Int J Adv Manuf Technol (2012) 58:309–325
DOI 10.1007/s00170-011-3368-5

ORIGINAL ARTICLE

Bi-objective hybrid flow shop scheduling with sequence-dependent setup times and limited buffers

Sina Hakimzadeh Abyaneh · M. Zandieh

Received: 3 September 2010 / Accepted: 25 April 2011 / Published online: 15 September 2011

Abstract The bi-objective hybrid flow shop problem with sequence-dependent setup times and limited buffers is mentioned in this paper. In this environment, there are limited buffer spaces between any two successive stages; thus, maybe there is not enough room for queues of jobs that are waiting in the system for their next operations. This problem is shown to be NP-hard in the strong sense. Up to now, some heuristic and metaheuristic approaches are proposed to minimize makespan or total tardiness of jobs. This paper presents several methods for optimization which consider two objectives simultaneously. The resolution of several specific instances from the open literature with the adaptations of non-dominated sorting genetic algorithm and sub-population genetic algorithm suggest that the proposed algorithms are effective and useful methods for solving this problem.

Keywords Bi-objective optimization · Hybrid flow shop scheduling · Limited buffer space · Sequence-dependent setup times

1 Introduction

Scheduling is one of the most important concerns in operation research. As a typical manufacturing and scheduling problem with strong industrial background, flow shop scheduling with limited buffers has appealed wide attention both in academic and engineering fields. Flow shop scheduling problem, or FSP, is a class of group shop scheduling problems in which the operations of every job must be processed on machines 1,2,…,m in this same order. A special case of FSP is permutation flow shop scheduling problem, in which the processing order of the jobs on the machines is the same for every machine. With the objective to minimize the total completion time (or makespan), such an issue is very hard to solve effectively due to the NP-hardness and the constraint on the intermediate buffer.

Different techniques have been presented for solving a flow shop optimization problem. Some of these are a simulation of natural events like ant colony optimization which is inspired by searching and foraging behavior of ants in real world. Genetic algorithm is another example that spots a primary population as initial solution and then use some operator inspired by genetic science to improve the answer. On the other hand, we could mention the simulated annealing that imitates the annealing process.

A hybrid flow shop model, which also referred to as flexible flow line or multiprocessor flow shop in the open literature, allows us to represent most of the industrial systems. The process industry such as chemical, powder making, refinery, pharmaceutical, oil, food, tobacco, textile, paper, and metallurgical industry can be modeled as a hybrid flow shop. A hybrid flow shop consists of a series of workshops, each of which has several machines in parallel. Some stages may have only one machine, but for the plant to be qualified as a hybrid flow shop, at least one stage must have several devices. These machines can be identical, uniform, or unrelated. Figure 1 shows the environment of hybrid flow shop with limited buffer spaces. Any job is processed at only one machine at each stage.

As stated by Pinedo [16], cited machine setup time is a significant factor for production scheduling in all flow patterns, and it may easily consume more than 20% of available machine capacity if not well handled. Also, the completion time of production and machine setups are influenced by production mix and production sequence. On

S. Hakimzadeh Abyaneh
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

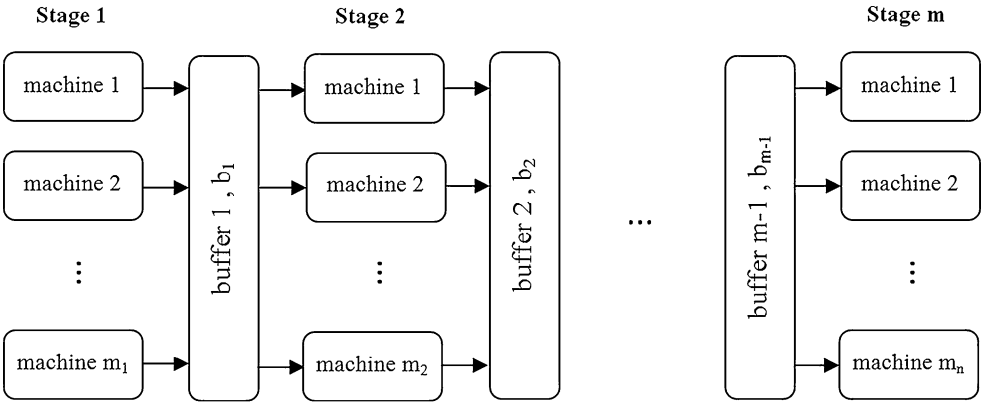

Fig. 1 The scheme of hybrid flow shop with limited buffer

the one hand, processing in large batches may increase machine utilization and reduce the total setup time. There is a tradeoff between flow time and machine utilization by selecting batch size and scheduling. Scheduling problems with sequence-dependent setup times are among the most difficult classes of scheduling problems. A one machine sequence-dependent setup scheduling problem is equivalent to a traveling salesman problem and is NP-hard [16]. Even for a small system, the complexity of this problem is beyond the reach of existing theories [12]. Sequence-dependent setup scheduling of a hybrid flow shop system is even more challenging. Also, limitation of buffer spaces increases this complexity extensively. Buffer capacity is an important issue in production lines such as the petrochemical processing industries and cell manufacturing. Most of the time, buffer is restricted or there is no buffer space between some stages due to limited room and storage facilities. So it could not be unlimited in real production processes, and if there is no efficient plan for job scheduling, this constraint may cause serious problems. Up to now, mathematical programming, constructive heuristics, and metaheuristics have been proposed for flow shop scheduling, where it is usually assumed that the buffer size between every two successive machines is infinite. So it is meaningful and functional to consider the case with limited buffers.

Moreover, if the multiple objectives of the combinatorial problems are considered, they become even more complex. A multiple objectives problem causes the single optimal solution to convert into a set of optimal solutions named Pareto optimal solutions. If there is no further information, one of these Pareto optimal solutions cannot be chosen as the best one. This demands a user to find as many Pareto optimal solutions as possible. Classical optimization methods like the multi-criterion decision-making methods suggest converting the multi-objective optimization problem to a single objective problem and try to find one optimal solution at each run. When such a method is used for finding multiple solutions, it has to be applied many times to find several different solutions after many simulations.

This study considers the problem of sequence-dependent setup time hybrid flow shop scheduling with limited buffer spaces and the objectives of minimizing the makespan and sum of the tardiness of jobs and presents two multi-phase methods to solve the problem. Finally, an extensive computational experience is carried out in order to analyze the different parameters of the algorithm. The results obtained from the computational study have shown the efficiency of the multi-phase algorithms. The key feature of these two algorithms is that they limit the search to the space of permutation vectors representing the order in which the given set of jobs is processed in the first stage. Naturally, such a vector does not specify the schedule by which the jobs are processed in subsequent stages. Thus, we need an appropriate constructive procedure for generating a complete and equal schedule associated with each permutation vector.

In this paper, adoptions of sub-population genetic algorithm II (SPGA II) and non-dominated sorting genetic algorithm II (NSGA II) are proposed to SDST hybrid flow shop problems with limited buffer capacities. In Section 2, we are going to reconsider some literatures of hybrid flow shop scheduling. Section 3 introduces the proposed adopted algorithms. Section 4 presents experimental design which compared the results achieved by proposed algorithms, and finally, Section 5 consists conclusions and future work.

2 Literature review

This section is going to reconsider some literatures of flow shop and hybrid flow shop scheduling. Here, some heuristic and metaheuristic approaches, which proposed to solve such these problems, are reviewed.

Among heuristic approaches, an algorithm which developed by McCormick et al. [13] and known as profile fitting tries to initially sequence jobs that lead to the minimum sum of idle times and blocking times on machines. Leisten [11] presented a more comprehensive approach and compares heuristics adapted from cases of no-wait, unlimited buffers, limited buffers, and two specially designed heuristics which attempt

to optimize the utilization of the available buffer storage. A heuristic for minimizing the steady-state cycle time presented by Abadi et al. [1] repetitively produces a minimal part set in an *m* machine blocking flow shop. The main idea is slowing down operations in order to make a connection between the no-wait flow shop, in which jobs do not wait between operations, and the blocking flow shop. In addition, this method might be practical to minimize the makespan. Caraffa et al. [3] used this idea to expand a genetic algorithm to minimize this measure.

Ronconi [17] proposed three constructive heuristics, which include a combination of the Profile Fitting heuristic and the enumeration procedure used by the NEH algorithm. Those suggested methods outperform the NEH algorithm in problems involving up to 500 jobs and 20 machines. More recently, Ronconi and Henriques [18] proposed FPDNEH constructive heuristic that explores the nonexistence of intermediate buffers, as well as the characteristics related to the tardiness criterion. They also suggested a greedy randomized adaptive search procedure (GRASP) metaheuristic, which is a multi-start and iterative process in which each iteration consists of two construction and local search phases, was also presented by them. The construction phase builds a feasible solution, whose neighborhood is explored until a local minimum is found during the local search phase.

As far as we know, few studies deal with limited buffer capacities in the flow shop environment. Nowicki [15] has developed a tabu search algorithm for this kind of problem in the special case where there is a single machine at each stage. Also, Wang et al. [22] proposed a hybrid genetic algorithm for flow shop scheduling with limited buffers. In the same context, Norman [14] also considers the problem with a sequence-dependent setup time for each job at each stage. Wittrock [24] and Sawik [19] are the only articles that directly addressing the FSPM/b (flow shop problem with limited buffer capacities). Wittrock [24] and after that Sawik [20] develop constructive procedures for solving this problem and solve six instances of the problem from a real production line.

In the literature, the notion of hybrid flow shop has emerged in the 1970s. Hung and Ching [7] addressed a scheduling problem taken from a label sticker manufacturing company which is a two-stage hybrid flow shop with the characteristics of sequence-dependent setup time at stage 1, dedicated machines at stage 2, and two due dates. The objective was to schedule 1 day's mix of label stickers through the shop such that the weighted maximal tardiness is minimized. They proposed a heuristic to find the near optimal schedule for the problem. The performance of the heuristic was evaluated comparing its solution with both the optimal solution for small sized problems and the solution obtained by the scheduling method used in the shop. Jolai et al. [8] tried to minimize tardiness, earliness, and maximize total profit, simultaneously. Davoudpour and Ashrafi [5] also solved multi-objective SDST flexible flow shop with GRASP algorithm. Behnamian et al. [2] suggested A multi-phase covering Pareto optimal front method to multi-objective scheduling in a realistic hybrid flow shop using a hybrid metaheuristic. For this reason, they proposed a new metaheuristic which combines simulated annealing and variable neighborhood search. The first phase concentrated on specific search space and prevented all individuals from being converted into a local optimal by means of decomposing the population into several sub-populations. Then, in order to improve non-dominated solutions and explore Pareto solution space found in the first phase, in second phase, sub-population was reunified as a single big population. The basic idea of phase 3 was developing a strategy that explores the Pareto space to improve previous phase solutions and finds a pathway that filled the gap between the disconnected pieces of the Pareto front using a novel e-constraint covering method.

In most real-world cases, setup times depend on both jobs, which is separable from processing. Up to now, there are few works published which address heuristics for flexible flow lines with sequence-dependent setup times. Yaurima et al. [25] proposed a genetic algorithm for hybrid flow shop with sequence-dependent setup time, unrelated parallel machines, machine availability, and limited buffer constraints together inside the same problem formulation, and they considered the production environment of a television assembly line for inserting electronic components. Kurz and Askin [9, 10] examined scheduling rules for SDST flexible flow lines. They explored three classes of heuristics. The first class of heuristics (cyclic heuristics) is based on simplistic assignment of jobs to machines with little or no regard for the setup times. The second class of heuristics is based on the insertion heuristic for the traveling salesman problem. The third class of heuristics is based on Johnson's rule. They proposed eight heuristics and compared the performance of those on a set of test problems. Moreover, they formulated the SDST flexible flow lines as an integer programming model. Because of the difficulty in solving the IP model directly, they developed a random keys genetic algorithm. Recently, Zandieh et al. [26] proposed an immune algorithm for hybrid flow shops scheduling with sequence-dependent setup times. This IA approach incorporates an accelerating mechanism as well as a restraining mechanism to assist the search for a near optimal solution. The accelerating mechanism redeploys the best solutions found so far in the subsequent generation. By doing so, the search for near optimal solutions can possibly converge rapidly.

In this paper, we use two adapted algorithms, which were inspired by SPGA II and NSGA II, for minimizing total tardiness and completion time of jobs in a hybrid flow shop with limited buffer capacities. After that, we will evaluate the performance of the proposed method with several different criteria.

3 Multi-objective evolutionary algorithms

3.1 Solution representation

For exploration a promising neighborhood, we carry out the search in the space of the permutation vectors that represent the order in which the given set of jobs are performed in the first stage. Also, an effective procedure is designed by Wardono and Fathi [23] to construct a complete schedule associated with a given permutation vector. This section began by debate about the concept of solution representation and the search space. Then, we introduce the procedure H_1 for constructing a complete schedule for a given permutation vector.

First of all, an appropriate format must be determined to represent a solution. There are two different formats that can be used for this problem, namely the matrix representation and the vector representation. Using the matrix representation, a solution is shown by a pair of $mN \times L$ matrices I and X whose components are defined below:

I_{jl}: the machine number to which job j is assigned at stage l
X_{jl}: the position of job j within the sequence of jobs that are assigned to machine I_{jl} at stage l

The major benefit of this representation is that it covers the whole solution space, i.e., there exists a unique pair of matrices (I, X) associated with every solution for the problem. In the context of a local search, however, there are two disadvantages associated with this format. The first weakness is that it is simply cumbersome to work with. See Wardono and Fathi [23] for an example of using this format in the context of the FSPM/b problem. This leads to subsequent difficulties in designing and implementing various features of the local search algorithm such as the neighborhood structure and the exploration and exploitations strategies in an effective manner. The second disadvantage of using this format, which is specific to the FSPM/b problem, is that it is computationally expensive to determine if a given solution in this format is feasible for the problem, i.e., whether or not the available buffer space is enough to implement the given schedule. Since we need to make this determination for every neighboring solution that we consider at each iteration, this is a particularly troublesome issue for the neighborhood search algorithm.

The alternative approach is to restrict the search to a subset of the solution space that can be represented in a simpler format for the local search. Ideally, this subset should contain the collection of all potentially good solutions for the problem. Apparently, it is complicated to recognize such a subset of the solution space in an exact and efficient manner. In the algorithms that we propose in this article, we restrict the search to the collection of solutions that are associated with the set of permutation vectors of size N. Each permutation vector S represents the order in which the given set of jobs is processed in the first stage. Such a vector does not indicate a complete schedule by itself. In a complete schedule, the jobs sequence at each stage and also the machine number which perform each job must be determined. Thus, we need to devise an associated procedure for constructing a complete and feasible schedule for any given vector S. The resulting schedule is expected to be related with the vector S. Naturally, the overall success of the local search algorithm strongly depends on the efficiency of this procedure to construct potentially good solutions for the given permutation vectors. Wardono and Fathi [23] showed that this approach results in a complete answer for the FSPM/b problem. Certainly, we use the matrix representation to show a complete schedule that obtained by H_1 procedure whenever we need to. In this article, we employ a similar format for solution representation to design a local search algorithm for the FSPM/b problem.

As mentioned above, in this paper, an adaption of procedure H_1 proposed by Wardono and Fathi [23] is employed to construct a complete schedule for the FSPM/b problem associated with any given permutation vector S. We only add a part to put setup times into calculations. According to this procedure, for any given permutation vector S, the corresponding job schedule at the first stage is obtained by using the first available machine (FAM) rule. For each of the subsequent stages, the jobs are first sequenced in the increasing order of their completion time at the previous stage and then assigned to the machines at the current stage according to the FAM rule. However, because of the limited buffer capacities and setup times, the starting time of a job on a machine at a given stage is not necessarily equal to either its completion time at the previous stage or the completion time of the previous job on the machine at this stage. In procedure H_1, when a machine start to process job j in stage l, the completion time of the job in this stage will be computed and added to event list. Because of SDST, we have to add setup time of the machine from its previous operation to job j while computing completion time (see [23] for more information).

3.2 Sub-population genetic algorithm II approach

This section concentrates on a multi-phase method named SPGA II which is proposed by Chang and Chen [4]. In the first phase, the population will be separated into several sub-populations in which each of them is designed for a scalar multi-objective. Multiple objectives are combined with min–max method, and then each sub-population evolves separately to obtain a good approximation of the Pareto front. In the second phase, for improvement of the Pareto front, non-dominant solutions will be unified as Pareto archive which is the initial solution for phase 2. Pioneering efforts [4] have shown that sub-population genetic algorithm is effective in solving the multi-objective combinatorial problems. Based on these previous researches, this paper adopts the SPGA

algorithm with a global Pareto archive technique and a two-stage approach to solve the multi-objective problems in a hybrid flow shop environment with limited buffer spaces and sequence-dependent setup times. In the first stage, the areas next to the two single objectives are searched, and solutions which explored around these two extreme areas are reserved in the global archive for later evolutions. Then, in the second stage, larger searching areas except the middle area are further extended to explore the solution space in finding the near optimal frontiers. There are two main characteristics of the sub-population-like method:

1. Numerous sub-populations explore the entire solution space
2. The multiple objectives are combined into a single objective for each sub-population

Because the sub-populations are planned to explore specific region, the Pareto optimal solutions are scattered uniformly over the frontier. If the sub-populations communicated to each other while they were exploring the different solution spaces, it might bring better convergence and diversity. Therefore, we employ the global Pareto archive concept to let each sub-population communicate with others during the algorithm's implementation. Another important thing about this procedure is the two-stage approach that can accelerate the convergence and improve the diversity of the solution quality at each different stages. The detailed procedure is explained in the following:

Algorithm 1: The main procedure of SPGA II
Parameters setting
ns : the total number of sub-populations
np : number of individuals in each sub-population
index : the number of sub-populations in stage 1
runtime, *crossover _ rate* and *mutation _ rate*
Initialize
Divide population
Assign weight to each objective
Stage 1: SPGA(1, *index* / 2, *ns* - *index* / 2, *ns*, *np*)
Initiate sub-populations (*index*, *ns*, *np*)
Stage 2: SPGA(1+ *index* / 2, *ns* - 1- *index* / 2, 0, 0, *np*)

Algorithm 2: SPGA (*start*1, *end*1, *start*2, *end*2)
$timer = 0$
while $timer < runtime/2$ **do**
  for $i = start1$ to $end1$ and $j = start2$ to $end2$ **do**
    Evaluate objectives and fitness function with H_1 procedure
    Update Pareto archive
    Selection with elitism strategy
    Two point crossover
    Moving position mutation
  end for
end while

The procedure *Initialize* generates a random set of chromosomes according to the population size at the beginning and determines the primary Pareto set. The procedure *Divide population* is to divide the original population into*ns* sub-populations. The procedure *Assign weight to each objective* is to assign different weight values to each sub-population and also the individuals in the same sub-population will share the same weight value. Since the problem is a bi-objective problem, the vector size is set to two. These weights should be changed gradually. In this research, combination of weight vector is defined by:

$$(w_1(t),\ w_2(t)) = (|\sin(2\pi t/R)|,\ 1 - w_1(t))$$

where t is the tth sub-population, R=200 and it is obvious that $0 \le w_1, w_2 \le 1$.

After the weight assignment, the corresponding scalarized objective value of the $f_1(x)$ and $f_2(x)$ objectives in sub-populations should be determined. For this reason, we mixed the min–max method for multi-objective problem with the weighting and because of the difference between scales of the two objectives, the values need to be normalized into a single unit interval. Consequently, the objective function for the tth sub-population is formulated as follows:

$$\left[w_1(t)\left(\frac{f_1(x)-f_1^*}{f_1^w-f_1^*}\right) + w_2(t)\left(\frac{f_2(x)-f_2^*}{f_2^*}\right)\right]$$

where f_1^* and f_2^* denote the individual minimum of each respective total tardiness and completion time of jobs and f_1^w is the worst (maximum) value of the first objective function. Notice that all instances are solved using 10 different seeds for each algorithm and the minimum solution in all runs are used as f^* and f^w for each objective.

The Elitism strategy adopted at the first stage randomly selects a number of individuals from non-dominant set into the mating pool. The tournament selection is employed at the selection operation. In this study, we also use ordinary two point crossover and moving position mutation which illustrated in Fig. 2.

When the first stage is finished, the algorithm initiates other sub-populations by tournament selection from all existing solutions generated at the first stage. So the fitness of these solutions will be recalculated according to the weight vector of each corresponding sub-population and the bests will be selected for that particular sub-population. In this phase, initial solutions for the sub-populations will be generated by selecting those better ones from those existing solutions, which will accelerate the convergent progression. Thus, those selected solutions will be further evolved during second stage. The pseudo code for initiating sub-populations for second stage is mentioned below:

Algorithm 3: Initiate sub-populations (*start*1, *end*1, *start*2, *end*2)

```
for i = 1  index/2 to  ns - 1 - index/2 do
        for j = 1 to np do
                solution = Tournament selection (np)
                Set solution (i, j, solution)
        end for
end for
```

This approach enables the system to recombine different species from different sub-populations, so more diversified solutions with better quality are produced. Another advantage of this approach is to exchange the information from existing sub-populations.

We have to inform that in all figures which illustrate final population and non-dominant members as Fig. 3, horizontal and vertical axes are shown completion time and total tardiness of jobs, respectively.

3.3 Non-dominated sorting genetic algorithm II approach

In this section, we are going to discuss a multi-phase method named NSGA II which suggested by Deb et al. [6]. Here, we adapt NSGA II by the environment considered in this study and employ it to resolve some instances. Simulation results show the efficiency of this method for this special case. To start with, a random set of chromosomes according to the population size will be generated. In order to spot solutions of the first non-dominant front, each solution can be compared with whole population based on their objective values which obtained by procedure H_1 [23]. In this way, it will be disclosed if any solution dominates entire population. Consequently, all individuals in the first non-dominant front are found, and this front is established. In order to identify the next non-dominant front, the solutions of the first front are temporarily removed and the above procedure is repeated. This action is continued for third and higher levels of non-domination. As we can see, it is a complex and time-consuming procedure for finding all non-dominant fronts. Therefore, we are going to use a fast non-dominated sorting approach proposed by Deb et al. [6] and described below.

First of all, for each solution, two entities must be calculated:

1. n_p: domination count, the number of solutions which dominate the solution p
2. S_p: set of solutions that the solution p dominates

All solutions with domination count equal to zero belong to the first non-dominant front named Pareto front. Now, for each solution with n_p=0, we check each member q of its

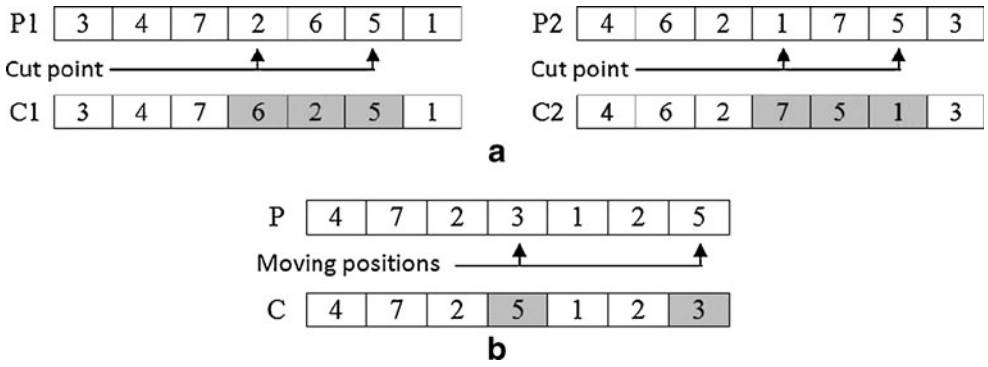

Fig. 2 Representation of two point crossover (**a**) and moving position mutation (**b**)

set S_p and reduce its domination count by one. In doing so, if the domination count becomes zero for any member, we put it in a separate list Q. These members belong to the second non-dominant front. Now, the above procedure is continued with each member of Q and the third front is identified. This process is true for finding all fronts.

Algorithm 4: Non-domination sort

for each $p \in P$ **do**
$S_p = \varnothing$
$n_p = 0$
for each $q \in P$ **do**
if $p \succ\succ q$ **then** $S_p = S_p \cup \{q\}$ ($p \succ\succ q$ i.e. p dominate q)
else if $q \succ\succ p$ **then** $n_p = n_p + 1$
end if
end for
if $n_p = 0$ **then**
$p_{rank} = 1$
$F_1 = F_1 \cup \{p\}$
end if
$i = 1$
while $F_i \neq \varnothing$ **do**
$Q = \varnothing$
for each $p \in F_i$ **do**
for $q \in S_p$ **do**
$n_p = n_p - 1$
if $n_p = 0$ **then**
$q_{rank} = i + 1$
$Q = Q \cup \{q\}$
end if
end for
end for
$i = i + 1$
$F_i = Q$

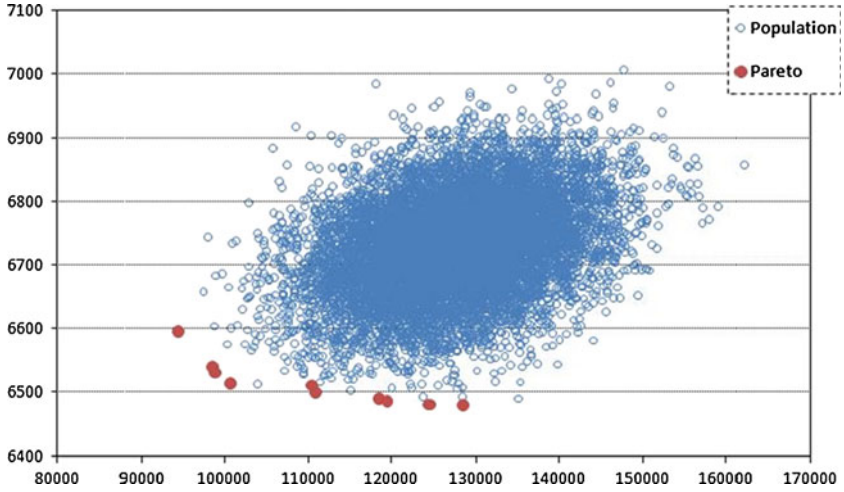

Fig. 3 Final population and Pareto archive of SPGA II algorithm

According to some empirical results, it is hard to obtain a widespread solution for multi-objective combinatorial problems. Along with convergence to the Pareto optimal set, it is also desired that an evolutionary algorithm maintains a good spread of solutions in the obtained set of solutions. In the NSGA II, crowded comparison approach is proposed to maintain sustainable diversity in the population. To describe this approach, we first define density estimation metric and then introduce the crowded comparison operator.

Density estimation To get an estimation of the density of solutions surrounding a particular solution in the population, we calculate the average distance of two points on either side of this point along each of the objectives. This quantity which called crowding distance, serves as an approximation of the perimeter of the cuboid formed by using the nearest neighbors as the vertices. Figure 4 illustrates the computation of crowding distance for a bi-objective problem. In Fig. 4, the crowding distance of the *i*th solution in its front (Points marked with bold circles are solutions of the same non-dominant front) is the average side length of the cuboid (shown with a dashed box).

The crowding distance computation needs sorting the population according to each objective function value in ascending order of quantity. Thereafter, for each objective function, the best solutions are assigned an infinite distance value. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. This calculation is continued with other objective functions. The overall crowding distance value is calculated as the sum of individual distance values corresponding to any objective. Each objective function is normalized before calculating the crowding distance. A solution with a smaller value of this distance measure is, in some sense, more crowded by other solutions. Crowding distance measure for front F_t is computed as below:

Algorithm 5: Algorithm 4: Crowding distance assignment

for each objective j **do** (total tardiness and completion time of jobs)

$l = |F_t|$

$F_s = Sort(F_t, j)$

$dis_1 = dis_l = \infty$

for $i = 2$ **to** $|F_t| - 1$

$$dis_i = dis_i + [F_s(i+1) - F_s(i-1)]/(F_s^{\max} - F_s^{\min})$$

end if

end for

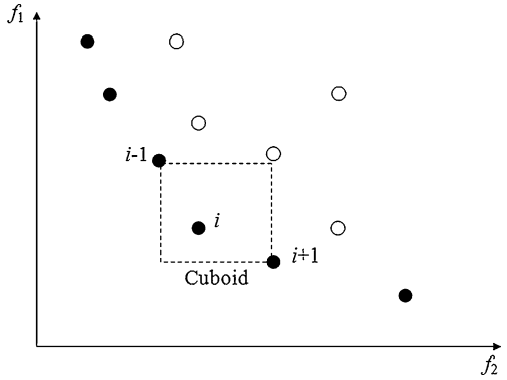

Fig. 4 Crowding distance illustration

Crowded comparison operator The crowded comparison operator ($<_n$) guides the selection process at the different stages of the algorithm in the direction of a uniformly scattered Pareto optimal front. Any individual in the population has two features:

1. Non-domination rank (i_{rank})
2. Crowding distance (i_{distance})

We now define a partial order $<_n$ as

$$i <_n j \text{ if } (i_{\text{rank}} < j_{\text{rank}}) \text{ or } ((i_{\text{rank}} = j_{\text{rank}}) \text{ and } (i_{\text{distance}} < j_{\text{distance}}))$$

It means that between two solutions with differing non-domination ranks, we prefer the solution with the lower rank. Otherwise, if both solutions belong to the same front, then the solution located in a lesser crowded region is preferred (Fig. 5).

Most of the time, a random initial population P_0 is created and sorted based on the non-domination in several independent groups. So each solution is assigned a rank equal to its non-domination level. At first, the two-point crossover and moving position mutation operators (Fig. 2) are used to create an offspring population Q_0 of size N. Now a combined population $R_0=P_0 \cup Q_0$ is established which is of size $2N$. Then the chromosomes of this mixed population R_0 is sorted according to non-domination level. Since R_0 includes all previous and current population members, elitism is ensured. Now, solutions belonging to the best non-dominant set F_1 are of best solutions in the combined population and must be emphasized more than any other solution in the combined population. If the size of F_1 is smaller than N, all members of this set have to be chosen for the new generation P_1. The remaining members of this population are chosen from subsequent non-dominant fronts in the order of their ranking. Thus, this is followed by solutions from the sets F_2, F_3, and so on. This procedure is continued until no more sets can be accommodated. Suppose that the count of solutions in all sets from F_1 to F_t would be larger than the population size N. To choose a population of size N, we sort the solutions of the last front F_t using the crowded comparison operator $<_n$ in descending order and choose the best solutions needed to fill all population slots. This procedure will be resumed for next generations.

In this method, we put a specific time for stopping criterion due to some empirical experiences. Whenever this condition is satisfied, the algorithm must be finished.

3.4 Local searching structure

Pareto archive which obtained by the main algorithms is the initial solution for this phase. For any permutation vector v, a vector v' is supposed to be in the neighborhood H of v (denoted as $H(v)$) if it is obtained from v by changing the position of job x and job y ($y \neq x$) in vector v. We choose these positions randomly among all jobs. The size of this neighborhood is $N(N-1)/2$. At the start of this neighborhood

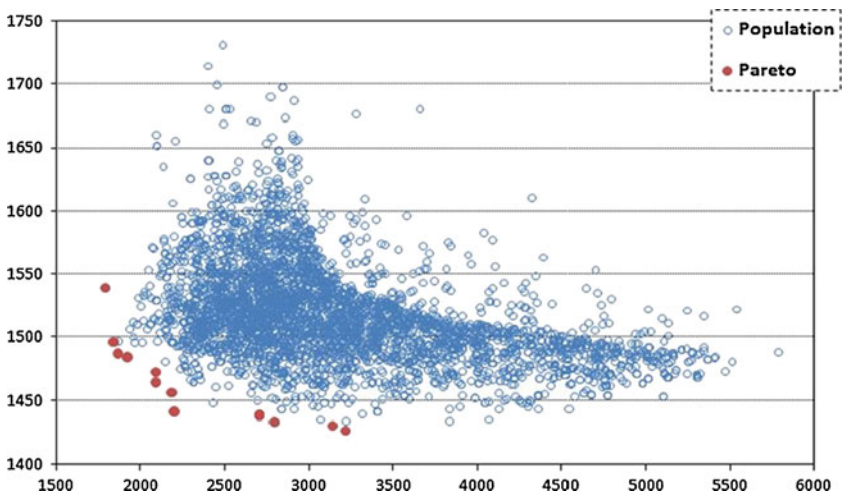

Fig. 5 Final population with Pareto front of NSGA II algorithm

exploration, the set of solutions obtained from main algorithms are considered as a promising area. During the implementation, if the local search algorithm achieved a better solution, the new solution would enter in the Pareto set, and it should be regarded as an updated promising area. In the continuance of this investigation, we will find that the quality of the solutions obtained via this local search is significantly improved. Also, the time needed for this search is less than $\frac{1}{3}$ of the time which consumed by the main algorithms.

In this study, after the usage of SPGA II and NSGA II methods until a specific time, the efficient local search algorithm based on moving position mutation operation (as represented in Fig. 2(b)) will be utilized to concentrate computing effort on exploring promising neighbor solutions. We named the combined algorithms as non-dominated sorting memetic algorithm (NSMA) and sub-population memetic algorithm (SPMA), respectively. The certain number of iteration could be a good stopping criterion for this search. Best improving and first improving are considered as two commonly used search techniques. In the best improving strategy, the entire neighborhood is analyzed and the current solution is replaced by the best one, while in the first improving strategy, the current solution is replaced by the first solution which improves its value. But none of these strategies are useful here. Due to presence of Pareto set, it is not rational to compare a new solution with its parent. Consequently, whenever mutation operator results in a new solution, we have to judge it against all Pareto archive members. If none of them dominates this solution, it should be entered the Pareto archive. Afterward, the old Pareto members have to be checked if the new member dominates them. If so, these old chromosomes must be removed from the Pareto front. The pseudo code for local search algorithm is given as follows:

Algorithm 6: Local search

for $i=1$ **to** $iteration\_number$ **do**
  $p = Random\ select(Pareto)$
  $q = mutate(p)$
  for each $p \in Pareto$ **do**
    if $p \succ\succ q$ **then**
      $k = 0$
      break for
    end if
  end for
  if $k=1$ **then** $Pareto = Pareto \bigcup \{q\}$ **end of**
  for $p \in Pareto$ **do**
    if $q \succ\succ p$ **then** $Pareto = Pareto - \{p\}$ **end if**
  end for
end for

The algorithm will be terminated whenever the total number of iterations reaches a specified value.

4 Experimental design

In this section, we are going to compare the results of different algorithms in a hybrid flow shop with limited buffer spaces and SDST constraint. All algorithms were implemented in MATLAB 7.6.0.324 (R2008a) and ran on a PC with a Pentium dual core 3.60 GHz processor with 2 GB of RAM and Microsoft Windows Vista environment.

4.1 Data generation and settings

In this part, we will test the performance of the proposed methods with several experiments by different sizes. These instances must include the number of jobs, number of stages, number of machines in each stage, range of processing times, the range of sequence-dependent setup

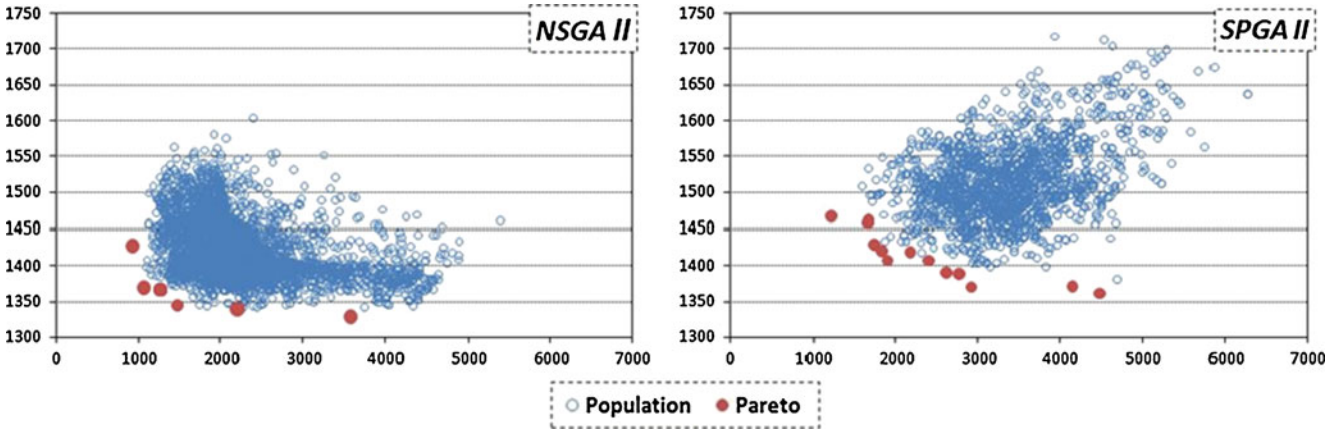

Fig. 6 Comparison of final population and Pareto set of *SPGA II* and *NSGA II* algorithms for the same problem

times, and the range of buffer capacities. Test problems were generated according to Taillard [21]. These instances are commonly used in the literatures. All instances are available at http://www.prd.usp.br/docentes/debora/. Ten different matrices of processing times were generated for each of the 12 size shown in Tables 2 and 3. For each of those matrices, four scenarios were built. But we have chosen the first processing times matrix and first scenario to test algorithms efficiency. Due to preventing applying remote answers, we ran each simulation for three times and employ the average of them as the main answer.

The setup times for the first stage are set to 0 for all jobs. However, in other stages, setup times are uniformly distributed from 0 to 25. Buffer capacities are other important things that we have to set. The important point is systems with larger buffer rooms tend to unlimited buffer problems. Because of this fact, it is better that the buffer spaces between successive stages be less than the number of jobs. Thus, examples with small size of buffer are recommended for our particular case. So we distribute buffer size uniformly from 0 to 4. Also, in general, all combinations of these levels will be tested. In addition, we have to mention some further restrictions. The variable machine distribution factor requires that at least one stage have more than one machine. Also, the largest number of machines in a stage must be less than the number of jobs.

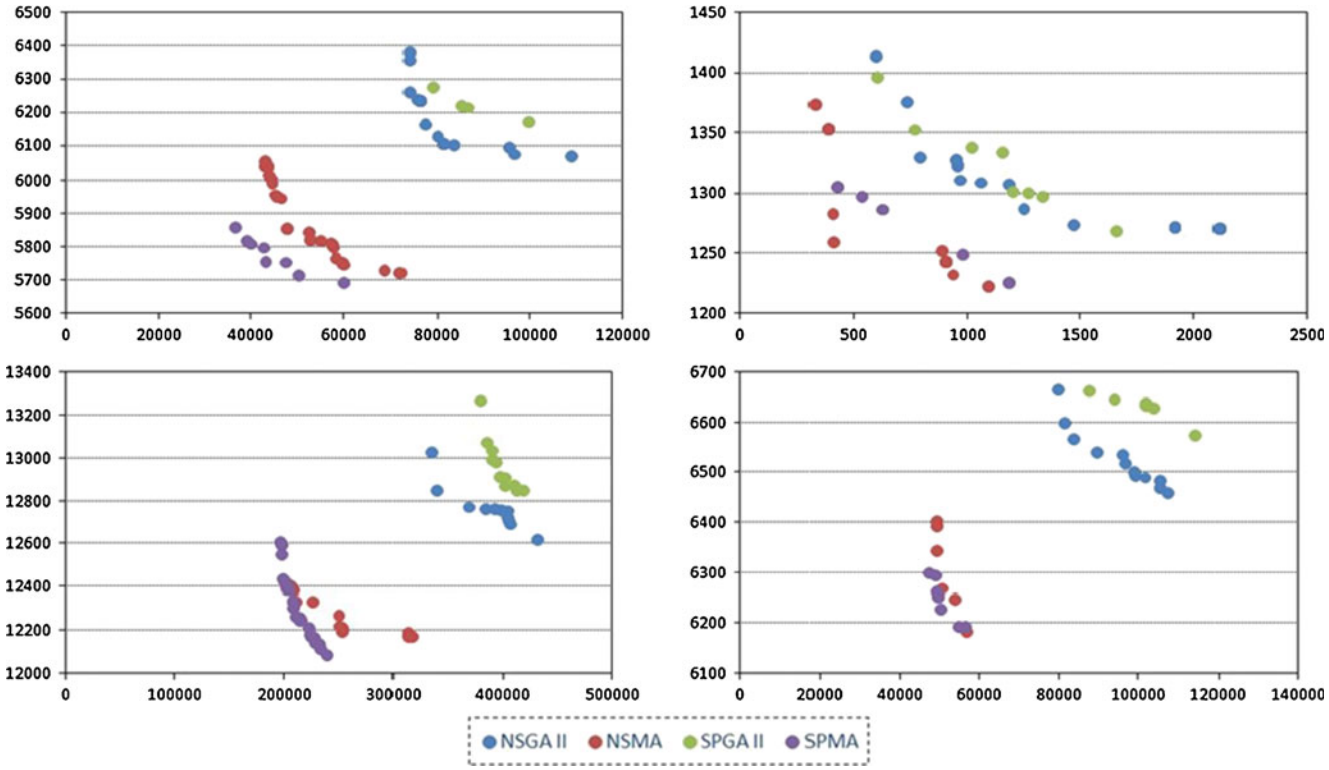

Fig. 7 Comparison of Pareto archives for *SPGA II*, *NSGA II*, *SPMA*, and *NSMA* for four different instances

Table 1 Mean evaluation metrics for each algorithm

Evaluation metrics	Algorithms			
	NSGA II	NSMA	SPGA II	SPMA
Computational time (s)	5171.5	1684.6	5296.0	1666.2
N	10.6	18.1	6.7	15.5
Spacing	1862.9	649.8	3080.6	874.2
MID	153065.5	126928.1	147732.8	123793.9
RAS	1.08	0.58	1.57	0.52
CS	0%	78.3%	0%	87.5%

The number of machines in each stage is randomly chosen between 1 and 5.

4.2 Evaluation metric

Most of the multi-objective optimization methods approximate the Pareto optimal front by a set of non-dominated solutions. It is important decision that how to evaluate the quality of these solutions, because the conflicting and incommensurable nature of some of the criteria makes this process more complex. In general, comparing the solutions of two different Pareto approximations coming from two

Table 2 Mean evaluation metrics for small sizes of problems in three runs

Problem size (Job×Stage)	Evaluation metrics	Algorithms			
		NSGA II	NSMA	SPGA II	SPMA
20×5	Computational time (s)	78.3	22.5	81.8	23.1
	N	20.3	42.7	7.3	29.7
	Spacing	89.8	34.6	147.8	47.9
	MID	1900.2	1532.9	1590.7	1605.7
	RAS	0.02	0.08	0.03	0.05
	CS	0%	38.3%	0%	10.3%
20×10	Computational time (s)	108.8	43.9	112.7	43.6
	N	7.0	29.7	6.3	17.3
	Spacing	309.5	20.8	389.0	47.8
	MID	2708.4	2210.7	2371.0	2237.2
	RAS	0.05	0.01	0.05	0.01
	CS	0%	42.9%	0%	61.2%
20×20	Computational time (s)	824.0	85.8	831.1	86.7
	N	9.3	13.7	5.0	6.3
	Spacing	197.2	79.6	377.0	266.5
	MID	6975.8	6358.6	6870.9	6438.7
	RAS	0.20	0.12	0.48	0.27
	CS	0%	24.4%	0%	90.3%
50×5	Computational time (s)	752.3	146.1	773.9	151.8
	N	11.0	19.0	7.3	21.3
	Spacing	294.0	148.8	1159.7	146.1
	MID	6776.9	5174.5	6712.2	4818.9
	RAS	0.13	0.02	0.20	0.02
	CS	0%	84.3%	0%	100%
50×10	Computational time (s)	908.3	268.8	953.6	269.3
	N	9.7	13.3	7.3	6.3
	Spacing	1151.6	127.3	1012.1	464.9
	MID	13529.5	9084.2	13081.7	11512.6
	RAS	0.32	0.12	0.65	0.44
	CS	0%	87.5%	0%	100%
50×20	Computational time (s)	1457.3	567.5	1495.4	570.6
	N	5.3	22.0	7.7	21.7
	Spacing	480.7	115.8	739.8	145.0
	MID	12595.8	9535.7	12209.5	9909.1
	RAS	0.56	0.06	0.42	0.08
	CS	0%	86.4%	0%	100%

Table 3 Mean evaluation metrics for large sizes of problems in three runs

Problem size	Evaluation metrics	Algorithms			
(Job×Stage)		NSGA II	NSMA	SPGA II	SPMA
100×5	Computational time (s)	4714.6	565.5	4772.5	597.3
	N	8.3	13.7	7.7	11.7
	Spacing	1031.5	511.3	1801.7	511.2
	MID	23673.5	8255.7	18311.4	11284.3
	RAS	0.42	0.07	0.45	0.04
	CS	0%	100%	0%	90.5%
100×10	Computational time (s)	5973.5	1177.1	6006.7	1198.7
	N	10.0	13.3	5.0	17.3
	Spacing	1090.0	378.9	3245.8	323.9
	MID	39312.5	22503.7	30686.6	23066.8
	RAS	0.94	0.16	1.61	0.14
	CS	0%	100%	0%	100%
100×20	Computational time (s)	7287.3	2242.4	7812.3	2248.3
	N	7.0	13.0	7.0	19.7
	Spacing	3538.4	780.5	1206.2	340.6
	MID	51131.3	40266.9	52896.2	39507.0
	RAS	1.07	0.43	1.82	0.23
	CS	0%	88.9%	0%	100%
200×10	Computational time (s)	7578.0	3424.3	7687.5	3711.4
	N	7.3	12.7	5.3	10.3
	Spacing	8535.7	800.0	2193.2	532.4
	MID	143730.2	91874.7	130701.3	88647.4
	RAS	3.18	0.55	3.76	0.87
	CS	0%	100%	0%	100%
200×20	Computational time (s)	11558.4	4613.4	11914.3	4356.6
	N	23.0	15.7	7.7	12.3
	Spacing	995.3	1234.4	8818.7	1599.4
	MID	246472.00	225078.4	236651.6	222150.4
	RAS	1.14	0.88	2.40	1.21
	CS	0%	100%	0%	100%
500×20	Computational time (s)	20816.7	7057.8	21110.7	6736.7
	N	8.7	8.3	6.7	12.3
	Spacing	4640.9	3565.9	15875.7	6064.7
	MID	1287978.9	1101260.9	1260710.1	1064348.3
	RAS	4.97	4.43	7.04	2.86
	CS	0%	86.7%	0%	97.2%

algorithms is not straightforward. In the 1990s, the quality of solutions to the Pareto optimal front (in the objective space) is being evaluated by visual inspection. This kind of evaluations is possible for two- and three-objective test functions. However, in general, the quality of the approximated sets must be measured by a quantitative metric. For this reason, in this paper, we proposed four indices as follows:

- *Spacing:* This metric allows us to measure the uniformity of the spread of the points in the solution set.
- Mean ideal distance (*MID*): The closeness between Pareto solution and ideal point (0, 0)
- *RAS*: The rate of achievement to two objectives simultaneously
- *CS*: dominance percentage of on solution compare with another one

The loss of diversity may mean premature of evolution algorithm. According to some empirical results, it is hard to obtain a widespread solution for multi-objective combinatorial problems. For this purpose, we will compute *Spacing* metric for algorithms as below:

$$\text{Spacing} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(d_i - \overline{d}\right)^2}$$

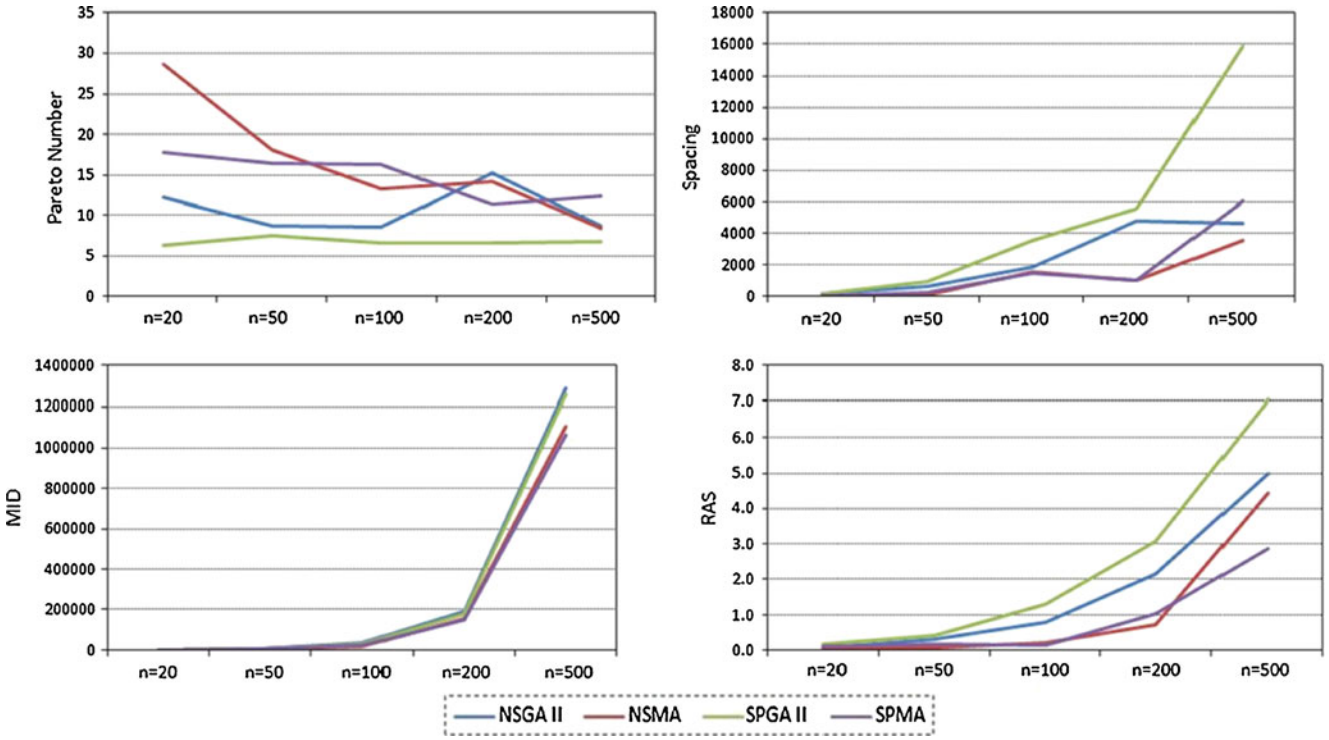

Fig. 8 Interaction between evaluation metrics and number of jobs for different types of algorithms

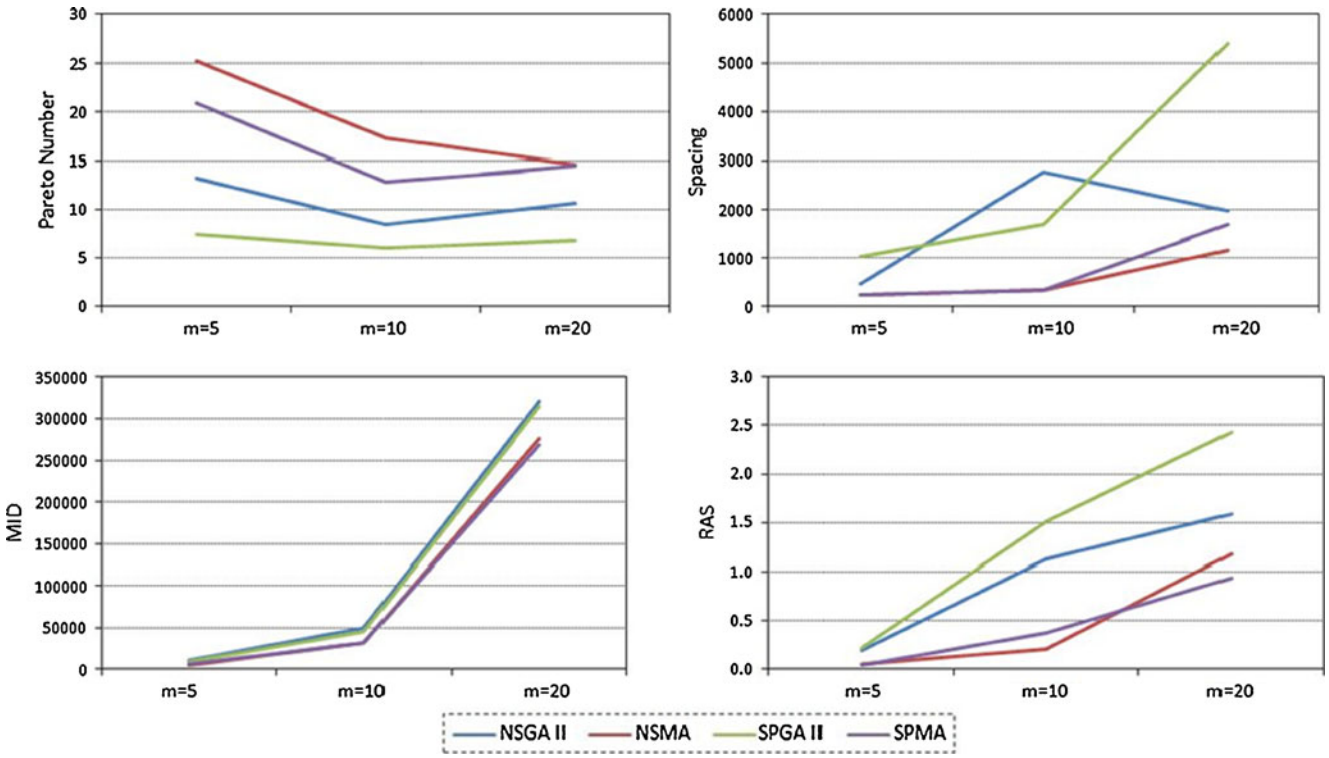

Fig. 9 Interaction between evaluation metrics and number of working stages for different type of algorithms

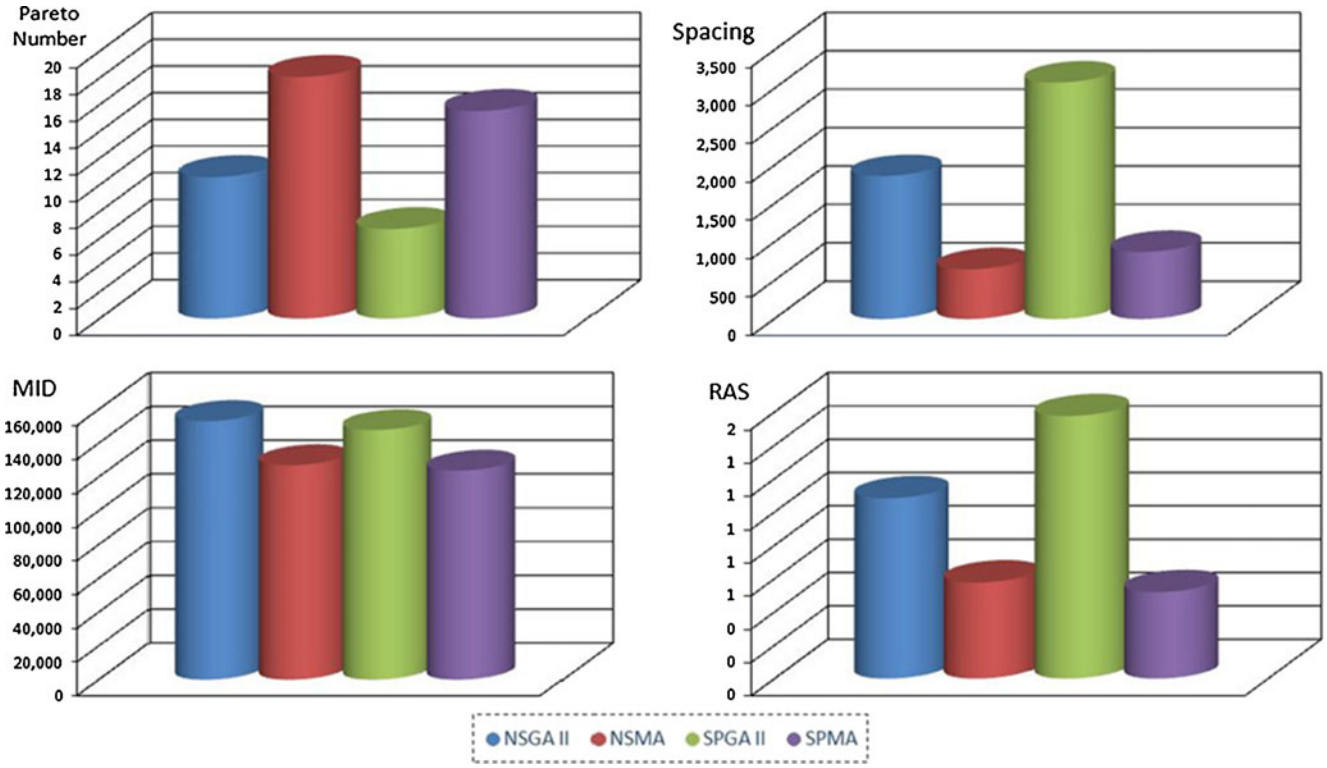

Fig. 10 Mean evaluation metrics for all algorithms

In which n is Pareto solutions number, d_i is define as

$$d_i = \min_j \left\{ \left| f_1^i(x) - f_1^j(x) \right| + \left| f_2^i(x) - f_2^j(x) \right| \right\};\ i, j = 1, 2, \ldots, n$$

And $\overline{d}$ is mean of d_i vectors. Also *MID* is defined as follows:

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

where n is the number of non-dominant solution and $c_i = \sqrt{f_{1i}^2 + f_{1i}^2}$. The lower value of *MID*, the better of solution quality we have. Another evaluation metric which we proposed in this paper is *RAS*. The equation of *RAS* is represented as following equation:

$$\text{RAS} = \frac{\sum_{i=1}^{n} \left\{ \left(\frac{f_{1i} - F_i}{F_i} \right) + \left(\frac{f_{2i} - F_i}{F_i} \right) \right\}}{n}$$

Where F_i=min $\{f_{1i}, f_{2i}\}$. The solution with lower value of *RAS* has better quality. Also, in all measures assumes positive objective values. Therefore, solutions with negative objectives must be transformed into the positive part of the axis. Another important criterion is *CS* which presents the percentage of the solutions number in a Pareto set which dominate the members of another Pareto set. If we suppose X and X' are two Pareto set, *CS* will be calculated with equation below:

$$\text{CS}(X, X') = \frac{\left| a \in X;\ \forall a' \in X' : a \succ\succ a' \right|}{|X|}$$

If CS=1, it means that X completely dominates X'. In addition, the number of Pareto solution and the time needed for implementation of an algorithm are significant.

4.3 Computational results

In this section, we are going to compare the proposed algorithms for hybrid flow shop scheduling. In this paper, SPGA II and NSGA II with an additional local search phase were adapted to solve the problem. The comparisons are performed on the basis of the sets of non-dominant solutions obtained by each algorithm. Figure 6 illustrates a comparison of final population of SPGA II and NSGA II algorithms for the same problem. As it shows, totally, the Pareto archive of NSGA II algorithm is better than the other one. It is closer to the ideal point. However, the Pareto front of the SPGA II algorithm is more crowded in this special instance.

Figure 7 presents the non-dominant solutions of a single run by SPGA II and NSGA II and local search phase for different sizes of problems. As it can be seen, the neighborhood exploration improves the Pareto archive

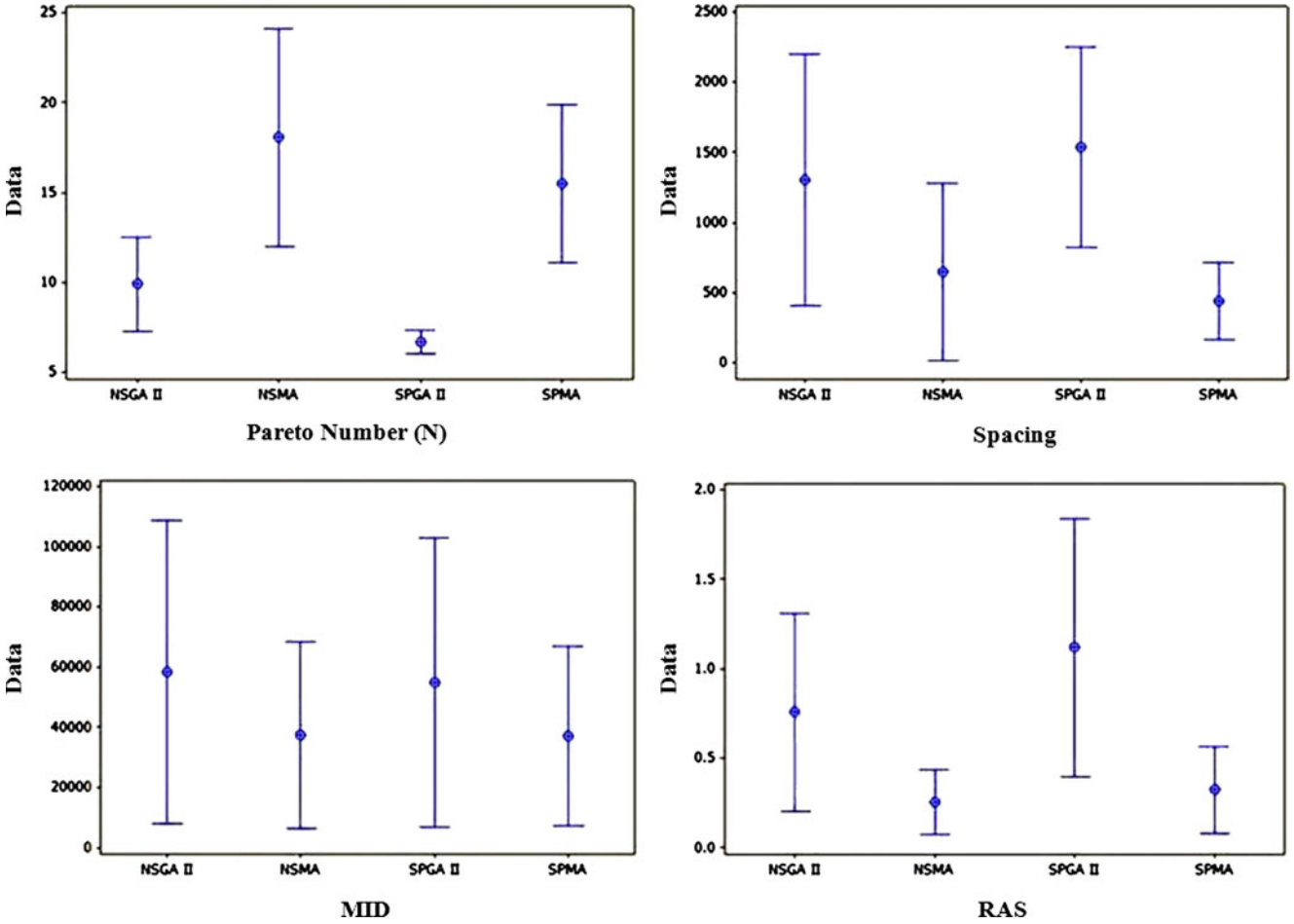

Fig. 11 Interval plots of *NSGA II*, *NSMA*, *SPGA II*, and *SPMA*

significantly. These results show that the proposed search algorithm works effectively in all size of problems.

In order to evaluate the efficacy and performance of the algorithms proposed in this paper, 12 different sizes of problems are used. As it can be seen in average row in Table 1, the SPMA and NSMA are superior to the basic version of the algorithms in all aspects. Tables 2 and 3 represent the values of the mean evaluation metrics for respectively small and large sizes of problems in three runs.

It is notable that the time measure mentioned for NSMA and SPMA algorithms is only the time period for the local search phase. It means the neighborhood exploration add this distinct time to the CPU time for achieving the final answer. Since we want to prove the efficiency of local search in advancement of Pareto archive, *CS* metric then shows the dominance percentage of non-dominant solution set which obtain by this phase in compare with the relevant basic algorithm. For example, notice *CS* for NSGA II and NSMA; Table 1 shows that 78.28% of Pareto members of NSMA dominate all members of the Pareto set which belongs to NSGA II. On the other side, there is no non-dominant solution of NSGA II that can dominate the Pareto archive of NSMA. Due to the proximity of the solutions attained by the basic version of algorithms and also the local search phases, it is not logical to compare this criterion for them.

Figures 8 and 9 show the effect of jobs and number of working stages on evaluation metrics for different algorithms. As we can see, in general, proposed local search algorithm significantly improves the solutions obtained by basic version of algorithms.

Figure 10 illustrates the mean evaluation metrics for all algorithms. As it can be seen above, the local search phase causes a meaningful improvement in all measures. In order to see the difference between algorithms, one-way ANOVA is applied for each criterion. All analyses in this section are at the 95% confidence level. Figure 11 shows *LSD* interval plots for the evaluation metrics.

5 Conclusions and future works

In this paper, we considered the minimization of two objectives which are the total tardiness and completion time

in a hybrid flow shop environment with restricted buffer space and also sequence-dependent setup time condition. For this reason, a new metaheuristic which combines SPGA II and NSGA II with a neighborhood search is proposed. First, the SPGA II and NSGA II were suggested for solving the problem in an exact time. Subsequently, in order to improve non-dominant solutions and to explore promising area for Pareto solution space, a functional local search algorithm started working with the Pareto archive reached by those algorithms.

To validate the proposed algorithm, we used 12 sizes of test problems, and different comparison metrics were proposed to judge and evaluate the performance of the algorithms. Experimental design shows that such a local search can be of great significance in improving the Pareto optimal solutions. Therefore, it is obvious that SPMA and NSMA are successful in this novel environment.

This paper illustrates the effectiveness of presented multi-phase algorithms in this particular problem. We believe that the methods proposed here are a good start in developing well-organized and useful algorithms for this special environment.

References

1. Abadi INK, Hall NG, Sriskandarajah C (2000) Minimizing cycle time in a blocking flowshop. Oper Res 48:177–180
2. 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 36:11057–11069
3. Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C (2001) Minimizing makespan in a flowshop using genetic algorithms. Int J Prod Econ 70:101–115
4. Chang P-C, Chen S-H (2009) The development of a sub-population genetic algorithm II (SPGA II) for multi-objective combinatorial problems. Appl Soft Comput 9:173–181
5. Davoudpour H, Ashrafi M (2009) Solving multi-objective SDST flexible flow shop using GRASP algorithm. Int J Adv Manuf Technol 44(7–8):737–747. doi:10.1007/s00170-008-1887-5
6. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm (NSGA-II). IEEE Trans Evol Comput 6:2
7. Hung TSL, Ching JL (2003) A case study in a two-stage hybrid flow shop with setup time and dedicated machines. Int J Prod Econ 86:133–143
8. Jolai F, Sheikh Sh, Rabbani M, Karimi M (2009) A genetic algorithm for solving no-wait flexible flow lines with due windows, and job rejection. Int J Adv Manuf Technol 42(5–6):523–532
9. Kurz ME, Askin RG (2003) Comparing scheduling rules for flexible flow lines. Int J Prod Econ 85:371–388
10. Kurz ME, Askin RG (2004) Scheduling flexible flow lines with sequence-dependent setup times. Eur J Oper Res 159(1):66–82
11. Leisten R (1990) Flowshop sequencing problems with limited buffer storage. Int J Prod Res 28:2085–2100
12. Luh PB, Gou L, Zhang Y, Nagahora T, Tsuji M, Yoneda K, Hasegawa T, Kyoya Y, Kano T (1998) Job shop scheduling with group dependent setups, finite buffers, and long time horizon. Ann Oper Res 76:233–259
13. McCormick ST, Pinedo ML, Shenker S, Wolf B (1989) Sequencing in an assembly line with blocking to minimize cycle time. Oper Res 37:925–936
14. Norman BA (1999) Scheduling flowshops with finite buffers and sequence-dependent setup times. Comput Ind Eng 36:163–177
15. Nowicki E (1999) The permutation flow shop with buffers: a tabu search approach. Eur J Oper Res 116:205–219
16. Pinedo M (1995) Scheduling theory, algorithms, and systems, 2nd edn. Prentice-Hall, Englewood Cliffs
17. Ronconi DP (2004) A note on constructive heuristics for the flowshop problem with blocking. Int J Prod Econ 87:39–48
18. Ronconi DP, Henriques LRS (2007) Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. Omega 37:272–281
19. Sawik TJ (1993) A scheduling algorithm for flexible flow lines with limited intermediate buffers. Appl Stoch Models Data Anal 9:127–138
20. Sawik TJ (1995) Scheduling flexible flow lines with no in-process buffers. Int J Prod Res 33(5):1357–1367
21. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64:278–285
22. Wang L, Zhang L, Zheng D-Z (2006) An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. Comput Oper Res 33:2960–2971
23. Wardono B, Fathi Y (2004) A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. Eur J Oper Res 155:380–401
24. Wittrock RJ (1988) An adaptable scheduling algorithm for flexible flow lines. Oper Res 36:445–453
25. Yaurima V, Burtseva L, Tchernykh A (2008) Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. Comput Ind Eng. doi:10.1016/j.cie.2008.09.004
26. 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