Int J Adv Manuf Technol (2010) 49:1129–1139
DOI 10.1007/s00170-009-2475-z

ORIGINAL ARTICLE

An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines

E. Rashidi · M. Jahandar · M. Zandieh

Received: 15 August 2009 / Accepted: 1 December 2009 / Published online: 27 February 2010

Abstract In this paper, we address the hybrid flow shop scheduling problems with unrelated parallel machines, sequence-dependent setup times and processor blocking to minimize the makespan and maximum tardiness criteria. Since the problem is strongly NP-hard, we propose an effective algorithm consisting of independent parallel genetic algorithms by dividing the whole population into multiple subpopulations. Each subpopulation will be assigned with different weights to search for optimal solutions in different directions. To further cover the Pareto solutions, each algorithm is combined with a novel local search step and a new helpful procedure called Redirect. The proposed Redirect procedure tries to help the algorithm to overcome the local optimums and to further search the solution space. When a population stalls over a local optimum, at first, the algorithm tries to achieve a better solution by implementing the local search step based on elite chromosomes. As implementing the local search step is time-consuming, we propose a method to speed up the searching procedure and to further increase its efficiency. If the local search step failed to work, then the Redirect procedure changes the direction and refreshes the population. Computational experiments indicate that our proposed improving procedures are thriving in achieving better solutions. We have chosen two measures to evaluate the performance of our proposed algorithms. The obtained results clearly reveal the prosperity of our proposed algorithm considering both measures we have chosen.

Keywords Hybrid flow shop · Multi-objective optimization · Non-dominated solution · Blocking processor · Sequence-dependent setup times

E. Rashidi
Department of Industrial Engineering,
Mazandaran University of Science and Technology,
Babol, Iran

M. Jahandar
Department of Industrial Management, Management Faculty,
University of Tehran,
Tehran, Iran

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

1 Introduction

Hybrid flow shop scheduling problem (HFS) is an extension of the classical flow shop. It incorporates all the difficulties and complexities of its predecessor flow shop (FS) and is more complex than the FS because of the additional need to determine the assignment of operations to the machines in each stage. Each stage may be separated by finite intermediate storages (capacities or buffers). Every job requires processing at exactly one machine per stage. Processors in each stage can be identical, uniform, or unrelated. A job, once its processing is completed on a machine, is transferred directly to either an available machine in the next stage or, if any buffer is available, goes to a buffer ahead of that stage [1]. In some cases, there is no buffer between stages. When no buffer exists or no buffer is available, a completed job remains on a machine and blocks it for other jobs until a downstream machine becomes available. It is usually assumed that the buffer size between every two successive stages is infinite. However, in real engineering practice, such as petrochemical industries and cell manufacturing, and in modern manufacturing industries, such as just-in-time production systems or flexible assembly lines and surface mount technology lines in the electronics industry for assembling printed circuit boards, buffer is either non-existent or with limited size [2–5]. There have been a lot of works on the hybrid or flexible flow shops. A taxonomy by Quadt and Kuhn [6] for the flexible flow line problems yields a thorough review in the case of flexible flow shop problems. In contrast, with lots of works on flexible and hybrid flow shops, there is not much work in the case of hybrid or flexible flow shops considering multi-objective approach. For example, Jungwattanakit et al. [7] considered flexible flow shop with unrelated parallel machines and setup times. Behnamian et al. [8] presented a multi-phase covering Pareto optimal front method to multi-objective scheduling in a realistic hybrid flow shop using a hybrid meta-heuristic. But this is not true for the case of flow shops. Tavakkoli-Moghaddam et al. [9] proposed a hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives, weighted mean completion time, and weighted mean tardiness. Pasupathy et al. [10] developed a multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. Varadharajan and Rajendran [11] introduced a multi-objective simulated annealing algorithm for the same problem. Chang et al. [12] developed a two-phase subpopulation genetic algorithm for parallel machine scheduling problem. Chang et al. [13] also presented a subpopulation genetic algorithm with mining gene structures for multi-objective flow shop scheduling problems. The majority of research on scheduling problems addresses only a single criterion, while the majority of real-life problems require the decision maker to consider more than a single criterion before arriving at a decision. The research on multiple criteria is mainly focused on the single machine scheduling problem; see [14]. The reason for this is that the scheduling problem with multiple machines is difficult even with a single criterion. Therefore, considering more than a single criterion makes the multiple-machine problem even more difficult to solve [15]. Although the flexible flow shop problem has been widely studied in the literature, most of the studies related to flexible flow shop problems are concentrated on problems with identical processors; see, for instance, Gupta et al. [16], Alisantoso et al. [17], Lin and Liao [18], and Wang and Hunsucker [19]. In a real-world situation, it is common to find newer or more modern machines running side by side with older and less efficient machines. Even though the older machines are less efficient, they may be kept in the production lines because of their high replacement costs. The older machines may perform the same operations as the newer ones, but they would generally require a longer operating time for the same operation. In this paper, the hybrid flow shop problem with unrelated parallel machines is considered; that is, there are different parallel machines at every stage, and the speeds of the machines are dependent on the jobs. Moreover, several industries encounter setup times which result in even more difficult scheduling problems. In this paper, sequence-dependent setup time restrictions are taken into account as well. We also assume that there is no buffer between stages and machines are blocked when the completed jobs cannot release themselves because of unavailability of machines in the next stage. As we said earlier in the case of hybrid or flexible flow shop, we found scarce work with multi-objective approach. To the best of our knowledge, this is the first work which approaches the HFS with unrelated parallel machines and sequence-dependent setup times and processor blocking to minimize makespan and maximum tardiness simultaneously. Makespan and maximum tardiness are two commonly used performance measures in the scheduling literature. Makespan is a measure of system utilization, while maximum tardiness is a measure of performance in meeting customer due dates.

The multi-criteria scheduling problem is one of the main research subjects in the field of multiple-objective programming. Several procedures have been developed to deal with this type of problems where some conflicting criteria have to be optimized simultaneously. Using constant weights is the most commonly used approach in the literature. In this way, a multi-objective problem is transformed into a single-objective problem through a linear combination of objectives and weights. The major imperfection of the transformation is that the weight of each objective must be given subjectively in advance. If the decision maker is not experienced enough, it leads to great gaps between the actual and expected solutions. An alternative idea used to solve the multi-objective problem does not specify the weights in advance. It offers various solutions to the decision makers to choose from. The solutions searched are called Pareto optimal solutions. The number of Pareto optimal solutions is often not unique, and decision makers can select one of the Pareto optimal solutions to meet their desire. It is a more flexible way than the constant weight weighting method. In this paper, we want to develop an effective algorithm to yield a set of good non-dominated solutions (in absence of known Pareto optimal solutions) for these complex problems.

The rest of the paper is organized as follows: Problem description and methodology will be described in Section 2. The proposed genetic algorithm will be discussed in Section 3. Computational evaluation will be examined in Section 4, and Section 5 will finally conclude the paper.

2 Problem description and methodologies

Hybrid flow shop is a flow line with several parallel machines on some or all production stages. All products follow the same linear path through the processing route starting at first stage and release in the last stage. Processors in each stage are unrelated, which means they are different in processing time but similar in operation. In this system, a set of n independent jobs, $j \in \{1, 2, \ldots, n\}$, with due dates, $d_1, \ldots, d_n$, has to be processed. Each job j has its fixed standard processing time ps^t_j for every stage t, $t \in \{1, 2, \ldots, g\}$. Moreover, at each stage t, there is a set of m^t unrelated parallel machines. Hence, machine i at stage t can process job j at the relative speed $v^t_{i,j}$. The processing time $p^t_{i,j}$ of job j on machine i at stage t is equal to $ps^t_j / v^t_{i,j}$. After releasing of job j and before beginning processing of the next job, job k, on machine i, some sort of setup must be performed; we assume this setup time equals to $s^{t,i}_{jk}$. In this research, setup times depend on sequencing of jobs, which means the setup times are sequence-dependent. We assume that all the setup times are known and constant. Also, we assume that there is no buffer between stages, so a completed job must remain on the machine and block it for other jobs until a downstream machine becomes available. The following two restrictions should also be fulfilled: (1) No machine can process more than one job at the same time and (2) no job can be processed by more than one machine at the same time. The objective is to simultaneously minimize the makespan ($C_{\max}$) and the maximum tardiness ($T_{\max}$).

The hybrid flow shop scheduling problem is a strongly NP-hard problem. As the size of problem becomes larger, it is time-consuming to solve it by exact algorithms, such as branch-and-bound algorithm and dynamic programming. It is even worse when we face multiple objectives. Therefore, recent development in evolutionary multi-objective optimization provides interesting results as discussed by Deb et al. [20], Zitzler et al. [21], and Chang et al. [12]. In addition, some multi-population-like approaches such as segregative genetic algorithms by Affenzeller [22], multi-sexual genetic algorithm by Lis and Eiben [23], multi-population genetic algorithm by Cochran et al. [24], hierarchical fair competition model by Hu et al. [25], and multi-objective particle swam optimization [26, 27] were proposed. Also, Chang et al. [13] proposed a subpopulation genetic algorithm with mining gene structures for multi-objective flow shop scheduling problems. They compared their proposed algorithm against the well-known NSGA II [20] and SPEA II [28]. Their computational experiments reveal that their proposed algorithm clearly supersedes these well-known algorithms. The prosperity of using the multi-population (also subpopulation) approach inspired us to use a similar approach. As we use different representations from what Chang et al. [13] used, we cannot adopt their algorithm for our considered problem. Instead, we propose a novel and effective procedure, called Redirect. In the following section, we describe our proposed algorithm in details.

3 Proposed methodology and algorithm

3.1 Methodology

There are a lot of studies in the literature which use genetic algorithm (GA) for the multi-objective problems. The primary reason for this is the empowering feature of GAs— population approach—that is highly suitable for use in multi-objective optimization. Since GAs work with a population of solutions, multiple Pareto optimal solutions can be found in a GA population [5]. Here, we want to simply describe how we want to handle the problem. Let w_1 and w_2 be the weights of $C_{\max}$ and $T_{\max}$, respectively, so the objective function can be calculated with the equation below:

$$\text{Min } y = w_1 C_{max} + w_2 T_{max}. \tag{1}$$

In this way, our bi-criteria problem is transformed into an ordinary single-objective problem, but as we mentioned before, determining the weights in advance is a delicate work. Instead, we can divide the population into some groups which all the individuals in each group seek for their own single-objective function, and all the groups work in parallel to seek for Pareto optimal solutions. By this way, we can transform the multi-objective problem to a single one with various weights. Here, we will consider a set of z different weights with a small deviation between each successive pairs of weights to transform our multi-objective problem to a single one.

$$W = \{(w^1{}_1, w^1{}_2), (w^2{}_1, w^2{}_2), \ldots, (w^z{}_1, w^z{}_2)\}. \tag{2}$$

In the same way, we can divide the population into z subpopulations which will be assigned with the above set of z different weights to search for optimal solutions in different directions. Each subpopulation, which is independent and unrelated to each other, works as parallel GAs with the same operators (elitism, crossover, and mutation) but with their own criterion (see Fig. 1). This way can help strengthen the solution diffusivity effectively. In this work, like many other investigations, a Pareto archive set is provided to explicitly maintain the set of non-dominated solutions. This approach is incorporated to prevent losing the set of non-dominated solutions during the optimization process. This archive is iteratively updated to get closer to the correct Pareto optimal front. When a new solution

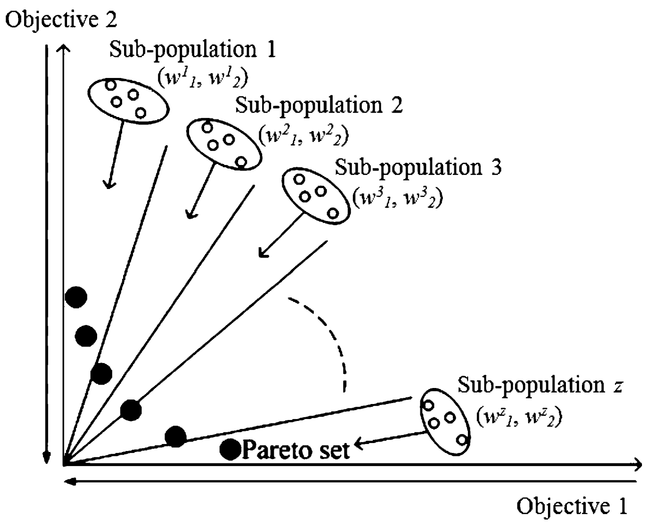

Fig. 1 The whole population is divided into several subpopulations [13]

enters the archive set, any solution in the archive dominated by this solution is removed from the set [16]. In the following subsection, we describe the procedures of the proposed algorithm.

3.2 Representation, operators, and other procedures

We use the random keys representation for our algorithm. This representation is widely used in the literature, e.g., [29–32]. In this representation, an individual that represents a solution is made of an array comprising n cells in series. Each cell indicates a job. Each job is assigned a real number whose integer part is the machine number to which the job is assigned and whose fractional part is used to sort the jobs assigned to each machine. Then, for each solution, we need n real numbers. Once the job assignments and order on each machine are found through the decoding, a schedule can be built incorporating additional factors such as blocking processor and sequence-dependent setup times. This representation is used for jobs in the first stage. In this strategy, the genetic operators and procedures only affect the assignments and sequences of jobs at the first stage. In other stages, jobs are supposed to transfer into the machines that allow them to complete at the earliest time (regarding the speed of the available machines and also the setup times). This greedy fashion strategy can be described in the following steps:

- Calculate the completion time of jobs in the first stage.
- Create a list by arranging jobs in the ascending order of their completion times.
- For the job that is earliest to complete in the list, calculate its completion times on each available machine in the next stage. This job should be assigned to the machine that completes it in the earliest time.
- If there were more than one job completed at the same time, for each job, calculate its completion time on each available machine in the next stage and assign each job to the machine that completes it in the earliest time. If there were more than one job that should be assigned to the same machine with this rule, choose the job that completes earlier.
- Continue with the next job on the list.

Job assignment and scheduling for the next stages is like above; it depends on the availability of machines, the arrival time of jobs, and the sum of their processing times and setup times (completion times) on the machines. In this process, we consider the fact that there is no buffer between successive stages, and a completed job may have to remain on a machine and block it for other jobs until a downstream machine becomes available.

Now we want to describe how our proposed GAs work. The algorithm begins with a random initial population. The random selection is used to select individuals to be parents. New population is made of children produced by the one point order crossover (OPX) and those chromosomes that reproduced and directly copied into the new population since *elitism* strategy is applied. To avoid premature convergence, a new individual will only replace its predecessor if its objective value is better and if its sequence is not already in the population. Also, some individuals are produced by implementing the swap mutation operator. After primary experiments, we set the following parameters: The population size is set to 1,050, and the number of different weights (also the number of subpopulation) is set to 21 (in Eq. 2, set $z = 21$), so $W = \{(0.0, 1.0), (0.05, 0.95), \ldots, (1.0, 0.0)\}$ $(W = \{(i \times s, 1.0 - i \times s) : i = 0, 1, \ldots, 20; s = 0.05)\})$ and the population size of each subpopulation will be 50. Also, the rate of crossover, mutation, and elitism are, respectively, set to 70%, 10%, and 20%. Now, our algorithm, multi-objective parallel genetic algorithm (we call it MOPGA), is ready to challenge problems. But primary experiments show that the algorithm still needs to be improved. By tracking the results of the algorithm, we observed that some subpopulations could not yield better solutions over time, and some sub-GAs have failed to properly search the solution space. It means that they have stalled over a local optimum. We suggest adding an improvement phase by applying a local search step to our proposed algorithm. The drawback of this approach is that applying local search to all individuals in every generation results in a very slow algorithm. Our proposal is to apply a local search to a limited number of individuals and not to each generation. As we work with independent and discriminate parallel sub-GAs, we suggest adding an independent and separate local search step for each sub-GA. For each subpopulation, we focus only on a leading

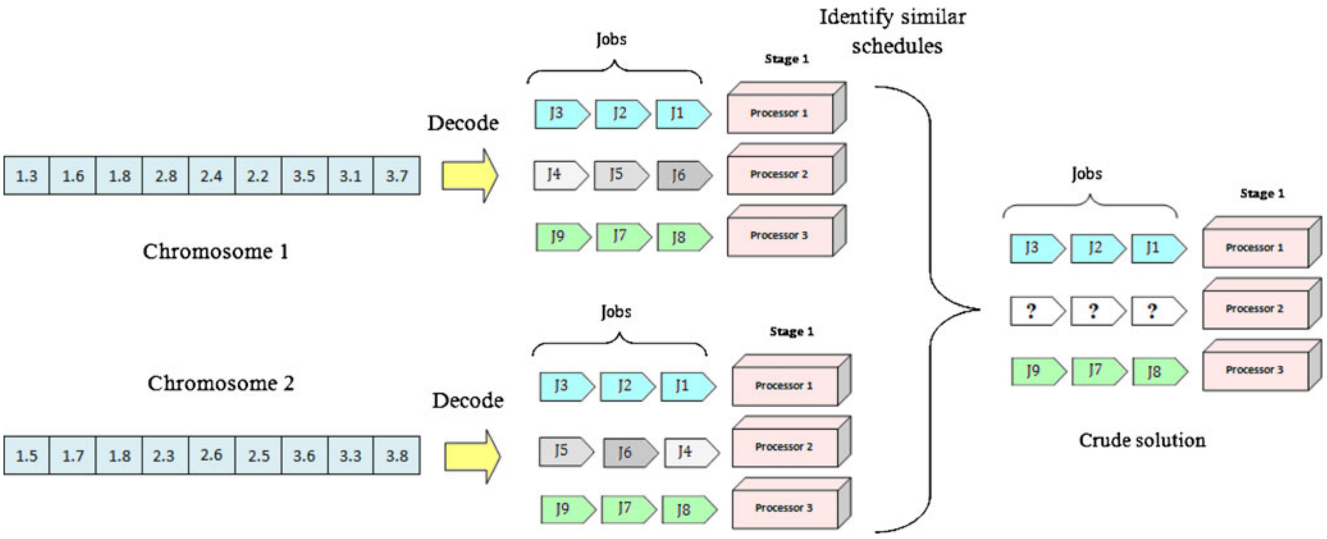

Fig. 2 Working procedure of identifying similar schedules to start local search step

group of chromosomes called elite chromosomes. For the case of trapping in local optimums if we can succeed to improve these leading individuals, then we have succeeded to pull out from these traps. The proposed local search works as follows:

- Local search step

About 10% of the best individuals in each subpopulation are considered for local search step. We randomly pick two chromosomes from these elite chromosomes. Then, a procedure decodes these chromosomes to identify the schedules they represent. By identifying the similar schedules between these two chromosomes and copying and coding these similarities to a new chromosome, we make a crude and incomplete solution (Fig. 2). Then, we investigate the ways we can complete this solution; each way is considered as an option. As there may exist a lot of options, it is extremely time-consuming to investigate all the options. Therefore, we define a counter ($\text{counter}_{\text{Option}}$) to count every option we exam. We consider an upper bound called $\text{Option}_{\max}$ for this step. To let the algorithm examine more options as the number of jobs increases, we set $\text{Option}_{\max}$ to $2 \times n$. We can continue this investigation with new chromosomes if the total number of options we have examined was less than the upper bound. The local search step stops when it achieves a better solution or reaches the upper bound. If there was no similarity between the two selected chromosomes, then the algorithm takes another two elite chromosomes. If the algorithm could not find any chromosomes with similar schedules, then the local search step would not be continued.

To further help the algorithm overcome local optimums and save diversification, we embed the local search step

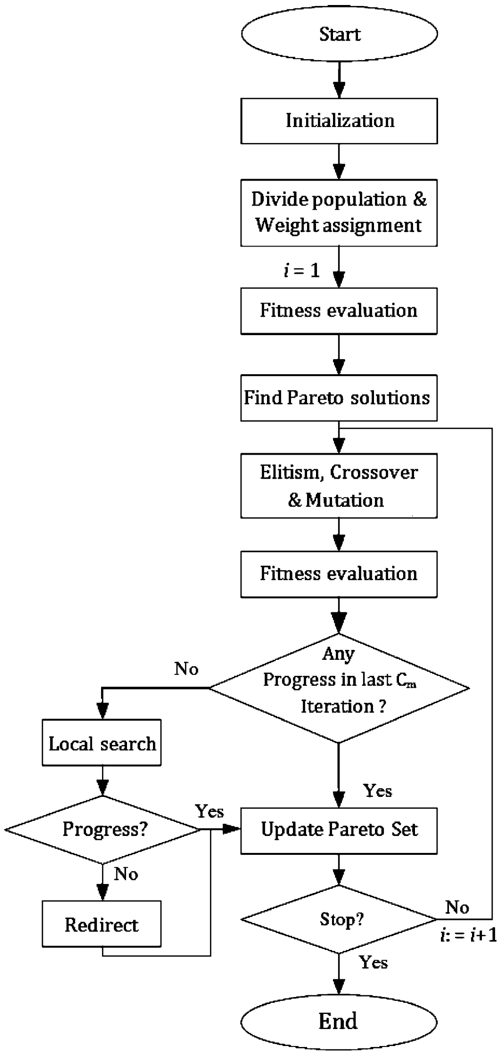

Fig. 3 Flowchart of IHMOPGA

Fig. 4 Pseudo-code of IHMOPGA

```
Initialization;          // population initialization
Divide population;
Weight assignment;
i: = 1;       // generation counter
While i <= Num_generation do
Begin       // while
    For j := 1 to Num_subpopulation do    // j: counter of sub_population
    Begin
        $func_{j,i}$ := Fitness_evaluation ();
        Random_selection();
        Eliticism();
        Crossover();      // OPX
        Mutation();    // Swap mutation
        If $func_{j,i}$ = $func_{j-1,i}$ then $Count_j$:= $Count_j$+1;
        If $count_j$>$C_m$ then
        Begin
          //Local_Search();
            Selection();         // Select two chromosomes from 10% of the best individuals of sub_ population i
            Find_similarities();         // identify similar schedules between the two selected chromosomes
            Make_crude_solution();
            For $counter_{option}$:=1 to $Option_{max}$ do
            Begin
               Complete_crude_solution();
               If $func_{j,i}$ < $func_{j-1,I}$ then break;
            End;  // end of if
            If $func_{j,i}$ < $func_{j-1,i}$ then $Count_j$= 0
            Else
            Begin
              // Redirect procedure
                Change_weights();
                Function_evaluation();
                Ascending_sort();
                Eliticism(20%);             // skip first 20% of chromosomes
                Swap_mutation(40%);
                     // replace 50% of remaining chromosomes (from 100% - 20%)
                    // by simple Swap mutations of the first 20% best individuals
                 Swap_mutation(40%);
                    // The remaining 50% are replaced by newly randomly
                    // generated schedules.
                 $Count_j$:= 0;
             End;    // end of else
          End;  // end of if
     End;    // end of for
    j:= j+1;
    Update_Pareto_solutions();
End;        //end of while
```

into a new proposed method called Redirect. This method works as follows:

1. For subpopulation j, at each generation i, store the related minimum value of objective function, $func_{j,i}$.
2. If $func_{j,i} = func_{j,i-1}$, then make $count_j = count_j + 1$. Otherwise, make $count_j$=0.
3. If $count_j > C_m$, then apply the local search step (we consider C_m= 10).
4. If the local search step could improve the solution, then make $count_j$= 0. Otherwise, apply the following procedures:
   - It seems that we have failed to find a better solution according to the objective function we have used; we suggest changing the objective function by changing the weights and then evaluating the individuals according to this new objective function. Let (w^j_1, w^j_2) be the weights we have used for subpopulation $j(1 < j < 21)$. We change these weights as follows: We generate a randomly chosen number called *Rand* between $(w^j_1-0.025,\ w^j_1+0.025)$. The new weights are $(w^j_1, w^j_2) = (Rand, 1 - Rand)$. For j=1 and j=21, we generate *Rand* in (0, 0.025) and (0.975, 1), respectively.
   - Sort the subpopulation in ascending order of its new objective function.
   - Skip the first 20% individuals from the sorted list (the best individuals).

- From the remaining 80% individuals, 50% of them are replaced by simple swap mutations of the first 20% best individuals. The remaining 50% are replaced by newly randomly generated schedules.
- Make $count_j = 0$.

We name this new algorithm improved hybrid multi-objective parallel genetic algorithm (IHMOPGA).The outline of OHMPOGA is summarized in Figs. 3 and 4. Table 1 illustrates all the parameters and their values used for our algorithms in this paper. The parameters that relate to local search and Redirect procedures are only for IHMOPGA. The selected crossover type was chosen from three types of crossovers: OPX (one-point order crossover), TPX (two-point order crossover), and PUX (parameterized uniform crossover). The mutation type was selected from two types: swap mutation and SHIFT mutation. We examined all these types and finally chose the best types with the best values that fit them; we summarized the results in Table 1.

To primarily test the performance of this new algorithm, we let it solve the same sample problems that the first algorithm (MOPGA) had solved. This primary test shows that our new proposed algorithm has performed better than its predecessor. As illustrated in Fig. 5 as a sample result of this test, our proposed IHMOPGA has reached a better and richer set of non-dominated solutions. To evaluate more the performance of our proposed algorithms, in the next section, we will carry out extensive experiments.

4 Computational evaluation

4.1 Generation of test data

Data required for a problem consist of the number of jobs (n), the range of setup and processing times, due dates, and the number of stages (g). We used problems with $n=\{6, 30, 100\}$ and $g=\{2, 4, 8\}$. The number of machines (m) in each

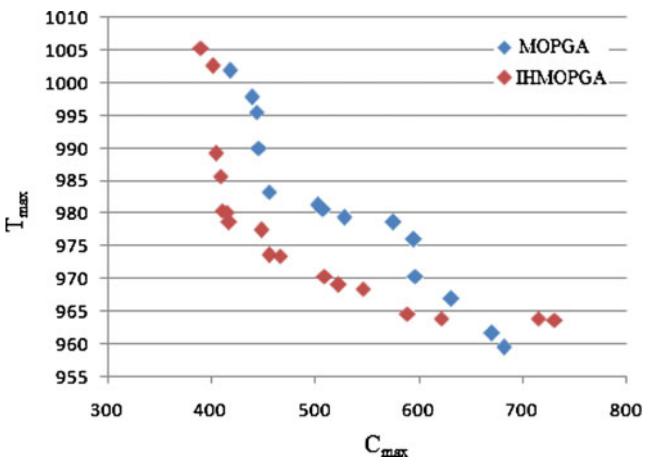

Fig. 5 Plot of obtained Pareto frontiers for IHMOPGA and MOPGA for a sample instance (the maximum number of generations is set to 300)

stage is distributed uniformly between interval (1, 4). Similar to [7], the standard processing times are generated uniformly from the interval [10, 100], and the relative speeds are distributed uniformly in the interval [0.7, 1.3]. The duration of sequence-dependent setup times generated uniformly from the intervals [1, 50], [1, 100], and [1, 125]. The due date of a job is set in a way that is similar to the approach used by Jungwattanakit et al. [7].

4.2 Comparison metrics

It is impossible to find the Pareto optimal solutions using the naive enumeration algorithm because of extreme complexity of the problems. In this regard, we use the following comparison metrics: (1) the number of non-dominated solutions that each algorithm can find and (2) the percent of non-dominated solutions obtained from each algorithm in the net non-dominated solutions (a set of non-dominated solutions (NDS) obtained after putting together the non-dominated solutions found by each algorithm) [10].

4.3 Experimental results

A set of 27 instances was generated as such: The set of instances comprises nine combinations of n and g, being $n=\{6, 30, 100\}$ and $g=\{2, 4, 8\}$. Three combinations for setup times are as previously defined. For each instance, we have generated five different problems. We set the stopping criterion to CPU time limit fixed to $n^2 \times m \times g \times 25$. This stopping criterion not only permits for more time as the number of jobs or machines increases, but also it is more sensitive toward rise in the number of jobs than the number of stages. We let both algorithms solve the generated instances. The algorithms are stochastic in nature; then to

Table 1 Parameters and their values

Where used	Parameter	Value
Main algorithm	Crossover rate	70%
	Mutation rate	10%
	Elitism rate	20%
	Population size	1050
	No. of subpopulations	21
Local search	C_m	10
Redirect procedure	Option_{max}	$2 \times n$

Table 2 Comparison results for g=2

Job	Setup time	No. net-NDS	IHMOPGA		MOPGA	
			No. NDS	Percent in the NET	No. NDS	Percent in the NET
N=6	ST [1, 50]	1	1	1	1	1
	ST [1, 100]	1.61	1.61	1	1.52	0.76
	ST [1, 125]	1	1	1	1	1
N=30	ST [1, 50]	5.23	5.13	0.42	3.54	0.58
	ST [1, 100]	3.43	3.19	0.56	2.98	0.65
	ST [1, 125]	6.74	6.05	0.54	5.37	0.46
N=100	ST [1, 50]	12.35	13.45	0.64	9.63	0.36
	ST [1, 100]	4.78	5.11	0.61	3.46	0.39
	ST [1, 125]	8.13	7.18	0.77	6.87	0.42
Average		4.92	4.86	0.73	3.93	0.62

have more reliable results, we have conducted five different replicates of each experiment to finally average the results. These results are summarized in Tables 2, 3, and 4. According to these tables, it is easy to verify the better performance of IHMOPGA against our other algorithm, MOPGA, in both two metrics especially in large instances. Although the number of non-dominated solutions for both algorithms increases, as the number of stages increases, on the other hand, the other criterion (percent in the net) decreases for MOPGA. This shows that as the number of stages increases, the superiority of IHMOPGA becomes more evident. As we can see, the more difficult instances are the better performance our IHMOPGA has against the MOPGA. This superiority indicates the effectiveness of our proposed Redirect procedure and also shows that the local search steps we have proposed are able to further search the solution space and find more non-dominated solutions.

We found that both algorithms, the MOPGA and the IHMOPGA, have nearly the same performance in the beginning of the computations, and it seems rational because both algorithms are the same when it is easy to find better solutions, and there is no challenging problem such as stalling over a local optimum for which the local search step and the Redirect procedure come into action for the IHMOPGA. As the number of generation increases, it is getting harder to find better solutions, and then it is more probable that the algorithms stall over local optimums. Then, it is time for the IHMOPGA to call its improvement phase to come into action. At first, a procedure tries to identify what is similar between the two randomly chosen elite chromosomes. The identified similar schedules are copied to a new chromosome to make a crude solution. Then, the local search step tries to search for a better solution by completing remaining schedules in the crude solution. If the local search step failed to improve the

Table 3 Comparison results for g=4

Job	Setup time	No. net-NDS	IHMOPGA		MOPGA	
			No. NDS	Percent in the NET	No. NDS	Percent in the NET
N=6	ST [1, 50]	2.43	2.43	1	2.43	1
	ST [1, 100]	3.14	3.14	1	2.34	0.31
	ST [1, 125]	4.53	4.53	1	3.15	0.75
N=30	ST [1, 50]	6.73	6.57	0.56	6.73	0.47
	ST [1, 100]	7.16	8.33	0.57	6.58	0.51
	ST [1, 125]	4.48	4.11	0.62	2.36	0.49
N=100	ST [1, 50]	6.24	7.34	0.65	6.08	0.35
	ST [1, 100]	11.35	9.26	0.74	8.13	0.46
	ST [1, 125]	4.13	4.01	0.75	3.47	0.25
Average		5.58	5.52	0.76	4.58	0.51

Table 4 Comparison results for $g=8$

Job	Setup time	No. net-NDS	IHMOPGA		MOPGA	
			No. NDS	Percent in the NET	No. NDS	Percent in the NET
$N=6$	ST [1, 50]	4.69	4.69	0.78	3.92	0.51
	ST [1, 100]	2.13	2.13	1	2.04	0.38
	ST [1, 125]	1	1	1	1	1
$N=30$	ST [1, 50]	5.74	5.53	0.56	5.17	0.44
	ST [1, 100]	4.27	4.67	0.68	3.95	0.32
	ST [1, 125]	6.64	6.47	0.83	4.13	0.17
$N=100$	ST [1, 50]	7.31	7.35	0.74	5.87	0.26
	ST [1, 100]	11.78	11.36	0.72	9.41	0.28
	ST [1, 125]	7.36	6.84	0.78	5.83	0.31
Average		5.66	5.56	0.79	4.59	0.41

solution in its determined iteration, then the IHMOPGA calls the Redirect procedure. This procedure changes the direction search, keeps elite individuals, and finally refreshes the population. These all help IHMOPGA to outperform MOPGA in larger number of generation.

As we said before, a common approach in solving multi-objective problems is transferring them into single-objective problems using constant weights. To compare our proposed algorithms against this approach (called weighted genetic algorithm, WGA), we implemented this approach too. For a fair comparison, we set the population size of this algorithm to 50 (equal to the population size of each sub-genetic algorithm in MOPGA and IHMOPGA), and we set the same parameters for WGA as the other two algorithms used. We considered the same set of weights for WGA as we used for other algorithms. Therefore, for each problem, we run 21 (equal to the number of weights) times WGA with these different weights. Then, we put together all the results and made the final set of non-dominated solution for WGA. The results indicate that WGA and MOPGA are equal in performance. In MOPGA, we indeed implement 21 sub-GAs with 21 different weights in parallel and run for just one time. On the other hand, in WGA, we have just one algorithm that it needs to run 21 times with 21 different weights in series. As illustrated in Fig. 6 as a sample instance, there is a tough competition between MOPGA and WGA, though implementing the MOPGA is very easier than WGA (remember we have to run WGA 21 times for each problem). Also, from this figure, it is easy to verify the better performance of IHMOPGA against the two other algorithms. As implementing the WGA is a tedious work and the performance of it is nearly equal to MOPGA, then we ignored this algorithm for the evaluation process.

5 Conclusion and future work

In this paper, we investigate the hybrid flow shop problems with unrelated parallel machines, sequence-dependent setup times, and blocking processor with two criteria: makespan and maximum tardiness. According to the good performance of genetic algorithms in multi-objective problems, we choose GA to introduce an effective algorithm using the population-based principle of GAs to tackle the complexity of these strongly NP-hard problems. We divided the whole population of GA into some groups, assigning each group a given weight and letting each group seek for its own single criterion, making MOPGA. To further search for non-dominated solutions and to overcome local optimums, we add a procedure called Redirect with an effective local search

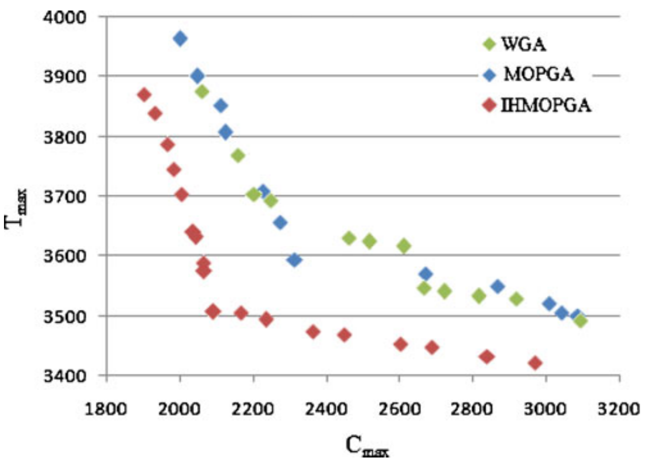

Fig. 6 Plot of obtained Pareto frontiers for WGA, MOPGA, and IHMOPGA for a sample instance (the maximum number of generations is set to 500)

step to our proposed algorithm, developing a new effective algorithm called IHMOPGA. We evaluate the performance of our proposed algorithms by generating random data. The computational results show that our proposed Redirect procedure combined with the local search step is thriving to extract and achieve better and more non-dominated solutions. We also compared our proposed algorithms against the commonly used approach constant weights method. The results indicate that our proposed IHMOPGA algorithm is superior to other WGA and MOPGA, which are nearly equal in performance. We also find out that the performance of IHMOPGA in comparison with MOPGA evidently increases over time.

As a direction for future research, it could be interesting to develop other meta-heuristic or heuristic algorithms and compare them with IHMOPGA. It also could be interesting to examine the performance of our proposed algorithm in other complex scheduling problems, such as job shop and open shop problems. Another clue for future research is the consideration of some other realistic assumptions such as limited buffers (non-zero), no-waiting environment, preceding constraints, and lag times. Another opportunity for research is the consideration of the problem with the other optimization objectives such as minimization of total completion time, total tardiness, early and tardy penalties, job waiting time variance, or even considering more than two criteria.

References

1. Pinedo M (2001) Scheduling: theory, algorithms, and systems, 2nd edn. Prentice-Hall, Englewood Cliffs
2. Thornton HW, Hunsucker JL (2004) A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage. Eur J Oper Res 152:96–114
3. Wang L, Liang Z, Da-Zhong Z (2006) An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. Comput Oper Res 33:2960–2971
4. Sawik T (2000) Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. Math Comput Model 31(13):39–52
5. Tavakkoli-Moghaddam R, Safaei N, Sasani F (2009) A memetic algorithm for the flexible flow line scheduling problem with processor blocking. Comput Oper Res 36(2):402–414
6. Quadt D, Kuhn H (2007) A taxonomy of flexible flow line scheduling procedures. Eur J Oper Res 178:686–698
7. Jungwattanakit J, Reodecha M, Chaovalitwongse P, Werner F (2008) Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. Int J Adv Manuf Technol 37:354–370
8. Behnamian J, Fatemi Ghomi SMT, Zandieh M (2009) A multi-phase covering Pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic. Expert Syst Appl 36(8):11057–11069. doi:10.1016/j.eswa.2009.02.080
9. Tavakkoli-Moghaddam R, Rahimi-Vahed A, Mirzaei AH (2007) A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives: weighted mean completion time and weighted mean tardiness. Inform Sciences 177 (22):5072–5090
10. Pasupathy T, Rajendran C, Suresh RK (2006) A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. Int J Adv Manuf Technol 27:804–815. doi:10.1007/s00170-004-2249-6
11. Varadharajan TK, Rajendran C (2005) A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. Eur J Oper Res 167(3):772–795
12. Chang PC, Chen SH, Lin KL (2005) Two phase subpopulation genetic algorithm for parallel machine scheduling problem. Expert Syst Appl 29(3):705–712
13. Chang PC, Chen SH, Liu CH (2007) Sub-population genetic algorithm with mining gene structures for multiobjective flowshop scheduling problems. Expert Syst Appl 33(3):762–771
14. Nagar A, Haddock J, Haragu S (1995) Multiple and bi-criteria scheduling: a literature review. Eur J Oper Res 81(1):88–104
15. Allahverdi A (2004) A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. Comput Oper Res 31(2):157–180
16. Gupta JND, Krüger K, Lauff V, Werner F, Sotskov YN (2002) Heuristics for hybrid flow shops with controllable processing times and assignable due dates. Comput Oper Res 29(10):1417–1439
17. Alisantoso D, Khoo LP, Jiang PY (2003) An immune algorithm approach to the scheduling of a flexible PCB flow shop. Int J Adv Manuf Technol 22(11–12):819–827
18. Lin H-T, Liao C-J (2003) A case study in a two-stage hybrid flow shop with setup time and dedicated machines. Int J Prod Econ 86 (2):133–143
19. Wang W, Hunsucker JL (2003) An evaluation of the CDS heuristic in flow shops with multiple processors. J Chin Inst Ind Eng 20(3):295–304
20. Deb K, Amrit P, Sameer A, Meyarivan T (2002) A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE T Evolut Comput 6(2):182–197
21. Zitzler E, Laumanns M, Bleuler S (2004) A tutorial on evolutionary multiobjective optimization. Proceedings of the Workshop on Multiple Objective Metaheuristics
22. Affenzeller M (2002) New generic hybrids based upon genetic algorithms. Lect Notes Comp Sci 2527:329–339
23. Lis J, Eiben AE (1997) A multi-sexual genetic algorithm for multicriteria optimization. Proceedings of the 4th IEEE Conference on Evolutionary Computation, pp 59–64
24. Cochran JK, Horng SM, Fowler JW (2003) A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. Comput Oper Res 30:1087–1102
25. Hu J, Goodman E, Seo K, Fan Z, Rosenberg R (2005) The hierarchical fair competition framework for sustainable evolutionary algorithms. Evolu Comput 13(2):241–277
26. Coello CA, Pulido GT, Lechuga MS (2004) Handling multiple objectives with particle swarm optimization. IEEE Trans Evol Comput 8(3):256–279
27. Mostaghim S, Teich J (2004) Covering Pareto-optimal fronts by subswarms in multi-objective particle swarm optimization. Evol Comput 2:1404–1411
28. Zitzler E, Laumanns M, Thiele L (2001) SPEA2: improving the strength Pareto evolutionary algorithm. Techni Rep Comp Eng Net Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland
29. Kurz ME, Askin RG (2004) Scheduling flexible flow lines with sequence-dependent setup times. Eur J Oper Res 159(1):66–82

30. Zandieh M, Fatemi Ghomi SMT, Moattar Husseini SM (2006) An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Appl Math Comput 180:111–127
31. Gholami M, Zandieh M, Alem-Tabriz A (2008) Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns. Int JAdv Manu Tech 42(1–2):189–201. doi:10.1007/s00170-008-1577-3
32. Naderi B, Zandieh M, Roshanaei V (2008) Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. Int J Adv Manu Tech 41(11–12):1186–1198. doi:10.1007/s00170-008-1569-3