithm for the risk-return trade-off in bank loan management. Int. Trans. Oper. Res. **9**(5), 583–597 (2002)
23. Schaffer, J.D.: Multiple objective optimization with vector evaluated genetic algorithms. In: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, pp. 93–100. Carnegie-Mellon University, Pittsburg (1985)
24. Murata, T.; Ishibuchi, H.; Tanaka, H.: Multi-objective genetic algorithm and its application to flow shop scheduling. Comput. Ind. Eng. **30**(4), 957–968 (1996)
25. Zitzler, E.; Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans. Evol. Comput. **3**(4), 257–271 (1999)
26. Zitzler, E.; Laumanns, M.; Thiele, L.: Spea2: improving the strength Pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Network Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Switzerland (2001)
27. Chang, P.C.; Chen, S.H.; Lin, K.L.: Two-phase sub population genetic algorithm for parallel machine scheduling problem. Expert Syst. Appl. **29**(3), 705–712 (2005)
28. Chang, P.C.; Chen, S.H.: The development of a sub-population genetic algorithm-II (SPGA-II) for multi-objective combinatorial problems. Appl. Soft Comput. **9**(1), 173–181 (2009)
29. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
30. Tamaki, H.; Kita, H.; Kobayashi, S.: Multi-objective optimization by genetic algorithms: a review. In: Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp. 517–522. IEEE Service Center, Piscataway (1996)
31. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA J. Comput. **6**(2), 154–160 (1994)
32. Norman, B.A.; Bean, J.C.A.: A genetic algorithm methodology for complex scheduling problems. Nav. Res. Logist. **46**, 199–211 (1999)
33. Goldberg, E.D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading (1989)
34. Sirnivas, N.; Deb, K.: Multi-objective function optimization using non-dominated sorting genetic algorithms. Evol. Comput. **2**(3), 221–248 (1995)