Int J Adv Manuf Technol (2013) 69:2799–2817
DOI 10.1007/s00170-013-5221-5

ORIGINAL ARTICLE

A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times

A. Costa · F. A. Cappadonna · S. Fichera

Received: 11 April 2013 / Accepted: 24 July 2013 / Published online: 18 August 2013
© Springer-Verlag London 2013

Abstract In this paper, the unrelated parallel machine scheduling problem with sequence-dependent setup times and limited human resources is addressed with reference to the makespan minimisation objective. Workers needed for setup operations are supposed to be a critical resource as their number is assumed to be lower than the number of workstations. In addition, each worker is characterised by a specific skill level, which affects setup times. Firstly, a mathematical model able to optimally solve small instances of the problem in hand is illustrated. Then, to deal with large-sized test cases, three different optimisation procedures equipped by different encoding methods are proposed: a permutation encoding-based genetic algorithm (GA), a multi-encoding GA and a hybrid GA that properly moves from a permutation encoding to a multi-encoding once a given threshold on the number of generations is achieved. In particular, three different hybrid GAs featured by different encoding switch thresholds were implemented. An extensive benchmark including both small- and large-sized instances was generated with the aim of both calibrating the genetic parameters and comparing the alternative GAs through distinct ANOVA analyses. Numerical results confirm the effectiveness of the hybrid genetic approach whose encoding switch threshold is fixed to 25 % of the overall generations. Finally, a further analysis concerning the impact of multi-skilled workforce on the performance of both production system and optimisation strategy is presented.

Keywords Parallel machines · Sequencing · Worker allocation · Genetic algorithms · Linear programming

A. Costa (✉) · S. Fichera
Dipartimento di Ingegneria Industriale, University of Catania, Catania, Italy
e-mail: costa@diim.unict.it

F. A. Cappadonna
Dipartimento di Ingegneria Elettrica, Elettronica e Informatica, University of Catania, Catania, Italy

1 Introduction

During the last few decades, the parallel machine problem has encountered an increasing interest in literature, as many real-world manufacturing systems can be described by means of such theoretical model [32]. Based on the regular parallel machine production environment, each job $j=1, \ldots, N$ has to be processed by only one machine $m=1, \ldots, M$; each workstation cannot process more than one job at a time. In addition, pre-emption is not allowed, i.e. each job cannot leave a particular machine until its processing is finished.

A pioneer study concerning the identical parallel machine scheduling problem has been carried out by Lenstra et al. [23] who demonstrated as makespan minimisation in such a kind of production system a nondeterministic polynomial (NP)-hard problem, even in the case of only two machines. Since then, various studies involving the use of heuristic techniques for tackling this scheduling issue have been presented. Cheng and Gen [9] proposed a Memetic Algorithm for minimising the maximum weighted absolute lateness. Cochran et al. [10] developed a two-stage Multi-population Genetic Algorithm with reference to various concurrent objectives: makespan, total weighted tardiness and total weighted completion time. Anghinolfi and Paolucci [1] presented a hybrid metaheuristic integrating features from a tabu search, simulated annealing (SA) and variable neighbourhood search with the aim of minimising the total tardiness. Recently, Gokhale and Mathirajan [14] proposed five different heuristic techniques for minimising the total weighted flowtime for a production system with identical parallel machines operating within an automotive manufacturing firm.

As concerns the most of recent literature focusing on unrelated parallel machines scheduling problem, job processing times are supposed to be machine dependent. In such kind of problems, assignment of jobs to workstations represents a crucial issue for determining the overall performances of the production system along with job sequencing issue. Thus, the degree of complexity gets considerably higher with respect to

the identical machine case. Kim et al. [21] coped with a batch-scheduling problem for an unrelated parallel machine system observed within a semiconductor-manufacturing environment. They developed a two-level heuristic and a SA for minimising the total weighted tardiness and compared those methods with two dispatching rules, namely the Earliest Weighted Due Date and the Shortest Weighted Processing Time. Pereira Lopes and Valério de Carvalho [30] studied an unrelated parallel machine system with sequence-dependent setup times, machine availability dates and jobs release dates with the aim of minimising the total weighted tardiness. They followed a branch-and-price approach elaborating a proper procedure able to solve instances of up to 50 machines and 150 jobs within a reasonable computational time. An unrelated parallel machine scheduling problem with sequence-dependent setup times was tackled also by Tavakkoli-Moghaddam et al. [35]. In addition, they considered precedence constraints among jobs and developed a two-step linear programming model, together with a genetic algorithm (GA) for minimising the number of tardy jobs primarily and the total completion time secondly. A GA-based approach has also been adopted by Vallada and Ruiz [37] with the aim of minimising makespan for an unrelated parallel machine scheduling problem with sequence-dependent setup times. After a proper calibration phase, they carried out an extensive comparison campaign involving both small and large instances to assess the performance of the proposed metaheuristic against a Mixed Integer Linear Programming (MILP) model and other algorithms arisen from literature, for small- and large-sized problems, respectively. The use of a GA approach for minimising makespan in an unrelated parallel machine production system has also been investigated by Balin [3] who developed a specific crossover operator along with a new optimality criterion and assessed the performance of the obtained algorithm against the Longest Processing Time rule. A new approach for the resolution of the unrelated parallel machine scheduling issue, though not involving sequence dependent setup times, has been given recently by Fanjul-Peyro and Ruiz [12] who developed several hybrid heuristic methods and algorithms for reducing the size of a given problem before solving it through a linear programming model. Fanjul-Peyro and Ruiz [13] have also studied two generalisations of the unrelated parallel machine model, namely the Not All Machine (NAM) and Not All Jobs variants; the former model allows one or more machines to be excluded by the production shop, whereas the latter handles as unnecessary some jobs to be processed. In both cases, a MILP model is given. In addition, three different algorithms are presented for the NAM model, as the complexity of the problem makes it difficult finding an exact solution within a polynomial time.

Whenever a scheduling theoretical model should be applied to the real-world manufacturing environment, the human factor should be taken into account to generate any reliable schedule. Indeed, a well-established literary trend known as Dual-Resource Constrained (DRC) focuses on the study of production systems wherein capacity constraints depend on both machines and human operators. Starting from the seminal work of Nelson [26], DRC systems have received many attentions by literature over the years. Treleven [36] and Hottenstein and Bowman [16] have thoroughly categorised the main topics and issues concerned with such manufacturing models. Recently, Xu et al. [38] provided a comprehensive review regarding the research works dealing with DRC systems presented during the last two decades.

Basically, the effect of human workers on DRC systems may be studied under two different viewpoints, i.e. worker flexibility and worker assignment. As far as the worker flexibility is concerned, it consists in taking into account the different technical skills related to every worker. Part of the relevant literature used a so-called flexibility level to consider the worker flexibility in manufacturing environments; it is a numerical index representing the number of machines where a worker may operate [11, 22]. Another research trend considered the so-called skill levels, i.e. numerical parameters able to characterise the efficiency of each worker in respecting specific quality requirements or productivity targets. This latter topic appears to be extremely relevant in fact, whenever manual operations should be executed, e.g. setups or removals, it could be crucial taking into account the impact of the different workers' skills on the production system performance. Kim and Bobrowsky [20] considered the crew skills as a random factor affecting setup times on a job-shop production system. Askin and Huang [2] proposed a mixed integer goal programming model for creating worker teams with high team synergy and proper technical and administrative skills in a cellular manufacturing environment. The same kind of production system was addressed by Norman et al. [27] who developed a mixed integer programming problem for assigning workers to manufacturing cells, to maximise the effectiveness of the organisation, also emphasising the possibility of enhancing skill levels through additional training. Pan et al. [29] tackled a job-shop scheduling problem pertaining to the precision engineering industry, in which each job requires specific technical skills as well as a certain grade of experience and seniority to be performed.

With regard to DRC systems entailing the worker assignment issue, two main trends may be noticed in the relevant literature as well. A first research area is concerned with the study of manufacturing configurations where each job processing time depends on the number of operators assigned to the machines where it has to be processed. Hu [17–19], Chaudhry and Drake [8] and Chaudhry [7] applied the same strategy to an identical parallel machine scheduling problem. Celano et al. [5] studied a mixed-model assembly line sequencing problem where the total line stoppage time may be reduced by applying specific help policies among workers. Further literary research involve the study of production systems where workers represent a critical resource, i.e. where the

number of operators is significantly lower than the number of available workstations. Celano et al. [4] studied the effect of reducing the human resource capacity on unrelated parallel manufacturing cells through an integrated simulation framework, also taking into account several performance measures. Zouba et al. [39] addressed the problem of scheduling jobs on an identical parallel machine manufacturing environment with less workers than machines, so that operators have simultaneously to supervise several machines.

The present paper focuses on the unrelated parallel machine scheduling problem with sequence dependent setup times along with limited and multi-skilled human resource that, to the best of our knowledge, never has been faced by literature. Making use of the well-known three-field notation $\alpha \mid \beta \mid \gamma$ proposed by Pinedo [31], it may be classified as $Rm|S_{ikjl}|C_{\max}$ wherein setup time of each job l depends on the machine i, on the skills of the operator k and on the preceding job j. It is worth pointing out that in the proposed sequencing-allocation problem, each worker is not assigned to a specific machine once and for all but, on the contrary, he may visit different machines along the production time horizon as the final schedule allocates each worker to a set of jobs and then jobs to machines.

A GA-based optimisation framework has been developed for the makespan minimisation of the aforementioned problem. As recognised by the body of literature over the past decades, GAs ensure a significance level of both efficacy and efficiency in solving NP-hard combinatorial scheduling problems. Furthermore, the aforementioned literary overview, which focuses on the parallel machine scheduling problem, highlights the extensive use of GAs for tackling such kind of topic [38].

In particular, three different hybrid GAs, each one involving two distinct encodings, have been compared with two regular single-encoding GAs in terms of makespan minimisation. Performance evaluation of the proposed metaheuristics was carried out by means of an ANOVA analysis [25] over a wide set of test cases. Finally, a further step of analysis highlights the impact of the multi-skilled human resources on the performance of the production system, as well as on the sequencing/allocation strategy of the proposed hybrid optimisation tool.

The remainder of the paper is organised as follows: in Section 2, a brief introduction to the proposed issue is presented; Section 3 deals with the MILP model describing the problem. In Section 4, all proposed GAs are discussed. Section 5 illustrates the calibration procedure applied to such metaheuristics, whereas in Section 6 numerical results arising from an extensive experimental campaign are presented. In Section 7, results obtained by taking into account multi-skilled human resources are compared with those pertaining to three different scenarios whose workers have identical skills. Finally, Section 7 concludes the paper.

2 Problem description

The proposed unrelated parallel machine problem with limited human resources can be stated as follows. A set of $j=1, \ldots, N$ jobs that has to be processed in a single production stage composed by $m=1, \ldots, M$ unrelated workstations, aiming at minimising the makespan, i.e. the maximum completion time among the scheduled jobs. Each job has to be processed by only one machine before it leaves the production system, and each machine cannot process more than one job at a time.

Setup operations performed on a given workstation by a single worker must precede each job processing on the same workstation. Setup times are sequence and machine dependent. Furthermore, a team $w=1, \ldots, W$ of workers, being $w \leq m$, is assumed to be involved just to setup operations. As a consequence, workers represent a critical resource as each setup operation required by a given job on a given machine could wait for a worker currently employed in another workstation for the setup of a different job. In addition, each worker is featured by a certain skill level, on the basis of which he is able to perform setup operations slower or faster than his colleagues. Thoroughly, setup time of a job l to be processed after a job j, on a specific machine i, by a worker k, may be defined as follows:

$$S_{ikjl} = \eta_k \cdot S_{ijl}, \tag{1}$$

where S_{ijl} is the standard setup time required by an average-skilled worker and $\eta_k \in [0.5,1.5]$ is a coefficient reflecting the skill level of operator kth. Expert workers are supposed to have η_k lower than 1 while novice workers are characterised by η_k greater than 1. It is worth pointing out that the way the skill levels have been modelled arises from the observation of a real manufacturing environment typified by different technologies like: injection moulding, compression moulding and HDPE pipe extrusion. To optimise the workforce utilisation and, at the same time, to be more flexible towards the changing customer demands, the observed firm adopts a sort of flexible labour management strategy. Instead of employing a number of workers equal to the total amount of production resources, the firm's policy aims to use a lower number of operators characterised by different skills, conforming to the different technologies installed in the shop floor.

3 MILP model

A first goal of the proposed research was the development of a MILP model, aiming both at optimally solving a set of small instances of the aforementioned problem and validating the performance of the provided GAs. The reported mathematical model includes a dummy job (denoted as job 0), which must be the first to be processed by all machines, though it has a

processing time equal to zero. Such approach is necessary to run setup times of each first job to be processed in a specific machine; thus, job 0 precedes each first job processed by each machine. The aforementioned mathematical model is reported as follows.

3.1 Indices

$h,j,l=0,1,\ldots,n$ Jobs
$i=1,2,\ldots,m$ Machines
$k=1,2,\ldots,w$ Workers

3.2 Parameters

T_{il} Processing time of job l on machine i
S_{ikjl} Setup time of job l performed by worker k on machine i, after machine i has processed job j
B A big number

3.3 Binary variables

X_{ikjl} 1 if job l is processed on machine i after job j and setup is performed by worker k; 0 otherwise
Q_{jl} Auxiliary variable for *either–or* constraint

3.4 Continuous variables

CS_l Setup completion time of job l
C_l Processing completion time of job l
$C_{\max}$ Makespan

3.5 Model

Minimise $C_{\max}$

Subject to:

$$\sum_{i=1}^{m}\sum_{k=1}^{w}\sum_{j=0}^{n} X_{ikjl} = 1 \qquad \forall l = 1,2,\ldots,n. \tag{2}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{w}\sum_{l=1}^{n} X_{ikjl} \le 1 \qquad \forall j = 1,2,\ldots,n. \tag{3}$$

$$\sum_{k=1}^{w}\sum_{l=1}^{n} X_{ik0l} \le 1 \qquad \forall i = 1,2,\ldots,m. \tag{4}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{w}\sum_{l=1}^{n} X_{ikll} = 0 \tag{5}$$

$$\sum_{k=1}^{w}\sum_{j=0}^{n} X_{ikjl} \ge \sum_{k=1}^{w}\sum_{h=1}^{n} X_{iklh} \qquad \forall i = 1,2,\ldots,m; l = 1,2,\ldots,n. \tag{6}$$

$$C_l - CS_l \ge \sum_{i=1}^{m}\sum_{k=1}^{w}\sum_{j=0}^{n} T_{il}\cdot X_{ikjl} \qquad \forall l = 1,2,\ldots,n. \tag{7}$$

$$CS_l - C_j \ge \sum_{i=1}^{m}\sum_{k=1}^{w} S_{ikjl}\cdot X_{ikjl} - B\cdot\left(1-\sum_{i=1}^{m}\sum_{k=1}^{w} X_{ikjl}\right) \qquad \forall j = 0,1,\ldots,n; l = 1,2,\ldots,n. \tag{8}$$

$$\begin{cases} CS_l - CS_j \ge \sum_{i=1}^{m}\sum_{\substack{h=0\\ h\neq j}}^{n} S_{ikhl}\cdot X_{ikhl} - B\cdot\left(2-\sum_{i=1}^{m}\sum_{h=0}^{n}\left(X_{ikhl}+X_{ikhj}\right)+Q_{jl}\right) \\ CS_j - CS_l \ge \sum_{i=1}^{m}\sum_{h=0}^{n} S_{ikhj}\cdot X_{ikhj} - B\cdot\left(2-\sum_{i=1}^{m}\sum_{h=0}^{n}\left(X_{ikhl}+X_{ikhj}\right)+1-Q_{jl}\right) \end{cases} \quad \begin{array}{l}\forall k = 1,2,\ldots w;\ \ j = 1,2,\ldots,n;\\ l = j+1, j+2,\ldots,n.\end{array} \tag{9}$$

$$C_0 = 0 \tag{10}$$

$$C_{\max} \ge C_j \quad \forall j = 1,2,\ldots,n. \tag{11}$$

$$X_{ikjl} \in \{0;1\} \quad \forall i = 1,2,\ldots,m; k = 1,2,\ldots,w;\ \ j = 0,1,\ldots,n; l = 1,2,\ldots,n. \tag{12}$$

$$Q_{jl} \in \{0;1\} \quad \forall j = 1,2,\ldots,n; l = j+1, j+2,\ldots,n; \quad (13)$$

Constraint (2) ensures each job is assigned to only one machine and is preceded by only one job, and only one worker performs its setup. Constraint (3) states that each job must precede at most one other job. Constraint (4) forces job 0 to precede at most one other job in each machine. Constraint (5) denotes that each job cannot precede itself. Constraint (6) ensures the feasibility of a job sequence for each machine: if job l precedes some other job, it must have a predecessor on the same machine. Constraint (7) states as, for a given job, the minimum time lag between the end of a setup task and the end of the corresponding processing task must be equal to the processing time required by the job itself. Constraint (8) ensures that if job l is processed immediately after job j on a given machine, the end of the processing task of job j and the end of the setup task of job l must be separated by a time interval equal to the setup time of job l, at least. The twofold constraints (9) handles the limited human resources: if setups of jobs j and l are performed by the same worker k, then setup of job j must be completed before setup of job l starts, or vice versa. Constraint (10) assigns a null completion time to job 0. Constraint (11) forces makespan to be equal to or greater than any job completion time. Finally, constraints (12) and (13) define the corresponding binary variables.

4 The proposed GAs

Three kinds of different metaheuristic procedures based on Gas have been developed to address the large-sized instances of the proposed problem.

GAs [15] are computational methods inspired by the process of natural evolution, which have largely been used to solve scheduling problems so far. Generally, a GA works with a set of solutions to the problem 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*, which is derived from the objective function value; thus, best solutions have greater chances of being selected. Crossover operator generates new solutions (*offspring*) by combining the genetic material of a couple of chromosomes (*parents*), whereas mutation operator generates a change into the genetic scheme of selected chromosomes, with the aim to avoid the procedure to remain trapped into local optima. The algorithm evolutionary process proceeds until a given stopping criterion is achieved.

Whenever an optimisation problem is addressed through evolutionary algorithms, the choice of a proper *encoding* scheme (i.e. the way a solution of a given problem is represented by a string of *genes*) plays a key role under the metaheuristics performance viewpoint; thus, the problem encoding strongly may affect both efficacy and efficiency of GAs [6]. In addition, a valid *decoding* procedure to be applied to every encoded solution needs to be provided.

In the following subsections, a detailed description of the GAs developed for solving the aforementioned unrelated parallel machine scheduling problem is reported.

4.1 Permutation-based GA

A first approach towards the resolution of the scheduling problem here investigated consisted in the development of a GA equipped with a permutation-based encoding scheme (PGA). In such optimisation procedure, each chromosome directly describes the order in which jobs have to be processed in the manufacturing stage, whereas both the jobs and the workers assignment issue are performed by means of a proper decoding procedure, based on the so-called list-scheduling method. A detailed description of such algorithm is reported in the following section.

4.1.1 Encoding/decoding scheme

In PGA, each solution is represented by a permutation string π of n elements, where n is the number of jobs to be scheduled. In detail, $\bar{l} = \pi(r)$ is the job in the rth position (r=1,2, … , n) of the considered permutation string; the unrelated parallel machine production system holds m machines and w workers, with $w \leq m$; $S_{ikj\bar{l}}$ denotes the time required by worker k (k=1, 2, … , w) to perform setup of job $\bar{l}$ on machine i (i=1, 2, … m), after machine i has processed job j (j=0, 1, … , n); $T_{i\bar{l}}$ is the processing time of job $\bar{l}$ on machine i. The decoding procedure considers jobs in the order they appear in the permutation and assigns them to the couple machine-worker that can complete them earlier than any other, according to the aforementioned list-scheduling method. Once all decoding steps for scheduling jobs preceding $\bar{l}$ in the permutation π have been completed, TM_i indicates the time at which machine i is ready to accept a new job; λ is the current job processed by machine i; TW_k denotes the time at which worker k is ready to start a new setup operation. At first decoding iteration: TM_i=0; λ=0; TW_k=0. Completion time $C_{\bar{l}}$ of job $\bar{l} = \pi(r)$ is calculated as follows:

$$C_{\bar{l}} = \min_{i,k} \left\{ E_{ik\bar{l}} \right\} \quad (14)$$

where, $E_{ik\bar{l}}$ indicates the estimated completion time of job $\bar{l} = \pi(r)$, whether it is processed on machine i and its setup is performed by worker k. Hence:

$$E_{ik\bar{l}} = \max\{TM_i; TW_k\} + S_{ik\lambda\bar{l}} + T_{i\bar{l}} \quad (15)$$

Then, denoting with i^* and k^* machine and worker to which job $\bar{l}$ is assigned, respectively (i.e., those able to minimise $E_{ik\bar{l}}$), quantities TM_{i^*} and TW_{k^*} are updated as follows:

$$TM_{i^*} = C_{\bar{l}} \quad (16)$$

$$TM_{k^*} C_{\bar{l}} - T_{i^*\bar{l}} \quad (17)$$

Finally, after the aforementioned procedure has been performed for the entire permutation string, the makespan is calculated according to the following formula:

$$C_{\max} = \max_{\bar{l}\in\pi(1),\ldots,\pi(n)} \left\{ C_{\bar{l}} \right\} \quad (18)$$

To provide a clear insight into the aforementioned encoding/decoding procedure, a brief example is reported. It consists of four jobs (n=4) to be scheduled on an unrelated parallel machine manufacturing system composed by three workstations (m=3) and characterised by having a workforce team with two workers (w=2) to be employed for setup operations. For the sake of simplicity, setup times are assumed to be sequence-independent. For each job j, Table 1 illustrates processing times as i (index of machine) changes, and setup times as i and k (indices of worker) change.

As the reader can notice, setup times concerning the first operator are always half of the corresponding times concerning the second one, as the two workers are supposed to be differently skilled and, in particular, the first worker has a greater level of expertise with respect to the second worker. Furthermore, it is worth highlighting that job processing times do not depend on the workers, as in the proposed model the human factor affects only setup operations. Whether solution π={4,1,2,3} is taken into account, Table 2 summarises the proposed decoding procedure applied to π. At each iteration r (r=1,2, … , n), the minimum value of $E_{ik\bar{l}}$, calculated according to Eq. (15), is denoted in italics. Both the corresponding machine and worker, denoted as i^* and k^*, are selected for processing job $\bar{l}=\pi(r)$, whereas TM_{i^*} and TW_{k^*} are updated according to Eqs. (16) and (17), respectively. Finally, a makespan equal to 11 occurs for the given solution, according to Eq. (18). Figure 1 shows the Gantt chart obtained by applying such decoding procedure.

Table 1 Processing and setup times for an example with n=4, m=3 and w=2

T_{ij}	j=1		j=2		j=3		j=4	
i=1	4		2		1		5	
i=2	3		5		8		2	
i=3	1		5		4		4	
S_{ikj}	k=1	k=2	k=1	k=2	k=1	k=2	k=1	k=2
i=1	3	6	5	10	2	4	2	4
i=2	1	2	1	2	5	10	3	6
i=3	1	2	5	10	2	4	3	6

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

r	$\bar{l}$	$E_{ik\bar{l}}$			i^*	k^*	TM_i			TW_k	
			k=1	k=2			i=1	i=2	i=3	k=1	k=2
1	4	i=1	7	9	2	1	0	5	0	3	0
		i=2	*5*	8							
		i=3	7	10							
2	1	i=1	10	10	3	2	0	5	3	3	2
		i=2	9	10							
		i=3	5	*3*							
3	2	i=1	*10*	14	1	1	10	5	3	8	2
		i=2	11	12							
		i=3	13	18							
4	3	i=1	13	15	3	2	10	5	11	8	7
		i=2	21	23							
		i=3	14	*11*							

Numbers in italics match the job completion times in Fig. 1

4.1.2 Selection, crossover and mutation operators for PGA

With regard to the selection mechanism, the well-known roulette-wheel scheme [24] has been considered, thus assigning to each solution a probability of being selected inversely proportional to makespan value. A position-based crossover [34] has been employed for generating new offspring from a couple of selected parents. In such procedure all selected genes from a parent are copied in the offspring, just preserving corresponding position and relative order; unselected genes are instead copied in the order they appear in the other parent, thus completing the structure of the new generated individual. Figure 2 shows an example of position-based crossover for an instance with n=10.

As far as the mutation procedure is concerned, a simple swap operator [28] has been chosen. Based on this technique, two genes randomly selected from the chromosome are mutually exchanged. However, the proposed algorithm has also been equipped with an *elitism* procedure, aiming to preserve the best two individuals of each generation from any alteration caused by crossover and mutation operators. Finally, a fixed number of makespan evaluations has been chosen as stopping criterion.

4.2 Multi-encoding GA

The main characteristic of the proposed PGA is reducing the computational burden by means of a basic problem encoding.

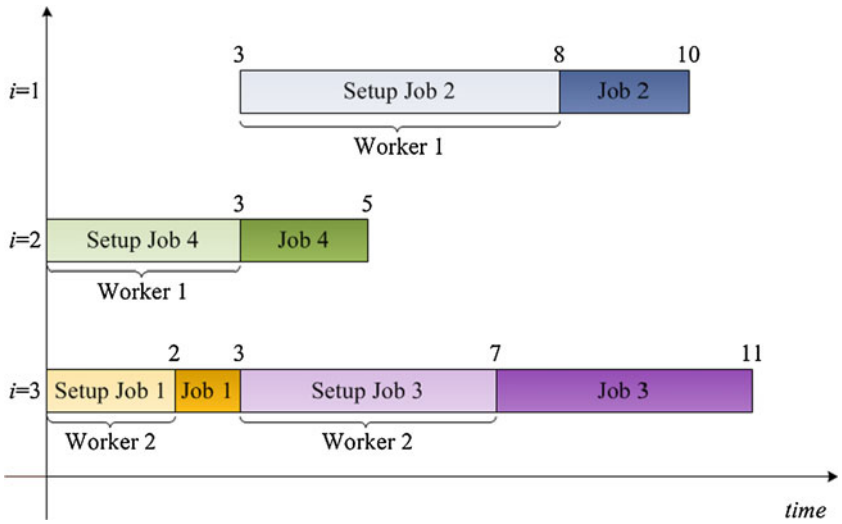

Fig. 1 Gantt chart obtained by PGA decoding procedure for $\pi=\{4,1,2,3\}$

Nevertheless, such algorithm moves over a search space that cannot embrace the overall set of solutions, due to the corresponding decoding procedure that, for each permutation, adopts a rigid criterion for assigning jobs to machines and workers. Hence, an alternative approach, towards the resolution of the proposed sequencing-allocation problem by means of metaheuristics, consisted in the development of a GA (MGA) equipped with a multi-stage encoding able to investigate a wider space of solution if compared with PGA. Three substrings compose such multi-stage encoding, the former being a regular permutation string to manage the job sequencing issue, and two adding strings necessary to drive the assignment of jobs to machines and workers, respectively. The following paragraph illustrates the proposed multi-stage encoding-based approach.

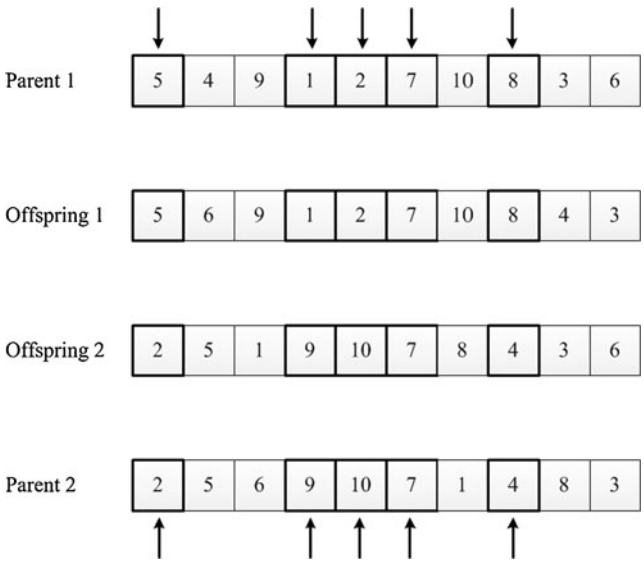

Fig. 2 Position-based crossover

4.2.1 Encoding/decoding scheme for MGA

To illustrate the encoding procedure exploited by MGA, the same nomenclature already defined for PGA may be adopted. Thus, assuming to have n jobs to be scheduled on an unrelated parallel machine system with m workstations and w workers ($w \leq m$), each chromosome is represented by the following substrings:

- A permutation string α of n elements;
- A string β of n integers ranging from 0 to m, driving the assignment of jobs to machines;
- A string γ of n integers ranging from 0 to w, driving the assignment of jobs to workers.

With regard to the initial population, it is worth highlighting that the maximum number of non-zero genes within each assignment substring (i.e. β and γ) is fixed a priori and equal to a fraction *pnz* of the total amount of genes every substring holds, i.e. n. In case the fraction of non-zero values is a real number, such number must be rounded up to the next integer. As for example, for a problem with ten machines, if *pnz* is equal to 1 % just one digit may assume a non-zero value within the β substring. Before introducing the decoding procedure, let $\bar{l}=\alpha(r)$ be the job on the rth position of the permutation α (r=1, 2, … , n); $\bar{i}=\beta(\bar{l})$ indicates the digit at position $\bar{l}$ of string β; $\bar{k}=\gamma(\bar{l})$ indicates the digit at position $\bar{l}$ of array γ. $S_{ikj\bar{l}}$ denotes the time required by worker k (k=1,2, … , w) to perform setup of job $\bar{l}$ on machine i (i=1,2,…m), after machine i has processed job j (j=0,1, … , n); $T_{i\bar{l}}$ indicates processing time of job $\bar{l}$ on machine i. The decoding procedure considers jobs in the order they appear in the permutation string α and uses corresponding information from arrays β and γ to perform the assignment of jobs to machines and workers; if no information is given by one or both arrays (i.e. if $\bar{i}$ =0 and/or $\bar{k}$ =0), the same list-scheduling

rule of PGA is used. Thus, let us assume to have completed all decoding steps for scheduling jobs preceding $\bar{l}$ in the permutation α. TM_i indicates the time at which machine i is ready to accept a new job; λ is the last job processed by machine i; TW_k denotes the time at which worker k is ready to start a new setup operation; first iteration states: $TM_i=0$, $\lambda=0$ and $TW_k=0$. Completion time $C_{\bar{l}}$ of job $\alpha(r)$ is calculated as follows:

$$C_{\bar{l}} = \begin{cases} E_{\bar{i}\bar{k}\bar{l}} & \text{if } \bar{i} \neq 0 \text{ and } \bar{k} \neq 0 \\ \min\limits_{k} \{E_{\bar{i}k\bar{l}}\} & \text{if } \bar{i} \neq 0 \text{ and } \bar{k} = 0 \\ \min\limits_{i} \{E_{i\bar{k}\bar{l}}\} & \text{if } \bar{i} = 0 \text{ and } \bar{k} \neq 0 \\ \min\limits_{i,k} \{E_{ik\bar{l}}\} & \text{if } \bar{i} = 0 \text{ and } \bar{k} = 0 \end{cases} \quad (19)$$

where $E_{ik\bar{l}}$ indicates the estimated completion time of job $\bar{l}$ if processed on machine i with setup performed by worker k. Its value is calculated according to the following formula:

$$E_{ik\bar{l}} = \max\{TM_i; TW_k\} + S_{ikl\bar{l}} + T_{i\bar{l}} \quad (20)$$

According to the decoding procedure above described, the job is assigned to machine $\bar{i}$ if $i \neq 0$ and to worker $\bar{k}$ if $k \neq 0$; in case either β or γ do not hold any specific value, job $\bar{i}$ allocation to machines and workers is managed by minimising the estimated completion time calculated through equation (20). Whether i^* and k^* denote machine and worker to which job $\bar{i}$ have to be assigned, TM_{i^*} and TW_{k^*} can be updated as follows:

$$TM_{i^*} = C_{\bar{l}} \quad (21)$$

$$TW_{k^*} = C_{\bar{l}} - T_{i^*\bar{l}} \quad (22)$$

Finally, after the aforementioned procedure has been performed for the entire set of jobs, the makespan is computed as follows:

$$C_{\max} = \max_{\bar{l} \in \alpha(1),\ldots,\alpha(n)} \left\{ C_{\bar{l}} \right\} \quad (23)$$

To better explain the proposed decoding procedure, the same example reported in Section 4.1.1 (see Table 1) can be taken into account. Supposing the solution to be decoded is $\pi=\{4,1,2,3|1,0,0,3|0,1,0,0\}$, Table 3 summarises the proposed decoding procedure. For each iteration r, the corresponding values of $E_{ik\bar{l}}$ are reported. Among these, the actual value of $C_{\bar{l}}$, calculated based on Eq. (19), is captured in italics; i^* and k^* are the selected machine and worker to process job $\bar{l}=\alpha(r)$, so that TM_i and TW_k can be updated conforming to Eqs. (21) and (22), respectively. Finally, a makespan equal to 10 is obtained by applying Eq. (23). Figure 3 shows the Gantt chart obtained by the proposed decoding procedure.

Table 3 MGA decoding procedure for $\pi=\{4,1,2,3|1,0,0,3|0,1,0,0\}$

r	$\bar{l}=\alpha(r)$	$\bar{i}$	$\bar{k}$	$E_{ik\bar{l}}$			i^*	k^*	TM_i			TW_k	
					$k=1$	$k=2$			$i=1$	$i=2$	$i=3$	$k=1$	$k=2$
1	4	1	0	$i=1$	*7*	9	1	1	7	0	0	2	0
				$i=2$	5	8							
				$i=3$	7	10							
2	1	0	1	$i=1$	14	17	3	1	7	0	4	3	0
				$i=2$	6	5							
				$i=3$	*4*	3							
3	2	0	0	$i=1$	14	19	2	2	7	7	4	3	2
				$i=2$	9	*7*							
				$i=3$	14	19							
4	3	3	0	$i=1$	10	16	3	1	7	7	10	6	2
				$i=2$	20	25							
				$i=3$	*10*	12							

Numbers in italics match the job completion times in Fig. 3

The proposed quantitative example puts in evidence the potential improvements that could arise from a more structured encoding scheme with respect to PGA. In fact, the same sequence of jobs elaborated by the PGA encoding/decoding generates a makespan equal to 11 (see Fig. 1), whereas MGA encoding/decoding yields a makespan equal to 10 times. For a given sequence of jobs α, a number of alternative solutions equal to $(m+1)^n(w+1)^n$ exist, thus emphasising the wider solution space that can be investigated by the MGA with respect to PGA.

4.2.2 Selection, crossover and mutation operators for MGA

Similarly being done by PGA, the same roulette wheel-based mechanism has been adopted as selection operator which assigns to each solution a probability of being selected inversely proportional to its makespan value. Conversely, crossover operator has been developed to separately run the mating between the three distinct parts composing the parents' structures (i.e. α, β and γ), with three distinct probabilities $p\alpha_{\text{cross}}$, $p\beta_{\text{cross}}$ and $p\gamma_{\text{cross}}$. Crossover between two parent α-substrings may be executed through a regular position-based operator conforming to PGA. As far as assignment substrings β and γ are concerned, a simple uniform crossover operator [33] has been employed. Such technique generates a couple of offspring by choosing, for each position, if the parents' genes will be swapped or not. Without loss of generality, Fig. 4 illustrates the application of such genetic operator for the β-substrings of two parents; bold-bordering genes are those to be swapped.

Similarly to crossover, mutation procedure has been performed by separately managing the three parts of the

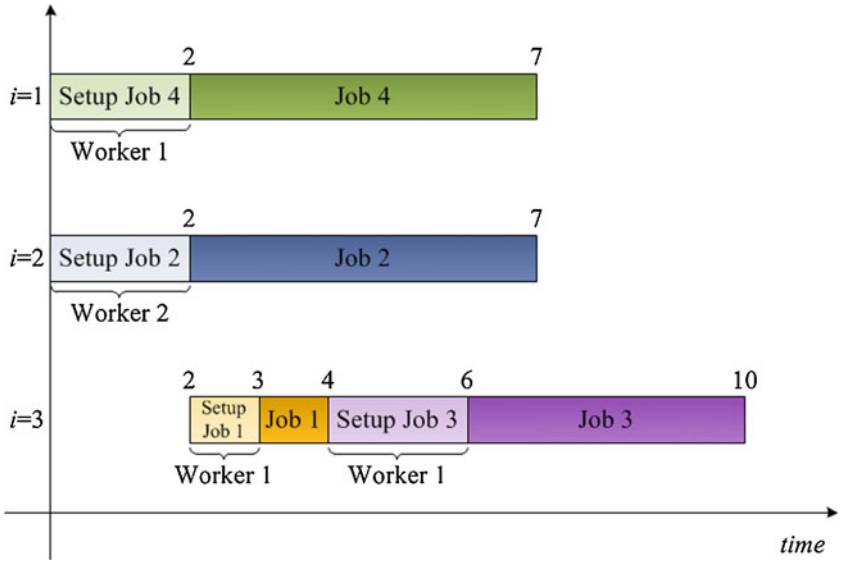

Fig. 3 Gantt chart obtained by MGA decoding procedure for $\pi = \{4,1,2,3|1,0,0,3|0,1,0,0\}$

chromosome, using three distinct probabilities $p\alpha_{mut}$, $p\beta_{mut}$ and $p\gamma_{mut}$. Mutation of the permutational substring has been performed through the same swap operator used in the PGA procedure. With reference to the assignment arrays, a simple uniform mutation operator [24] has been adopted. This technique randomly picks a gene and replaces it with a random value drawn from a uniform distribution in the provided domain (i.e. (0, m) for the first array, (0, w) for the second one). The same elitism procedure employed in PGA was embedded into the MGA so that, for each generation, the best two individuals are copied within the new population. The total number of makespan evaluations represents again the stopping criterion of the proposed MGA.

4.3 Hybrid GA

A hybrid genetic algorithm (HGA) which combines both the aforementioned meta-heuristics was developed as an alternative approach for solving the proposed unrelated parallel machine problem with limited human resources. In words, a twofold encoding-based GA has been arranged with the aim of combining the search and computational rapidity of PGA in the first phase and the ability of MGA in investigating a greater number of solutions in the second phase. PGA performs the first optimisation phase then, after a provided threshold is achieved and a proper encoding conversion procedure is executed, MGA starts until the stopping criterion is encountered. The encoding conversion procedure operates by adding the two assignment substrings to the chromosomes characterising the last population. Firstly, all values of new added assignment substring are set to zero; then, a fraction *pnz* of genes for each substring are replaced by elements drawn from an uniform distribution in the interval (1, m) or (1, w) for the first and the second substring, respectively. Figure 5 shows an example of such procedure for an instance having n=10, m=5 and w=2.

An elitism mechanism has been ensured whenever the population based on single-stage encoding must be converted into a multi-stage encoding-based population. In fact, the best two individuals included in the PGA final population are

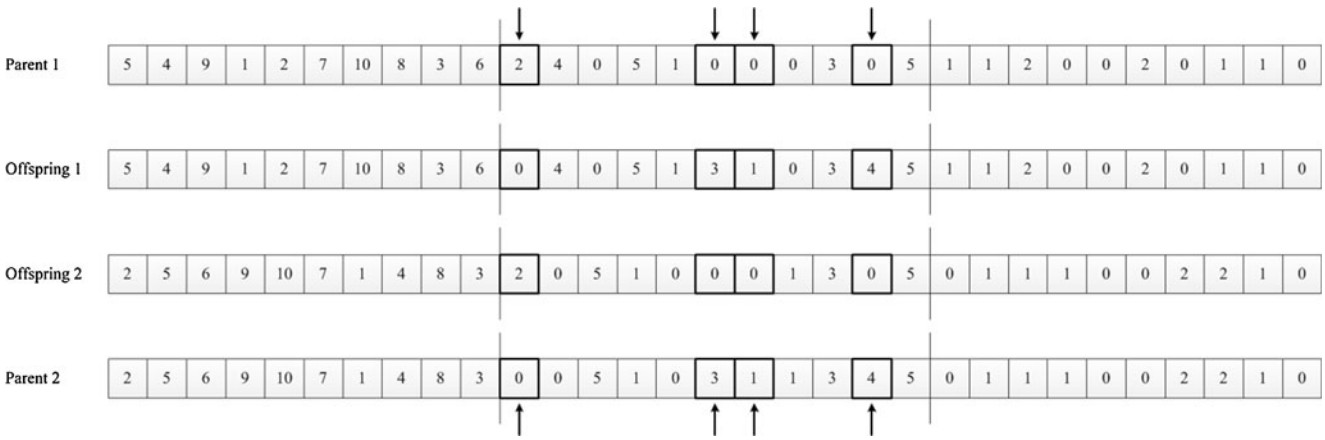

Fig. 4 Uniform crossover operator

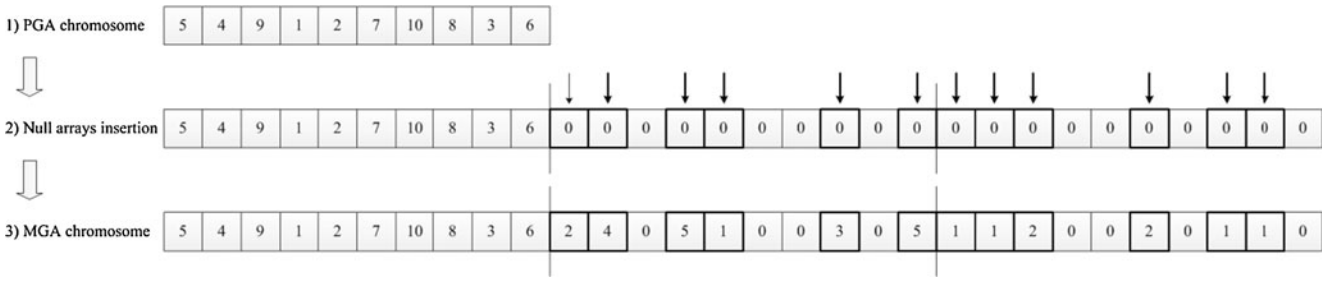

Fig. 5 Encoding conversion procedure

copied and updated into the new MGA population just by adding them two null assignment substrings (i.e. substring with only zeros). Once the encoding conversion procedure is completed, MGA cope with the second part of the optimisation process, until the total number of makespan evaluations is achieved.

5 Experimental calibration and test cases

Before carrying out an extensive comparison among the aforementioned metaheuristics, a comprehensive calibration phase has been fulfilled, with the aim of properly defining the best genetic parameters of the proposed algorithms. To this end, a benchmark of 100 problems has been created, combining number n of jobs, number m of machines and number w of workers in a full factorial design as illustrated in Table 4. The benchmark is characterised by $4\times2\times2=16$ small problems (with $n \leq 10$) and $7\times4\times3=84$ large problems (with $n \geq 20$), as to ensure the effectiveness of the tuned parameters over a wide range of test cases.

For each problem, one instance has been generated by extracting processing times from a uniform distribution in the range (1, 99). Setup times have been obtained by a two-steps procedure; first, a matrix S_{ijl} of both sequence-dependent and machine-dependent setup times was randomly generated by an uniform distribution U (1, 99). Then, for each worker k (k=1, 2, … , w), setup times have been multiplied by a factor η_k representing the skill weights of the operator itself, randomly extracted from the set {0.5, 0.75, 1, 1.25, 1.5}, thus obtaining the input parameters S_{ikjl}. A so configured set of skill weights is due to the observed real manufacturing environment wherein a highly skilled worker is able, on the average, to carry out a setup task in one third time with respect to a weakly skilled one.

Two separate calibration campaigns have been conducted for PGA and MGA, respectively. Table 5 illustrates parameters tested for each algorithm, and denotes in italics the best combination of values, chosen after an ANOVA analysis [25] at 95 % confidence level performed by means of Stat-Ease® Design-Expert® 7.0.0 commercial tool. The employed response variable was the relative percentage deviation (RPD), calculated according to the following formula:

$$RPD = \frac{GA_{\text{sol}} - BEST_{sol}}{BEST_{sol}} \times 100 \tag{24}$$

where GA_{sol} is the solution to be evaluated, i.e. corresponding to a GA with a specific setting of genetic parameters, and $BEST_{\text{sol}}$ is the best solution among those provided by the other GAs with

Table 4 Proposed benchmark of test problems

Scenario	Factor	Indices	Values
Small-sized problems	Number of jobs	n	7, 8, 9, 10
	Number of machines	m	4, 5
	Number of workers	w	2, 3
Large-sized problems	Number of jobs	n	20, 40, 60, 80, 100, 150, 200
	Number of machines	m	10, 12, 16, 20
	Number of workers	w	6, 8, 10

Table 5 Experimental calibration of PGA and MGA

Algorithm	Parameter	Notation	Values
PGA	Population size	P_{size}	20, 50, *100*
	Crossover probability	p_{cross}	0.2, *0.5*, 0.8
	Mutation probability	p_{mut}	0.05, *0.1*, 0.2
MGA	Population size	P_{size}	20, *50*, 100
	Percentage of non-zero genes of substrings β and γ (initial population)	*pnz*	*1 %*, 5 %, 10 %
	Crossover probability (permutation substring α)	$p\alpha_{\text{cross}}$	0.5, *0.8*
	Crossover probability (substring β)	$p\beta_{\text{cross}}$	*0.5*, 0.8
	Crossover probability (substring γ)	$p\gamma_{\text{cross}}$	*0.5*, 0.8
	Mutation probability (permutation substring α)	$p\alpha_{\text{mut}}$	0.05, *0.2*
	Mutation probability (substring β)	$p\beta_{\text{mut}}$	*0.05*, 0.2
	Mutation probability (substring γ)	$p\gamma_{\text{mut}}$	*0.05*, 0.2

Numbers in italics denote the most performing parameters to be adopted

different settings, for the same instance. GAs have been coded in MATLAB® language and executed on a 2-GB RAM PC powered by a dual-core 2.39-GHz processor. Stopping criterion was set to a total of 10,000 makespan evaluations. The use of such a stopping criterion arises from a small test campaign over a set of large-sized instances, wherein no any significant difference of performance has been observed by adopting 10,000 and 15,000 evaluations as stopping criterion, respectively.

As far as the HGA procedure is concerned, no any calibration has been provided since it consists of the combination of PGA with MGA; for such a reason, the same best parameters identified for the two distinct single-encoding algorithms (i.e. PGA and MGA) have been used for the two-phase hybrid metaheuristic. ANOVA outputs reported in Table 5 shows that MGA would require a smaller population than PGA; thus, the encoding conversion procedure exploited by HGA works by selecting the best performing 50 individuals of the final PGA population and by adding them the two provided assignment substrings, according to the procedure reported in Section 4.3.

6 Numerical examples and computational results

Once the calibration phase is completed, an extensive test campaign has been performed to compare the proposed GAs. In words, such comparison entails PGA, MGA and three variants of HGA, hereinafter HGA_{25}, HGA_{50} and HGA_{75}, whose encoding switch threshold p_{conv} was set to 25, 50 and 75 % of the total number of makespan evaluations chosen as stopping criterion, respectively. The same benchmark arrangement exploited for the tuning parameters analysis has been used. In particular, a total amount of ten different instances per each problem has been generated, using the same method employed in the calibration phase for defining both setup and processing times. Thus, a comparison analysis has been fulfilled over a total amount of 1,000 different instances. Indeed, as five runs characterised by five different random seeds have been considered for each metaheuristic algorithm, the effective number of investigated numerical instances is equal to 5,000.

Stopping criterion of each algorithm was set to 10,000 makespan evaluations and the same ICT computational equipment as described in Section 5 was employed for all metaheuristics. The following subsections reports the obtained results concerning small and large instances, respectively.

6.1 Small instances comparison analysis

Conforming to the calibration phase debated in Section 5, small-sized instances (i.e. those having $n \leq 10$) have been arranged in four scenario problems depending on the number of jobs (n=7, 8, 9 and 10). Each scenario problem includes 4 classes of problems, each one involving a different number of both machines (m=4 and 5) and workers (w=3 and 4). As each class of problems holds ten instances and each instance was replicated five times with different random seeds, 160×5=800 different runs have been considered. Both job and worker descriptors, e.g. processing times and setup times for each machine, have been randomly generated according to the same criteria discussed in Section 5.

The overall set of instances was optimally solved by means of the proposed MILP model, executed on an IBM® ILOG CPLEX Optimisation Studio 12.0 64-bit platform installed within a workstation powered by two quad-core 2.39-GHz processors with 24-Gb RAM. The response variable used for the comparison analysis was the RPD calculated according to Eq. (24). Considering that the optimality of the MILP-based approach $BEST_{sol}$ relates to the global optimum found by CPLEX® tool, Table 6 reports the average value of RPD obtained by each GA for a given class of problems (encompassing ten different instances replicated five times), together with the average computational time required, expressed in seconds. In parenthesis is reported the number of times out of 50 each algorithm hits the global optimum provided by the CPLEX® optimiser. Final row of Table 6 reports the grand averages (g_ave) in terms of RPD and number of optimal solutions concerning each metaheuristic.

Obtained results show that all the proposed metaheuristics are able to achieve a high level of performance in terms of both quality of solution and computational time. HGA_{25} seems to be very effective if compared with the other algorithms, although the difference of performance appears to be very narrow. To infer some statistical conclusion about the aforementioned difference of performance, a proper ANOVA analysis has been performed. Figure 6 reports the means plot with LSD intervals (α=0.05) obtained through Design-Expert® 7.0 platform.

The graph shows how MGA, HGA_{25} and HGA_{50} significantly outperform PGA and HGA_{75} under a statistical viewpoint, as LSD intervals of the winner algorithms are not overlapped with those of the looser ones. Nevertheless, ANOVA results do not allow drawing any conclusion with regards to the difference among the three best performing metaheuristics. However, this is an expected outcome due to the small size of problems addressed by the optimisation procedures. Larger-sized problems surely may highlight a significant performance difference among the GAs under examination.

6.2 Comparison for large instances

In this section a comparison among the proposed metaheuristics has been performed on the basis of an extended benchmark of larger-sized problems. Conforming to the calibration benchmark dealt with in Section 5, seven scenario problems depending on the number of jobs (n=20, 40, 60, 80, 100, 150 and 200) to be scheduled have been arranged. Each scenario problem includes four classes of problem at

Table 6 Average performances of GAs on small-sized instances

Problem ($n\times m\times w$)	Average RPD					Average CPU time (s)				
	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}
7×4×2	3.25 (18)	1.50 (31)	1.30 (35)	1.92 (29)	2.28 (27)	2.3	3.4	3.1	2.9	2.5
7×4×3	1.98 (27)	0.89 (38)	0.84 (33)	0.96 (37)	1.61 (33)	2.3	3.4	3.1	2.9	2.6
7×5×2	3.12 (18)	1.56 (27)	1.15 (32)	2.36 (21)	2.66 (22)	2.3	3.4	3.1	2.9	2.5
7×5×3	2.47 (21)	1.33 (30)	1.15 (32)	1.26 (27)	2.05 (22)	2.4	3.4	3.1	2.9	2.6
8×4×2	4.05 (16)	2.65 (25)	2.85 (20)	2.72 (21)	3.96 (16)	2.5	3.6	3.3	3.0	2.7
8×4×3	3.00 (22)	1.94 (31)	1.59 (35)	2.14 (29)	2.60 (26)	2.5	3.7	3.3	3.1	2.7
8×5×2	4.99 (15)	3.68 (18)	2.62 (21)	3.97 (19)	4.60 (16)	2.5	3.7	3.3	3.1	2.8
8×5×3	3.20 (22)	1.84 (28)	2.18 (26)	2.63 (26)	2.67 (25)	2.6	3.7	3.4	3.1	2.8
9×4×2	4.59 (13)	3.30 (19)	3.12 (20)	3.36 (18)	4.47 (14)	2.7	3.9	3.5	3.3	2.9
9×4×3	3.01 (15)	1.91 (26)	2.16 (19)	2.23 (23)	2.87 (16)	2.7	3.9	3.6	3.3	2.9
9×5×2	3.93 (18)	2.42 (29)	2.24 (30)	2.98 (25)	3.26 (20)	2.7	3.9	3.6	3.3	2.9
9×5×3	3.79 (13)	2.30 (24)	2.62 (19)	3.05 (17)	3.14 (14)	2.7	4.0	3.6	3.3	3.0
10×4×2	4.39 (10)	3.67 (12)	3.50 (12)	3.34 (14)	3.97 (13)	2.8	4.1	3.8	3.5	3.1
10×4×3	3.55 (18)	2.92 (23)	2.53 (29)	2.65 (23)	3.11 (22)	2.8	4.2	3.8	3.5	3.1
10×5×2	4.51 (12)	4.64 (12)	5.09 (15)	3.35 (16)	4.34 (14)	2.9	4.2	3.8	3.5	3.1
10×5×3	3.92 (12)	2.47 (23)	3.01 (19)	3.44 (17)	3.57 (14)	2.9	4.2	3.8	3.5	3.18
g_ave	3.61 (16.8)	2.44 (24.75)	2.37 (24.81)	2.65 (24.23)	3.20 (19.62)	2.6	3.5	3.2	2.9	3.8

varying number of machines (m=10, 12, 16 and 20). As far as the number of worker is concerned, it was varied according to three levels per each class of problems (w=6, 8 and 10). As ten randomly generated instances have been provided for class of problems and five replications with different random seeds have been arranged for each instance, a total amount of 840×5=4,200 runs have been considered for the proposed comparison analysis. Both job and worker descriptors, e.g. processing times and setup times for each machine, have been randomly generated according to the same criteria discussed in Section 5. Given the size of the problems included within this benchmark, MILP cannot achieve any global optimum. The RPD (see Eq. 24) has been taken into account as a performance indicator for the comparison in hand, the only difference being that $BEST_{sol}$ is the best solution among those obtained by the proposed metaheuristics for each instance. Table 7 reports the average RPD numerical results for each class of problems along with the related CPU time average percentage deviation with respect to PGA ($\Delta CPU_{\%PGA}$). Each row encompasses the averages results of 50 different runs (i.e. ten instances replicated five times). The last row shows the grand average of both RPD and $\Delta CPU_{\%PGA}$ over the entire set of investigated instances.

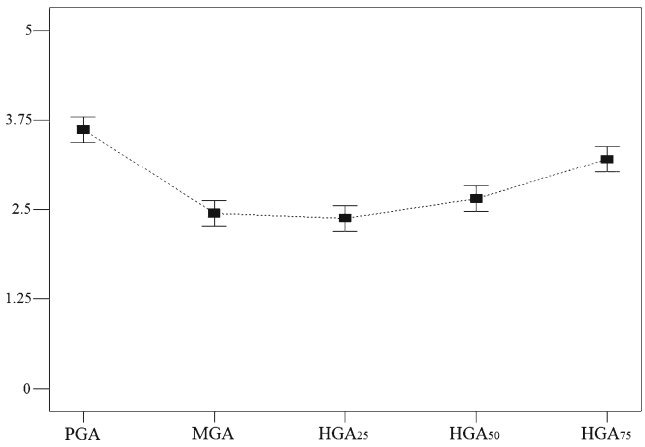

Fig. 6 Means plot and LSD intervals for small instances

HGA_{25} outperforms on the average the other competitors for optimising the proposed sequencing/allocation problem. This time, the advantage of using a hybrid approach clearly emerges, as all the three HGAs outperform both PGA and MGA. Of course PGA remains the best method under the computational time viewpoint because of the basic encoding scheme it adopts; however, average CPU times required by all other metaheuristics remain acceptable, especially in view of the complexity of problems solved. Figure 7 reports the means plot with 95 % confidence level LSD intervals.

Differently from the analysis concerning the small-sized scenario problems, it can be seen how the difference between HGA_{25} and other algorithms is statistically significant. MGA performance sensibly decreases if compared with the small-sized instances case.

Without loss of generality, Fig. 7 shows how all the three versions of HGA significantly outperform the two single encoding-based GAs (i.e. PGA and MGA). The reason of this outcome can be explained if the differences among exploration

Table 7 Average performances of GAs on large-sized instances

Problem ($n\times m\times w$)	Average RPD					Average $\Delta CPU_{\%PGA}$			
	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}	MGA (%)	HGA_{25} (%)	HGA_{50} (%)	HGA_{75} (%)
20×10×6	4.56	4.37	2.82	3.24	3.80	40	28	19	8
20×10×8	3.87	3.14	3.15	3.63	3.13	38	27	18	8
20×10×10	1.81	3.47	2.05	1.79	1.95	36	26	18	6
20×12×6	3.01	2.83	2.35	1.71	2.70	38	27	18	7
20×12×8	2.38	3.87	2.74	1.60	2.21	36	26	17	7
20×12×10	2.24	2.07	1.54	1.76	1.89	34	25	17	8
20×16×6	1.72	4.13	1.94	2.16	1.51	35	25	18	8
20×16×8	2.07	2.09	1.31	2.00	1.74	33	24	16	8
20×16×10	2.66	2.63	2.57	3.80	2.38	31	23	15	7
20×20×6	2.22	3.76	2.23	1.75	1.76	33	24	16	7
20×20×8	1.96	1.59	1.47	2.91	1.63	31	22	15	6
20×20×10	2.47	1.74	1.74	1.76	2.83	29	21	14	6
40×10×6	5.58	7.34	4.31	5.66	5.61	42	30	20	9
40×10×8	6.19	5.77	4.70	3.99	5.19	40	28	19	9
40×10×10	4.54	5.60	3.37	5.74	4.64	38	27	19	8
40×12×6	6.60	2.74	5.71	5.33	6.11	39	27	19	8
40×12×8	5.30	6.16	4.39	5.22	5.22	37	26	19	8
40×12×10	5.54	3.79	4.45	4.58	5.16	36	25	17	8
40×16×6	5.02	8.22	4.23	4.57	3.72	36	25	18	8
40×16×8	5.42	5.31	5.18	5.36	4.38	34	24	17	7
40×16×10	3.47	5.10	3.06	2.91	2.79	32	22	16	7
40×20×6	5.39	4.24	4.93	5.00	5.30	34	24	16	7
40×20×8	6.67	2.86	4.09	4.88	6.09	31	22	15	7
40×20×10	5.33	3.11	4.94	5.77	4.61	29	21	14	6
60×10×6	4.48	4.63	3.33	3.02	4.35	42	29	20	9
60×10×8	5.61	4.96	2.79	3.60	5.52	40	28	20	8
60×10×10	3.68	6.05	2.77	2.75	3.42	37	26	18	8
60×12×6	3.78	2.93	3.46	4.36	3.45	40	27	19	9
60×12×8	4.50	3.02	4.94	3.79	4.93	37	26	19	8
60×12×10	4.25	4.82	3.83	3.43	3.90	36	25	17	8
60×16×6	5.68	5.16	5.85	5.93	5.00	36	25	17	7
60×16×8	3.56	4.35	2.88	3.08	4.00	34	23	17	8
60×16×10	4.27	7.27	4.08	3.75	3.42	33	23	16	8
60×20×6	4.85	6.75	3.18	4.56	4.14	35	23	16	8
60×20×8	4.69	5.21	3.85	4.96	4.38	31	21	15	7
60×20×10	4.42	4.79	3.75	4.11	4.28	30	21	15	8
80×10×6	3.87	2.75	3.62	2.92	3.57	43	29	20	10
80×10×8	4.61	4.71	2.96	4.52	3.01	41	28	19	9
80×10×10	4.33	4.20	2.83	3.88	3.69	39	27	19	9
80×12×6	4.43	3.03	3.84	3.87	3.49	41	27	19	9
80×12×8	4.93	3.40	3.70	4.18	5.10	38	25	18	9
80×12×10	4.09	3.32	2.10	3.08	4.32	36	25	18	9
80×16×6	6.44	4.25	3.69	3.73	5.71	36	25	18	8
80×16×8	4.71	4.59	4.28	3.08	3.98	34	24	16	8
80×16×10	4.48	2.97	2.72	4.05	4.51	32	22	15	8
80×20×6	3.72	4.64	3.99	3.91	3.09	34	24	17	9
80×20×8	3.62	3.38	2.85	3.18	2.75	31	22	15	8

Table 7 (continued)

Problem ($n\times m\times w$)	Average RPD					Average $\Delta CPU_{\%PGA}$			
	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}	MGA (%)	HGA_{25} (%)	HGA_{50} (%)	HGA_{75} (%)
80×20×10	3.33	3.46	2.76	2.88	3.27	29	19	14	8
100×10×6	3.11	3.11	3.42	2.40	3.18	44	30	22	10
100×10×8	3.00	2.84	2.65	2.49	2.76	43	29	21	10
100×10×10	3.12	2.23	2.27	2.25	3.00	40	26	20	10
100×12×6	3.59	3.81	3.02	2.62	3.78	43	27	20	9
100×12×8	3.68	3.62	2.21	2.50	3.23	39	25	18	9
100×12×10	3.62	3.65	3.03	3.23	2.82	37	24	17	9
100×16×6	4.03	2.94	3.26	3.75	3.51	37	24	18	8
100×16×8	4.13	2.43	3.46	3.26	4.04	34	23	17	8
100×16×10	3.77	3.61	2.93	3.13	3.02	32	22	15	8
100×20×6	3.59	2.98	3.68	3.67	3.72	33	23	16	8
100×20×8	3.42	5.17	2.54	3.25	3.01	30	21	16	8
100×20×10	4.37	3.42	3.26	2.99	3.90	28	20	15	9
150×10×6	1.94	2.45	2.13	1.27	1.75	41	28	20	10
150×10×8	2.76	2.65	1.40	1.85	2.75	38	25	18	10
150×10×10	2.42	3.38	1.58	2.75	2.45	34	22	16	8
150×12×6	2.42	3.04	1.70	2.32	2.61	36	25	18	9
150×12×8	2.49	2.92	2.36	1.67	2.20	32	23	17	9
150×12×10	2.26	2.53	1.62	1.86	1.69	30	22	16	9
150×16×6	2.75	4.17	2.29	2.31	2.38	32	22	16	9
150×16×8	2.46	3.22	2.18	1.45	2.19	29	21	16	10
150×16×10	2.50	3.14	1.61	1.50	1.89	26	18	14	8
150×20×6	3.48	3.03	2.67	2.94	3.09	28	19	14	8
150×20×8	2.35	3.07	1.64	1.50	1.77	23	16	12	7
150×20×10	2.11	3.38	2.53	1.97	2.29	21	15	11	7
200×10×6	1.72	1.68	1.70	1.40	1.58	43	27	20	11
200×10×8	2.37	1.96	2.16	1.78	2.15	36	24	18	9
200×10×10	2.20	2.58	0.98	1.65	2.04	33	22	17	9
200×12×6	1.94	2.18	1.70	0.98	1.78	36	25	19	10
200×12×8	1.73	1.22	1.60	1.96	1.54	31	22	17	8
200×12×10	1.68	1.73	1.39	1.33	1.95	29	20	15	9
200×16×6	2.50	2.14	2.07	2.19	1.80	30	21	16	9
200×16×8	2.05	2.67	1.79	1.76	1.84	26	19	14	8
200×16×10	2.55	2.50	2.11	1.96	2.34	24	17	13	7
200×20×6	2.14	2.35	1.50	1.61	1.64	27	19	15	8
200×20×8	1.74	2.85	1.39	1.47	1.91	22	17	13	7
200×20×10	1.91	2.54	1.34	1.94	2.07	22	17	11	5
g_ave	3.57	3.62	2.89	3.06	3.25	34	24	17	8

and exploitation abilities of the proposed approaches are considered.

Regardless of the corresponding decoding procedure, the permutation encoding employed by PGA enhances the speed of the exploration phase. Indeed, as the same permutation string drives both the job sequencing and the worker assignment issue, every chromosome modification introduced by crossover and mutation operators may generate significant genetic distortions. As a consequence, there is a high probability that the search process moves towards entirely new areas of the search space.

Conversely, the multi-encoding scheme adopted by MGA may run a greater amount of information and an exploitation mechanism within the neighbourhood of the previously visited points may occur. Mutation and crossover operators should assist the exploitation phase as they can be applied to each

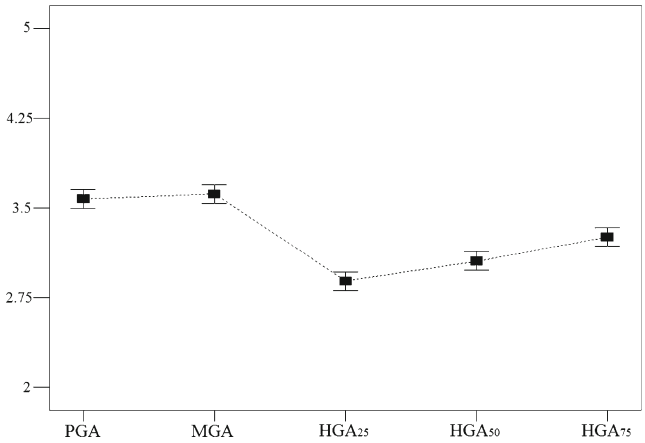

Fig. 7 Means plot and LSD intervals for the large instances campaign

single portion of a multi-encoded chromosome. Nevertheless, MGA keeps a fair ability in the exploration phase as demonstrated by the large-sized instances related numerical results (see Table 7; Fig. 7), where PGA and MGA achieve comparable results.

The twofold encoding employed by HGA ensures the best compromise between exploration and exploitation. In the first evolutionary phase, before the provided threshold is reached, the permutation encoding speed up the exploration mechanism of HGA over the entire search space. Then, after the switch threshold is reached, HGA may start to thoroughly investigate smaller spaces of solutions, making full use of the greater number of information included into the multi-encoded chromosomes. A 25 % encoding switch threshold seems to ensure the best trade-off between exploration and exploitation phases.

Table 8 Average performances of HGA_{25} at varying of workforce scenarios

Problem ($n\times m\times w$)	Average makespan				Average $\Delta C_{max\%HS}$		
	HS	AS	LS	MS	AS (%)	LS (%)	MS (%)
20×10×6	63.46	85.52	104.78	78.16	34.9	65.6	23.3
20×10×8	62.4	84.62	102.12	75.32	35.8	64.0	20.9
20×10×10	65.38	86.94	103.7	74.74	33.2	59.4	14.6
20×12×6	56.2	75.56	92.72	67.56	34.9	65.5	20.2
20×12×8	55.76	73.16	90.82	67.24	31.8	64.0	21.0
20×12×10	54.58	72.54	89.98	64.72	33.5	65.9	19.2
20×16×6	47.52	65.16	80.12	58.3	37.3	69.2	23.1
20×16×8	45.74	61.24	73.84	53.7	34.9	63.0	18.1
20×16×10	43.7	59.52	73.64	54.18	36.9	69.7	24.6
20×20×6	42.78	58.08	73.2	53.1	36.2	72.3	25.2
20×20×8	39.42	54.82	67.3	49.06	39.9	72.5	24.8
20×20×10	42.36	57.12	68.62	48.94	35.6	63.1	16.3
40×10×6	120.98	164.9	203.3	151.52	36.9	68.7	25.8
40×10×8	114.96	158.28	189.32	139.94	37.9	65.2	22.1
40×10×10	115.9	154.64	191.2	137.18	33.7	65.4	18.8
40×12×6	100.76	135.74	166.64	128.46	35.3	66.5	28.5
40×12×8	94.22	127.2	156.78	114.3	35.3	66.9	21.5
40×12×10	90.78	124.8	151.7	111.98	37.7	67.9	23.8
40×16×6	77.4	103.82	131.1	101.78	34.3	69.6	31.8
40×16×8	68.3	93.86	116.96	90.06	37.7	71.5	32.1
40×16×10	67.82	94.62	112.54	83.68	40.1	66.5	23.5
40×20×6	65.86	91.3	115.42	85.42	39.0	75.6	30.1
40×20×8	57.4	78.98	99.2	76.28	38.0	73.5	33.6
40×20×10	55.72	75.36	92.94	71.64	35.7	67.1	28.6
60×10×6	187.28	259.7	323.4	236.94	39.0	73.2	26.8
60×10×8	179.2	245.76	297.18	219.14	37.4	66.2	22.5
60×10×10	177.44	242.94	297.04	212.22	37.1	67.6	19.6
60×12×6	155.54	213.54	266.44	197.74	37.5	71.6	27.4
60×12×8	142.06	196.8	240.14	183.9	38.8	69.4	29.7
60×12×10	139.7	190.94	235.48	172.38	37.0	68.9	23.6
60×16×6	116.72	163.12	197.24	151.7	40.1	69.7	30.4

Table 8 (continued)

Problem ($n\times m\times w$)	Average makespan				Average $\Delta C_{max\%HS}$		
	HS	AS	LS	MS	AS (%)	LS (%)	MS (%)
60×16×8	104.3	143.68	179.5	137.32	37.9	72.5	31.9
60×16×10	96.56	135.94	166.92	125.12	41.0	73.4	29.9
60×20×6	96.9	135.42	166.9	124.64	40.2	72.8	28.8
60×20×8	85.54	116.98	147.42	114.48	37.0	72.6	34.0
60×20×10	79.02	111.6	137	103.88	41.5	73.7	31.7
80×10×6	252	354.04	434.78	337.14	40.7	72.8	33.9
80×10×8	240.16	326.56	404.2	296.38	36.3	68.7	23.5
80×10×10	238.76	327.72	401.86	292.08	37.6	68.7	22.7
80×12×6	209.12	292.2	362.18	268.72	39.9	73.4	28.7
80×12×8	191.5	264.12	327.24	247.66	38.1	71.1	29.5
80×12×10	189.26	260.5	318.4	235.7	37.8	68.5	24.7
80×16×6	160.24	223.14	276	216.1	39.4	72.4	34.9
80×16×8	140.08	196.18	242.66	184.36	40.3	73.6	31.9
80×16×10	133.1	184.76	227.88	168.1	39.0	71.5	26.5
80×20×6	131.82	183.22	230.06	176.18	39.4	75.0	34.1
80×20×8	115.68	160.08	201.28	151.08	38.6	74.3	30.7
80×20×10	106.9	147.88	183.56	135.3	38.6	72.2	26.8
100×10×6	320.92	447.42	551.2	422.5	39.5	71.9	31.8
100×10×8	304.68	423.18	520.92	392.04	39.1	71.2	28.9
100×10×10	308.66	420.36	511.48	374.62	36.3	65.9	21.5
100×12×6	269.58	375.48	464.26	344.12	39.4	72.3	27.7
100×12×8	242.44	341.62	415.04	306.72	41.0	71.4	26.7
100×12×10	236.24	331.44	401.34	297.04	40.4	70.0	26.0
100×16×6	205.74	289.54	353.36	276.72	40.9	71.9	34.7
100×16×8	178.76	248.62	307.66	234.06	39.2	72.3	31.0
100×16×10	165.88	231.34	285.66	213.38	39.6	72.3	28.7
100×20×6	169.5	241.74	300.28	235.98	42.8	77.4	39.4
100×20×8	144.78	203.72	253.46	192.36	40.9	75.3	33.0
100×20×10	131.34	184.4082	229.68	171.6	38.0	75.0	30.8
150×10×6	495.34	689.68	856.18	647.12	39.3	73.0	30.7
150×10×8	469.94	648.96	807.64	599.56	38.2	72.0	27.7
150×10×10	465.44	639.4	790.16	572.36	37.5	69.9	23.0
150×12×6	407.22	571.48	711.84	531.6	40.5	75.0	30.7
150×12×8	369.72	518.56	640.4	477.94	40.3	73.3	29.3
150×12×10	363.44	506.78	615.74	454	39.5	69.5	25.0
150×16×6	314.2	438.9	548.7	413.24	39.8	74.8	31.7
150×16×8	273.44	381.78	469.82	360.4	39.7	72.0	31.9
150×16×10	251.94	350.16	435.04	319.44	39.1	72.8	26.8
150×20×6	265.12	366.64	461.52	351.54	38.4	74.2	32.7
150×20×8	222.94	311.06	390.02	298.96	39.6	75.0	34.1
150×20×10	198.3	277.56	345.88	258.04	40.0	74.5	30.2
200×10×6	666.48	935.04	1,152.72	862.58	40.3	73.0	29.5
200×10×8	636.34	883.04	1,084.52	807.34	38.9	70.6	26.9
200×10×10	630.58	873.98	1,073.8	778.6	38.7	70.4	23.5
200×12×6	554.42	784.5	966.68	729.9	41.5	74.5	31.7
200×12×8	500.7	703.7	872.02	637.46	40.6	74.3	27.4
200×12×10	487.76	680.3	833.1	618.98	39.5	70.8	26.9

Table 8 (continued)

Problem ($n\times m\times w$)	Average makespan				Average $\Delta C_{max\%HS}$		
	HS	AS	LS	MS	AS (%)	LS (%)	MS (%)
200×16×6	431.56	607.48	753.12	559.34	40.8	74.6	29.7
200×16×8	366.78	515.7	634.38	477.58	40.7	73.1	30.3
200×16×10	339.22	473.58	585.82	431.62	39.7	72.8	27.2
200×20×6	360.78	505.88	630	471.58	40.3	74.7	30.8
200×20×8	300.68	420.12	522.52	404.52	39.8	73.9	34.6
200×20×10	268.52	373.54	465.26	349.56	39.2	73.4	30.2
g_ave	190.80	264.07	325.70	243.89	37.4	69.5	27.2

7 How the multi-skilled workforce affects productivity

The aim of this section is to assess the way the difference of technical skills among workers may influence the performance of the unrelated parallel machine production system under investigation. To this end, the results obtained by the best optimisation algorithm among those tested, i.e. HGA_{25}, have been selected for a further step of analysis.

To highlight the effect of the non-homogeneous workforce on the makespan minimisation objective, the proposed scenario problem (hereinafter denoted as the multi-skill (MS) scenario), which uses workers with skill levels randomly varying in the rage (0.5, 1), was compared with the following three configurations, each one featured by workers having identical skill levels:

- High-skill scenario (HS): each worker k has skill level η_k=0.5;
- Average-skill scenario (AS): each worker k has skill level η_k=1; and
- Low-skill scenario (LS): each worker k has skill level η_k=1.5.

For each one of the 840 large-sized instances included in the reference benchmark, three adding sets of setup times have been generated thus respecting the different skill levels pertaining to each scenario, i.e. HS, AS and LS. To emphasise the impact of the skill levels on the performance of the production system, small-sized instances have been ignored.

For each new instance, five runs of HGA_{25} with different random seeds have been carried out. As the three new scenarios do not involve any skill difference among human resources, substring-γ driving the assignment of jobs to workers has been removed from the multi-encoding scheme of HGA_{25}. A simple "first available worker" rule has been adopted for assigning each operator to each setup operation.

The average makespan values obtained for HS, AS, LS and MS scenarios are reported in Table 8, along with the relative percentage increment ($\Delta C_{max\%HS}$) of AS, LS and MS configurations with respect to the HS scenario.

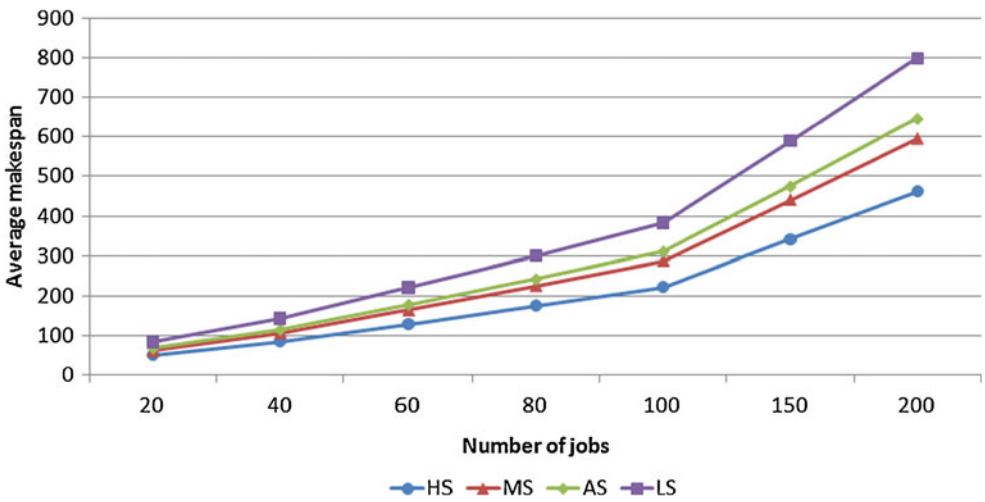

Fig. 8 Average makespan values at varying workforce scenarios

A trivial remark concerning the effect of workforce teams having different skill levels over the performances of the production system is that the HS scenario yields the lowest average makespan values for each class of problems. Conversely, LS configuration systematically provides the highest completion times.

A more interesting conclusion should entail the comparing between AS and MS scenarios. On the basis of what stated in the previous sections, MS configuration employs teams of workers whose skill levels are randomly extracted in the range {0.5, 0.75, 1, 1.25, 1.5}. Thus, the average-skill level of workforce teams generated for each problem is expected to be approximately equal to one. As the reader can see in Table 8, results obtained by the MS scenario always outperform those provided by the AS configuration, which involves workforce teams whose workers have a fixed skill level equal to one.

The aforementioned remarks may be clarified in Fig. 8, which reports the average makespan values for each scenario as the number of jobs changes.

Outperformance of MS with respect to the AS scenario should be justified by the optimisation strategy embedded within the proposed metaheuristic procedure, which allows to employ the best-skilled workers for the crucial activities, whereas the worst-skilled workers may be directed to setup tasks of other jobs that do not affect the entire system performance.

In conclusion, workforce skills strongly may influence the makespan minimisation issue for a parallel machine production system. Furthermore, a smart optimisation tool like the proposed HGA_{25} may ensure a significant exploitation of the skilled workforce, by assigning the best-skilled workers to the setup tasks of the key jobs that may affect the overall system performance.

8 Conclusions

In this paper, the unrelated parallel machine scheduling problem with limited and multi-skilled human resources has been addressed with regard to the makespan minimisation objective. To this aim, three different kinds of metaheuristic procedures based on GAs have been developed: a GA equipped with a single-stage permutation encoding, a GA powered by a multi-stage encoding, and a hybrid GA which encompasses both the single-stage and the multi-stage encodings. A calibration campaign has been carried out with the aim of selecting the best setting parameters of the proposed optimisation procedures; to this end, a benchmark of 100 classes of problem entailing 1,000 instances differing for number of jobs, workers and machines has been arranged. The proposed metaheuristics extensively have been compared by taking into account both small- and large-sized problems. In addition, a statistical analysis based on the ANOVA method has been fulfilled to evaluate the effects of the two kinds of encoding on the metaheuristics' performance.

Comparison regarding small instances revealed the effectiveness and the efficiency of all proposed algorithms in approaching the global optimum provided by the MILP model resolution. In particular, the hybrid GA, called HGA_{25}, which moves to the multi-stage encoding after 25 % of total makespan evaluations, outperformed all the other metaheuristics, though not always in a statistically significant manner.

Comparison conducted with reference to large-sized instances confirmed the effectiveness of the HGA_{25} hybrid optimisation approach, which achieved a statistically significant difference of performance with respect to any other competitor. Without loss of generality, it can be stated that GAs powered by a twofold encoding outperform the other optimisation procedures for the proposed sequencing/allocation optimisation problem. In fact, permutation encoding allows a fast exploration of the solution domain but it is not able to inspect the overall set of available solutions. By contrast, the multi-stage encoding makes full use of its larger string to investigate a wider space of solutions but, at the same time, it pays the penalty of a more complex encoding/decoding, which increases the computational time.

After the best optimisation procedure was selected, a further analysis comparing the production system performance at varying average workforce skill levels has been carried out. Such analysis highlighted the effect of a smart skilled-workers assignation strategy with regards to the makespan optimisation.

Future research may involve a comparison of the proposed hybrid approach with other kinds of metaheuristics or, alternatively, further methods of multi-encoding hybridisations may be studied. With reference to the manpower skills issue, the effects of training policies could be studied too.

References

1. Anghinolfi D, Paolucci M (2007) Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. Comput Oper Res 34:3471–3490
2. Askin RG, Huang RY (2001) Forming effective work teams for cellular manufacturing. Int J Prod Res 39(11):2431–2451
3. Balin S (2011) Non-identical parallel machine scheduling using genetic algorithm. Expert Syst Appl 38:6814–6821
4. Celano G, Costa A, Fichera S (2008) Scheduling of unrelated parallel manufacturing cells with limited human resources. Int J Prod Res 46(2):405–427
5. Celano G, Costa A, Fichera S, Perrone G (2004) Human factor policy testing in the sequencing of manual mixed model assembly lines. Comput Oper Res 31(1):39–59
6. Costa A, Celano G, Fichera S, Trovato E (2010) A new efficient encoding/decoding procedure for the design of a supply chain network with genetic algorithms. Comput Ind Eng 59(4):986–999
7. Chaudhry IA (2010) Minimizing flow time for the worker assignment problem in identical parallel machine models using GA. Int J Adv Manuf Technol 48:747–760

8. Chaudhry IA, Drake PR (2009) Minimizing total tardiness for the machine scheduling and worker assignment problems in identical parallel machines using genetic algorithms. Int J Adv Manuf Technol 42:581–594
9. Cheng R, Gen M (1997) Parallel machine scheduling problems using memetic algorithms. Comput Ind Eng 33(3–4):761–764
10. Cochran JK, Horng SM, Fowler JW (2003) A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. Comput Oper Res 30:1087–1102
11. ElMaraghy H, Patel V, Ben AI (2000) Scheduling of manufacturing systems under dual-resource constraints using genetic algorithms. J Manuf Syst 19(3):186–198
12. Fanjul-Peyro L, Ruiz R (2011) Size-reduction heuristics for the unrelated parallel machines scheduling problem. Comput Oper Res 38:301–309
13. Fanjul-Peyro L, Ruiz R (2012) Scheduling unrelated parallel machines with optional machines and jobs selection. Comput Oper Res 39:1745–1753
14. Gokhale R, Mathirajan M (2012) Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. Int J Adv Manuf Technol 60:1099–1110
15. Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor
16. Hottenstein MP, Bowman SA (1998) Cross-training and worker flexibility: a review of DRC system research. J High Technol Manag Res 9(2):157–174
17. Hu PC (2004) Minimising total tardiness for the worker assignment scheduling problem in identical parallel-machine models. Int J Adv Manuf Technol 23:383–388
18. Hu PC (2005) Minimizing total flow time for the worker assignment scheduling problem in the identical parallel-machines models. Int J Adv Manuf Technol 25:1046–1052
19. Hu PC (2006) Further study of minimizing total tardiness for the worker assignment scheduling problem in the identical parallel-machines models. Int J Adv Manuf Technol 29:165–169
20. Kim SC, Bobrowsky PM (1997) Scheduling jobs with uncertain setup times and sequence dependency. OMEGA Int J Manag Sci 25(4):437–447
21. Kim DW, Na DG, Chen FF (2003) Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. Robot Comput Integr Manuf 19:173–181
22. Kher HV, Fry TD (2001) Labour flexibility and assignment policies in a job shop having incommensurable objectives. Int J Prod Res 39(11):2295–2311
23. Lenstra JK, Rinnooy Kan AHG, Brucker P (1977) Complexity of machine scheduling problems. Ann Discret Math 1:342–362
24. Michalewicz Z (1994) Genetic algorithms+data structures=evolution programs, 2nd edn. Springer, Berlin
25. Montgomery D (2007) Design and analysis of experiments, 5th edn. Wiley, New York
26. Nelson RT (1967) Labor and machine limited production systems. Manag Sci 13(9):648–671
27. Norman BA, Tharmmaphornphilas W, Lascola Needy K, Bidanda B, Colosimo WN (2002) Worker assignment in cellular manufacturing considering technical and human skills. Int J Prod Res 40(6):1479–1492
28. Oliver IM, Smith DJ, Holland JRC (1987) A study of permutation crossover operators on the TSP. In: Grefenstette JJ (ed) Genetic algorithms and their applications: proceedings of the second international conference. Lawrence Erlbaum, Hillsdale
29. Pan QK, Suganthan PN, Chua TJ, Cai TX (2010) Solving manpower scheduling problem in manufacturing using mixed-integer programming with a two-stage heuristic algorithm. Int J Adv Manuf Technol 46:1229–1237
30. Pereira Lopes MJ, Valério de Carvalho JM (2007) A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. Eur J Oper Res 176:1508–1527
31. Pinedo M (2008) Scheduling: theory, algorithms and systems, 3rd edn. Prentice-Hall, New Jersey
32. Sule DR (2008) Production Planning and Industrial Scheduling: Examples, Case Studies and Applications, 2nd edn. CRC Press, Boca Raton
33. Syswerda G (1989) Uniform crossover in genetic algorithms. In: Schaffer JD (ed) Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann Publishers, San Mateo
34. Syswerda G (1991) Schedule optimization using genetic algorithms. In: Davis L (ed) Handbook of genetic algorithms. Van Nostrand Reinhold, New York
35. Tavakkoli-Moghaddam R, Taheri F, Bazzazi M, Izadi M, Sassani F (2009) Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. Comput Oper Res 36:3224–3230
36. Treleven M (1989) A review of the dual resource constrained system research. IIE Trans 21(3):279–287
37. Vallada E, Ruiz R (2011) A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. Eur J Oper Res 211:612–622
38. Xu J, Xu X, Xie SQ (2011) Recent developments in dual resource constrained (DRC) system research. Eur J Oper Res 215:309–318
39. Zoubaa M, Baptistea P, Rebaineb D (2009) Scheduling identical parallel machines and operators within a period based changing mode. Comput Oper Res 36:3231–3239