Int J Adv Manuf Technol (2014) 75:833–847
DOI 10.1007/s00170-014-6195-7

ORIGINAL ARTICLE

A novel genetic algorithm for the hybrid flow shop scheduling with parallel batching and eligibility constraints

Antonio Costa · Fulvio Antonio Cappadonna · Sergio Fichera

Received: 25 April 2014 / Accepted: 21 July 2014 / Published online: 7 August 2014

Abstract The hybrid flow shop with parallel batching (HFSPB) is a kind of flow shop production system wherein some stages may be populated by parallel processors that simultaneously process groups of jobs. This paper addresses the makespan minimization problem for a HFSPB system whose machines are characterized by both capacity and eligibility restrictions. Firstly, a mixed integer linear programming model concerning the proposed problem is presented. Then, a specific genetic algorithm (GA) that makes use of a permutation encoding scheme as well as a crossover operator specifically designed for effectively managing the batch processing is discussed. The relevant parameters of the developed algorithm were calibrated by means of a full factorial design of experiments, and an extensive comparison campaign has been carried out with the aim to statistically assess the performance of the proposed GA with respect to five alternative procedures, four of which arisen from the relevant literature. The obtained results, also supported by a properly developed ANOVA analysis, demonstrate the effectiveness of the proposed GA-based metaheuristics in tackling the HFSPB problem investigated, under both quality of solutions and computational burden viewpoints.

Keywords Hybrid flow shop · Batch processors · Parallel batching · Genetic algorithm · Linear programming

1 Introduction

In the last decades, several variants of the regular scheduling problems have been studied by the body of literature, with the aim of improving the applicability level of such research findings to the real industrial environment. One of the research topics which better interprets this trend consists in the study of hybrid configurations of flow shop manufacturing systems wherein production stages may be composed by multiple machines in parallel, such as flow shops with multiple processors (FSMP), flexible flow lines (FFL), and hybrid flow shops (HFS). According to what was stated by Ruiz and Vázquez-Rodríguez [1], these theoretical models have in common the following features:

- the number of processing stages I is greater than or equal to 2;
- each stage i has $M_i \geq 1$ parallel machines, and at least one stage $\bar{i}$ has $M_{\bar{i}} > 1$;
- all jobs follow the same unidirectional production flow, from stage 1 to stage I.

In the FSMP problem, all machines pertaining to a certain stage are assumed to be identical. FFL systems entail identical machines as well, but are also characterized by missing operations, i.e., some jobs may be allowed to skip one or more production stages. The HFS problem is instead a more complex variant of the regular flow shop, as it entails unrelated parallel machines at each stage. One of the first researches dealing with these hybrid configurations of flow shops was performed by Gupta [2], who demonstrated as FSMP problem is NP-Hard even in the case of only two production stages, one of them containing just one machine. The complexity of flow shops with parallel machines has also been addressed by Gourgand et al. [3], who stated that the total number of solutions of a hybrid flow shop problem can be calculated as $N!(\prod_{i=1}^{I} M_i)^N$, being N the number of jobs to be processed through I manufacturing stages, each one populated by M_i machines. In light of the computational complexity connected to these variants of the regular flowshop problem, most of the literature focused its efforts on the development of heuristic

A. Costa (✉) · F. A. Cappadonna · S. Fichera
University of Catania, Catania, Italy
e-mail: costa@diim.unict.it

methods as well as metaheuristic algorithms. Nevertheless, some exact approaches, mainly oriented to small-sized instances, can be found in literature [4–6].

Heuristic methods are often needed to deal with tricky scheduling problems as they can provide a quick solution without affecting the computational burden. However, they may suffer under the robustness viewpoint since obtained solutions may be drastically far from the global optimum. Guinet and Solomon [7] studied a flexible flow line with the aim of minimizing makespan and maximum tardiness; they developed a set of heuristics and used a benchmark based on 1,920 test instances to compare them with a set of specific lower bounds. Brah and Loo [8] investigated the effectiveness of five heuristics for solving a flow shop with multiple processor scheduling problem, in terms of both makespan and mean flow time minimization; at the end of their analysis, two remarkable methods have been identified. Botta-Genoulaz [9] tackled a FSMP problem with precedence constraints, time lags, and due dates by six heuristic methods that were compared over three different problem benchmarks to minimize both maximum lateness and mean completion time. More recently, Yang [10] addressed the problem of minimizing the total completion time on a two-stage flow shop with one only machine at stage 1 and two dedicated machines at stage 2. The same author developed three different heuristics and measured their performances against worst case bounds on relative errors. The use of heuristics in hybrid flow shops with dedicated machines has also been investigated by Wang and Liu [11] that proposed four constructive heuristics and a procedure based on branch and bound (B&B) algorithm for minimizing makespan. An extensive comparison campaign involving problems up to 100 jobs demonstrated the effectiveness of the B&B-based approach.

Differently from heuristic methods, metaheuristic algorithms can ensure a higher performance in finding near-optimal solutions of NP-Hard problems. Nevertheless, they often need a previous design phase aiming to select the most appropriate encoding/decoding schemes necessary to describe the real problem under investigation and to compute the objective function to be optimized, respectively. One of the first works concerning the use of metaheuristic algorithms for addressing flow shop systems with parallel machines is ascribable to Xiao et al. [12], who proposed a permutation encoding-based genetic algorithm (GA) for minimizing makespan in a FSMP problem. They tested the effectiveness of the proposed metaheuristics with respect to a random sampling procedure and a heuristic method arisen from literature. The ability of genetic algorithms in addressing hybrid configurations of flow shops has also been investigated by Ruiz and Maroto [13], who developed a GA for the makespan minimization of a HFS problem with sequence-dependent setup times and machine eligibility constraints. There, a single permutation encoding scheme was employed, together with an early finish machine decoding policy that assigns a given job to the machine able to earlier complete it. Authors carried out an extensive comparison campaign involving 10 alternative procedures arisen from literature and properly adapted to the problem in hand. A factorial design with 1,320 test problems confirmed the superiority of the proposed algorithm. Some years later, Yaurima et al. [14] employed a similar GA to cope with a HFS problem concerning a real production environment with sequence-dependent setup times, machine availability constraints, and limited buffers.

Apart from genetic algorithms, many other metaheuristic procedures have been devised for solving hybrid configurations of the regular flow shop scheduling problem. Wardono and Fathi [15] implemented a tabu search (TS) algorithm with the aim of minimizing makespan in a FSMP problem with limited buffer capacities. They equipped the proposed TS by both a permutation encoding scheme and a decoding procedure able to run the inter-operational buffer saturation. Engin and Döyen [16] dealt with the problem of minimizing makespan in a flow shop containing identical parallel machines at each stage through an artificial immune system (AIS) algorithm, inspired to clonal selection principles and affinity maturation mechanisms widely studied in theoretical immunology. Further, they carried out a comparison with lower bounds provided by a B&B procedure over a formalized benchmark of instances. Ying and Lin [17] faced a flow shop with multiprocessor tasks by means of an ant colony system (ACS) algorithm. They tested the performances of the proposed technique against a genetic algorithm and a tabu search from the relevant literature over two well-known benchmark datasets. A similar problem was studied by Tseng and Liao [18], who merged a particle swarm optimization (PSO) algorithm and a proper local search procedure, inspired to the flocking behavior of birds. In order to assess the effectiveness of such method, authors performed a comparison with three existing metaheuristics over a set of problems arisen from literature. A PSO approach for solving flow shops with parallel machines was also investigated by Singh and Mahapatra [19], who studied the makespan minimization problem for a FSMP production system. The authors employed a real number-based encoding scheme and used a chaotic number generation mechanism. A set of lower bounds formerly presented in literature was taken as reference for the final performance evaluation. A memetic algorithm (MA) combined with a local search engine was proposed by Tavakkoli-Moghaddam et al. [20] for minimizing the total completion time on an unbuffered flexible flow line with processor blocking. In this case, an encoding structure based on a matrix along with a vector was used to represent the problem solution. Then, a local search engine called nested variable neighborhood search (NVNS) was implemented to speed up performance of the MA algorithm. The obtained results were compared with those achieved by a classical genetic

algorithm, as to prove the effectiveness of the devised approach. Mirsanei et al. [21] developed a simulated annealing (SA) algorithm for minimizing makespan in a flow shop system with identical parallel machines at each stage and sequence-dependent setup times. Such method employs a permutation encoding scheme, joined with an effective decoding procedure inspired to the so-called shortest processing time cyclic heuristic (SPTCH) developed by Kurz and Askin [22]. Successively, SA was compared with a GA and an immune algorithm (IA) over three benchmarks of problems including small-, medium-, and large-sized instances, respectively. A biogeography-based optimization (BBO) procedure was recently proposed by Attar et al. [23] for minimizing makespan in a HFS problem with limited waiting times in inter-stage buffers and ready time of jobs. The authors demonstrated the superiority of the proposed approach against an imperialist competitive algorithm (ICA) and a population-based simulated annealing (PBSA) on a benchmark of 36 test problems including 20-, 50-, and 100-job instances.

Similarly to the hybrid flow shop manufacturing system topic, the research area connected with batch processing received many contributions during the last few decades, probably due to its ability in fitting the real-world production environments. Indeed, batch scheduling theory is quite common to be found in several manufacturing firms, as it allows both job setups and processing time reduction as well as benefits under the material handling viewpoint. As pointed out by Mathirajan and Sivakumar [24], a sharp distinction needs to be made between serial batching and parallel batching. In serial-batching problems, jobs that need the same setup operation on a given machine are grouped and then they are sequentially processed one-by-one by each processor [25–27]. In parallel-batching problems, a batch processor simultaneously processes all jobs grouped within the same batch. Batch processing time is equal to the highest processing time among jobs included in the batch. Such problem is also known in literature as scheduling of batch processors (SBP) or scheduling of batch processing machines. One of the first works dealing with the SBP problem is due to Azizoglu and Webster [28], who adopted a branch and bound procedure able to minimize the total weighted completion time on a batch processing machine with incompatible job families. In words, only jobs belonging to the same family (i.e., having identical processing times) can be processed in the same batch. Chang and Wang [29] addressed the problem of scheduling semiconductor burn-in operations on a batch processing machine able to simultaneously process different jobs within its capacity limit. The authors proposed a new heuristic algorithm to be employed for minimizing the total completion time.

Mathirajan et al. [30] analyzed a production process entailing heterogeneous batch processors and incompatible job families with the aim of maximizing machines' utilization. Mönch et al. [31] combined a genetic algorithm with both dispatching and scheduling rules for minimizing the total weighted tardiness on a production system characterized by parallel batch processors with incompatible job families and unequal ready times of jobs.

The study of hybrid configurations of flow shops including batch processors has recently started to be dealt as well, basically due to the affinity of this theoretical model to real production processes belonging to the microelectronics manufacturing industry. Amin-Naseri and Behesti-Nia [32] developed a three-dimensional genetic algorithm for minimizing makespan in a flow shop involving parallel identical batch processors. More recently, Costa et al. [33] investigated the effectiveness of a double-encoding metaheuristic procedure for tackling a constrained HFS problem with job overlapping under the makespan minimization viewpoint.

This paper focuses on the problem of scheduling incompatible job families into a hybrid flow shop production system equipped with unrelated parallel batch processors, these last being characterized by having both capacity and eligibility restrictions. It is worth pointing out that, to the best of our knowledge, the body of literature never addressed such a kind of scheduling problem so far.

Firstly, a mixed integer linear programming (MILP) model concerning the problem in hand has been devised. Then, a novel genetic algorithm for the makespan minimization was developed and an extensive experimental analysis has been performed with the aim of highlighting the effectiveness of the proposed optimization approach.

The remainder of the paper is organized as follows. The next section deals with the problem statement: in Section 3, MILP model is reported. Section 4 illustrates the proposed genetic algorithm; in Section 5, the outcomes of a calibration campaign needed to properly set the most relevant parameters of the developed GA are presented; in Section 6, a comparison campaign involving several optimization techniques arisen from literature and adapted to the problem in hand is presented. Finally, Section 7 concludes the paper.

2 Problem description

This paper focuses on the makespan minimization problem in a hybrid flow shop production system with unrelated batch processors at each stage, incompatible job families, and machine eligibility restrictions. In the regular HFS scheduling problem, a set of N jobs has to be processed through I manufacturing stages; each stage i (i=1, 2,…, I) entails a set of M_i unrelated parallel machines. Every job j (j=1,2,…, N) has to pass through all I stages following the same production flow, i.e., stage 1, stage 2, …stage I, being processed by exactly one machine at each stage. No precedence relationship exists among jobs. Pre-emption is not allowed, i.e., once a job

processing starts, it must be completed without any interruption. All jobs are ready to be worked at the beginning of the scheduling period, i.e., their release time is equal to zero. Machines are continuously available during the whole production session. Furthermore, setup times are assumed to be sequence-independent and included into processing times.

In addition to what was stated before, the following three features characterize the proposed research.

1. *Batch processors*. Parallel machines within each production stage have to be considered as batch processors with limited capacity, i.e., they may simultaneously work distinct batches of jobs instead of single jobs. The maximum size of each batch, expressed in terms of number of jobs to be processed together, is a priori known and depends on the selected processor, as machines are assumed to be unrelated.
2. *Incompatible job families*. The set of jobs to be processed may be partitioned into smaller subset, referred to as families. Jobs belonging to the same family are assumed to be identical, i.e., they have the same processing times for each machine of the whole manufacturing system. The batching option is allowed only for identical jobs; thus, a machine may simultaneously process several jobs at a time only if they belong to the same family. The batch processing time corresponds to the processing time of a single job pertaining to that batch.
3. *Machine eligibility constraints*. For each job j and for each production stage i, a set R_{ji} of eligible machines is defined; each job visiting a given stage can be processed only by an eligible machine. In addition, it must be $1 \le |R_{ji}| \le M_i$, being M_i the total number of machines included into the ith stage. Whether j and k are identical jobs of a given job family, it must be $R_{ji}=R_{ki}$.

It is worth noting that the proposed problem is the generalization of a scheduling issue truly observed in the backend assembly and test (BAT) area of a semiconductor manufacturing firm, where burn-in operations on IC chips have to be performed by means of batch processing ovens able to simultaneously process groups of identical jobs loaded onto boards.

3 MILP model

In the present section, a mixed integer linear programming formulation of the proposed HFS problem is given.

Minimize $Z=C_{\max}$

s.t

$$\sum_{m=1}^{M_i\left(m\in R_{ji}\right)} Y_{jim} = 1 \quad j=1,2,\ldots,N \quad i=1,2,\ldots,I \tag{1}$$

$$C_{j1} \ge \sum_{m=1}^{M_1} Y_{j1m}\cdot P_{j1m} \qquad j=1,2,\ldots,N \tag{2}$$

$$C_{ji} - C_{j(i-1)} \ge \sum_{m=1}^{M_i} Y_{jim}\cdot P_{jim} \quad j=1,2,\ldots,N \qquad i=2,3,\ldots,I \tag{3}$$

$$C_{ji} \le C_{ki} - \sum_{m=1}^{M_i} Y_{kim}\cdot P_{kim} + B\cdot\left(1-Q_{jki}\right) \quad j,k=1,2,\ldots,N \quad j\neq k \quad i=1,2,\ldots,I \tag{4}$$

$$\begin{cases} C_{ji} \le C_{ki} + B\cdot\left(1-W_{jki}\right) & j=1,2,\ldots,N \quad k=j+1, \\ C_{ki} \le C_{ji} + B\cdot\left(1-W_{jki}\right) & j+2\ldots,N \qquad i=1,2,\ldots,I \end{cases} \tag{5}$$

$$Q_{jki} + Q_{kji} + W_{jki} \ge 1 - B\cdot\left(2-Y_{jim}-Y_{kim}\right) \quad j=1,2,\ldots,N \quad k=j+1,j+2\ldots,N \quad i=1,2,\ldots,I \quad m=1,2,\ldots,M_i \tag{6}$$

$$\sum_{k=j+1}^{N\left(k\notin S_j\right)} W_{jki} = 0 \qquad j=1,2,\ldots,N \quad i=1,2,\ldots,I \tag{7}$$

$$\sum_{k=1}^{j-1} W_{kji} + \sum_{k=j+1}^{N} W_{jki} + 1 \le \sum_{m=1}^{M_i} BS_{im}\cdot Y_{jim} \quad j=1,2,\ldots,N \quad i=1,2,\ldots,I \tag{8}$$

$$C_{\max} \ge C_{jI} \qquad j=1,2,\ldots,N \tag{9}$$

$$Y_{jim}\in\{0,1\} \quad j=1,2,\ldots,N \quad i=1,2,\ldots,I \quad m=1,2,\ldots,M_i \tag{10}$$

$$Q_{jki}\in\{0,1\} \qquad j,k=1,2,\ldots,N \quad j\neq k \quad i=1,2,\ldots,I \tag{11}$$

$$W_{jki}\in\{0,1\} \quad j=1,2,\ldots,N \quad k=j+1,j+2\ldots,N \quad i=1,2,\ldots,I \tag{12}$$

Constraint (1) ensures that each job is processed by one machine per stage, selected among those eligible. Constraint (2) imposes that the completion time of a given job at the first stage must be greater than or equal to its corresponding processing time. Constraint (3) defines the relationship

between completion times of each job for two adjacent stages. Through constraint (4), it is imposed that if jobs j and k are worked by the same machine at stage i and j precedes k, job j must be completed before job k starts. Constraint (5) runs batch processing; it states that if job j and k are assigned to the same batch to be worked by a certain machine pertaining to stage i, their completion times must be the same. Constraint (6) establishes that for each couple of jobs j and k assigned to the same machine at a given stage, one condition among the following three must be fulfilled: j precedes k, k precedes j, j and k are simultaneously worked within the same batch. Constraint (7) ensures that batch processing is allowed only for identical jobs. Constraint (8) states that the total amount of jobs worked within the same batch of job j must not exceed the allowed maximum batch size. Constraint (9) forces makespan to be equal or greater than any job completion time at the last stage. Finally, constraints (10), (11), and (12) define the binary variables.

4 The proposed genetic algorithm

Genetic algorithms [34] are computational methods inspired by the process of natural evolution, which have largely been used to solve scheduling problems. Generally, a GA works with a set of problem solutions called *population*. At every iteration, a new population is generated from the previous one by means of two operators, *crossover* and *mutation*, applied to solutions (*chromosomes*) selected on the basis of their *fitness*, i.e., the objective function value; thus, best solutions have greater chances of being selected. Crossover operator generates new solutions (*offspring*) by coalescing the structures of a couple of existing ones (*parents*), while mutation operator brings a change into the scheme of selected chromosomes, with the aim to avoid the procedure to remain trapped into local optima. The algorithm proceeds by letting the population evolve through successive *generations* until a given stopping criterion is reached.

Whenever a real problem is addressed through an evolutionary algorithm, the choice of a proper *encoding* scheme (i.e., the way a solution is represented by a string of *genes*) plays a key role under both the quality of solutions and the computational burden viewpoints. In addition, a valid *decoding* procedure able to transform a given string into a feasible solution has to be provided.

In the following subsections, a detailed description of the proposed GA, named SGA, properly developed for solving the proposed HFSBP problem is reported.

4.1 Overview of the encoding/decoding strategy

Usually, the hybrid flowshop scheduling problem needs a specific encoding, which separately runs the job permutation and the assignment of jobs to machines at every stage [4, 35]. Nevertheless, it has recently been demonstrated how a simple permutation encoding scheme, linked to an effective decoding procedure, can drive metaheuristic methods to efficiently solve the HFS problem [13, 21].

In the present work, such a kind of encoding strategy has been adopted since the proposed genetic algorithm operates by means of N integers, where N is the number of jobs to be scheduled. As far as the decoding procedure is concerned, a method inspired to the SPT cyclic heuristic (SPTCH) proposed by Kurz and Askin [22] has been adopted. In fact, the permutation string of jobs drives the job allocation within the first stage while, as concerns the subsequent stages, jobs are processed according to an ERT (earliest release time) criterion, i.e., the job with the smallest completion time is the first job to be processed in the current stage and so on. At every stage, each job is assigned to the machine wherein it may be completed at the earliest time. The proposed method, differently from previous studies, is able to contemporarily run the batch processing issue (by assigning a set of identical jobs on a given workstation) as well as the machine eligibility restrictions. In the following paragraphs, a detailed description of the decoding procedure is provided.

Let a permutation π of N integers be the current solution of a hybrid flow shop problem with N jobs to be scheduled through I successive stages, each one made of M_i unrelated parallel machines (i=1, 2,…, I); π_i indicates the sequence of jobs to be scheduled at stage i; l denotes a general position within the sequence (l=1, 2,…, N) and $\pi_i(l)$ indicates the lth job of the sequence π_i which relates to stage i. The construction of a feasible schedule associated with a solution π occurs through the following steps:

For stage 1,

1. Set $\pi_1=\pi$.
2. For l=1 to N,
   (a) Let *bestmc*= *1*.
   (b) For m=1 to $M_1|m$ eligible for processing job $\pi_1(l)$,
      i. Detect the last job γ worked by machine m.
      ii. If job γ is identical to job $\pi_1(l)$ and machine m has enough capacity, then place job $\pi_1(l)$ within the same batch of job γ, else place job $\pi_1(l)$ after job γ
      iii. Calculate completion time of job $\pi_1(l)$.
      iv. If completion time of job $\pi_1(l)$ is less on machine m then on machine *bestmc*, then let *bestmc*= m.
   (c) Assign job $\pi_1(l)$ to machine *bestmc*.

For each stage, i=2,…, I:

1. Update the ready times in stage i to be the completion times in stage i–1.

2. Generate π_i by arranging jobs in ascending order of ready times.
3. For l=1 to N,

   (a) Let *bestmc*= *1*.
   (b) For m=1 to $M_i|m$ eligible for processing job $\pi_i(l)$,

      i. Detect the last job γ worked by machine m.
      ii. If job γ is identical to job $\pi_i(l)$, machine m has enough capacity and job $\pi_i(l)$ is ready to be worked when job γ starts, then place job $\pi_i(l)$ within the same batch of job γ, else place job $\pi_i(l)$ after job γ.
      iii. Calculate completion time of job $\pi_i(l)$.
      iv. If completion time of job $\pi_i(l)$ is less on machine m then on machine *bestmc*, then let *bestmc*= m.
   (c) Assign job $\pi_i(l)$ to machine *bestmc*.

In order to clarify how the aforementioned decoding procedure works, an illustrative example consisting of five jobs (N=5) to be scheduled on a hybrid flow shop system with two stages (I=2) and two machines per stage (M_1=M_2=2) has been reported. For each job j, Table 1 shows processing times (P_{jim}) as both i (index of stages) and m (index of machines within each stage) change.

It can be noticed that machine 2 at stage 1 is not eligible to process jobs 1 and 2; similarly, job 5 cannot be processed by machine 1 at stage 2. Jobs 1 and 2 are assumed to be identical as well as jobs 3 and 4; identical jobs have the same processing times through the whole manufacturing system, and they can be simultaneously worked within the same batch on a given eligible machine. Table 2 shows the maximum batch sizes allowed by each machine m pertaining to each stage i (BS_{im}).

Now, let us suppose to have solution π={3, 1, 4, 2, 5}. Table 3 summarizes the proposed decoding procedure applied to π. For each stage i, a new sequence π_i is generated. In particular, $\pi_i(l)$ indicates the job to be scheduled at the current iteration l. $EC_{\pi_i(l),i,m}$ denotes the expected completion time of job $\pi_i(l)$ whether it has to be scheduled on machine m at stage i, while *bestmc* indicates the machine selected to process a job $\pi_i(l)$ (i.e., the machine able to complete that job before the others). $C_{\pi_i(l),i}$ denotes the actual completion time of job $\pi_i(l)$ at stage i. Figure 1 illustrates the Gantt chart obtained by decoding π through the proposed procedure.

Table 1 Processing times for the HFS example

P_{jim}	i=1		i=2	
	m=1	m=2	m=1	m=2
j=1	2	–	2	5
j=2	2	–	2	5
j=3	4	1	3	4
j=4	4	1	3	4
j=5	3	2	–	2

Table 2 Maximum batch sizes for the HFS example

	i=1		i=2	
	m=1	m=2	m=1	m=2
BS_{im}	1	2	2	1

With reference to machine 1 at stage 1, it can be noticed that both jobs 1 and 2 are separately processed, though they are identical, since that machine can process no more than one job at a time, being BS_{11}=1; thus, it cannot allow any kind of batch processing. On the other hand, jobs 3 and 4 are simultaneously worked in both stages, since machine 2 at stage 1 and machine 1 at stage 2 can ensure an adequate batch capacity.

According to this *modus operandi*, whenever in a given solution π two identical jobs are close to each other, both of them will be assigned to the same batch as to be worked by the same machine on the first stage, supposing that such machine has enough capacity to work them together. In such case, those identical jobs will have the same first-stage completion time, and they will be placed again in adjacent positions within the sequence π_2, which runs the assignment of jobs to machines in the second stage. Following the same fashion, those identical jobs have a greater chance to be placed into the same batch and to be worked by the same machine in the successive production stages. It is quite clear that such a kind of decoding strategy should lead to the total completion time reduction.

4.2 Genetic operators

P_{size} individuals compose the initial population of the proposed SGA. P_{size}–1 permutation strings are randomly

Table 3 Proposed decoding procedure for π ={3, 1, 2, 4, 5}

i	l	$\pi_i(l)$	$EC_{\pi_i(l),i,m}$		*bestmc*	$C_{\pi_i(l),i}$
			m=1	m=2		
i=1→π_1={3, 1, 4, 2, 5}						
1	1	3	4	1	2	1
1	2	1	2	–	1	2
1	3	4	6	1	2	1
1	4	2	4	–	1	4
1	5	5	7	3	2	3
i=2 → π_2={3, 4, 1, 5, 2}						
2	1	3	4	5	1	4
2	2	4	4	5	1	4
2	3	1	6	7	1	6
2	4	5	–	5	2	5
2	5	2	6	10	2	6

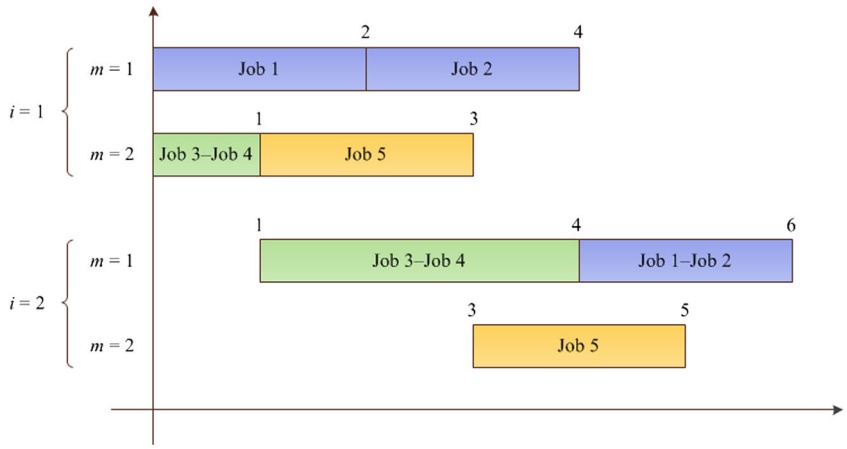

Fig. 1 Gantt chart obtained for $\pi = \{3, 1, 4, 2, 5\}$

extracted, while the missing individual is generated by applying the regular SPT rule just considering the job processing times related to the first stage. Indeed, this dispatching rule allows jobs belonging to the same family to be placed in adjacent positions, thus increasing the chances of both processing them through the same batch processor and reducing the total completion time.

As regards the selection mechanism, the well-known roulette-wheel scheme [36] has been adopted, thus assigning to each solution a probability of being selected inversely proportional to the makespan value.

A properly developed variant of the regular order crossover [37], hereinafter called *enclosed order crossover* (EOX), has been employed as crossover operator. According to such procedure, two cut points are randomly chosen for each couple of selected parents, with a restriction on their relative distance, which must be greater than two genes and lower than or equal to an a priori fixed value, hereinafter called MAX_{dist}. In such a way, a substring of genes to be rearranged, i.e., those positioned between the cut points, is identified for each parent; these genes are copied in the corresponding child according to the order in which they appear within the chromosome structure of the other parent. The remaining genes, which do not belong to the selected substring, are instead copied into the offspring without any change in their position as well as in their relative order. Figure 2 reports an example of EOX referred to an instance in which $N = 10$.

Basically, the effectiveness of the proposed crossover strategy would consist in keeping unchanged as much as possible any block of identical jobs, thus yielding a significance reduction of the total completion time. This circumstance has been confirmed by a preliminary test campaign, according to which a GA equipped with the proposed EOX crossover sensibly outperformed the same optimization algorithm powered by a regular position-based crossover. For the sake of brevity, such comparison analysis has not been included in the present paper.

As concerns the mutation operator, a simple shift operator [38] has been employed. A randomly selected gene is moved forward or backward along the sequence, on the basis of a randomly chosen number of positions. The maximum number of positions a single gene can be shifted is denoted as MAX_{shift}.

Finally, a proper elitism procedure, which preserves the best two individuals of each generation from any distortion caused by crossover or mutation operators, has been embedded within the proposed SGA.

5 Statistical calibration of genetic parameters

A proper calibration phase has been carried out, with the aim of selecting a suitable set of genetic parameters. To this end, a benchmark of 48 classes of large-sized problems has been arranged by combining in a full factorial design: four scenarios for the number N of jobs, three scenarios for the number I

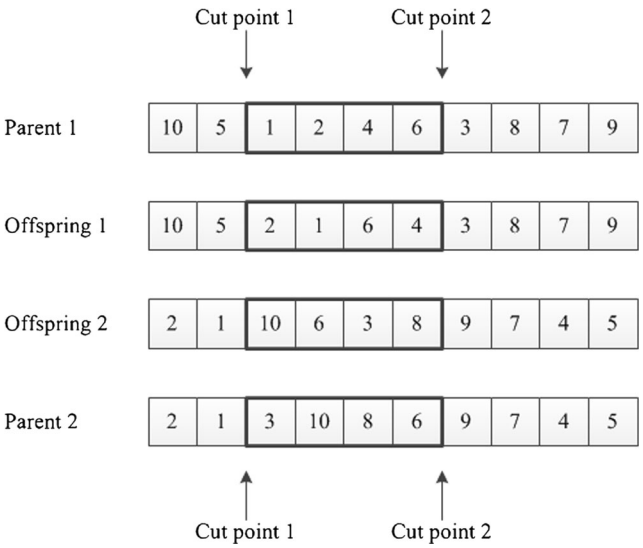

Fig. 2 Enclosed order crossover

Table 4 Structure of the benchmark for the calibration campaign

Factor	Notation	No. of levels	Levels
Number of jobs	N	4	(50, 100, 150, 200)
Number of stages	I	3	(3, 5, 10)
Number of machines in stage i	M_i	2	(U[1, 3], U[3, 5])
Maximum batch size allowed on machine m at stage i	BS_{im}	2	(U[1, 3], U[3, 5])

of stages, two scenarios for the number M_i of machines in each stage I, and two further scenarios concerning the batch processing capacity BS_{im} of each machine m within each stage i. Table 4 shows the structure of the calibration campaign, where symbol *U[a, b]* refers to a value extracted by a uniform distribution between a and b.

One instance has been generated for each class of problem. For each instance, the whole set of jobs has been partitioned into small subsets of identical jobs, each one made up of five elements at most. For each subset of identical jobs, processing times have been generated from a uniform distribution in the range [1, 99]. Eligibility restrictions have been taken into account by considering, for each machine, a 25 % probability of being ineligible for each subset of identical jobs. However, all instances have been generated as to ensure the eligibility of one machine per stage for the whole set of jobs, at least.

The devised benchmark aims to drive the tuning analysis on the following five parameters characterizing the developed SGA, i.e., P_{size}, p_{cross}, p_{mut}, MAX_{dist}, and MAX_{shift}. For each parameter, three different levels have been taken into account, as illustrated in Table 5, as to generate a total of 3^5=243 different configurations of the proposed metaheuristic algorithms. For each instance, all the provided algorithm configurations have been tested; therefore, a total of 48*243=11,664 runs have been considered. In order to identify the best combination of values for the aforementioned parameters, an ANOVA analysis [39] has been performed by means of Design Expert® 7.0.0 version commercial tool. The response

Table 5 Experimental calibration of proposed SGA

Parameter	Notation	No. of levels	Levels
Population size	P_{size}	3	(20, 50, 100)
Crossover probability	p_{cross}	3	(0.65, 0.8, 0.95)
Mutation probability	p_{mut}	3	(0.05, 0.1, 0.2)
Maximum relative distance between cut points (EOX crossover)	MAX_{dist}	3	(3, 5, 10)
Maximum shift width (mutation)	MAX_{shift}	3	(N/2, N/3, N)

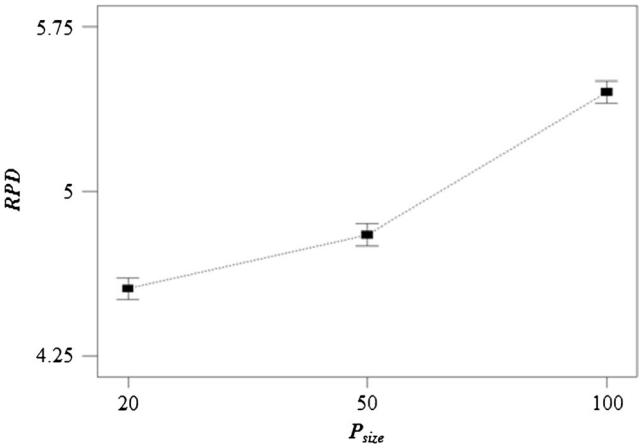

Fig. 3 Means plot with 95 % LSD intervals obtained for P_{size} parameter

variable studied was the relative percentage deviation (RPD), calculated according to the following formula:

$$\mathrm{RPD} = 100 \cdot \frac{\mathrm{SGA_{sol}} - \mathrm{BEST_{sol}}}{\mathrm{BEST_{sol}}} \tag{13}$$

where SGA_{sol} is the makespan solution found by an algorithm with a specific configuration in terms genetic parameters, and $BEST_{sol}$ is the best solution over the whole set of results concerning the same instance. The proposed SGA has been coded in MATLAB® language and executed on a 2-GB RAM virtual machine embedded on a workstation powered by two quad-core 2.39-GHz processors. Stopping criterion was set to a total of 10,000 makespan evaluations.

Figures 3, 4, 5, 6, and 7 show the means plots with LSD intervals at 95 % confidence level obtained for each one of the tuned genetic parameters.

Figure 3 shows that a population size (P_{size}) equal to 20 is the best parameter among the tested values. Probability of crossover (p_{cross}) should be set to 0.95, as confirmed by Fig. 4. The statistically significant difference highlighted in

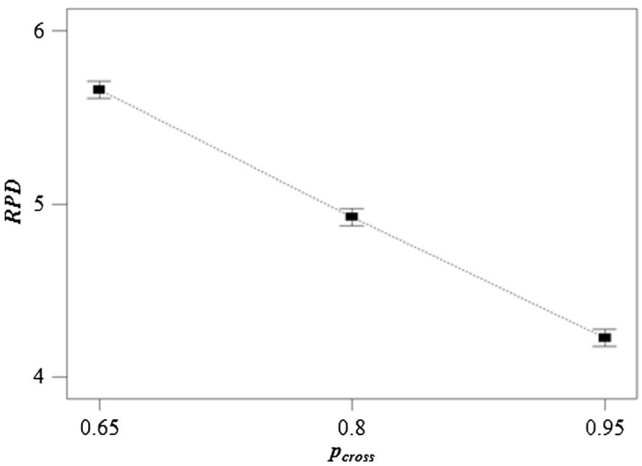

Fig. 4 Means plot with 95 % LSD intervals obtained for p_{cross} parameter

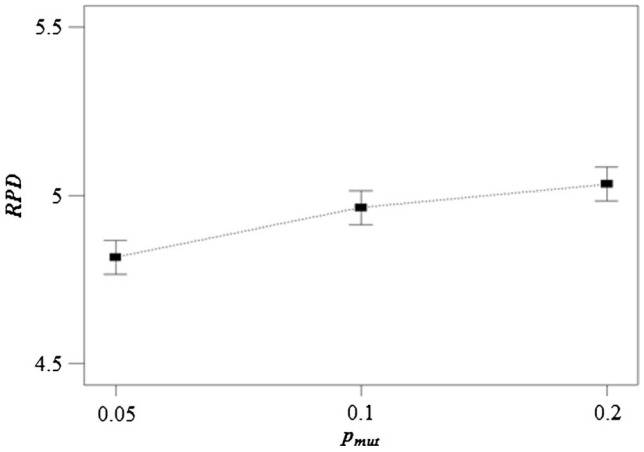

Fig. 5 Means plot with 95 % LSD intervals obtained for p_{mut} parameter

Fig. 5 indicates p_{mut} equal to 0.05 as the most effective choice for the proposed SGA. Figure 6 shows the statistical results related to the MAX_{dist} parameter, which characterizes the EOX crossover. The best performance should be achieved by adopting MAX_{dist} equal to 3. In light of the obtained results, it is worth pointing out that a frequent genetic recombination (p_{cross}=0.95) applied onto a small portion of chromosome (MAX_{dist}=3) may represent a valid strategy to preserve blocks of identical jobs and, at the same time, to enhance the performance of the SGA in terms of makespan minimization. Finally, means plot reported in Fig. 7 highlights as the best value for the maximum gene shift allowed in mutation operator, namely MAX_{shift}, is equal to N.

6 Numerical examples and computational results

In order to obtain an exhaustive evaluation regarding the effectiveness of the proposed SGA in solving the proposed HFSPB problem, an extensive comparative campaign has been performed, with the aim of assessing the proposed

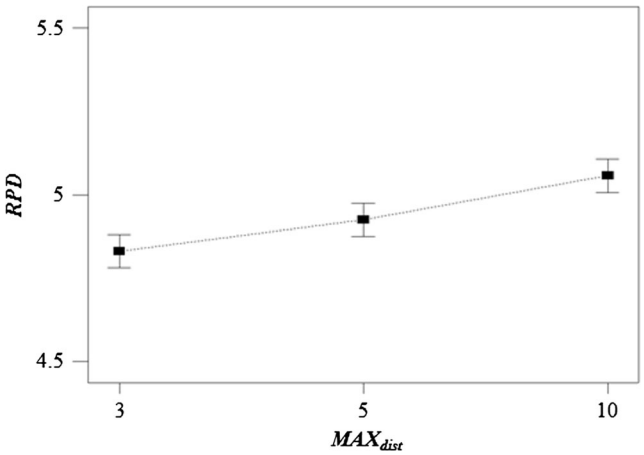

Fig. 6 Means plot with 95 % LSD intervals obtained for MAX_{dist} parameter

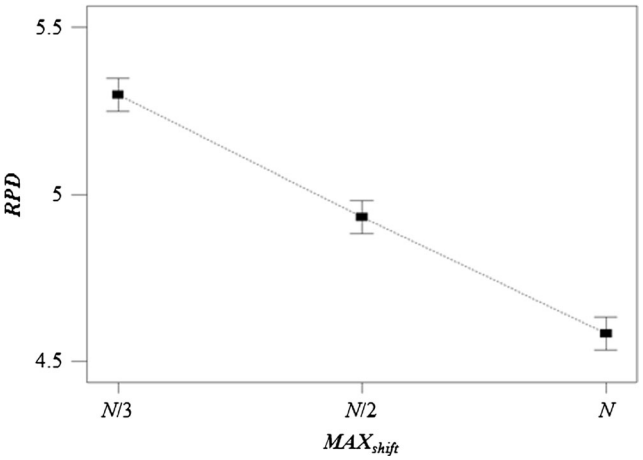

Fig. 7 Means plot with 95 % LSD intervals obtained for MAX_{shift} parameter

metaheuristic procedure with respect to a set of methods arisen from the relevant literature. A brief analysis of these methods is reported in the following paragraphs.

- The three-dimensional genetic algorithm (hereinafter coded as 3DGA) proposed by Amin-Naseri and Behesti-Nia [32] for solving the problem of minimizing makespan in a FSMP system with parallel-batching machines. Such method exploits a three-dimensional encoding that allows defining, for each stage of the production system and for each machine within a given stage, the exact scheduling sequence of jobs to be processed. The algorithm starts with an initial population partially composed by individuals obtained from properly developed heuristics; then, it employs a parameterized uniform crossover and two different mutation techniques, named swap and reverse operators, for the regular evolutionary path.
- The genetic algorithm (hereinafter coded as GAR) developed by Ruiz and Maroto [13] for minimizing makespan in a HFS problem with sequence-dependent setup times and machine eligibility. It exploits a single permutation encoding and an early finish machine decoding policy. Conforming to what was suggested by the authors, a block 2-point crossover and a regular shift-insertion

Table 6 Proposed benchmark of small-sized instances for the comparison campaign

Factor	Notation	No. of levels	Levels
Number of jobs	N	3	(10, 15, 20)
Number of stages	I	3	(3, 5, 10)
Number of machines in stage i	M_i	2	(U[1, 3], U[3, 5])
Maximum batch size allowed on machine m at stage i	BS_{im}	2	(U[1, 3], U[3, 5])

Table 7 Average RPD values for small-sized instances

Class	N	I	M_i	BS_{im}	SGA	3DGA	GAR	NSA	TSW	RND
1	10	3	U[1, 3]	U[1,3]	3.462	2.320	**1.654**	3.462	7.551	3.462
2	10	3	U[1,3]	U[1,5]	**0.977**	1.023	3.042	**0.977**	6.004	**0.977**
3	10	3	U[1,5]	U[1,3]	**4.907**	6.340	5.144	**4.907**	7.182	**4.907**
4	10	3	U[1,5]	U[1,5]	**1.074**	1.887	1.341	**1.074**	3.805	**1.074**
5	10	5	U[1,3]	U[1,3]	6.222	11.579	**5.983**	6.222	9.568	6.222
6	10	5	U[1,3]	U[1,5]	**1.082**	6.697	8.059	**1.082**	9.952	**1.082**
7	10	5	U[1,5]	U[1,3]	4.210	12.087	**3.705**	4.210	7.904	4.210
8	10	5	U[1,5]	U[1,5]	**4.886**	5.438	5.797	**4.886**	6.150	**4.886**
9	10	10	U[1,3]	U[1,3]	**4.842**	12.907	5.301	**4.842**	7.228	4.877
10	10	10	U[1,3]	U[1,5]	**5.843**	15.719	11.643	**5.843**	11.061	**5.843**
11	10	10	U[1,5]	U[1,3]	4.651	5.469	**4.370**	4.651	6.407	4.651
12	10	10	U[1,5]	U[1,5]	**7.307**	9.191	9.375	**7.307**	8.948	**7.307**
13	15	3	U[1,3]	U[1,3]	**0.925**	7.791	2.144	1.028	3.089	1.395
14	15	3	U[1,3]	U[1,5]	**0.000**	6.073	5.497	**0.000**	8.563	2.044
15	15	3	U[1,5]	U[1,3]	**0.582**	13.974	2.424	**0.582**	5.942	**0.582**
16	15	3	U[1,5]	U[1,5]	**0.614**	5.527	1.817	**0.614**	4.947	**0.614**
17	15	5	U[1,3]	U[1,3]	**0.111**	12.972	5.079	0.367	4.975	1.660
18	15	5	U[1,3]	U[1,5]	0.218	10.816	8.505	**0.000**	12.511	10.543
19	15	5	U[1,5]	U[1,3]	1.260	18.553	6.434	**1.036**	10.686	1.475
20	15	5	U[1,5]	U[1,5]	**0.040**	5.450	3.162	1.616	4.800	1.616
21	15	10	U[1,3]	U[1,3]	**1.288**	11.871	2.866	1.433	4.429	2.414
22	15	10	U[1,3]	U[1,5]	**0.037**	12.919	7.392	1.283	13.002	2.481
23	15	10	U[1,5]	U[1,3]	0.082	13.420	5.879	**0.000**	3.353	0.124
24	15	10	U[1,5]	U[1,5]	**0.000**	8.751	7.265	**0.000**	2.567	0.074
25	20	3	U[1,3]	U[1,3]	**0.017**	8.460	3.536	0.426	4.860	4.148
26	20	3	U[1,3]	U[1,5]	**0.000**	12.098	11.407	2.476	15.298	11.041
27	20	3	U[1,5]	U[1,3]	1.544	24.909	9.177	**1.315**	7.447	2.208
28	20	3	U[1,5]	U[1,5]	**0.465**	16.854	6.923	1.038	16.094	0.570
29	20	5	U[1,3]	U[1,3]	1.194	9.300	3.891	**0.259**	5.459	4.190
30	20	5	U[1,3]	U[1,5]	**0.348**	14.956	7.724	2.370	14.619	20.381
31	20	5	U[1,5]	U[1,3]	0.903	23.089	8.188	**0.605**	7.313	1.614
32	20	5	U[1,5]	U[1,5]	**0.000**	15.709	7.391	1.110	9.346	1.499
33	20	10	U[1,3]	U[1,3]	**0.379**	9.749	4.463	0.790	4.356	2.796
34	20	10	U[1,3]	U[1,5]	**0.208**	14.588	12.377	1.799	7.307	18.755
35	20	10	U[1,5]	U[1,3]	**0.348**	14.965	9.970	1.030	6.847	1.155
36	20	10	U[1,5]	U[1,5]	**0.268**	14.227	10.014	0.477	3.479	0.596
g_ave					**1.675**	11.047	6.082	1.975	7.585	3.985

operator have been employed as crossover and mutation operators, respectively.

– The novel simulated annealing (hereinafter NSA) elaborated by Mirsanei et al. [21] for scheduling jobs in a flow shop with multiple processors and sequence-dependent setup times with the aim of minimizing makespan. Similarly to the proposed SGA, such algorithm works with a single permutation solution encoding, also adopting a decoding scheme derived from the SPT cyclic heuristic (SPTCH) developed by Kurz and Askin [22]. Due to the modifications required by the SPTCH to satisfy both batch processing and eligibility restrictions characterizing the proposed HFSBP problem, it is worth pointing out that such a modified SPTCH decoding method coincides with the decoding scheme adopted by SGA. Finally, conforming to what was developed by the authors, a combination of two different local search techniques (pairwise interchange moving and inverse moving, respectively) has been embedded in the NSA algorithm.
– The tabu search devised by Wardono and Fathi [15] for addressing the makespan minimization in a FSMP problem with limited buffer capacity (hereinafter called as

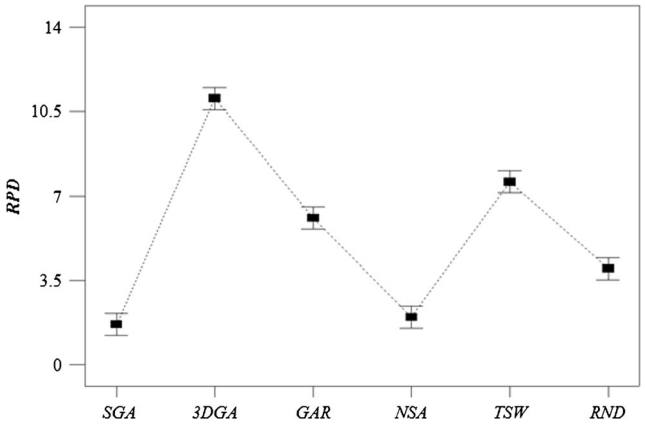

Fig. 8 Means plot with 95 % LSD intervals for small-sized instances

TSW). It employs a single permutation encoding scheme and a decoding procedure based on the first available machine rule for generating a feasible schedule from every coded solution. As proposed by the authors, a shift-insertion operator has been adopted to generate the neighborhood of the current solution, and a proper exploration/exploitation strategy has been implemented with the aim of enhancing the search mechanism throughout the solution space as well as intensifying the examination of the most promising areas.

In addition to the aforementioned algorithms, a simple random sampling algorithm (hereinafter named RND) has been included within the set of alternative methods to be compared with the proposed SGA. It is worth pointing out that such procedure exploits the same encoding/decoding strategy adopted by SGA. According to a regular random sampling mechanism, it starts with an initial randomly generated seed solution and then iteration by iteration, a new solution is obtained from the current one by applying the mutation operator used by SGA. Every time a new solution outperforms the current one, it assumes the role of a new seed solution.

The comparison campaign has been performed making use of two separate benchmarks, involving small- and large-sized instances, respectively. Parameters considered for generating small-sized problems are illustrated in Table 6, while the benchmark used for large-sized problems has been created similarly to what was done in the calibration campaign, whose parameters are reported in Table 4. On the whole, a total of 36 small-sized and 48 large-sized classes of problems have been created. For each class of problem, 10 different instances have been generated, following the same method adopted in the calibration phase wherein subsets of identical jobs, processing times, and machine eligibility restrictions have been configured. Thus, 36*10+48*10=840 instances have been generated. The overall set of instances has been solved by means of the six optimization procedures to be compared, namely SGA, 3DGA, GAR, NSA, TSW, RND. Therefore, a total of 840*6= 5,040 runs have been taken into account. Stopping criterion was set to 10,000 makespan evaluations.

The following subsections report the obtained results with reference to both small- and large-sized problems, respectively.

Table 8 Average CPU times (in seconds) for small instances

N	SGA	3DGA	GAR	NSA	TSW	RND
10	73.5	55.3	68.9	76.6	75.7	78.8
15	105.7	82.3	101.4	110.5	109.5	112.8
20	139.7	105.9	135.6	146.7	144.8	148.8
Average	106.3	81.2	102.0	111.3	110.0	113.5

6.1 Comparison analysis for small-sized problems

With reference to small-sized problems, 360 different instances have been considered for carrying out a comparison between the proposed SGA and the optimization procedures selected from relevant literature. In particular, for 120 instances of the benchmark (i.e., those having number of jobs N=10), the corresponding global optima have been obtained by means of the MILP model provided in Section 3. To this aim, an ILOG CPLEX® 12.0 64 bit platform installed within a workstation powered by two quad-core 2.39-GHz processors and with 24-GB RAM was used.

For each instance, and with reference to each metaheuristics, the relative percentage deviation from the best solution was calculated according to

$$\mathrm{RPD} = 100 \times \frac{\mathrm{ALG_{sol}\text{-}BEST_{sol}}}{\mathrm{BEST_{sol}}} \tag{14}$$

where $\mathrm{ALG_{sol}}$ is the solution provided by a given algorithm with reference to a given instance, while $\mathrm{BEST_{sol}}$ is the optimal solution obtained through the proposed MILP model. In case N>10, $\mathrm{BEST_{sol}}$ is equal to the lowest makespan among those obtained by the different algorithms. Table 7 illustrates for each class of problems the average values of RPD obtained by each algorithm, denoting in boldface the corresponding BESTsol value.

The obtained results highlight the effectiveness of SGA in solving small instances of the HFS problem in hand. The proposed SGA algorithm reaches the lowest average RPD value in 26 out of 36 classes of problems, also outperforming the other competitors under the RPD grand average viewpoint. NSA and RND reach the lowest average RPD values in 18 and 9 classes of problems, respectively. The fair result achieved by RND can be considered as a proof about the effectiveness of the single permutation encoding scheme

Table 9 Average RPD values for large-sized instances

Class	N	I	M_i	BS_{im}	SGA	3DGA	GAR	NSA	TSW	RND
1	50	3	U[1,3]	U[1,3]	**0.014**	11.690	3.468	2.276	3.188	12.970
2	50	3	U[1,3]	U[1,5]	**0.156**	14.392	3.241	2.234	2.910	18.690
3	50	3	U[1,5]	U[1,3]	**0.800**	15.243	3.249	2.442	3.661	11.644
4	50	3	U[1,5]	U[1,5]	**2.069**	26.517	8.077	6.427	10.557	10.907
5	50	5	U[1,3]	U[1,3]	**1.313**	18.516	7.927	4.257	8.880	11.239
6	50	5	U[1,3]	U[1,5]	**0.239**	19.997	10.155	3.251	7.028	6.445
7	50	5	U[1,5]	U[1,3]	**0.000**	16.222	7.526	9.718	33.254	66.995
8	50	5	U[1,5]	U[1,5]	**0.000**	16.094	8.950	9.131	23.156	63.745
9	50	10	U[1,3]	U[1,3]	**0.000**	17.344	7.161	8.411	13.926	57.713
10	50	10	U[1,3]	U[1,5]	**0.000**	30.407	10.087	13.541	50.968	53.822
11	50	10	U[1,5]	U[1,3]	**0.000**	28.401	15.310	14.529	27.467	31.655
12	50	10	U[1,5]	U[1,5]	**0.302**	25.567	14.361	8.426	22.065	27.403
13	100	3	U[1,3]	U[1,3]	**0.167**	5.257	1.593	4.034	13.978	29.787
14	100	3	U[1,3]	U[1,5]	**0.589**	7.950	2.404	4.734	14.897	22.392
15	100	3	U[1,5]	U[1,3]	4.156	17.690	**3.396**	8.250	17.076	22.288
16	100	3	U[1,5]	U[1,5]	**0.606**	21.680	11.426	10.409	24.049	22.855
17	100	5	U[1,3]	U[1,3]	**0.358**	22.356	10.932	8.389	21.114	20.745
18	100	5	U[1,3]	U[1,5]	**1.633**	19.987	8.679	4.445	10.117	8.624
19	100	5	U[1,5]	U[1,3]	**0.038**	10.097	3.805	5.547	55.966	85.335
20	100	5	U[1,5]	U[1,5]	**1.424**	12.988	3.970	9.994	70.491	103.699
21	100	10	U[1,3]	U[1,3]	**0.196**	13.760	4.757	8.880	57.547	93.224
22	100	10	U[1,3]	U[1,5]	**0.000**	24.442	9.135	10.445	76.347	76.572
23	100	10	U[1,5]	U[1,3]	**0.000**	21.555	12.692	10.381	57.798	61.016
24	100	10	U[1,5]	U[1,5]	**0.000**	19.237	16.630	10.636	56.870	55.118
25	150	3	U[1,3]	U[1,3]	**0.346**	7.558	1.552	4.237	14.764	20.641
26	150	3	U[1,3]	U[1,5]	**1.082**	9.290	1.422	4.147	19.185	25.854
27	150	3	U[1,5]	U[1,3]	2.444	9.688	**1.588**	4.445	10.634	10.013
28	150	3	U[1,5]	U[1,5]	**0.000**	22.399	10.848	11.766	37.484	39.753
29	150	5	U[1,3]	U[1,3]	**0.294**	17.891	9.886	9.012	24.015	26.185
30	150	5	U[1,3]	U[1,5]	**0.376**	15.915	9.423	6.731	21.158	19.259
31	150	5	U[1,5]	U[1,3]	**0.195**	8.184	1.460	5.657	100.431	120.425
32	150	5	U[1,5]	U[1,5]	**0.293**	11.679	3.423	6.651	108.273	131.338
33	150	10	U[1,3]	U[1,3]	**0.207**	11.415	4.411	8.224	99.735	112.116
34	150	10	U[1,3]	U[1,5]	**0.000**	16.979	11.298	13.128	109.914	112.294
35	150	10	U[1,5]	U[1,3]	**0.000**	19.977	11.316	12.820	93.191	92.266
36	150	10	U[1,5]	U[1,5]	**0.000**	18.771	11.849	8.524	74.281	72.358
37	200	3	U[1,3]	U[1,3]	**1.630**	6.452	2.208	5.303	33.018	36.855
38	200	3	U[1,3]	U[1,5]	**1.040**	7.686	1.409	1.944	14.478	15.362
39	200	3	U[1,5]	U[1,3]	1.468	8.773	**1.208**	2.032	6.560	7.435
40	200	3	U[1,5]	U[1,5]	**0.261**	11.559	6.664	7.480	25.463	25.101
41	200	5	U[1,3]	U[1,3]	**0.349**	18.597	9.505	8.519	26.703	25.919
42	200	5	U[1,3]	U[1,5]	**1.158**	12.703	8.002	5.580	17.015	15.427
43	200	5	U[1,5]	U[1,3]	**0.289**	11.046	1.588	4.459	121.572	129.520
44	200	5	U[1,5]	U[1,5]	**0.166**	8.658	2.636	6.837	115.519	122.766
45	200	10	U[1,3]	U[1,3]	**0.821**	11.867	2.628	8.531	101.831	106.480
46	200	10	U[1,3]	U[1,5]	**0.000**	21.755	9.073	10.792	126.768	130.429
47	200	10	U[1,5]	U[1,3]	**0.000**	16.087	7.842	8.547	100.610	101.764
48	200	10	U[1,5]	U[1,5]	**0.000**	13.633	8.815	8.968	79.681	80.035
g_ave					**0.552**	15.749	6.854	7.398	45.117	53.218

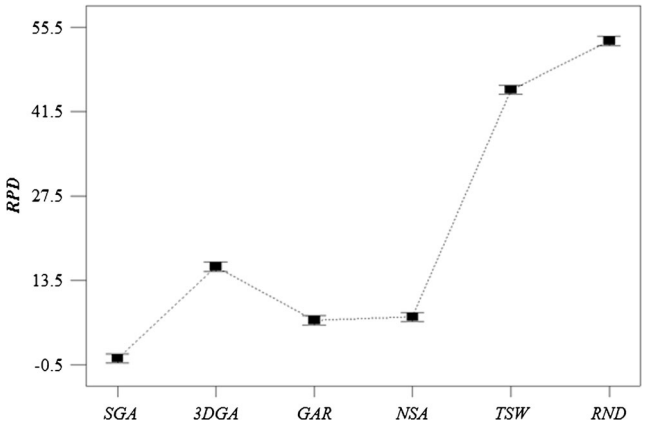

Fig. 9 Means plot with 95 % LSD intervals for large-sized instances

combined with the SPTCH-based decoding rule in approaching the HFSBP problem with machine eligibility restrictions.

In order to infer some statistical conclusion over the difference observed among the tested algorithms, an ANOVA analysis has been performed through Design Expert® 7.0.0 version commercial tool, calculating LSD intervals at 95 % confidence level for the RPDs connected to each optimization procedure. The corresponding chart is reported in Fig. 8.

The chart clearly shows as both SGA and NSA outperform the other algorithms in a statistically significant manner. Nevertheless, no conclusion can be drawn with reference to the difference observed between such algorithms, though SGA reports a lower average RPD. On the other hand, the narrow difference of performance between the two algorithms should depend on the small number of solutions characterizing the solution domain.

In order to evaluate the computational efficiency of the algorithms to be compared, the CPU time required by each method has been measured. Numerical results are reported in Table 8 and, for the sake of brevity, they are grouped according to the number N of jobs, which is clearly the most influent parameter affecting the computational burden. Due to its simple decoding procedure, the quickest metaheuristics is 3DGA, whose CPU times result quite smaller than the other competitors. Nevertheless, 3DGA seems to be the less effective procedure for solving the HFSBP problem at hand, as shown in Table 7 and Fig. 8. With reference to the other methods, no significant difference can be noticed in terms of time to convergence. However, it should be stated that SGA is on average faster than NSA, which represents the main competitor under the quality of solutions viewpoint.

6.2 Comparison analysis for large-sized problems

As for large-sized instances, the comparison between the proposed SGA and the alternative procedures has been performed on the basis of a benchmark composed by 48 classes of problems, which include 480 different instances. Differently from the previous analysis, four levels (50, 100, 150, 200) have been considered for the number N of jobs, in order to better fit with the real-world manufacturing context. The key performance indicator to compare the alternative metaheuristics is the relative percentage deviation from the best (See Eq.14), where $BEST_{sol}$ is the lowest makespan value among those gaining from every optimization procedure. Table 9 reports the average RPD values, highlighting in boldface the best results for each class of problem.

Numerical results reported in Table 9 clearly show the superiority of SGA in solving the large-sized instances of the HFSBP problem under investigation. The proposed algorithm ensures the best average RPD in 45 out of 48 classes of problems, and the grand average indicator (g_ave) further confirms the outperformance of the proposed SGA. It is also worth noting that for 15 classes of problems, the average RPD obtained by SGA is equal to zero, thus confirming the ability of the proposed SGA in achieving the best makespan among the overall set of tested procedures.

An ANOVA analysis has been carried out with the aim of defining LSD intervals at 95 % confidence level for RPDs obtained by each algorithm. Figure 9 related chart confirms the effectiveness of the proposed SGA, which statistically outperforms the other algorithms. As concerns the alternative optimization methods, an evident performance decrease of RND can be observed in Table 9 with respect to the small-sized instances, though it makes use of the same encoding/decoding strategy adopted by SGA. On the other hand, GAR significantly improves its performance in terms of quality of solutions, thus reaching a comparable efficacy with respect to the NSA method.

Under the computational burden perspective, Table 10 puts in evidence the average CPU times needed by each procedure, expressed in seconds and grouped by the number of jobs N. 3DGA is still the fastest method, while SGA gets slightly worse if compared to the case of small-sized instances. Whether SGA and GAR should be compared in terms of CPU times, the average advantage of the latter algorithm should be equal about to 10 %, regardless of any class of problems, as confirmed by Table 10. In fact, the CPU time

Table 10 average CPU times (in seconds) for large instances

N	SGA	3DGA	GAR	NSA	TSW	RND
50	382.0	321.2	342.6	361.0	372.2	358.0
100	779.5	635.3	712.1	736.0	819.8	730.9
150	1,184.5	959.6	1,088.4	1,116.6	1,203.9	1,112.4
200	1,618.0	1,321.5	1,488.4	1,524.2	1,613.9	1,539.3
Average	991.0	809.4	907.9	934.5	1,002.5	935.2

difference between SGA and GAR goes from less than 1 min (39 s for N=50) to about 2 min (130 s for N=200). In conclusion, it could be argued that the slight CPU time difference between SGA and the other algorithms may be ignored in light of the significant performance improvement that SGA assures in terms of makespan.

7 Conclusions

In this paper, a properly developed smart decoding-based genetic algorithm (SGA) has been employed for minimizing makespan for a hybrid flow shop scheduling problem with unrelated batch processors, limited machine capacity, and machine eligibility restrictions. The proposed technique makes use of a single permutation encoding scheme that, through a proper decoding procedure inspired to earlier literary works, can simultaneously manage job sequencing and allocation, batching of identical jobs, and machine eligibilities. The proposed SGA procedure was equipped with a properly developed crossover operator named enclosed order crossover (EOX), which tends to keep unchanged any set of identical jobs included in a given solution to be perturbed.

After an extensive calibration phase, the best combination of genetic parameters for the proposed algorithm has been selected. Then, a comparison campaign based on two separate benchmarks involving both small- and large-sized problems has been fulfilled in order to test the performance of SGA with respect to four different metaheuristics presented in literature and a regular random sampling algorithm. To this aim, an ANOVA analysis focusing on a statistical validation of the obtained outcomes has been performed. Numerical results highlighted the effectiveness of SGA in approaching the HFSBP problem under both quality of solutions and computational burden viewpoints. Results coming from the small-sized benchmark revealed a slight outperformance of SGA with respect to the alternative procedures. The global optima retrieved by the devised MILP model for a portion of the provided instances represented a valid support to the validation of the proposed metaheuristics. The narrow space of solutions characterizing the small-sized scenario problems surely justifies the small difference of performance among the alternative metaheuristics.

As for large-sized problems, SGA significantly outperformed the other metaheuristics. A slight gap of SGA under the CPU time perspective can be ignored whether the significant improvement of performance in terms of makespan reduction is considered.

Further research should involve the application of the developed SGA to other variants of hybrid flow shop problem. For instance, the hybrid flow shop problem with stage skipping or sequence-dependent setup times could be approached through the proposed algorithm.

8 Nomenclature

Indices	
j, k	Indices of jobs
i	Index of stages
m	Index of machines
Parameters/sets	
N	Number of jobs
I	Number of stages
M_i	Number of machines at stage i
S_j	Set of jobs identical to job j (i.e., belonging to the same family)
R_{ji}	Set of machines pertaining to stage i eligible for processing job j
P_{jim}	Processing time of job j on machine m at stage i
BS_{im}	Maximum batch size allowed on machine m at stage i
B	A big number
Binary variables	
Y_{jim}	1, If job j is worked on machine m at stage i; 0, otherwise
Q_{jki}	1, If jobs j and k are worked on the same machine at stage i, with j preceding k; 0, otherwise
W_{jki}	1, If jobs j and k are worked within the same batch on a given machine at stage i; 0, otherwise
Continuous variables	
C_{ji}	Completion time of job j at stage i
$C_{\max}$	Makespan
SGA decoding notation	
π	An encoded solution to the problem
π_i	Sequence of jobs to be processed at stage i
$\pi_i(l)$	lth job of sequence π_i
γ	Last job worked by a given machine at a certain step of the decoding phase
$bestmc$	Current best machine for processing a given job
$EC_{\pi_i(l),i,m}$	Expected completion time of job $\pi_i(l)$ on machine m at stage i
$C_{\pi_i(l),i}$	Actual completion time of job $\pi_i(l)$ at stage i
SGA parameters	
P_{size}	Population size
p_{cr}	Crossover probability
p_m	Mutation probability
$\mathrm{MAX}_{\mathrm{dist}}$	Maximum distance between two cut points for EOX crossover
$\mathrm{MAX}_{\mathrm{shift}}$	Maximum gene shift length for mutation

References

1. Ruiz R, Vázquez-Rodríguez JA (2010) The hybrid flow shop scheduling problem. Eur J Oper Res 205:1–18
2. Gupta JND (1988) Two-stage hybrid flow shop scheduling problem. J Oper Res Soc 39(4):359–364

3. Gourgand M, Grangeon N, Norre S (1999) Metaheuristics for the deterministic hybrid flow shop problem. Proc international conference on industrial engineering and production management, IEPM'99. FUCAMINRIA, Glasgow, pp 136–145
4. Sherali HD, Sarin SC, Kodialam MS (1990) Models and algorithms for a two-stage production process. Prod Plann Contr 1(1):27–39
5. Brah SA, Hunsucker JL (1991) Branch and bound algorithm for the flow-shop with multiple processors. Eur J Oper Res 51(1):88–99
6. Carlier J, Néron E (2000) An exact method for solving the multi-processor flowshop. RAIRO, Oper Res 34(1):1–25
7. Guinet AGP, Solomon MM (1996) Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. Int J Prod Res 34(6):1643–1654
8. Brah SA, Loo LL (1999) Heuristics for scheduling in a flow shop with multiple processors. Eur J Oper Res 113:113–122
9. Botta-Genoulaz V (2000) Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. Int J Prod Econ 64:101–111
10. Yang J (2011) Minimizing total completion time in two stage hybrid flow shop with dedicated machines. Comput Oper Res 38:1045–1053
11. Wang S, Liu M (2013) A heuristic method for two-stage hybrid flow shop with dedicated machines. Comput Oper Res 40:438–450
12. Xiao W, Hao P, Zhang S, Xu X (2000) Hybrid flow shop scheduling using genetic algorithms. Proc 3rd world conference on intelligent control and automation, Hefei, pp 537–541
13. Ruiz R, Maroto C (2006) A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. Eur J Oper Res 169:781–800
14. Yaurima V, Burtseva L, Tchernykh A (2009) Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. Comput Ind Eng 56:1452–1463
15. Wardono B, Fathi Y (2002) A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. Eur J Oper Res 155:380–401
16. Engin O, Döyen A (2004) A new approach to solve hybrid flow shop scheduling problems by artificial immune system. Future Gener Comp Sy 20:1083–1095
17. Ying KC, Lin SW (2006) Multiprocessor task scheduling in multi-stage hybrid flowshops: an ant colony system approach. Int J Prod Res 44(16):3161–3177
18. Tseng CT, Liao CJ (2008) A particle swarm optimization for hybrid flow-shop scheduling with multiprocessor tasks. Int J Prod Res 46(17):4655–4670
19. Singh MR, Mahapatra SS (2012) A swarm optimization approach for flexible flow shop scheduling with multi-processor tasks. Int J Adv Manuf Technol 62:267–277
20. Tavakkoli-Moghddam R, Sfaei N, Sassani F (2009) A memetic algorithm for the flexible flow line scheduling problem with processor blocking. Comput Oper Res 36:402–414
21. Mirsanei HS, Zandieh M, Moayed MJ, Khabbazi MR (2011) A simulated annealing approach to hybrid flow shop scheduling with sequence-dependent setup times. J Intell Manuf 22:965–978
22. Kurz ME, Askin RG (2004) Scheduling flexible flow lines with sequence dependent setup times. Eur J Oper Res 159:66–82
23. Mathirajan M, Sivakumar AI (2006) A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. Int J Adv Manuf Technol 29:990–1001
24. Attar SF, Mohammadi M, Tavakkoli-Moghaddam R (2013) Hybrid flexible flowshop scheduling problem with unrelated parallel machines and limited waiting times. Int J Adv Manuf Technol 68:1583–1599
25. Webster S, Baker KR (1995) Scheduling groups of jobs on a single machine. Oper Res 43(4):692–703
26. Yuan JJ, Yang AF, Cheng TCE (2004) A note on the single machine serial batching scheduling problem to minimize maximum lateness with identical processing times. Eur J Oper Res 158:525–528
27. Shen L, Buscher U (2012) Solving the serial batching problem in job shop manufacturing systems. Eur J Oper Res 221:14–26
28. Azizoglu M, Webster S (2001) Scheduling a batch processing machine with incompatible job families. Comput Ind Eng 39:325–335
29. Chang PC, Wang HM (2004) A heuristic for a batch processing machine scheduled to minimize total completion time with non-identical job sizes. Int J Adv Manuf Technol 24:615–620
30. Mathirajan M, Sivakumar AI, Chandru V (2004) Scheduling algorithms for heterogeneous batch processors with incompatible job-families. J Intell Manuf 15:787–803
31. Mönch L, Balasubramanian H, Fowler JW, Pfund ME (2004) Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. Comput Oper Res 32:2731–2750
32. Amin-Naseri MR, Beheshti-Nia MA (2009) Hybrid flow shop scheduling with parallel batching. Int J Prod Econ 117:185–196
33. Costa A, Cappadonna FA, Fichera S (2013) A dual encoding-based meta-heuristic algorithm for solving a constrained hybrid flow shop scheduling problem. Comput Ind Eng 64(4):937–958
34. Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor
35. Rajendran C, Chaundhuri D (1992) A multi-stage parallel processor flowshop problem with minimum flowtime. Eur J Oper Res 57:111–122
36. Michalewicz Z (1994) Genetic algorithms + data structures=evolution programs, 2nd edn. Springer, Berlin
37. Davis L (1985) Applying adaptive algorithms to epistatic domains. Proc ninth international joint conference on artificial intelligence. IJCAI-85, Los Angeles, pp 162–164
38. Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22(1):5–13
39. Montgomery D (2007) Design and analysis of experiments, 5th edn. Wiley, New York