UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI INGEGNERIA ELETTRICA, ELETTRONICA E INFORMATICA

DOTTORATO DI RICERCA IN INGEGNERIA DEI SISTEMI

XXVI CICLO

FULVIO ANTONIO CAPPADONNA

Application of exact and approximate optimization methods to novel scheduling problems

Ph. D. Thesis

Coordinator: Prof. Ing. L. Fortuna

Tutor: Prof. Ing. S. Fichera

Contents

Introduction

The application of exact and heuristic optimization techniques to scheduling problems pertaining to production processes has been widely investigated over the last decades by the relevant scientific literature in the field of industrial systems design and analysis. In general, the term *scheduling* is used with reference to the allocation of resources to tasks over time, so to execute all planned activities according to a given performance objective (minimization of costs, minimization of production time, due dates fulfilment, etc.). Even though basic scheduling problems have been effectively solved long time ago, this topic still remains attractive for expert and practitioners, as the technological innovation of production processes and the need for effective planning activities emerging from new sectors still set new frontiers to the scheduling optimization research.

The aim of the present Thesis is to investigate three scheduling problems that have not been addressed yet by the literature, even though they have a clear correspondence to real-world manufacturing environments. After an introduction to the purpose and the main practical aspects connected to the scheduling activity reported in Chapter 1, the outcomes of the research conducted are reported in Chapters 2 to 4.

In Chapter 2 the minimization of makespan in an unrelated parallel machine system with sequence-dependent setup times and limited human resources is addressed. Workers are needed to perform setup operations before each job is processed; they are supposed to be a critical resource as their number is assumed to be lower than the number of workstations available in the production shop. In addition, each worker is characterized by a provided skill level, which affects the time required for completing setup operations. Firstly, a Mixed Integer Linear Programming (MILP) model suitable for tackling small instances of the problem in hand is illustrated. Then, an optimization framework based on Genetic Algorithms (GAs) is presented with the aim of effectively addressing larger test cases. Three different procedures are proposed, namely a permutation based GA, a multi-encoding GA, and a hybrid GA which exploits both the former methods in two successive steps, changing the encoding scheme when a fixed number of evaluations is reached. An extensive benchmark including both small-and large-sized instances is taken as reference for both calibration and comparison of the proposed

methods, which have been performed by means of ANOVA analysis. Comparison conducted with reference to small-sized instances reveals the superiority of the hybrid approach using an encoding switch threshold equal to the 25% of the total evaluations. This outcome becomes more evident with respect to large-sized test cases, where such hybrid algorithm outperforms all other procedures in a statistically significant manner. Finally, a remark on the effect of multi skilled human resources on the performance of the investigated production system in comparison with three scenarios involving homogeneous workforce is proposed.

Chapter 3 is dedicated to the minimization of makespan in a Flow Shop Sequence Dependent Group Scheduling (FSDGS) problem entailing the worker allocation issue. As first, a Mixed Integer Linear Programming (MILP) formulation for the problem is given. Then, a well-known benchmark arisen from literature is adopted for carrying out an extensive comparison campaign among three specifically developed metaheuristic methods based on a GA framework. Afterwards, the best procedure among those tested is compared with a well-performing algorithm recently proposed in the field of FSDGS problems, properly adapted to manage the Skilled Workforce Assignment (SWA) issue. Finally, a further analysis dealing with the trade-off between manpower cost and makespan improvement is proposed.

In Chapter 4, the research conducted with reference to the minimization of makespan in a hybrid flow shop inspired to a truly observed micro-electronics manufacturing environment, is illustrated. Overlap between jobs of the same type, waiting time limit of jobs within inter-stage buffers as well as machine unavailability time intervals represent just a part of the constraints which characterize the problem here investigated. A MILP model of the problem in hand has been developed with the aim to validate the performance concerning the proposed optimization technique, based on a two-phase metaheuristics (MEs). In the first phase the proposed ME algorithm evolves similarly to a genetic algorithm equipped with a regular permutation encoding. Subsequently, since the permutation encoding is not able to investigate the overall space of solutions, a random search algorithm equipped with an m-stage permutation encoding is launched for improving the algorithm strength in terms of both exploration and exploitation. Extensive numerical studies on a benchmark of problems, along with a properly arranged ANOVA analysis, demonstrate the statistical outperformance of the proposed approach with respect to the traditional optimization approach based on a single encoding. Finally, a comprehensive comparative analysis involving the proposed algorithm

and several metaheuristics developed by literature demonstrates the effectiveness of the dual encoding based approach for solving HFS scheduling problems.

Chapter 1

The problem of scheduling and sequencing

1.1 Preliminaries

It has been a long time ago since the role of scheduling was recognized to be crucial in most of manufacturing and service industries. The scheduling issue has been constantly attracting the attention of researchers and practitioners and still represents one of the most active field of research connected to industrial systems design and analysis.

Of course the word *scheduling* deals with the allocation of tasks to resources over time, but what is the concrete purpose of the scheduling activity? Baker and Trietsch (2009) provided an answer to such a question through an exhaustive dissertation. According to the authors, it is actually not *scheduling* that is a common concept in our everyday life, rather it is *schedules*. A schedule is a plan that tells us when things are supposed to happen, i.e., the time at which certain activities should start and end according to the most likely scenario. Typical examples of schedules are bus or flights timetables. That being stated, the scheduling process in industrial environments involves considerations about the set of activities or tasks to be performed and the set of resource available to fulfill them; in addition to resource requirements, each task may be described in terms of information such as its expected duration, the earliest time at which it may start and the time at which it is due to complete. The aim of the scheduling activity consists thus in determining the timing of the tasks while recognizing the capability of the resources, so to effectively fulfill all the activities according to a given performance measure. It can be easily argued that the scheduling process entails *sequencing* decisions, as the existence of limited resources implies the determination of the order by which activities have to be executed.

As far as performance objectives are concerned, three kinds of decision-making goals are common to be found in the industrial context: *turnaround, timeliness* and *throughput*. Turnaround refers to the time required to complete a task. Timeliness measures the

conformance of a particular task's completion to a given deadline. Throughput is related to the amount of work completed during a fixed period of time. In the following Section, an overview of the most commonly used indicators which put such goals into quantitative measure will be provided.

One of the most important conditions influencing the reliability of the scheduling process is *uncertainty*. Although information characterizing each task to be performed are often exactly retrievable, there exist many industrial processes in which a some uncertainty has to be taken into account. In such cases, a probability distribution model describing the expected variability of data needs to be employed, and the scheduling process is called *stochastic*. On the other hand, when conditions are assumed to be known with certainty, the problem is known as *deterministic*. Another important distinction needs to be made between *static* and *dynamic* scheduling, the former referring to a set of activities which are all known in advance, the latter involving timing and sequencing decisions for tasks that appear over time.

One of the simplest and most widely used tools for supporting the scheduling process is the *Gantt chart*, which visually depicts a solution to a given scheduling problem, showing resource allocation to activities over time. In a classical Gantt chart, the horizontal axis reports the time scale, while resources available are shown along the vertical axis (see Figure 1.1). Activities are displayed as rectangles, whose positions denote the time at which tasks should be performed through a given resource. A fundamental assumptions of the Gantt chart is that processing times are known with certainty.

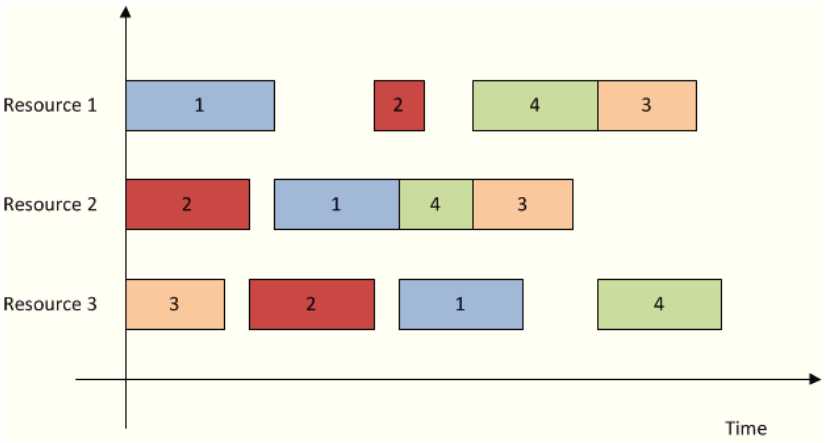

Figure 1.1. A Gantt Chart.

The execution of the scheduling activity in real-world industrial environments often needs to cope with the computational complexity of the solution algorithms that have to be employed when approaching the scheduling issue. This topic has been extensively analyzed in the well-known work of Garey and Johnson (1979). According to the authors, the *time complexity function* for a solution algorithm is a useful metric for selecting the most effective way to approach an optimization problem. Given a solution algorithm, the time complexity function expresses its time requirements by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size. A *polynomial time algorithm* has a complexity function whose order of magnitude is polynomial as n increases, being n the input length. Any algorithm whose time complexity function cannot be so bounded is called an *exponential time algorithm*. For instance, if the time complexity function is bounded by n^2 for each value of n, the algorithm is polynomial. If the function is bounded by 2^n the algorithm is exponential. Of course, polynomial algorithms are to be preferred, being ultimately faster than exponential ones. Nevertheless, there exists many optimization problems for which no polynomial time solution algorithms have been found, to the point that they are not supposed to exist. Such problems are known as $\mathcal{NP}$hard. Most of the common scheduling problems pertain to such a class, as it will hereinafter be shown. When medium to large-sized $\mathcal{NP}$hard problems have to be addressed, the chance of finding the optimal solution through an exact algorithm is quite unlikely. Thus, the best approach consists in employing *heuristic* or *metaheuristic* algorithms which do not guarantee the convergence towards the global optimality, but may be completed within a reasonable amount of time. A brief overview of the most common heuristic and metaheuristic procedures employed in the field of scheduling problems will hereinafter be presented.

1.2 Classification of scheduling problems

Although sequencing and allocation issues are frequently dealt with in many economic areas, the need for an effective scheduling activity basically arises for the manufacturing sector. The number of scheduling problems pertaining to real or theoretical manufacturing situations that have been studied by the relevant literature over the last decades is impressive. For this reason, the vocabulary of manufacturing is universally employed when addressing scheduling

problems. Activities or tasks are called *jobs* and resources are usually called *machines*. The environment in which jobs are processed is called the *production shop*, or simply the *shop*.
A comprehensive notation to be used for describing most of the deterministic scheduling models that have been considered so far has been provided by Pinedo (2012). Following such scheme, the number of jobs should be denoted by n and the number of machines by m. The subscript j refers to a job ($j = 1,2,\ldots,n$), while the subscript i is used with reference to machines ($i = 1,2,\ldots,m$). The following data are associated with job j.

- Processing time (p_{ij}). The symbol p_{ij} denotes the time required for processing job j on machine i. Whether exists one only machine, or processing times are not machine-dependent, subscript i is omitted.
- Release date (r_j). The release date denotes the time at which job j is ready to undergo the first processing operation, i.e., the time at which it arrives at the system.
- Due date (d_j). The due date is the time at which job j is due to be completed. If any deviation from the due date occurs, a penalty is incurred (e.g., holding, inventory or delivery costs).
- Weight (w_j). The weight expresses the relative priority of job j compared to the other jobs. Whether weights are defined in a scheduling problem, the objective function must be primarily met for the highest-weighted jobs.

According to Graham, Lawler, Lenstra and Rinnooy Kan (1979) a three-field problem classification $\alpha \mid \beta \mid \gamma$ may be used for describing the characteristic of a scheduling problem. The field α specifies the machine environment. It may contain one of the following entries.

- Single machine (1). The production shop is made by one only machine that has to process all jobs. Of course, this is the simplest machine environment to cope with.
- Identical parallel machines (Pm). Each job needs to be processed by only one of the m machines belonging to the manufacturing environment. Since machines are supposed to be identical, processing times only depend on jobs.
- Uniform parallel machines (Qm). The m parallel machines of the system have different speeds. Denoting by v_i the speed of machine i, processing time of job j on machine i is

equal to p_j/v_i, being p_j the standard processing time of job j, i.e., the time required for processing the job on a machine having speed equal to 1.

- Unrelated parallel machines (*Rm*). In this case, the speed at which machine i can process job j depends on the job itself, and is denoted as v_{ij}. Thus, processing time p_{ji} of job j on machine i is calculated as p_j/v_{ij}.
- Flow shop (*Fm*). A flow shop manufacturing system is made of m machines in series. Each job has to be processed on each machine. All jobs follow the same processing order, i.e., from the first to the last machine.
- Flexible flow shop (*FFc*). The flexible flow shop is a more general case of the flow shop environment. Instead of m machines, there are c production stages in series. Each production stage is made of a bank of identical parallel machines. All jobs visit the production stages according to the same order.
- Job shop (*Jm*). A job shop manufacturing system entails m different machines. Differently from the flow shop manufacturing environment, each job follows its own predetermined route.
- Flexible job shop (*FJc*). The flexible job shop is a more general case of the job shop environment. Instead of m machines, there are c production stages, each one made of a bank of identical parallel machines. Each job visits the production stages according to its own predetermined route.
- Open shop (*Om*). In an open shop environment there are m machines, even though some jobs are allowed to skip one or more of them. The processing route of each job has to be determined by the scheduler, as no predetermined processing order exists.

The β field holds data describing job characteristics or particular scheduling constraints. It may contain a single entry, multiple entries or no entry at all. Possible symbols to be found in such field are as follows.

- Release dates (r_j). Whether release dates are defined, each job cannot start to be processed prior to its arrival at the system. If the symbol r_j is not specified in the β field, all jobs are ready to be processed as the manufacturing session starts.

- Preemptions (*prmp*). If preemptions are allowed, a given job processing operation may be interrupted and the job moved to another machine, where it will be completed later.
- Precedence constraints (*prec*). Precedence relationships imply that some jobs have to wait for other jobs to be completed before their processing can start.
- Sequence dependent setup times (s_{jk}). The symbol s_{jk} denotes the time required for performing setup operations of job k immediately after job j has been completed on the same machine. If s_{jk} does not appear in the β field, setup times are assumed not to exist or to be sequence independent and included in processing times.
- Job families (*fmls*). The jobs to be processed belong to different groups or families. Setup operations are required only when switching from one family to another, and not between jobs of the same group. Setup times between families may be sequence dependent. In this case, the symbol s_{gh} will appear in the β field to denote the time required for performing setup operations between families g and h.
- Batch processing (*batch*(*b*)). Whether batch processing is allowed, a machine can process up to b jobs simultaneously. The batch processing time corresponds to the processing time of the longest job pertaining to that batch.
- Breakdowns (*brkdwn*). Machines are supposed not to be continuously available during the whole manufacturing session, as they may undergo maintenance operations or failures.
- Machine eligibility restrictions (M_j). This constraint arises in parallel machine manufacturing environments. The symbol M_j denotes the subset of machines capable to process job j. All machines not belonging to M_j cannot be selected for working that job.
- Permutation (*prmu*). In many flow shop manufacturing environments it is required to leave unaltered the order in which jobs go through each machine, thus following a *First In First Out* (*FIFO*) rule to manage the queues between successive workstations.
- Blocking (*block*). The blocking constraint often arises in permutation flow shop manufacturing environments. Whether such restriction exists, the capacity of buffers located between consecutive machines is limited. When a buffer is full, the upstream machine is not allowed to release a completed job. The job will have to wait on the

machine until one position on the buffer has been cleared, i.e. until the downstream machine has accepted a new job.

- No-wait (*nwt*). In no-wait flow shop manufacturing systems jobs are not allowed to wait between two consecutive machines. Therefore, they have to be scheduled so as to let each processing operation start immediately after that job has been completed on the previous machine.
- Recirculation (*rcrc*). Recirculation condition may be found in job shop or flexible job shop manufacturing environments. It occurs when some jobs visit certain machines or production stages more than once.

The γ field reports the function to be minimized in a scheduling problem. As observed by Leung (2004) the objective is always connected to the completion times of the jobs. With respect to a given schedule, let C_j denote the completion time of job j, i.e. the time at which the job exits the system. The lateness of job j is calculated as:

$$L_j = C_j - d_j \tag{1.1}$$

The tardiness of job j is defined as:

$$T_j = \max (C_j - d_j, 0) = \max (L_j, 0) \tag{1.2}$$

The unit penalty of job j is defined as

$$U_j = \begin{cases} 1 \text{ if } C_j > d_j \\ 0 \text{ otherwise} \end{cases} \tag{1.3}$$

Possible entries for the γ field are the following.

- Makespan (C_{max}). The makespan is the highest among the completion times of all jobs, i.e., $C_{max} = \max (C_1, C_2, \ldots, C_n)$.
- Maximum lateness (L_{max}). The maximum lateness is defined as the highest among the lateness of all jobs, i.e., $L_{max} = \max (L_1, L_2, \ldots, L_n)$.

- Total weighted completion time ($\sum w_j C_j$). Such indicator is also known as the total weighted flow time. The unweighted sum of completion times is usually referred to as the total completion time or the total flow time.
- Total weighted tardiness ($\sum w_j T_j$).
- Weighted number of tardy jobs ($\sum w_j U_j$). This function sums the weights of all jobs completed after their due dates.

1.3 Solution methods for scheduling problems

As it has been already pointed out, the resolution of scheduling issues often implies high computational burdens. Most of the scheduling problems analyzed by the relevant literature are $\mathcal{NP}$hard, even in the case of single machine manufacturing environments. Therefore, exact solution algorithms are suitable to be employed only in case of small-sized instances.

The *linear programming* technique is one of the most used approach for exactly solving scheduling problems involving a small number of variables and constraints. A linear program models an optimization problem in which the objective and the constraints are linear function of variables (Lopez & Roubellat, 2008). The two most important types of algorithms able to solve linear programs in continuous variables are the *simplex* method and the *interior point* method. The former is powerful, yet characterized by an exponential time complexity function. The latter owns a class of methods among which the algorithm proposed by Karmakar (1984) has been proved to solve linear programming models within a polynomial time. Nevertheless, the linear programs which describe scheduling problems involve integer variables (e.g., binary decision variables) in addition to continuous ones. Such Mixed-Integer Linear Programming (MILP) models entail a large number of constraints and no polynomial algorithm has been found to solve them.

When the complexity of a scheduling problem is too high to find an exact solution within a polynomial time, *heuristic* methods should be employed. Differently from exact procedures, heuristics do not guarantee the convergence towards optimality, although they can provide valid solutions in a reasonable amount of time. Heuristic methods elaborate data characterizing each job to generate a feasible solution to a given scheduling problem, i.e., a

sequence of jobs together with machine assignment rules if parallel workstations exist. Several heuristic methods have been presented in literature over the last decades. Those proposed by Nawaz, Enscore and Ham (1983), by Campbell, Dudek and Smith (1970) and by Palmer (1965) for the flow shop scheduling problem have been widely employed over years, also being adapted to cope with many other scheduling problems than the regular flow shop.

Although heuristic methods may deal with complex scheduling problems by quickly providing a solution without affecting the computational burden, there are two main disadvantages connected to the use of such methods. Firstly, each heuristic procedure needs to be properly adapted to the specific problem it is applied to, or results may be very poor. Secondly, heuristics provide unique solutions, while scheduling problems usually involve wide solution spaces (Jarboui, Siarry & Teghem, 2013). Thus, the chances of being dramatically far for the global optimum are quite high, even if the solution obtained can be considered satisfactory. To overcome these shortcomings, independent problem research strategies, called metaheuristics have been proposed. Metaheuristic search methods can be defined as upper level general methodologies that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems (Talbi, 2009). Similarly to heuristic methods, metaheuristics do not guarantee the convergence towards optimality. Nevertheless, instead of creating unique solutions, each metaheuristic method explores a part of the global solution space pertaining to a given optimization problem, employing specific criteria to drive the search. The most commonly used metaheuristic in the field of scheduling problems are *simulated annealing*, *tabu search* and *genetic algorithms*.

Since its introduction by Kirkpatrick, Gelatt and Vecchi (1983), the Simulated Annealing (SA) algorithm has been successfully employed for addressing a huge number of discrete optimization problems. The SA takes its name from the process of physical annealing with solids, in which a crystalline solid is heated and then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuration. At each iteration, the algorithm compares the current solution with a newly generated one. Improving solutions are always accepted, while non-improving ones may be accepted according to a probability based on a parameter called temperature, which is typically non-increasing with each iteration of the algorithm (Nikolaev and Jacobson, 2010). The acceptance of worse solutions, which is called a *hill-climbing* move, may allow to skip from local optima in search of the global optima. As

the temperature decreases to zero, hill climbing moves occur less frequently, and the procedure tends to converge to the final solution.

The tabu search (TS) algorithm proposed by Glover (1986) is an extension of the classical local search (LS) method. A LS procedure iteratively tries to improve the current solution by performing small local modifications. Whether the new solution leads to a better objective function value, it replaces the current one and is used for the next iteration. Differently from LS, tabu search allows non-improving moves. At each iteration, all the local transformations that can be applied to the current solution, denoted S, define a set of neighboring solutions, called $N(S)$. The algorithm selects the best-performing neighbor as the new current solution, even if it is worse than S. Such strategy allows to escape from the local optima on which classical LS methods quickly converge, thus allowing a wider exploration of the solution space. One of the distinctive elements of TS compared to LS are the tabu moves, which are used to prevent cycling when moving away from local optima through non-improving moves. When such a situation occurs, something needs to be done to prevent the search from tracing back its steps to where it came from. Therefore, the moves that reverse the effect of the most recent ones are declared tabu, and then disallowed. Tabus are stored in a short-term memory of the search and usually only a limited amount of information is recorded (Gendreau and Potvin, 2010).

Genetic algorithms (Holland, 1975) are search methods inspired by the process of natural evolution, that have largely been used to solve scheduling problems. Differently from SA and TS, which handle single solutions, 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 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. Through such method, a GA performs a multi-directional search by maintaining a population of potential solutions and encourages information formation exchange between these directions. The evolutionary process of a GA allows to let

die relatively “bad” solutions, favoring survival and reproduction of the “good” ones (Michalewicz, 1994).

Chapter 2

The parallel machine scheduling problem with limited human resources

2.1 Preliminaries

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 (Sule, 2008). According to 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 given machine until its processing is finished.

A pioneer study concerning the identical parallel machine scheduling problem has been carried out by Lenstra, Rinnooy Kan and Brucker (1977) who demonstrated as makespan minimization in such a kind of production system is a $\mathcal{NP}$-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 (1997) proposed a Memetic Algorithm (MA) for minimizing the maximum weighted absolute lateness. Cochran, Horng and Fowler (2003) developed a two-stage Multi Population Genetic Algorithm (MPGA) with reference to various concurrent objectives: makespan, total weighted tardiness and total weighted completion time. Anghinolfi and Paolucci (2007) presented a hybrid metaheuristic integrating features from Tabu Search (TS), Simulated Annealing (SA) and Variable Neighborhood Search (VNS) with the aim of minimizing the total tardiness. Recently, Gokhale and Mathirajan (2012) proposed five different heuristic techniques for minimizing 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 in respect of the identical machine case. Kim, Na and Chen (2003) coped with a batch-scheduling problem for an unrelated parallel machine system observed within a semiconductor-manufacturing environment. They developed a two-level heuristic (TH) and a SA for minimizing the total weighted tardiness, and compared those methods with two dispatching rules, namely the Earliest Weighted Due Date (EWDD) and the Shortest Weighted Processing Time (SWPT). Pereira Lopes and Valério de Carvalho (2007) studied an unrelated parallel machine system with sequence-dependent setup times, machine availability dates and jobs release dates with the aim of minimizing the total weighted tardiness. They followed a branch-and-price approach elaborating a proper procedure able to solve instances 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 by Tavakkoli-Moghaddam, Taheri, Bazzazi, Izadi and Sassani (2009) as well. Also, they considered precedence constraints among jobs, and developed a two-step linear programming model, together with a Genetic Algorithm (GA) for minimizing the number of tardy jobs primarily, and the total completion time secondly. A GA-based approach has also been adopted by Vallada and Ruiz (2011) with the aim of minimizing 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 in order to assess performances of the proposed metaheuristic against a Mixed Integer Linear Programing (MILP) model and other algorithms arisen from literature, for small-sized and large-sized problems respectively. The use of a GA approach for minimizing makespan in an unrelated parallel machine production system has also been investigated by Balin (2011), 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 (LPT) rule. A new approach for the resolution of the unrelated parallel machine scheduling issue, though not involving sequence dependent setup times, has been recently given by Fanjul-Peyro and Ruiz (2011), 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 (2012) have also studied two generalizations of the unrelated parallel machine model, namely the Not All Machine (NAM) and Not All Jobs (NAJ) variants; the former model allows one or more machines to be excluded by the production shop, while 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.

In order to make the parallel machine theoretical model suitable for describing real-world manufacturing environments in an accurate way, the effect of human factors on the performance of the production shop should be taken into account as well. Indeed, a well-established trend in literature has been focusing on the study of production systems known as Dual Resource Constrained (DRC), i.e., in which capacity constraints arise from both machines and human operators. DRC systems have received many attentions over the years since the seminal work of Nelson (1967). Treleven (1989) and Hottenstein and Bowman (1998) have thoroughly categorized the main topics and issues concerned with such manufacturing models. Recently, Xu, Xu and Xie (2011) provided a comprehensive review regarding the research works dealing with DRC systems presented during the last two decades. The effect of human workers on DRC systems may be basically studied under two different viewpoints, i.e., the worker flexibility and the worker assignment issues. As far as the worker flexibility is concerned, the existence of differences among technical skills and abilities of each operator should be taken into account. Part of the relevant literature embodied the concept of worker flexibility in manufacturing situations by means of the flexibility level, i.e., a numerical index representing the number of machines on which a worker is capable of operating (ElMaraghy, Patel & Ben Abdallah, 2000; Kher & Fry, 2001). Another research trend considered the so-called skill levels, i.e., numerical parameters assessing the efficiency of workers in processing jobs with respect to quality or productivity. This latter topic appears to be extremely relevant, as when one or more operation concerning the production phase should be manually executed, e.g. setups or removals, it could be crucial to take into account the impact of the different technical abilities pertaining to each worker on the global performances of the production system. Kim and Bobrowsky (1997) considered the crew skills

as a random factor affecting setup times on a job-shop production system. Askin and Huang (2001) 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, Tharmmaphornphilas, Lascola Needy, Bidanda, Colosimo Warner and Worker (2002), who developed a mixed integer programming problem for assigning workers to manufacturing cells in order to maximize the effectiveness of the organization, and emphasized the possibility of enhancing skill levels through additional training. Pan, Suganthan, Chua and Cai (2010) tackled a job-shop scheduling problem pertaining to the precision engineering industry, in which each job requires specific technical skills and a certain grade of experience and seniority to be performed.

With respect 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 the processing time of each job depends on the number of operators assigned to the machines where it is actually processed. Hu (2004, 2005, 2006), Chaudhry and Drake (2009) and Chaudhry (2010) addressed this kind of topic with respect to the identical parallel machine manufacturing environment. Celano, Costa, Fichera and Perrone (2004) studied a mixed-model assembly line sequencing problem in which workers are allowed to help their colleagues in completing operations with the aim of reducing the conveyor stoppage time. A further trend arisen from literature involves 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, Costa and Fichera studied the effect of reducing the human resource capacity on unrelated parallel manufacturing cells through an integrated simulation framework, taking into account several performance measures. Zouba, Baptistea and Rebaineb (2009) addressed the problem of scheduling jobs on an identical parallel machine manufacturing environment having less workers than machines, i.e., wherein operators may have to supervise simultaneously several machines.

To the best of our knowledge, the study of the unrelated parallel machine scheduling problem with sequence dependent setup times along with limited and multi-skilled human resource has not been taken into account yet. The present chapter addresses such topic under the makespan minimization viewpoint. Making use of the well-known three-field notation $\alpha \mid \beta \mid \gamma$ proposed

by Pinedo (2012), it may be classified as $Rm \mid S_{ikjl} \mid C_{\max}$ 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 given 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 workers to a set of jobs and then jobs to machines.

A GA-based optimization framework has been developed for investigating the problem in hand. Genetic algorithms represent a well-performing approach for solving combinatorial scheduling problems, as it has been widely recognized by the literature over the past decades. The aforementioned research works dealing with the parallel machine scheduling issue demonstrate as GAs have been extensively employed for tackling such kind of topic. Furthermore, as reported by Xu, Xu and Xie (2011), the use of GA-based optimization procedures to solve problems pertaining to DRC systems is a well-established trend as well. In the present chapter, three different hybrid GAs, each one involving two distinct encodings have been compared with two regular single-encoding GAs in terms of makespan minimization. Performance evaluation of the proposed metaheuristics was carried out by means of an ANOVA analysis (Montgomery, 2007) over a wide set of test cases. A further step of analysis has then been performed in order to highlight the impact of the multi skilled human resources on the performance of the production system under investigation.

The remainder of the chapter is organized as follows: in Section 2.2 a brief introduction to the proposed issue is presented; Section 2.3 deals with the MILP model describing the problem; in Section 2.4 all proposed GAs are discussed; Section 2.5 illustrates the calibration procedure applied to such metaheuristics, while in Section 2.6 numerical results arisen from an extensive experimental campaign are presented; in Section 2.7, the results obtained by taking into account multi skilled human resources are compared with those pertaining to three different scenarios featured by workers having identical skills.

2.2 Problem description

The proposed unrelated parallel machine problem with limited human resources can be stated as follows. Let us consider 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 minimizing 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 $w = 1,\ldots,W$ 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 to be executed for a given job on a given machine needs a worker that could be employed in another workstation for the setup task required by 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 given machine i, by a worker k, may be defined as follows:

$$S_{ikjl} = \eta_k \cdot S_{ijl}, \tag{2.1}$$

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 k-th. Expert workers are supposed to have η_k lower than 1 while novice workers are characterized 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 populated by different transforming technologies like injection moulding, compression moulding and PE pipe extrusion. In order to optimize the workforce utilization 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 characterized by different skills, conforming to the different technologies installed in the shop-floor.

2.3 MILP model

A first goal of the proposed research was the development of a Mixed Integer Linear Programming (MILP) model aiming at both 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 is assumed to be processed by all machines, and to be always processed as first, though it has a processing time equal to zero. Such approach is necessary to run setup times of jobs which are processed as first in a given machine; thus, job 0 precedes each job processed as first in a each machine. The aforementioned mathematical model is reported as follows.

Indices

$h, j, l = 0,1,\ldots,n$ Jobs;

$i = 1,2,\ldots,m$ Machines;

$k = 1,2,\ldots,w$ Workers;

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;

Binary variables

X_{ikjl} $\begin{cases} 1 \text{ if job } l \text{ is processed on machine } i \text{ after job } j \text{ and setup is performed by worker } k \\ 0 \text{ otherwise} \end{cases}$

Q_{jl} auxiliary variable for *either-or* constraint;

Continuous variables

CS_l setup completion time of job *l;*

C_l processing completion time of job *l;*

$C_{\max}$ Makespan;

Model

minimize $C_{\max}$

subject to:

$$\sum_{i=1}^{m}\sum_{k=1}^{w}\sum_{j=0}^{n} X_{ikjl} = 1 \qquad \forall\, l = 1,2,...,n. \tag{2.2}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{w}\sum_{l=1}^{n} X_{ikjl} \le 1 \qquad \forall\, j = 1,2,...,n. \tag{2.3}$$

$$\sum_{k=1}^{w}\sum_{l=1}^{n} X_{ik0l} \le 1 \qquad \forall\, i = 1,2,...,m. \tag{2.4}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{w}\sum_{l=1}^{n} X_{ikll} = 0 \tag{2.5}$$

$$\sum_{k=1}^{w}\sum_{j=0}^{n} X_{ikjl} \ge \sum_{k=1}^{w}\sum_{h=1}^{n} X_{iklh} \qquad \forall\, i = 1,2,...,m;\ l = 1,2,...,n. \tag{2.6}$$

$$C_l - CS_l \ge \sum_{i=1}^{m}\sum_{k=1}^{w}\sum_{j=0}^{n} T_{il} \cdot X_{ikjl} \qquad \forall\, l = 1,2,...,n. \tag{2.7}$$

$$CS_l - C_j \ge \sum_{i=1}^{m}\sum_{k=1}^{w} S_{ikjl} \cdot X_{ikjl} - B \cdot (1 - \sum_{i=1}^{m}\sum_{k=1}^{w} X_{ikjl}) \qquad \forall\, j = 0,1,...,n;\ l = 1,2,...,n. \tag{2.8}$$

$$\begin{cases} CS_l - CS_j \ge \sum_{i=1}^{m}\sum_{h=0}^{n} S_{ikhl} \cdot X_{ikhl} - B \cdot (2 - \sum_{i=1}^{m}\sum_{h=0}^{n} (X_{ikhl} + X_{ikhj}) + Q_{jl}) \\ CS_j - CS_l \ge \sum_{i=1}^{m}\sum_{h=0}^{n} S_{ikhj} \cdot X_{ikhj} - B \cdot (2 - \sum_{i=1}^{m}\sum_{h=0}^{n} (X_{ikhl} + X_{ikhj}) + 1 - Q_{jl}) \end{cases} \tag{2.9}$$

$$\forall\, k = 1,2,...w;\ j = 1,2,...,n;$$
$$l = j+1, j+2,...,n.$$

$$C_0 = 0 \tag{2.10}$$

$$C_{\max} \ge C_j \qquad \forall\, j = 1,2,...,n. \tag{2.11}$$

$$X_{ikjl} \in \{0;1\} \qquad \forall\, i = 1,2,...,m;\ k = 1,2,...,w;\ j = 0,1,...,n;\ l = 1,2,...,n. \tag{2.12}$$

$$Q_{jl} \in \{0;1\} \qquad \forall\, j = 1,2,...,n;\ l = j+1, j+2,...,n; \tag{2.13}$$

Constraint (2.2) ensures that each job is assigned to one and only one machine, one and only one worker performs its setup, and such job is preceded by one and one only job. Constraint (2.3) states that each job must precede at most one other job. Constraint (2.4) forces job 0 to precede at most one other job in each given machine. Constraint (2.5) denotes that each job cannot precede itself. Constraint (2.6) 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 (2.7) 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 (2.8) 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 (2.9) 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 (2.10) assigns a null completion time to job 0. Constraint (2.11) forces makespan to be equal to or greater than any job completion time. Finally, constraints (2.12) and (2.13) define the corresponding binary variables.

2.4 The proposed genetic algorithms

Three kinds of different metaheuristic procedures based on genetic algorithms (GAs) have been developed in order to address the large-sized instances of the proposed problem.

Genetic algorithms (Holland 1975) 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 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 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 in letting population evolve through successive generations until some stopping criterion is reached.

Whenever an optimization problem is addressed through evolutionary algorithms, 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 effectiveness and the efficiency viewpoints (Costa, Celano, Fichera & Trovato, 2010). 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.

Permutation-based GA

A first approach towards the resolution of the scheduling problem here investigated consisted in the development of a genetic algorithm equipped with a permutation-based encoding scheme, hereinafter called PGA. In such optimization procedure each chromosome directly describes the order in which jobs have to be processed in the manufacturing stage, while both the jobs and the workers assignment issue are performed by means of proper decoding procedure, on the basis of a time-saving (also known as *list-scheduling*) rule.

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, let $\bar{l} = \pi(r)$ be the job in the r-th position (r = 1,2,…,n) of the considered permutation string to be scheduled within an unrelated parallel machine production system with 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 a list-scheduling procedure. Thus, let us assume to have completed all decoding steps for scheduling jobs preceding $\bar{l}$ in the permutation π. 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: $TM_i = 0$; $\lambda = 0$; $TW_k = 0$. Completion time $C_{\bar{l}}$ of job $\bar{l} = \pi(r)$ is calculated as follows:

$$C_{\bar{l}} = \min_{i,k} \left\{ E_{ik\bar{l}} \right\} \tag{2.14}$$

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:

$$E_{ik\bar{l}} = \max\left\{ TM_i ; TW_k \right\} + S_{ik\lambda\bar{l}} + T_{i\bar{l}} \tag{2.15}$$

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

$$TM_{i^*} = C_{\bar{l}} \tag{2.16}$$

$$TW_{k^*} = C_{\bar{l}} - T_{i^*\bar{l}} \tag{2.17}$$

Finally, after the aforementioned procedure has been performed for all jobs in the permutation, the makespan is calculated according to the following formula:

$$C_{\max} = \max_{\bar{l} \in \pi(1),\ldots,\pi(n)} \{C_{\bar{l}}\} \tag{2.18}$$

In order to provide a clear insight into the aforementioned encoding/decoding procedure, let us consider a brief example consisting of four jobs (n = 4) to be scheduled on an unrelated parallel machine manufacturing system composed by three workstations (m = 3) and characterized by having a workforce team with two workers (w = 2) to be employed for setup operations. For sake of simplicity, setup times are assumed to be sequence-independent. For each job j, Table 2.1 illustrates processing times as i (index of machine) changes, as well as setup times as i and k (index of worker) change.

Table 2.1. Processing and setup times for an example with n = 4, m = 3, w = 2.

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

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. With reference to the reported example, let us suppose to take into account the solution $\pi = \{4,1,2,3\}$. Table 2.2 summarizes the proposed decoding procedure applied to π. At each iteration r ($r = 1,2,\ldots,n$), the minimum value of $E_{ik\bar{l}}$, calculated according to equation (2.15), is denoted in bold type. Both the corresponding machine and worker, denoted as i^* and k^* respectively, are selected for processing job $\bar{l} = \pi(r)$, and both TM_{i^*} and TW_{k^*} are updated as well, according to equations (2.16) and (2.17). Finally, a makespan equal to 11 is obtained for the given solution, according to equation (2.18). Figure 2.1 shows the Gantt chart obtained by applying such decoding procedure.

Table 2.2. PGA decoding procedure for $\pi = \{4,1,2,3\}$.

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
		i=1	7	9							
1	4	i=2	**5**	8	2	1	0	5	0	3	0
		i=3	7	10							
		i=1	10	10							
2	1	i=2	9	10	3	2	0	5	3	3	2
		i=3	5	**3**							
		i=1	**10**	14							
3	2	i=2	11	12	1	1	10	5	3	8	2
		i=3	13	18							
		i=1	13	15							
4	3	i=2	21	23	3	2	10	5	11	8	7
		i=3	14	**11**							

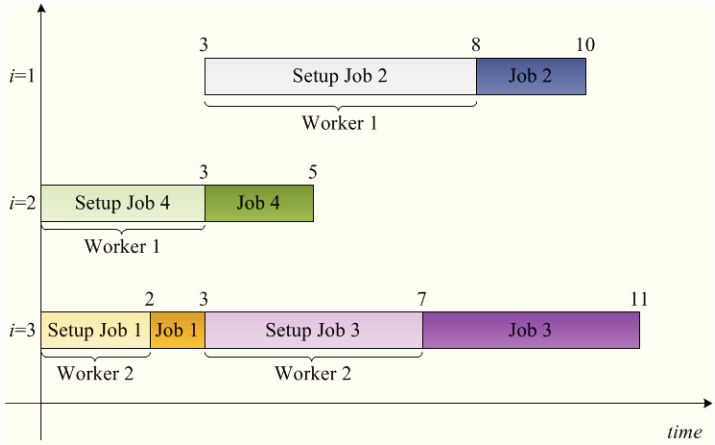

Figure 2.1. Gantt chart obtained by PGA decoding procedure for π={4,1,2,3}.

With regards to the selection mechanism, the well-known roulette-wheel scheme (Michalewicz 1994) has been considered, thus assigning to each solution a probability of being selected inversely proportional to makespan value. A position-based crossover (Syswerda 1991) 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. For sake of brevity, Figure 2.2 shows an example of position-based crossover for an instance with n = 10.

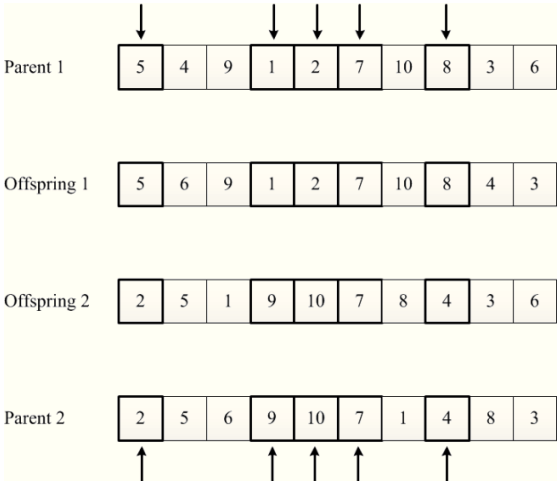

Figure 2.2. Position-based crossover.

As far as the mutation procedure is concerned, a simple swap operator (Oliver, 1987) has been chosen. According to 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.

Multi-encoding GA

The main characteristic of the proposed PGA is reducing the computational burden by means of a basic problem encoding. 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 genetic algorithm (hereinafter called MGA) equipped with a multi-stage encoding able to investigate a wider space of solution if compared to 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.
In order 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.

As concerns 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 10% 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 *r*-th position of the permutation α ($r = 1,2,\ldots,n$); $\bar{i} = \beta(\bar{l})$ 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* ($k = 1,2,\ldots,w$) to perform setup of job $\bar{l}$ on machine *i* ($i = 1,2,\ldots m$), after machine *i* has processed job *j* ($j = 0,1,\ldots,n$); $T_{i\bar{l}}$ 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 time-saving 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: $TM_i = 0$; $\lambda = 0$; $TW_k = 0$. Completion time $C_{\bar{l}}$ of job $\alpha(r)$ is calculated as follows:

$$C_{\bar{l}} = \begin{cases} E_{\bar{i}\bar{k}\bar{l}} & \text{if } \bar{i} \neq 0 \text{ and } \bar{k} \neq 0 \\ \min\limits_{k}\left\{E_{\bar{i}k\bar{l}}\right\} & \text{if } \bar{i} \neq 0 \text{ and } \bar{k} = 0 \\ \min\limits_{i}\left\{E_{i\bar{k}\bar{j}}\right\} & \text{if } \bar{i} = 0 \text{ and } \bar{k} \neq 0 \\ \min\limits_{i,k}\left\{E_{ik\bar{l}}\right\} & \text{if } \bar{i} = 0 \text{ and } \bar{k} = 0 \end{cases} \tag{2.19}$$

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:

$$E_{ik\bar{l}} = \max\left\{TM_i; TW_k\right\} + S_{ik\lambda\bar{l}} + T_{i\bar{l}} \tag{2.20}$$

According to the decoding procedure above described, the job is assigned to machine $\bar{i}$ if $\bar{i} \neq 0$ and to worker $\bar{k}$ if $\bar{k} \neq 0$; in case either β or γ do not hold any specific value, job $\bar{l}$ allocation to machines and workers is managed by minimizing the estimated completion time calculated through equation (2.20). Whether i^* and k^* denote machine and worker to which job $\bar{l}$ have to be assigned respectively, TM_{i^*} and TW_{k^*} can be updated as follows:

$$TM_{i^*} = C_{\bar{l}} \tag{2.21}$$

$$TW_{k^*} = C_{\bar{l}} - T_{i^*\bar{l}} \tag{2.22}$$

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

$$C_{\max} = \max_{\bar{l} \in \alpha(1),\ldots,\alpha(n)} \left\{ C_{\bar{l}} \right\} \tag{2.23}$$

To better explain the proposed decoding procedure, the same example reported in Section 4.1.1 (see Table 2.1) can be taken into account. Supposing the solution to be decoded is π={4,1,2,3|1,0,0,3|0,1,0,0}, Table 2.3 summarizes 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 according to equations (2.19), is highlighted by a bold type; 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 equations (2.21) and (2.22), respectively. Finally, a makespan equal to 10 is obtained by applying equation (2.23). Figure 2.3 shows the Gantt chart obtained by the proposed decoding procedure.

Table 2.3. MGA decoding procedure for π={4,1,2,3|1,0,0,3|0,1,0,0}.

r	$\bar{l}=\alpha(r)$	$\bar{i}$	$\bar{k}$	$E_{ik\bar{l}}$	k=1	k=2	i*	k*	TM_i	i=1	i=2	i=3	TW_k	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									

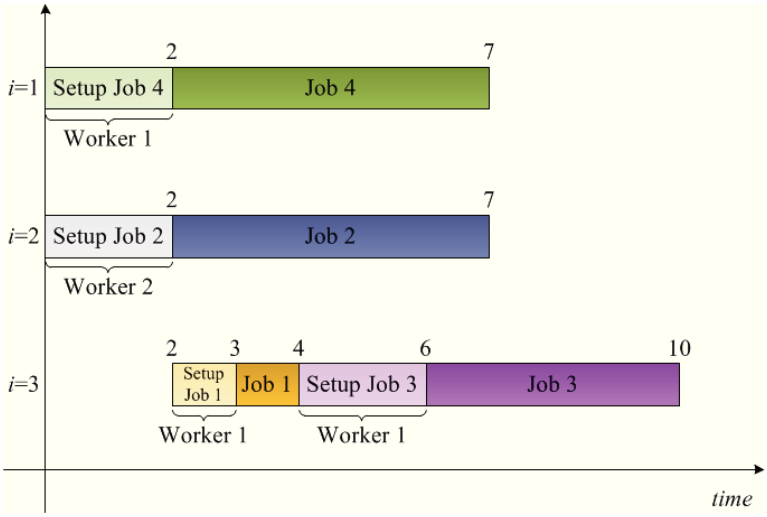

Figure 2.3. Gantt chart obtained by MGA decoding procedure for π={4,1,2,3|1,0,0,3|0,1,0,0}.

This 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 PGA encoding/decoding generates makespan equal to 11 (see Figure 2.1) while MGA encoding/decoding yields a makespan equal to 10 time units. For a given sequence of jobs α there exist a total number of alternative solutions equal to $(m+1)^n \cdot (w+1)^n$, thus

emphasizing the wider solution space that can be investigated by MGA encoding with respect to PGA.

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. On the other hand, crossover operator has been developed to separately run the mating between the three distinct parts composing the parents' structures (i.e., α, β, γ), with three distinct probabilities $p\alpha_{cross}$, $p\beta_{cross}$, $p\gamma_{cross}$. Crossover between two parent α-substrings has been 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 (Syswerda 1989) 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, Figure 2.4 illustrates the application of such genetic operator for the β-substrings of two parents; bold-bordering genes are those to be swapped.

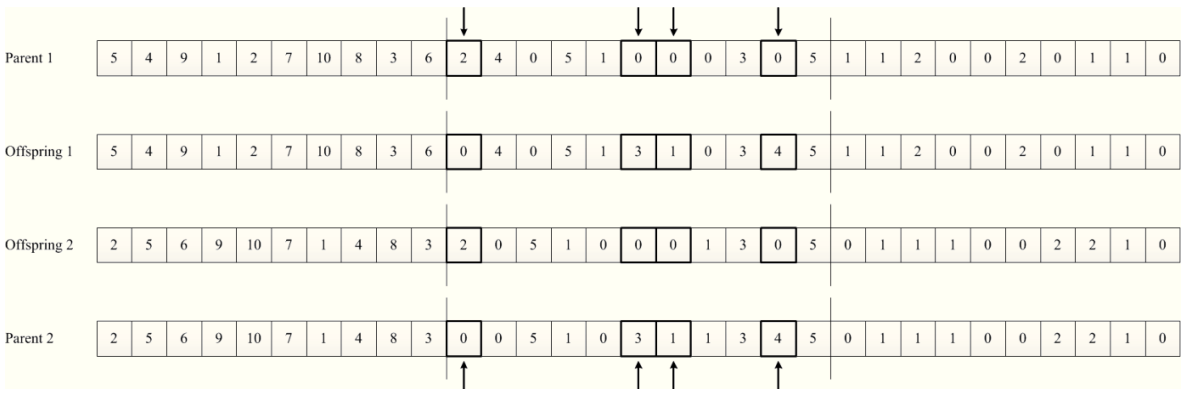

Figure 2.4. Uniform crossover operator.

Similarly to crossover, mutation procedure has been performed by separately managing the three parts of the chromosome, using three distinct probabilities $p\alpha_{mut}$, $p\beta_{mut}$, $p\gamma_{mut}$. Mutation of the permutational substring has been performed through the same swap operator used in the PGA procedure. With reference to assignment arrays, a simple uniform mutation operator (Michalewicz 1994) 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.

Hybrid GA

A hybrid genetic algorithm, hereinafter named HGA, which combines both the aforementioned metaheuristics 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 computational rapidity of PGA in the first phase and the ability of MGA in investigating a wider space of solutions. PGA performs the first optimization phase then, after a provided threshold is achieved and a proper encoding conversion procedure is executed, MGA is launched until the stopping criterion is encountered. The encoding conversion procedure operates by adding the two assignment substrings to the chromosomes characterizing the last population. Firstly, all values of new added assignment substring are set equal 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 2.5 shows an example of such procedure for an instance having n=10, m=5, w=2.

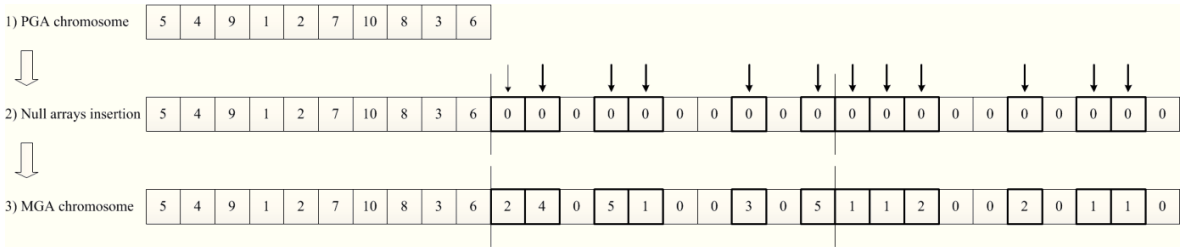

Figure 2.5. Encoding conversion procedure.

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 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 optimization process, until the total number of makespan evaluations is achieved.

2.5 Experimental calibration and test cases

Before carrying out an extensive comparison among the metaheuristics above discussed, 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 2.4. The benchmark is characterized by 4 x 2 x 2 = 16 small problems (with $n \leq 10$) and 7 x 4 x 3 = 84 large problems (with $n \geq 20$), as to ensure the effectiveness of the tuned parameters over a wide range of test cases.

Table 2.4. 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)

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 2.5

illustrates parameters tested for each algorithm, and denotes in bold type the best combination of values, chosen after an ANOVA analysis (Montgomery, 2007) 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:

$$RPD = \frac{\mathrm{GA}_{sol} - \mathrm{BEST}_{sol}}{\mathrm{BEST}_{sol}} \cdot 100 \tag{2.24}$$

Where GA_{sol} is the solution found by a given GA setting (i.e., a GA characterized by a specific combination of genetic parameters), and BEST_{sol} is the best solution among those provided by the other differently set GAs for the same instance. GAs have been coded in MATLAB® language and executed on a 2GB RAM PC powered by a dual-core 2,39 GHz processor. Stopping criterion was set to a total of 10,000 makespan evaluations. A preliminary experimental analysis has shown a substantial convergence of all proposed GAs within this limit. Furthermore, the use of such a stopping criterion allows to reach a good compromise between quality of solutions and computational burden.

Table 2.5. 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)	$p\alpha_{cross}$	(0.5; **0.8**)
	Crossover probability (substring β)	$p\beta_{cross}$	(**0.5**; 0.8)
	Crossover probability (substring γ)	$p\gamma_{cross}$	(**0.5**; 0.8)
	Mutation probability (permutation substring α)	$p\alpha_{mut}$	(0.05; **0.2**)
	Mutation probability (substring β)	$p\beta_{mut}$	(**0.05**; 0.2)
	Mutation probability (substring γ)	$p\gamma_{mut}$	(**0.05**; 0.2)

As far as the HGA procedure is concerned, no 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-phases hybrid metaheuristic. ANOVA outputs reported in Table 2.5 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 further assignment substrings.

2.6 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}, 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 10 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, comparison analysis has been fulfilled over a total amount of 1,000 different instances. Indeed, as five runs characterized 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.

Small instances comparison analysis

Conforming to the calibration phase debated in Section 2.5, small-sized instances (i.e., those having $n \leq 10$) have been arranged in 4 scenario problems depending on the number of jobs (n = 7, 8, 9, 10). Each scenario problem includes 4 classes of problems, each one involving a different number of both machines (m = 4, 5) and workers (w = 3, 4). Since each class of problems holds 10 instances and each instance was replicated 5 times with different random

seeds, 160 x 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 2.5.

The overall set of instances was optimally solved by means of the proposed MILP model, executed on an IBM® ILOG CPLEX Optimization 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 Relative Percentage Deviation (RPD) calculated according to equation (2.24). Due to the optimality of the MILP-based approach $BEST_{sol}$ relates to the global optimum found by CPLEX® tool. In Table 2.6, the average value of RPD obtained by each GA for a given class of problems (encompassing 10 different instances replicated 5 times) is reported, 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® optimizer. Final row of Table 2.6 reports the grand averages (*g_ave*) in terms of RPD and number of optimal solutions concerning each metaheuristic.

Table 2.6. Average performances of GAs on small-sized instances

Problem (n x m x w)	Average RPD					Average CPU time (s)				
	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}
(7 x 4 x 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 x 4 x 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 x 5 x 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 x 5 x 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 x 4 x 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 x 4 x 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 x 5 x 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 x 5 x 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 x 4 x 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 x 4 x 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 x 5 x 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 x 5 x 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

Table 2.6. (Continued)

Problem (n x m x w)	Average RPD					Average CPU time (s)				
	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}
(10 x 4 x 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 x 4 x 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 x 5 x 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 x 5 x 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**

Obtained results show 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. In order to infer some statistical conclusion about the aforementioned difference of performance, a proper ANOVA analysis has been performed. Figure 2.6 reports the means plot with LSD intervals ($\alpha = 0.05$) obtained through Design-Expert® 7.0 platform.

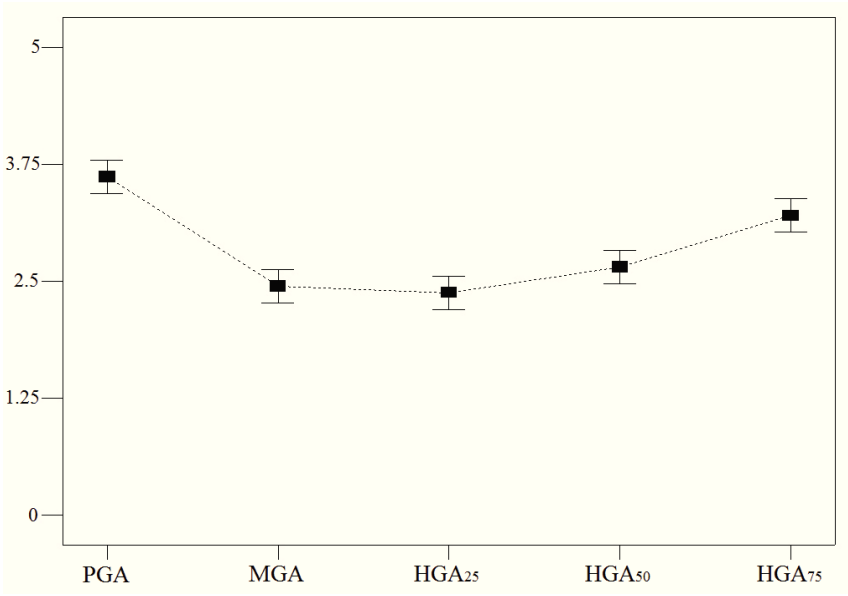

Figure 2.6. Means plot and LSD intervals for small instances

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 optimization procedures. Larger-sized problems surely may highlight a significant performance difference among the genetic algorithms under examination.

Large instances comparison analysis

In this subsection 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 2.5, seven scenario problems depending on the number of jobs (n = 20, 40, 60 , 80, 100, 150, 200) to be scheduled have been arranged. Each scenario problem includes four classes of problem at varying number of machines (m = 10, 12, 16, 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, 10). Since ten randomly generated instances have been provided for class of problems and 5 replications with different random seeds have been arranged for each instance, a total amount of 840 x 5 = 4,200 runs have been taken into account 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 2.5. Due to the size of the problems included within this benchmark, MILP cannot achieve any global optimum. The Relative Percentage Deviation (see eq. 2.24) has been taken into account as 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 2.7 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., 10 instances replicated 5 times). Last row shows the grand average of both RPD and $\Delta CPU_{\%PGA}$ over the entire set of investigated instances.

Table 2.7. Average performances of GAs on large-sized instances

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

Table 2.7. (continued)

Problem (n x m x w)	Average RPD					Average $\Delta CPU_{\%PGA}$			
	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}	MGA	HGA_{25}	HGA_{50}	HGA_{75}
(60 x 20 x 8)	4.69	5.21	3.85	4.96	4.38	31%	21%	15%	7%
(60 x 20 x 10)	4.42	4.79	3.75	4.11	4.28	30%	21%	15%	8%
(80 x 10 x 6)	3.87	2.75	3.62	2.92	3.57	43%	29%	20%	10%
(80 x 10 x 8)	4.61	4.71	2.96	4.52	3.01	41%	28%	19%	9%
(80 x 10 x 10)	4.33	4.20	2.83	3.88	3.69	39%	27%	19%	9%
(80 x 12 x 6)	4.43	3.03	3.84	3.87	3.49	41%	27%	19%	9%
(80 x 12 x 8)	4.93	3.40	3.70	4.18	5.10	38%	25%	18%	9%
(80 x 12 x 10)	4.09	3.32	2.10	3.08	4.32	36%	25%	18%	9%
(80 x 16 x 6)	6.44	4.25	3.69	3.73	5.71	36%	25%	18%	8%
(80 x 16 x 8)	4.71	4.59	4.28	3.08	3.98	34%	24%	16%	8%
(80 x 16 x 10)	4.48	2.97	2.72	4.05	4.51	32%	22%	15%	8%
(80 x 20 x 6)	3.72	4.64	3.99	3.91	3.09	34%	24%	17%	9%
(80 x 20 x 8)	3.62	3.38	2.85	3.18	2.75	31%	22%	15%	8%
(80 x 20 x 10)	3.33	3.46	2.76	2.88	3.27	29%	19%	14%	8%
(100 x 10 x 6)	3.11	3.11	3.42	2.40	3.18	44%	30%	22%	10%
(100 x 10 x 8)	3.00	2.84	2.65	2.49	2.76	43%	29%	21%	10%
(100 x 10 x 10)	3.12	2.23	2.27	2.25	3.00	40%	26%	20%	10%
(100 x 12 x 6)	3.59	3.81	3.02	2.62	3.78	43%	27%	20%	9%
(100 x 12 x 8)	3.68	3.62	2.21	2.50	3.23	39%	25%	18%	9%
(100 x 12 x 10)	3.62	3.65	3.03	3.23	2.82	37%	24%	17%	9%
(100 x 16 x 6)	4.03	2.94	3.26	3.75	3.51	37%	24%	18%	8%
(100 x 16 x 8)	4.13	2.43	3.46	3.26	4.04	34%	23%	17%	8%
(100 x 16 x 10)	3.77	3.61	2.93	3.13	3.02	32%	22%	15%	8%
(100 x 20 x 6)	3.59	2.98	3.68	3.67	3.72	33%	23%	16%	8%
(100 x 20 x 8)	3.42	5.17	2.54	3.25	3.01	30%	21%	16%	8%
(100 x 20 x 10)	4.37	3.42	3.26	2.99	3.90	28%	20%	15%	9%
(150 x 10 x 6)	1.94	2.45	2.13	1.27	1.75	41%	28%	20%	10%
(150 x 10 x 8)	2.76	2.65	1.40	1.85	2.75	38%	25%	18%	10%
(150 x 10 x 10)	2.42	3.38	1.58	2.75	2.45	34%	22%	16%	8%
(150 x 12 x 6)	2.42	3.04	1.70	2.32	2.61	36%	25%	18%	9%
(150 x 12 x 8)	2.49	2.92	2.36	1.67	2.20	32%	23%	17%	9%
(150 x 12 x 10)	2.26	2.53	1.62	1.86	1.69	30%	22%	16%	9%
(150 x 16 x 6)	2.75	4.17	2.29	2.31	2.38	32%	22%	16%	9%
(150 x 16 x 8)	2.46	3.22	2.18	1.45	2.19	29%	21%	16%	10%

Table 2.7. (continued)

Problem (n x m x w)	Average RPD					Average $\Delta CPU_{\%PGA}$			
	PGA	MGA	HGA_{25}	HGA_{50}	HGA_{75}	MGA	HGA_{25}	HGA_{50}	HGA_{75}
(150 x 16 x 10)	2.50	3.14	1.61	1.50	1.89	26%	18%	14%	8%
(150 x 20 x 6)	3.48	3.03	2.67	2.94	3.09	28%	19%	14%	8%
(150 x 20 x 8)	2.35	3.07	1.64	1.50	1.77	23%	16%	12%	7%
(150 x 20 x 10)	2.11	3.38	2.53	1.97	2.29	21%	15%	11%	7%
(200 x 10 x 6)	1.72	1.68	1.70	1.40	1.58	43%	27%	20%	11%
(200 x 10 x 8)	2.37	1.96	2.16	1.78	2.15	36%	24%	18%	9%
(200 x 10 x 10)	2.20	2.58	0.98	1.65	2.04	33%	22%	17%	9%
(200 x 12 x 6)	1.94	2.18	1.70	0.98	1.78	36%	25%	19%	10%
(200 x 12 x 8)	1.73	1.22	1.60	1.96	1.54	31%	22%	17%	8%
(200 x 12 x 10)	1.68	1.73	1.39	1.33	1.95	29%	20%	15%	9%
(200 x 16 x 6)	2.50	2.14	2.07	2.19	1.80	30%	21%	16%	9%
(200 x 16 x 8)	2.05	2.67	1.79	1.76	1.84	26%	19%	14%	8%
(200 x 16 x 10)	2.55	2.50	2.11	1.96	2.34	24%	17%	13%	7%
(200 x 20 x 6)	2.14	2.35	1.50	1.61	1.64	27%	19%	15%	8%
(200 x 20 x 8)	1.74	2.85	1.39	1.47	1.91	22%	17%	13%	7%
(200 x 20 x 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%***

HGA_{25} outperforms on the average the other competitors for optimizing 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 2.7 reports the means plot with 95% confidence level LSD intervals.

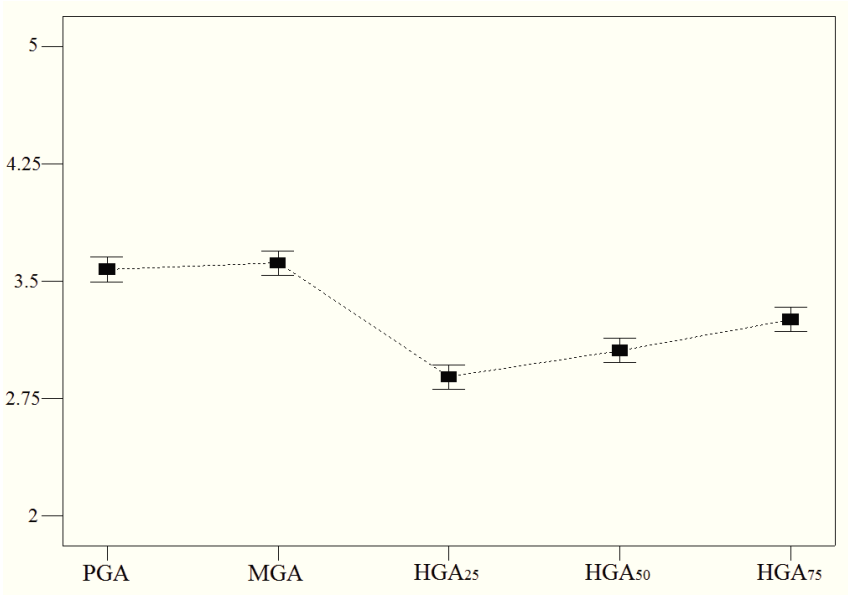

Figure 2.7. Means plot and LSD intervals for the large instances campaign.

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, and gets even worse than those obtained by PGA.

Without loss of generality, Figure 2.7 shows as all the three versions of HGA significantly outperform the two single encoding-based GAs (i.e. PGA and MGA). The reason for such an outcome can be explained if the differences among exploration and exploitation power of the proposed approaches are considered.

The encoding scheme employed by PGA generates a high exploration pressure. In fact, since job sequencing and assignment policies of jobs to machines and workers are all driven by the same permutation string, every chromosome modification introduced by crossover or mutation operators generates substantial changes in individuals. As a consequence, the search process if often pushed towards new areas of the solution space. On the other hand, the multi-encoding scheme adopted by MGA allows a higher exploitation power. In fact, whenever mutation or crossover operators are applied, new solutions may be generated by singularly changing the sequence of jobs, or the assignment of jobs to machines, or the allocation of workers to setup operations. Thus, the ability of visiting new solutions in the immediate neighborhood of the previously generated ones gets considerably higher, at the point of being excessive for coping

with large-sized instances, where an extensive search over different regions of the search space is required. The hybrid approach employed by HGA allows to fairly combine exploration and exploitation power, following the principle that the exploration process should be performed first in order to discover the potential search zones, then exploitation should fine-tune in order to reach a valid final solution. The 25% encoding switch threshold seems to be the one that provides the best balance between the two phases.

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 optimization algorithm among those tested, i.e. HGA_{25}, have been selected for a further step of analysis. The benchmark employed for evaluating the performances of such metaheuristic involved a set of instances wherein the skill level of each worker was assumed to randomly range between 0.5 and 1.5. Such configuration will be hereinafter denoted as the Multi Skill (MS) scenario.
In order to highlight the effect of the non-homogeneous workforce on the makespan minimization objective, the following three further configurations, each one featured by workers having identical skill levels, have been taken into account:

- High Skill scenario (HS): each worker k has skill level $\eta_k = 0.5$;
- Average Skill scenario (AS): each worker k has skill level $\eta_k = 1$;
- Low Skill scenario (LS): each worker k has skill level $\eta_k = 1.5$;.

For each one of the 840 large-sized instances generated within the reference benchmark, three more variants have thus been created, the only difference being in the matrix of setup times, generated according to the skill level pertaining to each scenario, respectively. Small sized instances have not been taken into account, as the effect of workforce teams having different skill levels is negligible in such kind of test cases, due to the short number of setup operations required.
Each new instance has been solved 5 times by HGA_{25} optimization procedure. Since the three new scenarios do not involve multi skilled human resources, the substring γ driving the

assignment of jobs to workers has been removed from the encoding scheme of the metaheuristic. A simple "first available worker" rule has been coded for selecting the operator to be assigned to each setup operation.

The average makespan values obtained for HS, AS, LS and MS scenarios are reported in Table 2.8, along with the relative percentage increment of completion times calculated for AS, LS and MS configurations with respect to the high skill scenario ($\Delta Cmax_{\%HS}$).

Table 2.8. Average performances of HGA_{25} at varying of workforce scenarios.

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

Table 2.8. (continued)

Problem (n x m x w)	Average makespan				Average $\Delta Cmax_{\%HS}$		
	HS	AS	LS	MS	HS	AS	LS
(60 x 10 x 10)	177.44	242.94	297.04	212.22	37.1%	67.6%	19.6%
(60 x 12 x 6)	155.54	213.54	266.44	197.74	37.5%	71.6%	27.4%
(60 x 12 x 8)	142.06	196.8	240.14	183.9	38.8%	69.4%	29.7%
(60 x 12 x 10)	139.7	190.94	235.48	172.38	37.0%	68.9%	23.6%
(60 x 16 x 6)	116.72	163.12	197.24	151.7	40.1%	69.7%	30.4%
(60 x 16 x 8)	104.3	143.68	179.5	137.32	37.9%	72.5%	31.9%
(60 x 16 x 10)	96.56	135.94	166.92	125.12	41.0%	73.4%	29.9%
(60 x 20 x 6)	96.9	135.42	166.9	124.64	40.2%	72.8%	28.8%
(60 x 20 x 8)	85.54	116.98	147.42	114.48	37.0%	72.6%	34.0%
(60 x 20 x 10)	79.02	111.6	137	103.88	41.5%	73.7%	31.7%
(80 x 10 x 6)	252	354.04	434.78	337.14	40.7%	72.8%	33.9%
(80 x 10 x 8)	240.16	326.56	404.2	296.38	36.3%	68.7%	23.5%
(80 x 10 x 10)	238.76	327.72	401.86	292.08	37.6%	68.7%	22.7%
(80 x 12 x 6)	209.12	292.2	362.18	268.72	39.9%	73.4%	28.7%
(80 x 12 x 8)	191.5	264.12	327.24	247.66	38.1%	71.1%	29.5%
(80 x 12 x 10)	189.26	260.5	318.4	235.7	37.8%	68.5%	24.7%
(80 x 16 x 6)	160.24	223.14	276	216.1	39.4%	72.4%	34.9%
(80 x 16 x 8)	140.08	196.18	242.66	184.36	40.3%	73.6%	31.9%
(80 x 16 x 10)	133.1	184.76	227.88	168.1	39.0%	71.5%	26.5%
(80 x 20 x 6)	131.82	183.22	230.06	176.18	39.4%	75.0%	34.1%
(80 x 20 x 8)	115.68	160.08	201.28	151.08	38.6%	74.3%	30.7%
(80 x 20 x 10)	106.9	147.88	183.56	135.3	38.6%	72.2%	26.8%
(100 x 10 x 6)	320.92	447.42	551.2	422.5	39.5%	71.9%	31.8%
(100 x 10 x 8)	304.68	423.18	520.92	392.04	39.1%	71.2%	28.9%
(100 x 10 x10)	308.66	420.36	511.48	374.62	36.3%	65.9%	21.5%
(100 x 12 x 6)	269.58	375.48	464.26	344.12	39.4%	72.3%	27.7%
(100 x 12 x 8)	242.44	341.62	415.04	306.72	41.0%	71.4%	26.7%
(100 x 12 x10)	236.24	331.44	401.34	297.04	40.4%	70.0%	26.0%
(100 x 16 x 6)	205.74	289.54	353.36	276.72	40.9%	71.9%	34.7%
(100 x 16 x 8)	178.76	248.62	307.66	234.06	39.2%	72.3%	31.0%
(100 x 16 x10)	165.88	231.34	285.66	213.38	39.6%	72.3%	28.7%
(100 x 20 x 6)	169.5	241.74	300.28	235.98	42.8%	77.4%	39.4%
(100 x 20 x 8)	144.78	203.72	253.46	192.36	40.9%	75.3%	33.0%
(100 x 20 x10)	131.34	184.4082	229.68	171.6	38.0%	75.0%	30.8%

Table 2.8. (continued)

Problem (n x m x w)	Average makespan				Average $\Delta Cmax_{\%HS}$		
	HS	AS	LS	MS	HS	AS	LS
(150 x 10 x 6)	495.34	689.68	856.18	647.12	39.3%	73.0%	30.7%
(150 x 10 x 8)	469.94	648.96	807.64	599.56	38.2%	72.0%	27.7%
(150 x 10 x10)	465.44	639.4	790.16	572.36	37.5%	69.9%	23.0%
(150 x 12 x 6)	407.22	571.48	711.84	531.6	40.5%	75.0%	30.7%
(150 x 12 x 8)	369.72	518.56	640.4	477.94	40.3%	73.3%	29.3%
(150 x 12 x10)	363.44	506.78	615.74	454	39.5%	69.5%	25.0%
(150 x 16 x 6)	314.2	438.9	548.7	413.24	39.8%	74.8%	31.7%
(150 x 16 x 8)	273.44	381.78	469.82	360.4	39.7%	72.0%	31.9%
(150 x 16 x10)	251.94	350.16	435.04	319.44	39.1%	72.8%	26.8%
(150 x 20 x 6)	265.12	366.64	461.52	351.54	38.4%	74.2%	32.7%
(150 x 20 x 8)	222.94	311.06	390.02	298.96	39.6%	75.0%	34.1%
(150 x 20 x10)	198.3	277.56	345.88	258.04	40.0%	74.5%	30.2%
(200 x 10 x 6)	666.48	935.04	1152.72	862.58	40.3%	73.0%	29.5%
(200 x 10 x 8)	636.34	883.04	1084.52	807.34	38.9%	70.6%	26.9%
(200 x 10 x10)	630.58	873.98	1073.8	778.6	38.7%	70.4%	23.5%
(200 x 12 x 6)	554.42	784.5	966.68	729.9	41.5%	74.5%	31.7%
(200 x 12 x 8)	500.7	703.7	872.02	637.46	40.6%	74.3%	27.4%
(200 x 12 x10)	487.76	680.3	833.1	618.98	39.5%	70.8%	26.9%
(200 x 16 x 6)	431.56	607.48	753.12	559.34	40.8%	74.6%	29.7%
(200 x 16 x 8)	366.78	515.7	634.38	477.58	40.7%	73.1%	30.3%
(200 x 16 x10)	339.22	473.58	585.82	431.62	39.7%	72.8%	27.2%
(200 x 20 x 6)	360.78	505.88	630	471.58	40.3%	74.7%	30.8%
(200 x 20 x 8)	300.68	420.12	522.52	404.52	39.8%	73.9%	34.6%
(200 x 20 x 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%**

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. On the other hand, LS configuration systematically provides the highest completion times. A more significant conclusion can be drawn from comparing AS and MS scenarios. The multi skilled configuration involves teams of workers whose skill levels are randomly extracted from 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 1. As the reader can notice, results obtained by the MS scenario always outperform those provided by the AS configuration, which entails workforce teams with the same average skill level, but does not assume any difference among technical abilities featuring each operator. This outcome is visually presented in Figure 2.8, which reports the average makespan value for each scenario as the number of jobs changes.

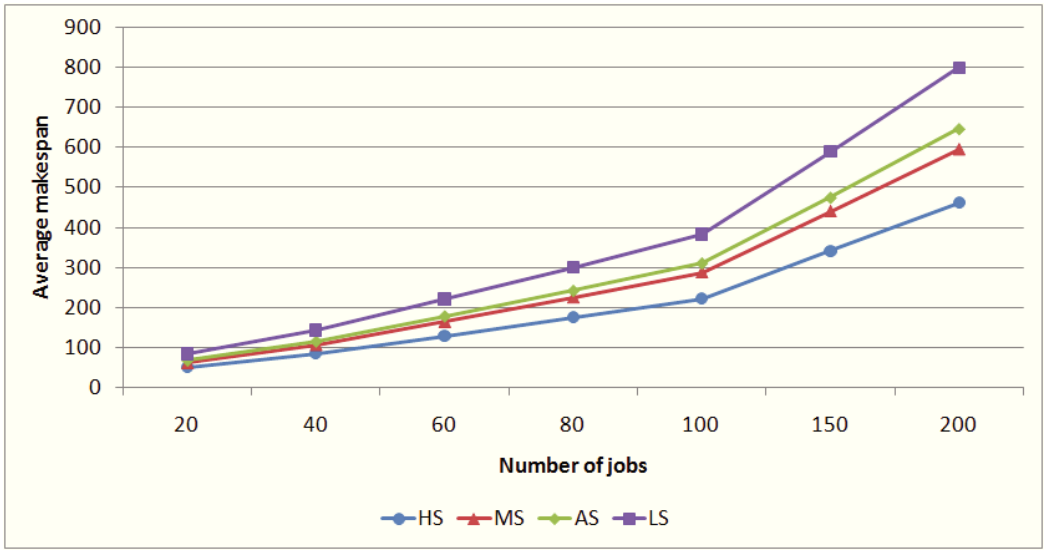

Figure 2.8. Average makespan vs. number of jobs at varying of workforce scenarios

The reason for such an effect lies in the fact that the optimization strategy embedded within the proposed metaheuristic procedure allows to reach a fair balance between the workload assigned to each operator and his own technical abilities. As a consequence, the most experienced workers are committed in performing the majority of setup operations, and this eventually leads to a substantial improvement of performances compared with the homogeneous workforce scenario. In conclusion, the impact of a multi skilled workforce on the unrelated parallel machine system under investigation may be significantly positive if a proper strategy for optimally managing human resources is adopted.

Chapter 3

The flow shop sequence dependent group scheduling problem with skilled workforce assignment

3.1 Preliminaries

Flow Shop Group Scheduling (FSGS) problems have attracted several researchers due to their frequent industrial implications. Similarly to classical flow shop manufacturing environments, FSGS systems are made of serial machines that have to be visited by all jobs following the same pre-determined path. Moreover, FSGS models entail Group Technology (GT) manufacturing principles, according to which the total set of jobs may be divided into different subsets, called families or groups, wherein each set of jobs have similar technological requirements in terms of tooling and setups. Since different groups of jobs need different tooling, a setup is often necessary whenever a job of a new group has to be produced. If such setup time depends on the technological features of the previously processed group, the issue may be classified as a Flow Shop Sequence-Dependent Group Scheduling (FSDGS) problem. As jobs belonging to the same group have the same technological requirements, setup times from one part to another are often negligible, as in this study. On the other hand, since there exists a major setup change between part families, there is an advantage to process together parts belonging to the same group. This is the key difference between the flow shop group scheduling model and regular flow shops. Nevertheless, FSGS problems still are permutation scheduling issues, as each feasible solution may be described as a simple sequence of jobs passing through each machine belonging to the shop floor.

As far as group technology is concerned, production planning involves short-term operational decisions which entail both job sequencing within each group and group sequencing, in order to achieve a specific goal fixed by the company's management. Though it is well-known as production scheduling based on job sequencing constitutes a driver towards cost saving and productivity growth as well, workforce management in terms of worker allocation to machines

may represent a further competitive leverage for manufacturing companies. In fact, whether job setup times are sequence-dependent, and setup operations must be performed by human resources having different skills, the worker allocation issue strongly may influence both the performance of the overall production system and the resources exploitation along the provided planning horizon.

Typical examples of manufacturing environments where human resource is critical for the set-up activities are the flexible manufacturing cells, where mechanical parts (hereinafter called jobs) are produced by CNC work centres: jobs to be processed are grouped into families due their similarity, (e.g. same/similar morphology, same/similar technological features), so that those jobs have to visit the working machines in the same order. Setup time of a single job is negligible, but setup time of a group may be significant and depends on the technological requirements of the previously processed group, i.e. setup times among groups are sequence-dependent. In addition, as mentioned before, each worker assigned to a machine, (according to his specific skills) manually performs the setup operations required by a given group of jobs (e.g. job fixing on the pallet, tool path and machining parameters programming). The completion time of a given job within a given machine arises from the sum of a fixed processing time and a variable setup time. Job setup time depends on the order of the groups and on the ability of the worker performing the set-up of those groups. Decision problems to be faced are: sequencing of jobs within each group, sequencing of groups and assignment of each worker with specific skills to a given machine.

It is worth noting as both flow shop and workforce assignment problems separately have been studied by literature so far, also due to the high level of complexity characterizing the mixed problem.

Both FSGS and FSDGS problems have been extensively dealt with in the recent literature, as confirmed by the reviews performed by Allahverdi, Gupta and Aldowaisian (1999), Cheng, Gupta and Wang (2000), and Zhu and Wilhelm (2006), respectively. Schaller, Gupta and Vakharia (2000) proposed a heuristic algorithm and a lower bound aiming to test the efficiency of their procedure. França, Gupta, Mendes, Moscato and Veltink (2005) developed a Genetic Algorithm (GA) and a Memetic Algorithm (MA) to solve the FSDGS problem; they also emphasized the performance of their optimization techniques with respect to the heuristics proposed by Schaller, Gupta and Vakharia (2000). Hendizadeh, Faramarzi,

Mansouri, Gupta and Elmekkawy (2008) and Salmasi and Logedran (2008), proposed a metaheuristic algorithm based on a Tabu Search (TS) optimization procedure. Celano, Costa and Fichera (2010) developed an evolutionary algorithm to solve a real group scheduling problem which pertains to the inspection department of a semiconductors company; the issue was handled as a permutation flow shop group scheduling problem with sequence-dependent setup times and limited inter-operational buffer capacity. Salmasi, Logendran and Skandari (2010) studied a FSDGS problem by proposing a MILP formulation and two metaheuristic procedures, namely a TS and a Hybrid Ant Colony Optimization (HACO) algorithm, which were compared over an extensive benchmark of test cases. Obtained results revealed the superiority of the latter procedure with respect to the total flow time minimization objective. One year later, Salmasi, Logendran and Skandari (2011) adopted the HACO approach for minimizing makespan in a similar production environment; such algorithm was also taken as reference by Hajinejad, Salmasi and Mokhtari (2011) who succeeded in getting superior performances through the use of a fast hybrid Particle Swarm Optimization (PSO) algorithm. Finally, Naderi and Salmasi (2012) proposed a comprehensive study regarding the employment of different MILP modelling techniques to cope with the FSDGS issue; their best performing mathematical model revealed its effectiveness in tackling problems up to 10 groups and almost 60 jobs in total.

With reference to the impact of human workers on the global performances of a production system, it can be noticed that the study of such a field is establishing a new trend in both operational and production research. Nevertheless, a wide literature addressed this kind of issue for a lot of years. A pioneer research was carried out by Hunter (1986) who proposed a model to measure worker ability in learning and obtaining information from the process. This model allows a classification of the workers on the basis of individual differences, called skills, and evaluates the potential of cross-training and productivity. Fitzpatrick, Askin and Ronald (2005) presented a mathematical model that exploits labour skill pools for arranging any team of workers; performance of such a team depends on individual behaviours and interpersonal interactions of workers as well as on their technical competences. Lodree, Christofer, Geiger and Xiaochun (2009) presented a survey on the human factor literature and constructed a framework for scheduling human tasks accounting for physical and/or cognitive human characteristics and behaviours; workers were characterised by specific skills and

assigned to tasks or to teams. Askin and Huang (2001) proposed a heuristic algorithm to solve the team formation and worker assignment problem minimising a multi-objective model consisting of training costs, the requirements of the task and the team synergy. Norman, Tharmmaphornphilas, Lascola Needy, Bidanda, Colosimo Warner and Worker (2002) proposed a mathematical model to assign each operation to each worker and to quantify the amount of training each worker should receive for enhancing his skill level for each ability. As far as profit maximization is concerned, McDonald, Ellis, Van Aken and Koelling (2009) developed a model that assigns workers to tasks within a lean manufacturing cell, thus minimizing the net present cost. In determining how to assign workers to tasks, the model matches production requirements with customer demand, skill level required by tasks, quality levels based on skill levels and job rotation to retain skills for a cross-trained workforce. In a labour intensive context like service centres where task scheduling and workforce assignment play a key role for increasing their efficiency, this kind of problem is denoted as the Skilled Workforce Project Scheduling (SWPS) problem and entails constraints on task times and variable task durations, depending on the worker efficiency. Valls, Perez and Quintanilla (2009) proposed a hybrid genetic algorithm to obtain a feasible scheduling plan and a balanced and efficient assignment of workforce to tasks for a SWPS problem. Corominas, Olivella and Pastor (2010) considered the problem of assigning and scheduling a set of tasks to a set of workers, according to which worker's performance for a given task depends on the worker experience.

Solving problems that mix both scheduling and planning is a tricky challenge as described by Perron (2010) who tried to solve the following problem: assembling teams of skilled workers to perform jobs that require these skills, breaking up these teams and then assembling new ones to perform more jobs. However, the joint problem involving both job sequencing and assignment of a set of workers with different skills to machines has not been discussed by the body of literature so far

In this chapter, a *m*-machines, *n*-jobs flow shop is considered as the reference scheduling problem: different numbers of jobs are grouped into families accordingly to group technology principles and set-up time of each group is both sequence and human-skill dependent. The objective is to minimize the completion time of the last job of the last scheduled group on the last machine, i.e. the makespan. Thus, the optimization problem can be stated as minimizing

the makespan by selecting the optimal sequence of both jobs within groups and groups each other and by finding the most efficient assignment of the skilled workers to each machine. Once a worker is assigned to a machine he will work there for the overall set of groups/jobs to be performed. The problem in hand may be defined as a flow shop sequence-dependent group scheduling problem with Skilled Workforce Assignment (SWA).

To the best of our knowledge, the integration of FSDGS and SWA has never been studied by literature; thus, this chapter has multiple goals: formalizing a mathematical model for the joint problem (FSDGS-SWA); evaluating a set of optimization strategies by comparing them over a well-known benchmark arisen from literature; assessing the effectiveness of the best-performing method against a reference algorithm developed in the field of classical FSDGS problems. Finally, as different skills may affect both productivity and manpower cost, an extensive analysis concerning the trade-off between manpower cost and makespan improvement has been developed in order to chart the guidelines a decision maker should follow for identifying the best configuration of workforce to be employed in a serial production system.

The remainder of the chpater is organized as follows: Section 3.2 presents the mixed integer linear programming mathematical model for the proposed FSDGS-SWA problem. Section 3.3 shows the structure and the adopted operators of three proposed Genetic Algorithms, characterized by different ways to run the sequencing and the worker allocation problem, properly developed for the problem in hand. Section 3.4 reports the results of an extensive comparison among the three GAs carried over a reference benchmark arisen from literature. In Section 3.5 the best heuristic procedure among those proposed is tested against a well-performing algorithm recently presented with reference to classical FSDGS problems. Finally, Section 3.6 shows how workforce skills may affect the productivity of a serial production system like a flow shop and illustrates the guidelines for selecting the best trade-off between makespan reduction and manpower skill upgrade cost.

3.2 The FSDGS-SWA mathematical model

The proposed mathematical model integrates multi-skilled workforce assignment with the flow shop sequence-dependent group scheduling problem by means of a Mixed Integer Linear Programming (MILP) approach. According to the formalization proposed in Pinedo (2012),

this problem can be denoted as $Fm|fmls, S_{igh}, prmu|C_{max}$, where Fm indicates a flow-shop with m machines in series, *fmls* indicates that the jobs are assigned to different groups (or families), S_{igh} means that setup times of groups are machine and sequence-dependent, i.e., that setup time of group h on machine i depends on the preceding group g, *prmu* refers to a permutation type process (that is, all jobs and groups are processed by respecting the same order on each machine), while $C_{\max}$, i.e., the makespan, is the objective to be minimized.

Workers are required to manually perform setup operations on each group of jobs to be processed on each machine. An anticipatory setup is allowed: that is, a worker can start the setup of a group, even if the first job of the group is not yet available on the machine. Workers do not constitute a limited resource; that is, a worker from the team is assigned to a machine and he is always available for setup operations, thus preventing that machine from starvation and consequently avoiding any makespan increase. Setup times between two jobs pertaining to the same group are included into their own processing time and are both sequence and worker independent. Workers have different skill levels, which may vary from one machine to another. Three levels of expertise are considered for a worker: senior, normal and junior. Senior level denotes a worker who is, on the average, more experienced than lower-level workers, thanks to his longer work experience. Senior worker can complete a setup task in a shorter time than his lower-level colleagues. Normal level concerns a worker enough experienced but with lacking skills with respect to a senior worker. Finally, the junior level can be associated to a not-experienced worker, e.g., a worker recently hired. Of course, a junior worker requires a longer time to complete a set-up. Multiple skill levels can be defined for each level of expertise: the larger is the skill level index ($SL_{wm} \leq 1$) of a worker w performing a set-up on machine m, the shorter is the required set-up time since SL_{wm} is a divisor coefficient of set-up times. The optimization problem consists of assigning the set of workers to machines on one hand, and scheduling both groups of jobs and jobs within each group on the other hand.

Mathematical model proposed in this chapter may be denoted as a development of the model proposed by Naderi and Salmasi (2012), the main difference being represented by the adaptation to the SWA issue. One of the key elements of the proposed model is the so-called "slot". A slot is a position of the sequence of jobs within each group that should be occupied just by a single job. For this reason, a one-to-one correspondence between jobs of a given

group and slots of that group must be considered. In order to properly describe the partition of the total number of jobs into g different groups, the parameter G_k (k = 1,2,...,g) is employed, i.e., G_k denotes the set of jobs belonging to group k.

The indices and parameters, the decision variables and the mathematical model are as follows:

Indices/parameters:

n_0	number of jobs
m	number of machines
g	number of groups
$j = 1,2,\ldots,n_0$	index of jobs
$i = 1,2,\ldots,m$	index of machines
$q = 1,2,\ldots,m$	index of workers
$k, t = 0,1,\ldots,g$	index of groups
G_k	set of jobs belonging to group k
$n_k = \|G_k\|$	number of jobs belonging to group k
$l = 1,2,\ldots n_k$	index of slots
p_{ji}	processing time of job j on machine i
a_{tkiq}	setup time of group k performed by worker q on machine i, if group k is processed immediately after group t,
M	a big number

Decision variables:

$$X_{lj} \quad \begin{cases} 1 \text{ if job } j \text{ is assigned to slot } l \text{ of group } k \\ 0 \text{ otherwise} \end{cases} \qquad \begin{array}{l} k = 1,2,\ldots,g \\ l = 1,2,\ldots,n_k \\ j \in G_k \end{array}$$

$$U_{tk} \quad \begin{cases} 1 \text{ if group } k \text{ is processed immediately after group } t \\ 0 \text{ otherwise} \end{cases} \qquad \begin{array}{l} t = 0,1,\ldots,g \\ k = 1,2,\ldots,g \\ t \neq k \end{array}$$

Z_{iq} $\begin{cases} 1 \text{ if worker } q \text{ is assigned to machine } i \\ 0 \text{ otherwise} \end{cases}$ $i, q = 1,2,\ldots,m$

C_{kli} completion time of job processed in slot l of group k on machine i

F_{ki} finishing time of group k on machine i

S_{ki} starting time of group k on machine i

$C_{\max}$ makespan

Model

minimize $C_{\max}$

Subject to:

$$\sum_{l=1}^{n_k} X_{jl} = 1 \qquad k = 1,2,...,g \quad j \in G_k \tag{3.1}$$

$$\sum_{j \in G_k} X_{jl} = 1 \qquad k = 1,2,...,g \quad l = 1,2,...n_k \tag{3.2}$$

$$C_{kli} \geq C_{k(l-1)i} + \sum_{j \in G_k} X_{jl} \cdot p_{ji} \qquad k = 1,2,...,g \quad l = 2,3,...n_k \quad i = 1,2,...,m \tag{3.3}$$

$$C_{kli} \geq C_{kl(i-1)} + \sum_{j \in G_k} X_{jl} \cdot p_{ji} \qquad k = 1,2,...,g \quad l = 1,2,...n_k \quad i = 2,3,...,m \tag{3.4}$$

$$C_{k1i} \geq S_{ki} + \sum_{j \in G_k} X_{j1} \cdot p_{ji} \qquad k = 1,2,...,g \quad i = 1,2,...,m \tag{3.5}$$

$$F_{ki} \geq C_{kn_k i} \qquad k = 1,2,...,g \quad i = 1,2,...,m \tag{3.6}$$

$$S_{ki} \geq F_{ti} + \sum_{q=1}^{m} Z_{iq} \cdot a_{tkiq} - (1 - U_{tk}) \cdot M \qquad t = 0,1,...g \quad k = 1,2,...,g \quad k \neq t \quad i = 1,2,...,m \tag{3.7}$$

$$\sum_{\substack{t=0 \\ t \neq k}}^{g} U_{tk} = 1 \qquad k = 1,2,...,g \tag{3.8}$$

$$\sum_{\substack{k=1 \\ k \neq t}}^{g} U_{tk} \leq 1 \qquad t = 1,2,...,g \tag{3.9}$$

$$\sum_{k=1}^{g} U_{0k} = 1 \tag{3.10}$$

$$\sum_{i=1}^{m} Z_{iq} = 1 \qquad q = 1,2,...,m \tag{3.11}$$

$$\sum_{q=1}^{m} Z_{iq} = 1 \qquad i = 1,2,...,m \tag{3.12}$$

$$C_{\max} \geq F_{km} \qquad k = 1,2,...,g \tag{3.13}$$

$$C_{kli}, S_{ki}, F_{ki} \geq 0 \tag{3.14}$$

$$X_{lj}, U_{tk}, Z_{iq} \in \{0;1\} \tag{3.15}$$

Constraint (3.1) ensures that each job is assigned to exactly one slot within the group it belongs to. Constraint (3.2) states that each slot of any given group must be occupied by only one job. Through constraint (3.3) it is imposed that each job cannot start before the job assigned to the previous slot of the same group has been finished. Constraint (3.4) forces each job to start on a given machine after it has been completed on the previous one. Constraint (3.5) links the completion time of the job processed as first in each group to the starting time of the group itself, while constraint (3.6) links the completion time of the job processed as last in each group to the finishing time of the group itself. Constraint (3.7) states that each group can start to be processed on a given machine after the preceding group has been completed and the setup has been performed. Constraints (3.8), (3.9) and (3.10) define precedence relationships among groups: they ensure that each group is preceded by exactly one group, being followed by one other group at most, and that the dummy group 0 precedes only one group. Constraint (3.11) forces each machine to be assigned to one only worker; conversely, constraint (3.12) assigns each worker to only one machine. Through constraint (3.13), the makespan is set to be equal to the highest among finishing times of all groups. Constraints (3.14) establishes the non-negativity of continuous variables, while constraint (3.15) defines the binary variables.

3.3 The genetic algorithm approach

On the basis of what stated by Schaller, Gupta and Vakharia (2000), makespan minimization of a FSDGS problem is $\mathcal{NP}$ hard. As a consequence, the proposed FSDGS-SWA problem may be denoted as $\mathcal{NP}$ hard too, due to the larger domain of solutions arising from the SWA issue. Whenever a $\mathcal{NP}$ hard combinatorial problem needs to be solved, metaheuristic algorithms may represent an effective and timesaving alternative to other exhaustive approaches. Since Costa, Celano, Fichera and Trovato (2010) demonstrated both efficacy and efficiency of Genetic Algorithms (GAs) for solving the FSDGS problem, this chapter focuses on three modified GAs fitting the group scheduling and multi-skilled workforce assignment joint problem.

A GA is a metaheuristic optimization procedure that mimics the natural evolution of individuals. It starts with a set of feasible solutions (the *initial population*) and iteratively replaces the current population by a new population generation based on a mechanism that preserves the most promising solutions. It requires a suitable *representation* of the solution domain (*chromosome encoding*) and a *fitness function* that measures the quality of each possible solution. The population evolution proceeds by means of a *reproduction mechanism* called *crossover*, which selects one or more couples of chromosomes (the *parents*) and recombines them to generate the *children* (two new chromosomes). Children having higher fitness performance than parents replace them into the new population. Search towards unexplored areas of the solutions domain is assured by means of a mutation operator, which can randomly alter one or more chromosomes within a population. A termination condition allows the generational process to be stopped within a finite computational time.

The first proposed algorithm for solving the FSDGS-SWA problem in hand is a hybrid genetic algorithm, hereinafter called GAI, which integrates the evolutionary optimization strategy with a local search mechanism aiming to enhance the overall performances of the procedure. A detailed description of the algorithm is reported in the following sub-sections.

Problem encoding

Problem encoding is the way a given problem to be optimized through metaheuristics is represented by a numerical chromosome. For the proposed GAI, each solution is a

combination of three distinct portions of chromosome and each chromosome is encoded by means of a properly arranged matrix. Making use of the same notation introduced in the MILP model Section, the first part of each chromosome comprises the sequences π_l^k (k = 1,2,...,g; l = 1,2,...,n_k) of jobs to be assigned to each slot within each group; the second portion entails the sequence of groups Ω_k (k = 1,2,...,g); finally, workers assignment to each machine is managed by the third portion named Θ_i (i = 1,2,...,m). Basically, the overall chromosome is encoded by means of a partitioned $(g + 2) \times n_{\max}$ matrix where $n_{\max} = \max_{k=1}^{g} \{n_k\}$:

$$\left[\begin{array}{c} \pi_1^1,....,\pi_{n_1}^1 \\ \cdots \\ \pi_1^k,....,\pi_{n_k}^k \\ \cdots \\ \pi_1^g,....,\pi_{n_g}^g \\ \hline \Omega_1,....,\Omega_g \\ \hline \Theta_1,....,\Theta_m \end{array}\right] \tag{3.16}$$

Each row r (r = 1,2,…,g) of the partitioned matrix codes a specific schedule concerning the problem in hand and it is worth pointing out that it is independent from the other sequences; hereinafter it will be denoted as *sub-chromosome*. Thus, a certain sub-chromosome r corresponds to the sequence of jobs scheduled within group r; sub-chromosome $r = g + 1$ identifies the sequence Ω of groups. Finally, sub-chromosome $r = g + 2$, namely Θ, runs the workforce assignment to the provided m machines. For example, a feasible solution for a problem in which m = 5, g = 3 and $n_{\max}$ = 3 can be represented by the following chromosome [C1]:

$$[C1] = \left[\begin{array}{ccccc} 1 & 2 & 0 & 0 & 0 \\ 3 & 1 & 2 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ \hline 1 & 2 & 3 & 0 & 0 \\ \hline 3 & 1 & 5 & 2 & 4 \end{array}\right] \tag{3.17}$$

Sub-chromosomes from 1 to 3 hold the schedules of jobs within each group (i.e., schedule 1-2 for group 1, schedule 3-1-2 for group 2, schedule 2-1 for group 3); sub-chromosome $r = g + 1$ fixes the sequence of groups Ω = 1-2 -3, while sub-chromosome corresponding to the sequence $\Theta = 3{-}1{-}5{-}2{-}4$ assigns worker 3 to machine1, worker 1 to machine 2 and so on. All the digits equal to zero neither take part to the solution decoding nor to the genetic evolutionary process. Once the problem encoding is defined, the fitness function of N_s individuals pertaining to the genetic population may be computed.

Crossover operator

Crossover works by recombining the genetic material of two parent chromosomes selected from the current population with a probability that is directly proportional to their own fitness value. Basically, a couple of sub-chromosomes is selected from the parents based on an a priori fixed probability, hereinafter called $pcross_{sel}$. Two methods of crossover operators have been adopted to recombine alleles within each couple of sub-chromosomes and to generate offsprings: they are denoted as *Position Based Crossover* (PBC) and *Two Point Crossover* (TPC), respectively. Both these two crossover operators have been largely adopted by literature within GAs applied to combinatorial problems; PBC generates offspring by considering the relative order in which some alleles are positioned within the parents. Indeed, it works on a couple of sub-chromosomes (P1) and (P2) as follows: 1) one or more alleles are randomly selected; 2) the alleles genetic information of *parent 1* (P1) are reordered in the *offspring 1* (O1) in the same order as their appear within the second *parent 2* (P2); 3) remaining elements are positioned in the sequence by copying directly from the *parent 1* the unselected alleles. The same procedure is followed in the second parent, i.e., *parent 2*, to obtain offspring (O2). Figure 3.1 shows the implementation of the PBC on a couple of parents where alleles in positions {2}, {3} and {6} have been selected.

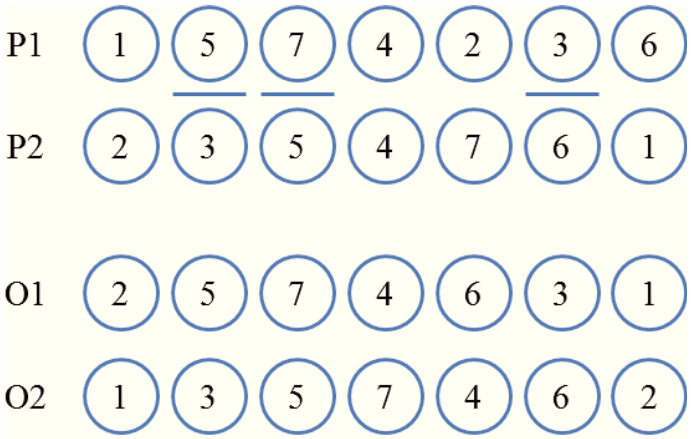

Figure 3.1. Position Based Crossover (PBC).

As far as is concerned with the TPC method, two positions are randomly selected and each sub-chromosome parent is divided into three blocks of alleles: both head and tail blocks are copied directly in the corresponding offspring, while the alleles belonging to the middle block are reordered within the offspring in the same order as they appear in the other parent sub-chromosome (see Figure 3.2). Probability of selecting either PBC or TPC crossover is denoted as p_{cr}.

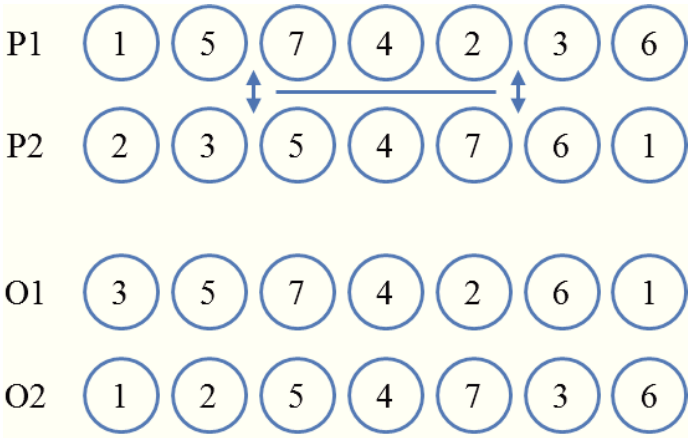

Figure 3.2. Two Point Crossover (TPC).

Mutation operator

Mutation may be applied with probability p_m to a chromosome belonging to the current population; such a chromosome is randomly selected on the basis of its fitness and it is

substituted by a new chromosome generated by means of the mutation operator. Each sub-chromosome of the selected solution is considered for mutation according to a probability $pmut_{sel}$; two kind of operators have been adopted in the present research: an *Allele Swapping Operator* (ASO), which performs an exchange of two randomly selected alleles of the sub-chromosome; and a *Block Swapping Operator* (BSO), which performs a block exchange (see Figure 3.3). To avoid any loss of the current best genetic information, the survival of the two current fittest individual within the population is ensured by an elitist strategy. Probability of selecting either ASO or BSO mutation operator has been denoted as p_{cm}.

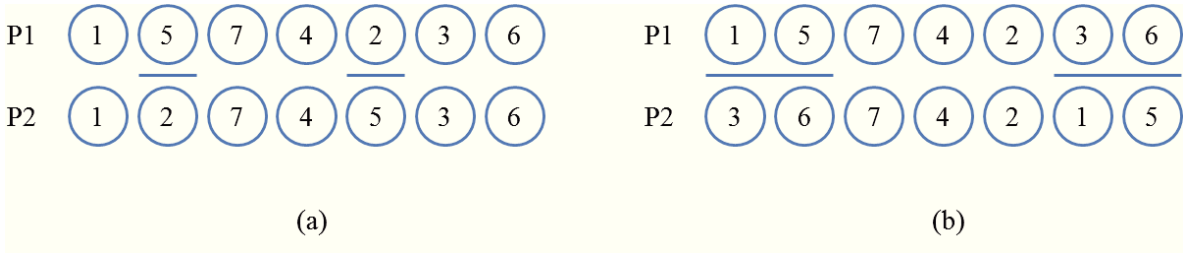

Figure 3.3. (a) Allele Swapping (ASO) and (b) Block Swapping (BSO) mutation Operators.

Diversity operator

A population diversity control technique has been embedded within the proposed optimization procedure, in order to mutate those identical chromosomes exceeding a pre-selected value $D_{\max}$ in the current population.

Local search and termination rule

In order to enhance the performances of the proposed genetic algorithm, a Group-based Local Search (GLS) scheme has been embedded within the evolutionary optimization strategy of GAI. Such procedure operates only on a fixed number (hereinafter called N_{GLS}) of the best individuals obtained after each generation. For each selected chromosome C_s (s = 1,2,...,N_{GLS}), it works by modifying the sequence Ω^{C_s} of groups, i.e., the sub-chromosome r = g + 1, pertaining to that solution, through a gene-swapping operator. Whether the new obtained solution leads to a better makespan value, it is used for the next iteration. After a total number

of iterations (It_{GLS}) is reached, the last accepted solution is used for replacing solution C_s in the newly generated population.
For sake of clarity, GLS pseudo-code is reported in detail in Procedure 3.1.

Procedure 3.1. Group-based Local Search (GLS) pseudo-code.

for s = 1 to N_{GLS}
    $C_s \leftarrow$ s-th best solution of last generated population;
    for it = 1 **to** It_{GLS}
        $\Omega^{C_s} \leftarrow$ sequence of groups pertaining to solution C_s;
        randomly select k and t in {1;2;...;g};
        generate C_s' from C_s by exchanging $\Omega_k^{C_s}$ with $\Omega_t^{C_s}$;
        calculate *makespan*(C_s');
        if *makespan*(C_s') < *makespan*(C_s) then
            $C_s \leftarrow C_s$'
        end if
    next it
    update C_s in the last generated population;
next s

The termination rule of the proposed GAI consists in 30 s of CPU time. This is the same criterion adopted by Hajinejad, Salmasi and Mokhtari (2011) and can thus allow a fair comparison with the metaheuristic procedure presented in such reference paper, as it will hereinafter be enlightened; moreover, preliminary experimental tests, here not reported for sake of brevity, have shown a substantial convergence of GAI within such time limit.
To sum up, the whole optimization procedure followed by GAI can be described through the steps reported in Procedure 3.2.

Procedure 3.2. GAI optimization procedure

Step 1: Initialization of parameters $pcross_{sel}$, p_{cr}, p_m, $pmut_{sel}$, p_{cm}, D_{max};
Step 2: Generation of initial random population of chromosomes;

Step 3: Application of Crossover Operator, chosen between Position Based and Two Point Crossover, to a couple of chromosomes chosen on the basis of their fitness (makespan);

Step 4: Generation of the new population after Crossover Operator through the insertion of the two best chromosomes individuated between parents and offspring: the two individuals with best values of fitness are introduced in the population;

Step 5: Evaluation of p_m. If it is verified go to Step 6 else go to Step 7;

Step 6: Application of Mutation Operator, chosen among two different Operators: Allele Swapping and Block Swapping. The operator is applied randomly to a chromosome of the population;

Step 7: Population Control: a mutation operator is applied on duplicates exceeding $D_{\max.}$;

Step 8: Application of GLS procedure to the N_{GLS} best individuals of the population;

Step 9: Updating of the current population, then return to Step 3.

Alternative optimization strategies

A further step regarding the study of the genetic algorithm approach for tackling the FSDGS-SWA problem in hand consisted in the comparison of the proposed GAI with two distinct modified GAs, each one characterized by a different way to run the SWA problem. It is worth pointing out that all the optimization procedures involved in such comparison analysis are subject to the same stopping criterion, namely 30 s of CPU time.

The former modified GA, hereinafter coded as GAR, consists of a regular genetic algorithm, integrated with the same GLS procedure exploited by GAI, optimizing the FSDGS problem; as concerns the SWA problem, a random assignation of skilled workers to machines is provided once and for all before the evolutionary process starts. GAR flow chart is reported in Figure 3.4 a).

The latter genetic technique works on the basis of a hierarchical optimization strategy; thus, from now on it will be defined as GAH. It consists of a regular GA combined with GLS as well as GAI that approaches the FSDGS problem at a first stage and then, once a threshold equal to half the time limit, i.e., 15 s of CPU time, is reached, it moves on the only SWA optimization problem for further 15 seconds. GAH flow chart is reported in Figure 3.4 b).

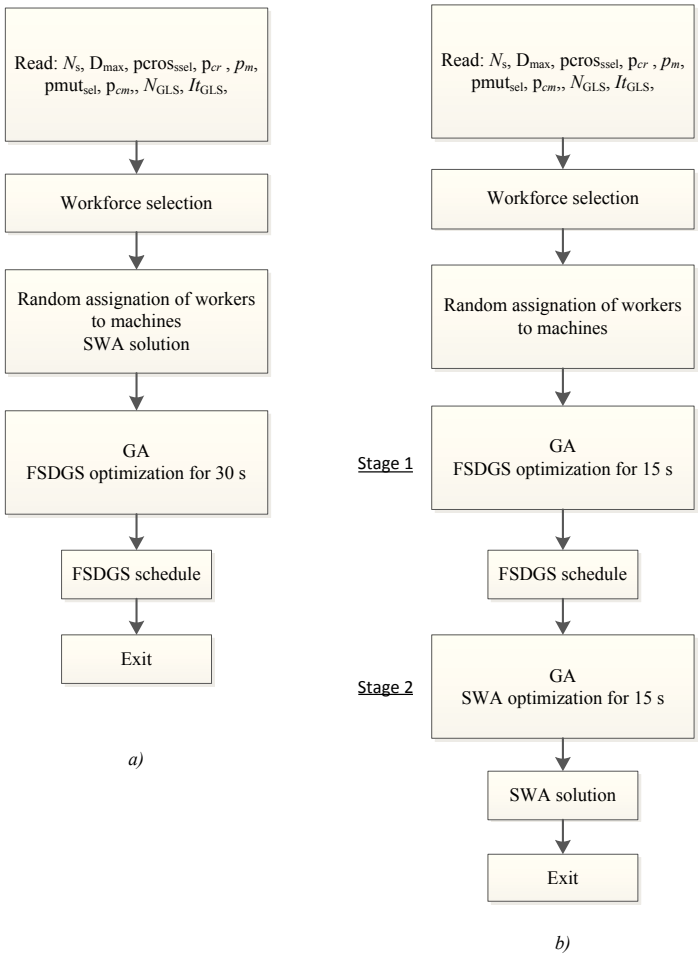

Figure 3.4. a) GAR and b) GAH flow charts.

3.4 Overview of the problem benchmark

In order to perform a fair comparison among the three proposed GAs, an extensive set of instances has been generated according to the approach delineated by Salmasi, Logendran and Skandari (2010) with respect to the FSDGS problem. Then, an adding set of data related to the skilled workforce assignment has been arranged. Basically, three separate sub-benchmarks characterized by different numbers of machines, i.e., 2, 3 and 6, respectively, have been considered. Within each sub-benchmark, three distinct factors, namely number of groups, number of jobs within groups and setup times of groups on each machine have been combined so to obtain a full factorial experimental plan as shown in Tables 3.1, 3.2 and 3.3.

Table 3.1. Benchmark of instances for the FDSGS problem with 2 machines.

Factor	Level	Value	
Number of groups *(g)*	1	U [1,5]	
	2	U [6,10]	
	3	U [11,16]	
Number of jobs in a group (*n*)	1	U [2,4]	
	2	U [5,7]	
	3	U [8,10]	
Setup times of machine M_i *(a)*	1	$M_1 \rightarrow$ U [1,50]	$M_2 \rightarrow$ U [17,67]
	2	$M_1 \rightarrow$ U [1,50]	$M_2 \rightarrow$ U [1,50]
	3	$M_1 \rightarrow$ U [17,67]	$M_2 \rightarrow$ U [1,50]

Table 3.2. Benchmark of instances for the FDSGS problem with 3 machines.

Factor	Level	Value		
Number of groups *(g)*	1	U [1,5]		
	2	U [6,10]		
	3	U [11,16]		
Number of jobs in a group (*n*)	1	U [2,4]		
	2	U [5,7]		
	3	U [8,10]		
Setup times of machine M_i *(a)*	1	$M_1 \rightarrow$ U [1,50]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [45,95]
	2	$M_1 \rightarrow$ U [17,67]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [17,67]
	3	$M_1 \rightarrow$ U [45,95]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [1,50]
	4	$M_1 \rightarrow$ U [1,50]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [17,67]
	5	$M_1 \rightarrow$ U [1,50]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [1,50]
	6	$M_1 \rightarrow$ U [17,67]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [45,95]
	7	$M_1 \rightarrow$ U [17,67]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [1,50]
	8	$M_1 \rightarrow$ U [45,95]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [45,95]
	9	$M_1 \rightarrow$ U [45,95]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [17,67]

Table 3.3. Benchmark of instances for the FDSGS problem with 6 machines.

Factor	Level	Value		
Number of groups *(g)*	1	U [1,5]		
	2	U [6,10]		
	3	U [11,16]		
Number of jobs in a group (*n*)	1	U [2,4]		
	2	U [5,7]		
	3	U [8,10]		
Setup times of machine M_i *(a)*	1	$M_1 \rightarrow$ U [1,50]	$M_2 \rightarrow$ U [17,67]	$M_3 \rightarrow$ U [45,95]
		$M_4 \rightarrow$ U [92,142]	$M_5 \rightarrow$ U [170,220]	$M_6 \rightarrow$ U [300,350]
	2	$M_1 \rightarrow$ U [1,50]	$M_2 \rightarrow$ U [1,50]	$M_3 \rightarrow$ U [1,50]
		$M_4 \rightarrow$ U [1,50]	$M_5 \rightarrow$ U [1,50]	$M_6 \rightarrow$ U [1,50]
	3	$M_1 \rightarrow$ U [300,350]	$M_2 \rightarrow$ U [170,220]	$M_3 \rightarrow$ U [92,142]
		$M_4 \rightarrow$ U [45,95]	$M_5 \rightarrow$ U [17,67]	$M_6 \rightarrow$ U [1,50]

It can be noticed that a total of 27 + 81 + 27 = 135 separate instances describing a consistent data set for the FSDGS problem have been generated. With reference to the SWA issue, additional data have been obtained by considering the relationship between skill level and expertise level of each worker. Three distinct skill levels have been provided for each worker and, in addition, every skill level may assume further three values according to the expertise level characterizing such worker. As reported before, the skill level of each worker depends on the experience he has accumulated over time by using a given manufacturing equipment or a given technology; thus, a specific SL_{qi} skill level out of three can be been assigned to each worker q for each machine i, once his level of expertise (senior, normal or junior) has been established. Skill levels of each worker allocated to a given machine may assume the following values: [1.0, 0.91, 0.83] for senior workers, [0.77, 0.71, 0.67] for normal workers and [0.63, 0.59, 0.56] for junior workers. The way the skill level affects the setup times consists of dividing such time a_{tki} of a given group k, following a group t in machine i, for the skill level SL_{qi} of worker q assigned to the same machine i. As for example, whenever a setup task on a given machine i is performed by a fully experienced senior worker q, $a_{tkiq} \leftarrow a_{tki}$ as $SL_{qi} = 1$; in alternative, whether the same task should be performed by a less skilled junior worker, setup time should get $a_{tki}/0.56$ as $SL_{qi} = 0.56$. In order to associate the provided nine

skill levels with the corresponding level of performance a worker can ensure for setup operations, hereinafter a normalized range of values as reported in Table 3.4 should be taken into account. Thus, it could be said that a senior worker with a skill level equal to 1.0 has a Normalized Work Ability (*NWA*) in setup operations equal to 100%, while a Junior worker with skill level equal to 0.59 may also be considered as a worker whose *NWA* is equal to 12.5% (see Table 3.4). In words, a further scale of normalized values may be adopted for emphasizing the ability level of each worker.

Table 3.4. Relationship between skill level and normalized work ability.

	Senior			Normal			Junior		
Skill levels	1.0	0.91	0.83	0.77	0.71	0.67	0.63	0.59	0.56
Normalized Work Ability (*NWA*) %	100.0	87.5	75.0	62.5	50.0	37.5	25.0	12.5	0.0

As the number of workers *w* employed on a flow-shop production system must be equal to the number of machines *m*, *m* workers compose each workforce team. For every one out of 135 instances generated by the FSDGS benchmark, nine workforce teams have been randomly assembled, each one having an average *NWA* level, hereinafter called $\overline{NWA}$, equal to one of the values presented in Table 3.4. Therefore, a workforce team with $\overline{NWA}$ = 100.0 is composed by only *m* fully skilled senior workers with SL_{qi} = 1.0, while a team having $\overline{NWA}$ = 25.0 is composed by *m* differently skilled workers whose average skill level, denoted as $\overline{SL}$, is equal to 0.63. Due to the specific average value of skill levels a workforce team should get, a proper procedure named Workforce Team Generation (WTG) has been developed (see Procedure 3.3); it is worth noting that a tolerance ξ = 0.1% has been considered with reference to the absolute gap between the $\overline{SL}$ value obtained through WTG method, and the average skill level considered as a target for each one out of the nine teams created. Since each FSDGS problem of the reference benchmark should be investigated at varying workforce teams, the benchmark concerning the proposed FSDGS-SWA problem holds a total amount of 135 $\times$ 9 = 1,215 instances.

Procedure 3.3: Workforce Team Generation (WTG) procedure.

for $q = 1$ **to** m

  for $i = 1$ **to** m

    $SL_{qi} = \mathrm{U}[0,1]$;

  next i

next q

calculate $\overline{SL}$;

$\Delta = SL_{\text{target}} - \overline{SL}$;

while $|\Delta| > \xi \cdot SL_{\text{target}}$ **do**

  randomly select q and i;

  if $\Delta > 0$ then

    $\varepsilon = \mathrm{U}[0, \min(\Delta; 1 - SL_{qi})]$;

  else

    $\varepsilon = -\mathrm{U}[0, \min(-\Delta; SL_{qi} - 0.56)]$;

  endif

  $SL_{qi} = SL_{qi} + \varepsilon$;

  calculate $\overline{SL}$;

  $\Delta = SL_{\text{target}} - \overline{SL}$;

end while

3.5 Computational experiments and results

After a comprehensive a set of experimental data has been generated, the proposed genetic algorithms have been thoroughly compared in order to identify the most appropriate optimization strategy for the FSDGS-SWA problem in hand. Then, the best metaheuristc method available in literature for the FSDGS problem, i.e. the fast Hybrid Particle Swarm Optimization (hereinafter HPSO) algorithm proposed by Hajinejad, Salmasi and Mokhtari (2011), has been taken as reference for assessing performances of the selected genetic algorithm. Computational details of such analysis are reported in the following sub-sections.

Selection of the best optimization strategy

On the basis of the aforementioned FSDGS-SWA benchmark, an extensive comparison has been performed among GAI, GAH, and GAR, running the parameters reported in Table 3.5. It is worth noting that such parameters have been selected after a preliminary tuning analysis not reported here for sake of brevity. As the quality of solutions of the three algorithms cannot be affected by the skilled workforce profiles equal to $NWA = 100$ and $NWA = 0$, only the remaining 7 skill levels have been taken into account for the comparison analysis. Moreover, 2 different test problems have been randomly generated for each instance of the considered benchmark drawing processing times of jobs in the range [1,20]. Therefore, $135 \times 7 \times 2 = 1{,}890$ problems have been created in all. All genetic algorithms have been coded in MATLAB® 7.6 and executed on a 2GB RAM virtual machine embedded on a workstation powered by two quad-core 2,39 GHz processors.

Table 3.5. Parameters of proposed genetic algorithms.

Parameter	Notation	Value
Population size	N_s	30
Maximum number of duplicates at each generation	$D_{\max}$	2
Probability of selecting a sub-chromosome for crossover	$pcross_{sel}$	0.85
Probability of selecting either PBC or TPC crossover operator	p_{cr}	0.5
Mutation probability	p_m	0.13
Probability of selecting a sub-chromosome for crossover	$pmut_{sel}$	0.5
Probability of selecting either ASO or BSO mutation operator	p_{cm}	0.5
Number of individuals subject to GLS	N_{GLS}	10
GLS iterations for each solution	It_{GLS}	4

In order to highlight the performance of each optimization strategy, let us denote with ΔIR and ΔIH the percentage deviation of completion times between the solutions provided by GAI vs. GAR and by GAI vs. GAH, respectively:

$$\Delta IR = \frac{\overline{C_{\max}}(\text{GAR}) - \overline{C_{\max}}(\text{GAI})}{\overline{C_{\max}}(\text{GAI})} \times 100 \tag{3.18}$$

$$\Delta IH = \frac{\overline{C_{\max}}(\text{GAH}) - \overline{C_{\max}}(\text{GAI})}{\overline{C_{\max}}(\text{GAI})} \times 100 \tag{3.19}$$

where $\overline{C_{\max}}(\text{GAX})$ refers to the average completion time yielded by a given GA (GAI, GAR, GAH) over 2 different test problems generated for each instance of the considered benchmark. A large positive value of ΔIR or ΔIH means that workforce allocation managed by the GAI, i.e. based on an integrated encoded strategy, may significantly affect the process performance in terms of makespan reduction. Tables 3.6, 3.7 and 3.8 show the average values of ΔIR and ΔIH for problems with 2, 3 and 6 machines, respectively.

Table 3.6. GAs percentage deviation analysis for 2-machine problems.

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			ΔIH	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*
1	1	1	0.0%	4.4%	0.0%	6.4%	0.0%	7.8%	0.0%	8.7%	0.0%	9.9%	0.0%	11.3%	0.0%	6.1%
1	1	2	0.0%	2.9%	0.0%	3.7%	0.0%	4.3%	0.0%	5.0%	0.0%	5.5%	0.0%	7.1%	4.0%	7.1%
1	1	3	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%	2.0%	0.0%	2.0%	0.2%	0.4%	0.0%	0.0%
1	2	1	0.0%	0.7%	0.0%	0.0%	0.1%	0.0%	0.0%	0.0%	0.8%	1.8%	4.0%	5.6%	1.2%	3.7%
1	2	2	0.0%	1.8%	0.0%	1.6%	0.0%	1.0%	0.0%	0.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
1	2	3	0.0%	0.9%	0.0%	0.0%	0.0%	2.3%	0.0%	6.0%	0.0%	9.5%	0.0%	8.2%	0.0%	4.9%
1	3	1	0.0%	0.0%	0.0%	0.1%	-0.2%	0.2%	0.2%	0.7%	0.1%	0.8%	1.2%	1.8%	0.0%	2.8%
1	3	2	0.0%	0.8%	0.5%	2.5%	0.8%	3.1%	1.1%	3.7%	1.4%	3.2%	1.9%	2.5%	1.1%	2.0%
1	3	3	0.0%	0.0%	0.0%	0.6%	0.2%	2.5%	0.1%	6.8%	0.0%	6.3%	0.0%	3.4%	0.0%	0.0%
2	1	1	1.5%	1.4%	3.8%	2.9%	2.0%	3.1%	5.1%	5.1%	4.3%	4.3%	0.6%	2.7%	1.1%	1.5%
2	1	2	0.0%	4.4%	0.0%	5.9%	0.2%	5.4%	0.0%	4.6%	0.5%	3.6%	0.2%	1.1%	0.0%	0.0%
2	1	3	0.3%	8.9%	0.0%	9.3%	1.4%	8.5%	1.0%	8.5%	-0.3%	4.2%	0.0%	0.1%	3.1%	0.4%
2	2	1	-0.9%	3.9%	1.5%	8.6%	1.7%	11.0%	-0.2%	10.3%	1.3%	10.1%	2.1%	7.9%	-0.1%	4.5%
2	2	2	0.9%	1.3%	2.3%	2.5%	1.0%	4.8%	-0.6%	2.4%	0.4%	2.3%	-0.3%	1.2%	0.4%	0.5%
2	2	3	0.1%	-0.6%	0.4%	0.0%	0.6%	0.4%	0.1%	0.0%	1.9%	1.9%	2.1%	3.5%	1.7%	1.3%
2	3	1	0.0%	3.3%	2.4%	5.6%	0.5%	6.1%	-0.3%	5.7%	-0.5%	5.9%	0.3%	4.7%	-0.2%	2.5%
2	3	2	0.0%	0.5%	0.0%	1.0%	0.1%	1.8%	0.2%	1.6%	0.0%	1.7%	0.4%	1.3%	0.6%	1.0%
2	3	3	0.4%	0.5%	0.5%	0.0%	0.5%	0.5%	1.5%	1.1%	0.2%	0.2%	0.8%	0.0%	0.0%	0.0%
3	1	1	3.1%	2.1%	0.1%	0.1%	2.9%	4.0%	1.9%	3.2%	5.8%	6.1%	4.2%	4.3%	2.7%	4.3%
3	1	2	-0.2%	-0.2%	-0.5%	0.8%	5.8%	4.8%	6.1%	6.1%	2.1%	2.1%	2.3%	4.1%	-1.1%	0.8%
3	1	3	2.8%	2.8%	-3.5%	-3.5%	-0.6%	-1.1%	-3.9%	-3.9%	-0.1%	-0.1%	-1.5%	-1.5%	3.5%	1.2%
3	2	1	1.0%	0.5%	1.0%	0.9%	2.3%	1.0%	0.6%	1.9%	2.4%	5.0%	1.7%	2.1%	1.7%	2.7%
3	2	2	3.5%	4.4%	2.3%	4.6%	2.8%	4.5%	1.8%	5.6%	2.7%	8.5%	2.7%	5.0%	4.1%	4.8%
3	2	3	0.4%	3.9%	0.7%	6.0%	3.4%	9.1%	2.7%	9.0%	0.6%	5.9%	-1.5%	2.6%	-0.1%	2.6%
3	3	1	1.0%	3.1%	1.6%	3.3%	5.1%	4.5%	5.7%	5.2%	2.1%	3.0%	2.9%	4.5%	2.2%	2.3%
3	3	2	-0.4%	-1.2%	-0.5%	-0.5%	2.0%	1.7%	2.6%	2.2%	-0.8%	-1.0%	-0.6%	0.0%	0.7%	1.4%
3	3	3	-0.2%	2.4%	1.6%	2.9%	0.4%	2.0%	-1.2%	2.0%	0.6%	3.0%	-0.1%	0.5%	2.7%	2.7%
average			***0.5%***	***2.0%***	***0.5%***	***2.4%***	***1.2%***	***3.5%***	***0.9%***	***3.8%***	***0.9%***	***3.9%***	***0.9%***	***3.1%***	***1.1%***	***2.3%***
		min	*-0.9%*	*-1.2%*	*-3.5%*	*-3.5%*	*-0.6%*	*-1.1%*	*-3.9%*	*-3.9%*	*-0.8%*	*-1.0%*	*-1.5%*	*-1.5%*	*-1.1%*	*0.0%*
		max	*3.5%*	*8.9%*	*3.8%*	*9.3%*	*5.8%*	*11.0%*	*6.1%*	*10.3%*	*5.8%*	*10.1%*	*4.2%*	*11.3%*	*4.1%*	*7.1%*

Table 3.7. GAs percentage deviation analysis for 3-machine problems.

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			ΔIH	ΔIR	ΔIH	ΔIR	ΔIH	ΔIR	ΔIH	ΔIR	ΔIH	ΔIR	ΔIH	ΔIR	ΔIH	ΔIR
1	1	1	0.0%	0.0%	0.6%	1.8%	0.0%	4.3%	0.0%	11.3%	0.3%	12.3%	0.0%	1.6%	0.1%	3.7%
1	1	2	1.3%	1.3%	0.0%	1.5%	0.7%	1.8%	1.1%	2.5%	0.0%	1.3%	0.6%	1.4%	0.0%	1.5%
1	1	3	0.0%	2.9%	0.0%	1.7%	0.0%	1.5%	0.0%	2.8%	0.0%	2.7%	0.0%	0.9%	0.0%	0.0%
1	1	4	3.5%	8.2%	4.3%	8.1%	3.1%	6.4%	3.7%	5.1%	2.8%	4.5%	0.1%	1.8%	0.0%	0.0%
1	1	5	0.0%	3.0%	0.0%	3.3%	0.0%	2.8%	0.0%	3.1%	0.2%	2.5%	0.0%	0.8%	0.0%	0.0%
1	1	6	0.0%	5.3%	0.0%	7.1%	0.0%	6.1%	0.0%	6.4%	0.0%	4.4%	0.0%	4.1%	0.0%	3.5%
1	1	7	1.1%	3.6%	0.0%	4.4%	1.3%	1.6%	1.0%	1.0%	0.0%	0.0%	0.0%	2.8%	0.2%	0.7%
1	1	8	6.8%	11.4%	3.3%	9.0%	9.4%	9.4%	3.3%	9.9%	5.1%	8.4%	5.8%	6.8%	6.1%	6.1%
1	1	9	0.0%	0.0%	0.0%	1.6%	0.0%	3.0%	0.0%	4.6%	0.0%	7.7%	0.0%	2.7%	0.0%	0.0%
1	2	1	0.0%	1.8%	0.0%	11.8%	0.0%	19.3%	0.0%	21.9%	0.0%	17.2%	0.0%	11.7%	0.0%	4.5%
1	2	2	0.0%	3.3%	0.0%	3.4%	0.0%	2.3%	1.1%	1.7%	0.4%	0.7%	0.0%	0.0%	0.0%	0.0%
1	2	3	0.0%	3.5%	1.2%	8.5%	-0.2%	11.8%	0.1%	17.7%	1.6%	18.2%	0.1%	10.2%	0.0%	0.6%
1	2	4	0.8%	1.6%	0.0%	0.0%	0.1%	0.1%	0.0%	-0.1%	0.1%	0.1%	0.7%	0.7%	2.3%	2.3%
1	2	5	0.0%	1.7%	0.8%	4.0%	0.6%	6.6%	1.5%	9.5%	4.2%	11.5%	2.7%	8.5%	1.7%	3.2%
1	2	6	5.7%	8.3%	2.5%	8.1%	5.6%	10.0%	4.8%	10.5%	3.6%	9.4%	3.1%	5.6%	0.0%	4.2%
1	2	7	3.4%	6.0%	2.2%	5.5%	2.0%	3.5%	0.0%	1.6%	0.0%	2.3%	2.3%	4.8%	0.0%	1.8%
1	2	8	0.0%	4.5%	0.0%	1.8%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
1	2	9	0.2%	1.4%	1.2%	3.5%	1.0%	3.5%	0.0%	4.0%	0.0%	4.2%	0.1%	5.1%	0.3%	0.9%
1	3	1	0.0%	2.8%	0.0%	4.9%	0.0%	4.2%	0.0%	6.2%	0.0%	7.8%	0.0%	6.1%	0.0%	3.6%
1	3	2	-0.1%	-0.1%	0.9%	1.0%	0.8%	2.9%	4.8%	5.0%	2.1%	6.3%	1.7%	6.8%	2.0%	4.1%
1	3	3	1.5%	6.1%	1.5%	6.7%	1.4%	7.2%	0.4%	5.7%	0.0%	3.8%	0.0%	0.4%	0.0%	0.0%
1	3	4	0.1%	4.0%	1.5%	8.4%	4.6%	8.6%	5.2%	9.7%	4.7%	9.4%	3.3%	7.9%	-1.1%	2.8%
1	3	5	0.4%	1.5%	2.5%	3.8%	2.9%	5.1%	2.3%	4.7%	1.2%	6.4%	0.9%	7.3%	1.1%	4.3%
1	3	6	1.3%	7.8%	0.0%	7.9%	0.2%	9.7%	0.5%	9.9%	0.1%	9.0%	0.0%	7.4%	0.0%	3.2%
1	3	7	0.0%	1.6%	0.1%	2.0%	0.0%	1.4%	0.3%	2.1%	0.6%	1.9%	0.0%	0.7%	0.0%	0.5%
1	3	8	1.3%	1.5%	1.9%	1.9%	3.0%	3.0%	3.6%	3.6%	1.0%	2.7%	2.4%	2.4%	0.9%	0.9%
1	3	9	2.9%	7.3%	3.6%	9.8%	2.7%	10.1%	0.7%	11.4%	0.7%	11.6%	1.3%	10.9%	2.6%	6.6%
2	1	1	0.0%	7.9%	1.6%	10.2%	1.8%	11.3%	2.2%	13.6%	2.3%	13.3%	1.0%	10.3%	1.6%	5.2%
2	1	2	4.1%	4.1%	5.8%	5.8%	2.1%	3.1%	2.5%	2.8%	2.0%	2.0%	2.0%	2.0%	0.8%	0.8%
2	1	3	0.0%	9.3%	-0.6%	17.7%	1.8%	23.6%	4.7%	24.0%	1.5%	20.8%	5.0%	18.9%	6.5%	14.0%
2	1	4	0.4%	0.8%	4.3%	4.3%	2.9%	2.7%	5.0%	5.6%	2.7%	2.2%	0.2%	0.7%	2.1%	1.1%
2	1	5	5.3%	7.8%	3.2%	9.7%	2.4%	6.1%	0.1%	3.3%	0.0%	1.7%	-0.3%	-0.3%	0.2%	0.6%
2	1	6	0.0%	29.7%	0.0%	31.9%	0.9%	28.6%	1.6%	22.6%	2.9%	18.0%	2.2%	13.2%	0.6%	6.1%
2	1	7	1.4%	6.1%	0.7%	10.1%	-0.5%	8.2%	0.1%	10.3%	0.0%	8.9%	0.0%	4.7%	-1.2%	1.6%
2	1	8	0.9%	4.8%	5.2%	11.9%	6.0%	18.8%	3.1%	14.9%	1.9%	12.6%	0.1%	11.8%	0.0%	6.9%
2	1	9	2.9%	7.8%	3.8%	13.1%	3.5%	15.2%	5.0%	13.8%	0.5%	9.5%	1.9%	3.9%	0.6%	0.6%
2	2	1	0.8%	9.2%	1.8%	14.9%	4.6%	18.8%	11.3%	24.6%	10.0%	24.5%	7.7%	22.1%	3.3%	15.5%
2	2	2	1.1%	2.8%	2.2%	4.5%	-0.1%	3.1%	0.9%	4.7%	1.8%	5.5%	4.7%	7.4%	5.4%	5.9%
2	2	3	0.2%	2.1%	0.1%	0.3%	0.0%	1.4%	1.4%	2.3%	0.9%	4.1%	0.7%	5.0%	-0.1%	1.0%
2	2	4	0.6%	1.9%	2.1%	2.8%	2.0%	3.5%	3.2%	4.7%	0.8%	2.1%	2.8%	3.4%	-0.4%	0.1%
2	2	5	-0.8%	2.6%	1.0%	3.6%	1.0%	4.6%	0.4%	4.8%	1.4%	5.7%	2.0%	4.1%	-0.2%	1.3%
2	2	6	1.1%	2.9%	2.9%	8.2%	1.8%	8.2%	2.2%	9.7%	0.8%	9.0%	0.9%	9.3%	0.3%	5.5%
2	2	7	3.1%	2.9%	2.7%	3.4%	6.6%	6.3%	3.2%	6.6%	0.5%	5.0%	1.5%	4.4%	2.8%	2.7%
2	2	8	4.2%	8.0%	6.5%	11.2%	5.8%	14.5%	3.2%	14.7%	2.1%	11.6%	2.8%	9.7%	2.7%	3.3%
2	2	9	1.3%	0.9%	1.6%	2.4%	1.3%	1.3%	0.9%	0.9%	0.9%	2.7%	0.5%	4.8%	0.1%	0.4%
2	3	1	-0.5%	2.5%	0.6%	2.9%	1.4%	2.5%	1.1%	1.1%	0.4%	0.7%	-0.4%	-0.4%	0.5%	0.5%
2	3	2	0.5%	3.1%	4.8%	5.6%	2.0%	3.7%	3.1%	5.8%	2.0%	3.9%	0.2%	3.9%	2.1%	2.8%
2	3	3	-0.1%	-0.1%	-0.1%	3.0%	1.0%	2.4%	0.0%	3.4%	-0.1%	3.9%	-0.1%	4.8%	0.0%	3.1%
2	3	4	0.4%	5.2%	1.5%	8.7%	1.9%	7.2%	0.8%	7.5%	2.2%	8.1%	1.4%	3.9%	2.7%	2.4%
2	3	5	-0.1%	0.3%	0.4%	0.8%	-0.4%	1.4%	1.4%	3.9%	0.3%	4.2%	0.8%	3.0%	-0.1%	1.0%
2	3	6	0.3%	3.1%	0.5%	5.9%	0.7%	6.3%	3.6%	6.1%	3.4%	6.1%	2.5%	7.4%	-0.3%	4.5%

Table 3.7. (continued)

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			ΔIH	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*
2	3	7	-0.1%	0.7%	3.5%	3.5%	2.7%	2.7%	0.9%	2.8%	2.5%	3.7%	0.7%	2.7%	0.7%	0.9%
2	3	8	1.7%	6.4%	2.9%	9.0%	3.2%	11.8%	1.9%	10.9%	2.3%	12.6%	1.1%	10.0%	0.3%	4.3%
2	3	9	0.1%	2.0%	0.8%	2.8%	-0.2%	2.5%	0.1%	3.9%	-0.1%	4.9%	0.4%	4.8%	0.0%	2.7%
3	1	1	1.2%	2.7%	1.0%	3.9%	0.7%	10.7%	1.7%	8.8%	1.4%	5.1%	-0.4%	0.2%	0.2%	0.2%
3	1	2	4.5%	3.5%	7.8%	7.8%	6.9%	7.9%	6.7%	7.3%	8.1%	8.8%	3.5%	0.5%	3.0%	0.0%
3	1	3	0.3%	14.8%	-0.5%	24.5%	2.8%	31.5%	-0.3%	27.6%	1.4%	19.9%	2.0%	13.0%	-0.3%	-0.4%
3	1	4	5.4%	6.1%	3.5%	9.4%	5.7%	9.3%	7.9%	9.1%	9.1%	10.6%	8.5%	9.1%	2.8%	3.9%
3	1	5	-3.4%	-3.4%	3.8%	5.3%	8.0%	9.4%	6.1%	7.8%	6.5%	9.9%	2.9%	3.2%	4.1%	5.3%
3	1	6	0.4%	22.6%	2.0%	21.0%	4.6%	25.8%	3.2%	24.1%	11.8%	25.3%	13.2%	24.1%	10.5%	16.6%
3	1	7	4.0%	8.3%	10.0%	10.3%	6.8%	8.5%	9.6%	8.6%	6.0%	4.8%	4.7%	4.6%	0.7%	0.7%
3	1	8	1.3%	10.3%	6.8%	17.9%	8.2%	18.9%	5.0%	15.8%	9.1%	13.6%	10.4%	15.1%	3.0%	8.6%
3	1	9	2.7%	4.3%	2.2%	7.5%	0.0%	7.5%	0.7%	8.4%	4.9%	12.3%	6.2%	18.3%	5.8%	9.9%
3	2	1	1.1%	2.1%	2.8%	9.0%	0.4%	16.2%	-0.7%	21.7%	1.3%	21.1%	1.3%	16.0%	1.0%	3.7%
3	2	2	0.7%	0.5%	1.7%	-0.8%	0.1%	-0.7%	2.9%	4.6%	4.9%	4.6%	2.0%	2.2%	5.5%	5.0%
3	2	3	2.0%	7.8%	0.8%	12.1%	2.1%	13.4%	2.0%	15.8%	4.9%	14.8%	6.0%	11.8%	4.3%	8.4%
3	2	4	3.5%	4.0%	4.1%	5.5%	1.2%	2.8%	3.0%	5.4%	5.3%	3.9%	-0.9%	-0.6%	-1.5%	-1.8%
3	2	5	-1.4%	0.1%	0.0%	4.0%	-3.2%	-0.1%	1.2%	3.9%	2.5%	4.9%	-0.5%	-0.7%	2.3%	2.3%
3	2	6	2.4%	7.0%	3.1%	7.3%	5.1%	7.8%	7.8%	11.2%	8.0%	12.0%	2.3%	7.3%	0.6%	2.7%
3	2	7	1.5%	1.8%	1.1%	1.6%	2.2%	2.5%	3.2%	3.4%	2.1%	3.8%	0.4%	1.6%	2.0%	3.8%
3	2	8	0.3%	-0.4%	1.3%	0.8%	2.7%	1.5%	1.2%	0.6%	1.3%	2.2%	2.9%	2.5%	0.4%	-0.3%
3	2	9	1.7%	12.1%	2.8%	14.0%	3.0%	16.8%	1.8%	18.2%	5.2%	19.2%	-1.5%	12.9%	-0.3%	7.7%
3	3	1	0.2%	1.4%	3.6%	3.5%	-0.2%	4.1%	0.0%	3.4%	1.4%	5.8%	0.9%	4.7%	0.9%	2.1%
3	3	2	3.7%	3.7%	0.8%	-0.1%	0.5%	0.5%	0.7%	0.2%	-0.9%	-1.0%	-0.8%	-1.6%	2.1%	0.3%
3	3	3	1.9%	7.2%	-1.3%	4.9%	0.1%	5.7%	3.1%	7.5%	-0.4%	7.0%	-0.8%	6.3%	2.9%	6.1%
3	3	4	1.3%	0.9%	0.5%	0.5%	-0.2%	-0.4%	0.2%	0.0%	-1.0%	-1.2%	0.8%	0.2%	-1.8%	-2.3%
3	3	5	2.2%	1.4%	3.2%	2.8%	1.2%	1.4%	-0.4%	0.4%	2.5%	2.4%	3.7%	2.4%	2.9%	3.6%
3	3	6	1.4%	-0.1%	0.0%	0.0%	2.0%	5.2%	3.9%	10.7%	2.7%	9.0%	1.2%	8.0%	1.8%	5.5%
3	3	7	1.2%	4.0%	3.0%	3.2%	0.5%	1.3%	2.4%	3.1%	1.2%	0.9%	-1.2%	-1.5%	0.8%	0.3%
3	3	8	0.6%	0.1%	-0.5%	-0.9%	1.4%	1.7%	1.8%	4.4%	-0.8%	3.4%	1.9%	6.9%	2.3%	5.9%
3	3	9	0.9%	3.4%	1.4%	3.7%	2.4%	6.0%	2.2%	7.6%	0.8%	10.7%	0.4%	8.8%	1.6%	6.9%
average			***1.2%***	***4.5%***	***1.9%***	***6.4%***	***2.0%***	***7.2%***	***2.2%***	***7.9%***	***2.1%***	***7.5%***	***1.7%***	***5.8%***	***1.3%***	***3.2%***
		min	*-3.4%*	*-3.4%*	*-1.3%*	*-0.9%*	*-3.2%*	*-0.7%*	*-0.7%*	*-0.1%*	*-1.0%*	*-1.2%*	*-1.5%*	*-1.6%*	*-1.8%*	*-2.3%*
		max	*6.8%*	*29.7%*	*10.0%*	*31.9%*	*9.4%*	*31.5%*	*11.3%*	*27.6%*	*11.8%*	*25.3%*	*13.2%*	*24.1%*	*10.5%*	*16.6%*

Table 3.8. GAs percentage deviation analysis for 6-machine problems.

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			ΔIH	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*
1	1	1	0.0%	7.8%	0.0%	20.0%	0.0%	22.3%	0.0%	21.4%	0.0%	20.2%	0.0%	18.6%	0.0%	0.0%
1	1	2	0.0%	0.1%	2.6%	3.0%	2.5%	3.2%	1.9%	7.5%	5.4%	9.8%	3.4%	8.7%	1.6%	7.2%
1	1	3	0.0%	30.8%	0.0%	40.5%	0.0%	40.8%	0.0%	39.1%	0.0%	36.2%	0.0%	31.2%	0.0%	18.9%
1	2	1	0.0%	7.1%	0.0%	18.6%	0.0%	27.2%	0.0%	40.2%	0.0%	38.7%	0.0%	35.5%	0.0%	30.0%
1	2	2	-0.6%	-0.6%	0.3%	1.2%	0.3%	1.3%	0.1%	1.1%	0.5%	0.9%	2.9%	2.9%	-0.2%	1.5%
1	2	3	0.0%	9.9%	0.0%	18.4%	0.0%	18.7%	0.0%	17.0%	0.0%	17.6%	0.0%	17.7%	0.0%	10.8%
1	3	1	0.0%	0.0%	0.5%	0.5%	0.5%	0.7%	0.3%	1.8%	0.0%	4.9%	0.0%	10.0%	0.0%	9.7%
1	3	2	1.6%	2.0%	1.2%	3.4%	0.2%	4.4%	1.4%	6.0%	-0.3%	5.1%	0.4%	4.1%	2.5%	4.9%

Table 3.8. (continued)

Level of g	Level of n	Level of a	$\overline{NWA}$ = 87.5		75		62.5		50		37.5		25		12.5	
			ΔIH	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*	*ΔIH*	*ΔIR*
1	3	3	0.0%	3.4%	-0.1%	17.7%	0.0%	27.3%	-0.1%	34.3%	0.0%	34.0%	0.1%	32.2%	0.2%	16.6%
2	1	1	0.0%	11.7%	0.0%	15.6%	0.0%	23.3%	0.0%	23.2%	0.0%	19.7%	0.0%	15.4%	0.0%	8.7%
2	1	2	0.0%	2.4%	1.0%	5.7%	0.8%	9.2%	-0.8%	8.9%	0.4%	8.9%	5.8%	9.3%	0.7%	5.6%
2	1	3	0.2%	6.9%	0.1%	19.4%	0.2%	26.1%	-0.1%	18.5%	-0.1%	16.8%	-0.2%	16.8%	0.1%	17.5%
2	2	1	0.1%	7.2%	0.0%	13.5%	0.1%	15.7%	-0.4%	12.1%	-0.1%	12.5%	-0.1%	12.8%	0.2%	6.3%
2	2	2	5.3%	5.2%	3.9%	9.3%	0.3%	7.9%	2.5%	8.6%	1.8%	8.3%	-0.6%	3.5%	-1.6%	1.8%
2	2	3	0.4%	25.6%	0.1%	37.6%	-0.2%	44.1%	0.0%	44.7%	-0.1%	43.1%	0.3%	32.4%	0.2%	13.1%
2	3	1	0.0%	0.7%	-0.1%	9.7%	-0.4%	9.1%	0.0%	9.2%	0.1%	10.5%	0.1%	12.3%	0.0%	5.3%
2	3	2	0.4%	1.5%	0.1%	2.1%	1.8%	3.4%	0.6%	3.0%	0.3%	2.9%	1.0%	3.0%	2.8%	3.2%
2	3	3	0.9%	14.9%	0.0%	19.1%	0.5%	23.1%	-0.1%	23.6%	0.8%	22.8%	0.4%	23.1%	-0.1%	18.5%
3	1	1	0.1%	12.4%	0.3%	35.0%	0.3%	46.2%	0.4%	53.3%	0.2%	47.2%	-0.2%	38.8%	0.4%	23.2%
3	1	2	2.3%	3.4%	1.6%	4.6%	0.2%	5.2%	1.1%	8.5%	2.7%	6.3%	0.8%	3.8%	1.7%	2.3%
3	1	3	-0.7%	-0.7%	0.1%	6.7%	0.1%	10.9%	-0.3%	18.2%	0.2%	16.7%	0.1%	10.8%	0.5%	7.6%
3	2	1	0.1%	9.1%	-0.1%	23.7%	0.2%	31.5%	-0.2%	36.9%	-0.4%	38.6%	0.0%	31.1%	0.0%	16.8%
3	2	2	1.8%	2.8%	2.0%	4.3%	3.8%	6.4%	3.5%	8.5%	0.1%	5.2%	0.4%	6.6%	0.2%	2.5%
3	2	3	-0.1%	-0.2%	0.4%	3.2%	-0.1%	0.5%	0.7%	0.3%	0.2%	-0.3%	-0.2%	-0.3%	0.1%	-0.3%
3	3	1	0.1%	5.6%	-0.2%	11.0%	0.0%	14.5%	-0.2%	18.9%	0.1%	24.5%	0.1%	21.7%	0.0%	15.1%
3	3	2	1.1%	1.2%	3.2%	3.9%	1.5%	2.9%	1.7%	1.4%	3.3%	4.7%	2.6%	4.4%	1.4%	2.8%
3	3	3	-0.3%	1.9%	0.2%	3.2%	0.3%	3.2%	0.3%	4.4%	0.1%	6.6%	0.2%	10.8%	0.1%	16.0%
average			***0.5%***	***6.4%***	***0.6%***	***13.0%***	***0.5%***	***15.9%***	***0.5%***	***17.4%***	***0.6%***	***17.1%***	***0.6%***	***15.5%***	***0.4%***	***9.8%***
			-0.7%	*-0.7%*	*-0.2%*	*0.5%*	*-0.4%*	*0.5%*	*-0.8%*	*0.3%*	*-0.4%*	*-0.3%*	*-0.6%*	*-0.3%*	*-1.6%*	*-0.3%*
			5.3%	*30.8%*	*3.9%*	*40.5%*	*3.8%*	*46.2%*	*3.5%*	*53.3%*	*5.4%*	*47.2%*	*5.8%*	*38.8%*	*2.8%*	*30.0%*

As the reader can notice GAI on the average outperforms the two alternative metaheuristics regardless of the number of machines. In terms of average results, as concerns the comparison between GAI and GAH (ΔIH) the minimum value is equal to 0.4% (6-machine problems with $\overline{NWA}$ = 12.5) while the biggest value is 2.2% for problems with 3 machines and $\overline{NWA}$ = 50. As regards ΔIR, it puts in evidence a larger deviation as the smallest value is equal to 2.0% (2-machine problems with $\overline{NWA}$ = 87.5) while the largest difference (17.4%) arises from the scenario problems with 6 machines and $\overline{NWA}$ = 50. Numerical results also reveal as GAH outperforms GAR as well, since the average ΔIH is always lower than the corresponding ΔIR. Moreover, ΔIR reaches a maximum punctual value equal to 53.3% (6-machine problems with $\overline{NWA}$ = 50, Level 3 of g, Level 1 of n, Level 1 of a), while ΔIH is never higher than 13.2% (3-machine problems with $\overline{NWA}$ = 25, Level 3 of g, Level 1 of n, Level 6 of a). Further remarks may regard the way ΔIR deviations reported in Tables 3.7, 3.8 and 3.9 vary from a lower to a higher $\overline{NWA}$ value; it may be noticed that a low skill diversification on the workforce team (which coincided with $\overline{NWA}$ = 87.5 and $\overline{NWA}$ = 12.5) reduces the performance gap between

GAI and GAR, regardless of the number of machines. As for example, ΔIR achieves an average value equal to 6.4% and 9.8% in case $\overline{NWA}$ = 87.5 and $\overline{NWA}$ = 12.5, respectively, for 6-machine problems. On the other hand, if $\overline{NWA}$ assumes values from 25 to 75, the average deviation varies from 13% to 17.4%. A similar trend can be noticed for problems with 2 and 3 machines, as well. The reason of such an outcome could be justified by the way the different skills of a given workforce affect the space of solutions. In fact, whether the workforce team is characterized by a roughly uniform skill level (most workers have high skills, most worker have low skills) a random assignation of worker to machines, as performed by GAR, may just yield to slightly worse performances. On the contrary, whenever a workforce team is typified by workers with so different skills, exploring the space of solution with reference to worker allocation policies in addition to the sequence of jobs and groups, becomes crucial for effectively solving the problem in hand.

Comparison with HPSO algorithm

Since GAI proved its superiority in tackling the FSDGS-SWA problem in hand, a multiple comparative analysis between such procedure and the HPSO procedure proposed by Hajinejad, Salmasi and Mokhtari (2011) has been performed. HPSO algorithm employs a real-number encoding scheme able to separately manage the sequence of groups and the sequences of jobs within each group. Moreover, it integrates the standard particle swarm optimization strategy with an Individual Enhancement (IE) neighbourhood search technique, aimed at improving performances in terms of both exploration and exploitation. In order to adapt the HPSO algorithm to the combined FSDGS-SWA problem in hand, a further substring of real numbers allocating workers to machines has been added to the encoding scheme proposed by the authors. HPSO has been coded on the same computational platform used for GAI, and the same stopping criterion, i.e., 30 s of CPU time, has been adopted.

A first comparison between GAI and HPSO has been executed through the assessment of their performances against the optimal results given by the MILP model reported in Section 3.2. To this aim, a set of experimental data has been generated by considering 90 out of 135 instances belonging to the FSDGS benchmark, i.e., only those with Level 1 or Level 2 for the number of groups. In fact, according to what stated by Naderi and Salmasi (2012), the proposed MILP model can be used for solving problems with 10 groups at most. For each instance, two test

problems have been randomly generated. Moreover, 3 different levels of $\overline{NWA}$, i.e., 12.5, 50 and 87.5, respectively, have been considered for each problem. Therefore, a total of 90 × 3 × 2 = 540 experimental test cases have been taken into account for such analysis. The MILP model has been implemented on an IBM® ILOG CPLEX 12.0 64 bit platform installed within a workstation powered by two quad-core 2.39 GHz processors with 24 GB RAM. All considered problems have been optimally solved by the mathematical model within a computational time of 180 s. Performances of both metaheuristics have been measured by means of the average percentage deviations ΔGAI and $\Delta HPSO$ reported by GAI and HPSO, respectively, vs. the global optimum:

$$\Delta GAI = \frac{\overline{C_{\max}}(\text{GAI}) - \overline{C_{\max}}(\text{MILP})}{\overline{C_{\max}}(\text{MILP})} \times 100 \tag{3.20}$$

$$\Delta HPSO = \frac{\overline{C_{\max}}(\text{HPSO}) - \overline{C_{\max}}(\text{MILP})}{\overline{C_{\max}}(\text{MILP})} \times 100 \tag{3.21}$$

where $\overline{C_{\max}}(\text{GAI})$, $\overline{C_{\max}}(\text{HPSO})$ and $\overline{C_{\max}}(\text{MILP})$ refers to the average completion time yielded by GAI HPSO, and the mathematical model, respectively, over the 2 different test problems randomly generated for each instance of the considered benchmark. Also, the number of times each algorithm reached the global optimum given by the MILP model, denoted as *opt_GAI* and *opt_HPSO*, has been computed. Results obtained are shown in Tables 3.9, 3.10 and 3.11 for problems with 2, 3 and 6 machines, respectively.

Table 3.9. Comparison of metaheuristics with MILP model for problems with 2 machines.

Level of g	Level of n	Level of a	$\overline{NWA}$ = 87.5				$\overline{NWA}$ = 50				$\overline{NWA}$ = 12.5			
			ΔGAI	$\Delta HPSO$	*opt_GAI*	*opt_HPSO*	ΔGAI	$\Delta HPSO$	*opt_GAI*	*opt_HPSO*	ΔGAI	$\Delta HPSO$	*opt_GAI*	*opt_HPSO*
1	1	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	3	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	1	0.00%	0.25%	2	1	0.00%	0.18%	2	1	0.00%	0.00%	2	2
1	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	3	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	3	1	0.00%	0.00%	2	2	0.09%	0.00%	1	2	0.00%	0.00%	2	2
1	3	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	3	3	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
2	1	1	0.00%	0.00%	2	2	0.00%	0.14%	2	1	0.31%	0.31%	1	1
2	1	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
2	1	3	0.11%	0.11%	1	1	0.00%	0.20%	2	1	0.00%	0.00%	2	2

Table 3.9. (continued)

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5				50				12.5			
			ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO	ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO	ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO
2	2	1	1.05%	1.05%	1	1	0.21%	0.17%	1	1	0.96%	0.66%	1	1
2	2	2	0.00%	0.16%	2	1	0.96%	0.30%	0	0	0.18%	0.18%	1	1
2	2	3	1.24%	0.57%	0	1	0.74%	0.55%	0	0	0.27%	1.32%	1	0
2	3	1	0.00%	0.73%	2	1	0.24%	0.19%	1	1	0.25%	0.47%	1	1
2	3	2	0.00%	0.07%	2	1	0.13%	0.21%	1	0	0.00%	0.00%	2	2
2	3	3	0.44%	0.39%	1	1	0.00%	0.87%	1	0	0.18%	0.51%	1	0
*average/***TOT**			*0.16%*	*0.18%*	**31**	**28**	*0.13%*	*0.16%*	**27**	**23**	*0.12%*	*0.19%*	**30**	**28**

Table 3.10. Comparison of metaheuristics with MILP model for problems with 3 machines.

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5				50				12.5			
			ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO	ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO	ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO
1	1	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	2	0.00%	2.08%	2	1	0.00%	0.99%	2	1	0.00%	0.00%	2	2
1	1	3	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	4	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	5	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	6	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	7	0.00%	0.43%	2	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	8	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	9	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	3	0.00%	0.77%	2	1	0.02%	0.02%	1	1	0.00%	0.82%	2	1
1	2	4	0.00%	0.00%	2	2	0.09%	0.13%	1	1	0.00%	0.00%	2	2
1	2	5	0.00%	0.13%	2	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	6	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	7	0.24%	0.24%	1	1	0.09%	0.29%	1	1	0.00%	0.00%	2	2
1	2	8	0.00%	0.10%	2	1	0.00%	0.18%	2	1	0.00%	0.00%	2	2
1	2	9	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.25%	2	1
1	3	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	3	2	0.10%	0.10%	1	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	3	3	0.00%	0.22%	2	1	0.15%	0.28%	1	0	0.12%	0.12%	1	1
1	3	4	0.16%	0.00%	1	2	0.00%	0.00%	2	2	0.89%	0.89%	1	1
1	3	5	0.13%	0.56%	1	0	0.58%	1.33%	0	0	0.00%	0.00%	2	2
1	3	6	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	3	7	0.00%	0.00%	2	2	0.00%	0.02%	2	1	0.35%	0.36%	1	1
1	3	8	0.51%	0.52%	0	0	0.00%	0.76%	2	1	0.10%	0.02%	1	1
1	3	9	0.00%	0.47%	2	0	0.38%	0.01%	1	1	0.00%	0.06%	2	1
2	1	1	0.59%	0.00%	1	2	1.20%	1.36%	1	1	0.22%	0.29%	1	1
2	1	2	0.00%	0.00%	2	2	0.00%	0.62%	2	1	0.00%	0.71%	2	1
2	1	3	0.12%	0.74%	1	1	0.73%	0.25%	1	1	0.12%	0.00%	1	2
2	1	4	0.00%	0.98%	2	1	0.82%	0.00%	1	2	0.21%	0.71%	1	1
2	1	5	0.00%	0.00%	2	2	0.00%	0.04%	2	1	0.00%	0.69%	2	1
2	1	6	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.38%	2	1
2	1	7	1.06%	0.00%	1	2	0.00%	0.16%	2	1	1.22%	1.46%	1	1
2	1	8	0.85%	0.62%	1	1	0.05%	0.38%	1	1	0.37%	0.34%	1	1
2	1	9	0.00%	0.00%	2	2	0.00%	0.08%	2	1	0.00%	0.16%	2	1
2	2	1	0.27%	0.16%	1	1	0.00%	0.00%	2	2	0.12%	0.52%	1	1
2	2	2	0.48%	1.73%	1	0	0.75%	1.05%	0	0	1.08%	1.08%	0	0
2	2	3	0.14%	0.51%	1	0	0.06%	0.00%	1	2	0.27%	0.61%	1	0
2	2	4	0.25%	0.25%	1	1	0.05%	1.78%	1	0	1.00%	0.09%	0	1
2	2	5	1.01%	0.00%	1	2	1.14%	0.74%	0	0	1.43%	0.69%	0	0
2	2	6	0.00%	0.07%	2	1	0.00%	0.74%	2	1	0.79%	0.79%	1	1
2	2	7	0.73%	0.65%	1	0	0.15%	1.41%	0	0	0.00%	0.20%	2	1

Table 3.10. (continued)

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5				$\overline{NWA}$ 50				$\overline{NWA}$ 12.5			
			ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO	ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO	ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO
2	2	8	0.27%	0.80%	0	0	1.46%	0.84%	1	0	0.58%	1.10%	1	0
2	2	9	0.71%	0.47%	0	0	1.42%	0.79%	1	1	0.44%	0.92%	1	0
2	3	1	1.40%	0.64%	1	1	0.05%	0.37%	1	1	0.29%	0.65%	1	1
2	3	2	1.60%	1.24%	0	0	0.85%	1.31%	0	0	1.34%	1.41%	0	0
2	3	3	0.14%	0.18%	1	0	0.00%	0.38%	2	0	0.12%	0.34%	1	1
2	3	4	1.15%	0.96%	0	0	0.62%	0.43%	0	0	0.54%	0.32%	0	1
2	3	5	0.66%	0.43%	0	0	0.52%	1.25%	0	0	1.09%	0.79%	0	1
2	3	6	0.41%	0.65%	0	1	0.24%	0.35%	1	0	1.01%	0.93%	1	0
2	3	7	0.85%	0.46%	1	1	1.14%	0.50%	0	1	0.49%	1.15%	0	0
2	3	8	0.01%	0.82%	1	0	0.64%	0.14%	1	1	0.28%	0.00%	1	2
2	3	9	0.09%	0.00%	1	2	0.22%	0.38%	1	1	0.00%	0.04%	2	1
average/**TOT**			*0.26%*	*0.33%*	**75**	**65**	*0.25%*	*0.36%*	**75**	**63**	*0.27%*	*0.35%*	**76**	**68**

Table 11. Comparison of metaheuristics with MILP model for problems with 6 machines.

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5				$\overline{NWA}$ 50				$\overline{NWA}$ 12.5			
			ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO	ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO	ΔGAI	$\Delta HPSO$	opt_GAI	opt_HPSO
1	1	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	1	2	0.00%	0.06%	2	1	0.11%	0.16%	1	1	0.00%	0.66%	2	0
1	1	3	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	2	2	1.20%	1.39%	1	0	0.27%	1.86%	0	1	1.48%	2.61%	0	0
1	2	3	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	3	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
1	3	2	0.87%	1.48%	1	1	1.01%	2.68%	1	0	0.21%	1.80%	1	1
1	3	3	0.13%	0.06%	0	1	0.25%	0.04%	0	1	0.02%	0.06%	1	1
2	1	1	0.00%	0.00%	2	2	0.00%	0.00%	2	2	0.00%	0.00%	2	2
2	1	2	0.32%	1.20%	1	1	1.39%	2.24%	1	0	0.96%	1.00%	0	0
2	1	3	0.00%	0.22%	2	0	0.23%	0.13%	0	0	0.00%	0.16%	2	0
2	2	1	0.00%	0.05%	2	1	0.34%	0.04%	1	1	0.00%	0.17%	2	1
2	2	2	1.95%	3.71%	0	0	2.46%	3.40%	0	0	4.35%	5.47%	0	0
2	2	3	0.14%	0.16%	0	0	0.07%	0.59%	0	0	0.07%	0.11%	1	0
2	3	1	0.13%	0.16%	1	1	0.12%	0.01%	1	1	0.04%	0.04%	1	1
2	3	2	1.75%	3.18%	0	0	2.22%	3.85%	0	0	2.12%	2.71%	0	0
2	3	3	0.27%	0.17%	0	0	0.46%	0.53%	1	0	0.53%	0.76%	0	0
average/**TOT**			*0.38%*	*0.66%*	**22**	**18**	*0.50%*	*0.86%*	**18**	**17**	*0.54%*	*0.86%*	**22**	**16**

It can be noticed that GAI shows a higher effectiveness in approaching the global optimum compared to HPSO, since the average gap from the MILP solution obtained through GAI is always lower or equal than the corresponding average value of $\Delta HPSO$. Moreover, the proposed genetic algorithm reaches the global optimum in a higher number of cases than the particle swarm procedure.

In addition to the assessment of GAI and HPSO performances against the global optima provided by the MILP model, a more comprehensive comparison between the two procedures has been fulfilled by taking into account the same benchmark adopted in the GAs evaluation Section, composed by 1,890 test cases. In this case the Relative Percentage Deviation (RPD) from the best metaheuristic solution has been calculated with respect to each problem for both GAI and HPSO, according to the following formula:

$$\mathrm{RPD}_{\mathrm{GAI}} = \frac{C_{\max}(\mathrm{GAI}) - BestC_{\max}}{BestC_{\max}} \times 100 \tag{3.22}$$

$$\mathrm{RPD}_{\mathrm{HPSO}} = \frac{C_{\max}(\mathrm{HPSO}) - BestC_{\max}}{BestC_{\max}} \times 100 \tag{3.23}$$

where $C_{\max}(\mathrm{GAI})$ and $C_{\max}(\mathrm{HPSO})$ are the makespan values obtained by the two metaheuristics with reference to each test problem and $BestC_{\max}$ is the lowest among them, i.e., the best metaheuristic solution available for a given test case. Tables 3.12, 3.13 and 3.14 illustrate the average results obtained. for 2-, 3- and 6-machine problems, respectively. Each main row reports the average RPD calculated over the two randomly generated problems pertaining to a given instance, while bold numbers represent the grand average RPDs for a given value of $\overline{NWA}$.

Table 3.12. Comparison between GAI and HPSO for problems with 2 machines.

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			$\mathrm{RPD}_{\mathrm{GAI}}$	$\mathrm{RPD}_{\mathrm{HPSO}}$	$\mathrm{RPD}_{\mathrm{GAI}}$	$\mathrm{RPD}_{\mathrm{HPSO}}$	$\mathrm{RPD}_{\mathrm{GAI}}$	$\mathrm{RPD}_{\mathrm{HPSO}}$	$\mathrm{RPD}_{\mathrm{GAI}}$	$\mathrm{RPD}_{\mathrm{HPSO}}$	$\mathrm{RPD}_{\mathrm{GAI}}$	$\mathrm{RPD}_{\mathrm{HPSO}}$	$\mathrm{RPD}_{\mathrm{GAI}}$	$\mathrm{RPD}_{\mathrm{HPSO}}$	$\mathrm{RPD}_{\mathrm{GAI}}$	$\mathrm{RPD}_{\mathrm{HPSO}}$
1	1	1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	3	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	2	1	0.00%	0.25%	0.00%	0.00%	0.00%	0.00%	0.00%	0.18%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	2	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	2	3	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	3	1	0.00%	0.00%	0.00%	0.08%	0.03%	0.00%	0.10%	0.00%	0.00%	0.08%	0.00%	0.00%	0.00%	0.00%
1	3	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	3	3	0.00%	0.00%	0.00%	0.00%	0.00%	0.03%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
2	1	1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.14%	0.00%	0.00%	0.00%	0.23%	0.00%	0.00%
2	1	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.93%	0.00%	0.00%	0.00%	0.00%	0.28%	0.00%	0.00%	0.00%
2	1	3	0.00%	0.00%	0.08%	0.00%	0.00%	0.00%	0.00%	0.20%	0.28%	0.19%	0.00%	0.00%	0.00%	0.00%
2	2	1	0.00%	0.00%	0.00%	1.07%	0.00%	0.58%	0.03%	0.00%	0.00%	0.63%	0.23%	0.00%	0.29%	0.00%
2	2	2	0.00%	0.16%	0.20%	0.07%	0.50%	0.25%	0.66%	0.00%	0.91%	0.42%	0.00%	0.43%	0.00%	0.00%
2	2	3	0.79%	0.12%	0.14%	0.31%	0.00%	0.59%	0.22%	0.03%	0.00%	1.70%	0.00%	1.36%	0.00%	1.05%
2	3	1	0.00%	0.72%	0.00%	0.00%	0.31%	0.00%	0.06%	0.00%	0.00%	0.00%	0.00%	0.80%	0.25%	0.48%
2	3	2	0.00%	0.07%	0.00%	0.07%	0.00%	0.03%	0.00%	0.09%	0.00%	0.06%	0.00%	0.03%	0.00%	0.00%
2	3	3	0.05%	0.00%	0.30%	0.56%	0.00%	0.66%	0.00%	0.86%	0.44%	0.13%	0.00%	0.55%	0.00%	0.34%
3	1	1	0.00%	3.80%	0.71%	2.82%	0.00%	2.51%	0.00%	4.84%	0.63%	0.72%	0.00%	2.69%	0.00%	3.51%

Table 3.12. (continued)

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}
3	1	2	0.00%	2.83%	0.00%	3.90%	0.00%	6.19%	0.00%	3.82%	0.00%	1.36%	0.00%	5.49%	1.64%	3.16%
3	1	3	0.09%	2.72%	1.04%	0.43%	1.68%	0.00%	0.65%	1.12%	0.00%	3.19%	0.66%	4.06%	0.10%	2.35%
3	2	1	0.17%	1.98%	0.00%	3.27%	0.00%	2.35%	0.25%	1.73%	0.12%	3.49%	0.00%	2.53%	0.00%	4.88%
3	2	2	0.00%	4.32%	0.00%	4.54%	0.00%	3.35%	0.00%	3.31%	0.00%	5.05%	0.00%	1.39%	0.00%	3.27%
3	2	3	0.45%	0.71%	0.21%	0.50%	0.54%	0.86%	0.09%	0.34%	1.02%	1.07%	0.27%	0.31%	0.41%	1.03%
3	3	1	0.00%	1.51%	0.20%	0.00%	0.61%	1.08%	0.20%	0.94%	0.32%	0.48%	0.00%	0.81%	1.40%	2.41%
3	3	2	0.00%	1.82%	0.00%	2.58%	0.00%	1.80%	0.00%	3.82%	0.00%	1.76%	0.00%	1.24%	0.00%	3.86%
3	3	3	0.00%	3.13%	0.00%	2.11%	0.00%	1.76%	0.00%	2.97%	0.00%	3.03%	0.00%	2.15%	0.00%	5.04%
grand average			***0.06%***	***0.89%***	***0.11%***	***0.83%***	***0.14%***	***0.85%***	***0.08%***	***0.90%***	***0.14%***	***0.86%***	***0.05%***	***0.89%***	***0.15%***	***1.16%***

Table 3.13. Comparison between GAI and HPSO for problems with 3 machines.

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}
1	1	1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	2	0.00%	2.08%	0.00%	0.00%	0.00%	0.00%	0.00%	0.99%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	3	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	4	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	5	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	6	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	7	0.00%	0.43%	0.00%	0.00%	0.00%	0.29%	0.00%	0.00%	0.00%	0.57%	0.00%	0.00%	0.00%	0.00%
1	1	8	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1	9	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	2	1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	2	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	2	3	0.00%	0.77%	0.00%	0.52%	0.00%	0.06%	0.00%	0.00%	0.00%	0.03%	0.00%	0.00%	0.00%	0.82%
1	2	4	0.00%	0.00%	0.00%	0.00%	0.00%	0.09%	0.00%	0.03%	0.00%	0.22%	0.00%	0.00%	0.00%	0.00%
1	2	5	0.00%	0.13%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.39%	0.00%	0.00%
1	2	6	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	2	7	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.19%	0.00%	0.02%	0.00%	0.72%	0.00%	0.00%
1	2	8	0.00%	0.11%	0.00%	0.00%	0.00%	0.00%	0.00%	0.18%	0.00%	0.00%	0.00%	0.40%	0.00%	0.00%
1	2	9	0.00%	0.00%	0.00%	0.11%	0.00%	0.40%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.25%
1	3	1	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	3	2	0.00%	0.00%	0.00%	0.00%	0.00%	0.07%	0.00%	0.00%	0.00%	0.00%	0.00%	0.55%	0.00%	0.00%
1	3	3	0.00%	0.22%	0.00%	0.52%	0.00%	0.10%	0.00%	0.12%	0.00%	0.55%	0.00%	0.18%	0.00%	0.00%
1	3	4	0.16%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.04%	0.00%	0.00%	0.00%	0.00%	0.00%
1	3	5	0.00%	0.42%	0.00%	0.25%	0.00%	0.16%	0.00%	0.75%	0.63%	0.02%	0.05%	0.00%	0.00%	0.00%
1	3	6	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	3	7	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%	0.00%	0.00%	0.00%	0.00%	0.00%	0.01%
1	3	8	0.00%	0.01%	0.00%	0.12%	0.00%	0.54%	0.00%	0.76%	0.00%	0.00%	0.00%	0.16%	0.08%	0.00%
1	3	9	0.00%	0.47%	0.00%	1.86%	0.00%	0.31%	0.38%	0.00%	0.00%	0.00%	0.00%	0.05%	0.00%	0.06%
2	1	1	0.58%	0.00%	0.00%	0.05%	1.33%	0.00%	0.00%	0.16%	0.96%	0.00%	0.50%	0.07%	0.00%	0.07%
2	1	2	0.00%	0.00%	0.29%	0.00%	1.56%	0.00%	0.00%	0.61%	0.14%	2.05%	0.00%	0.00%	0.00%	0.71%
2	1	3	0.00%	0.62%	0.00%	0.27%	0.23%	0.59%	0.47%	0.00%	0.90%	0.00%	0.00%	0.02%	0.12%	0.00%
2	1	4	0.00%	0.98%	0.00%	0.04%	0.45%	0.00%	0.81%	0.00%	0.00%	0.87%	0.00%	0.14%	0.00%	0.49%
2	1	5	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.04%	0.00%	0.00%	0.26%	0.00%	0.00%	0.69%

Table 3.13. (continued)

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}
2	1	6	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.38%
2	1	7	1.06%	0.00%	0.21%	0.59%	1.25%	0.63%	0.00%	0.16%	0.00%	0.00%	0.00%	0.00%	0.00%	0.23%
2	1	8	0.23%	0.00%	0.27%	0.00%	0.44%	0.00%	0.00%	0.32%	0.00%	0.31%	0.09%	0.00%	0.02%	0.00%
2	1	9	0.00%	0.00%	0.00%	0.03%	0.00%	0.23%	0.00%	0.08%	0.00%	0.00%	0.00%	0.00%	0.00%	0.16%
2	2	1	0.11%	0.00%	0.27%	0.00%	0.00%	0.00%	0.00%	0.00%	0.57%	0.00%	0.71%	0.00%	0.00%	0.41%
2	2	2	0.00%	1.25%	0.00%	1.28%	0.95%	0.00%	0.02%	0.33%	0.51%	0.00%	0.63%	0.35%	0.10%	0.10%
2	2	3	0.00%	0.36%	0.00%	0.80%	0.00%	0.10%	0.06%	0.00%	0.09%	0.00%	0.26%	0.06%	0.09%	0.43%
2	2	4	0.00%	0.00%	0.43%	0.64%	0.18%	0.78%	0.00%	1.73%	0.00%	0.79%	0.00%	0.25%	0.92%	0.01%
2	2	5	1.00%	0.00%	0.00%	0.03%	0.00%	0.38%	0.41%	0.00%	0.00%	1.43%	0.00%	0.88%	0.74%	0.00%
2	2	6	0.00%	0.07%	0.00%	0.55%	0.77%	0.00%	0.00%	0.74%	0.00%	0.62%	0.00%	0.00%	0.00%	0.00%
2	2	7	0.54%	0.45%	1.01%	0.00%	0.04%	0.74%	0.00%	1.27%	0.26%	0.14%	0.10%	0.21%	0.00%	0.20%
2	2	8	0.00%	0.53%	0.06%	0.57%	0.00%	1.31%	1.37%	0.75%	0.49%	0.32%	0.26%	0.00%	0.00%	0.51%
2	2	9	0.24%	0.00%	0.00%	0.61%	0.98%	0.00%	0.62%	0.00%	1.01%	0.00%	0.00%	0.26%	0.00%	0.48%
2	3	1	0.76%	0.00%	0.00%	0.40%	0.69%	0.00%	0.00%	0.32%	0.00%	0.07%	0.37%	0.00%	0.00%	0.36%
2	3	2	1.23%	0.88%	0.00%	1.79%	1.38%	0.00%	0.00%	0.45%	0.33%	0.00%	1.18%	0.20%	0.04%	0.11%
2	3	3	0.00%	0.05%	0.18%	0.23%	0.00%	0.44%	0.00%	0.38%	0.01%	0.26%	0.00%	0.00%	0.00%	0.22%
2	3	4	0.22%	0.05%	0.00%	1.00%	0.07%	0.34%	0.23%	0.04%	0.25%	0.55%	0.28%	0.17%	0.21%	0.00%
2	3	5	0.24%	0.00%	0.32%	0.86%	0.00%	1.07%	0.00%	0.73%	0.78%	1.73%	0.90%	0.17%	0.91%	0.61%
2	3	6	0.16%	0.41%	0.07%	0.56%	0.16%	0.12%	0.00%	0.12%	0.00%	0.51%	0.00%	0.53%	0.69%	0.62%
2	3	7	0.38%	0.00%	0.00%	0.28%	0.66%	0.06%	1.01%	0.37%	0.50%	0.19%	0.12%	0.06%	0.00%	0.66%
2	3	8	0.00%	0.81%	0.00%	0.05%	0.00%	0.76%	0.49%	0.00%	0.00%	0.18%	0.00%	0.74%	0.28%	0.00%
2	3	9	0.09%	0.00%	0.05%	0.50%	0.48%	0.00%	0.00%	0.15%	0.26%	0.00%	0.00%	0.37%	0.00%	0.04%
3	1	1	0.00%	3.38%	0.53%	1.86%	0.17%	0.96%	0.35%	1.15%	0.00%	0.85%	0.15%	1.64%	0.00%	0.96%
3	1	2	0.00%	2.59%	0.44%	0.69%	0.00%	1.61%	0.01%	1.81%	0.00%	5.94%	1.16%	2.16%	0.00%	2.65%
3	1	3	0.00%	2.01%	0.00%	2.15%	0.00%	3.04%	0.00%	1.01%	1.04%	1.03%	0.32%	3.32%	0.00%	3.80%
3	1	4	1.18%	0.41%	0.38%	1.37%	2.45%	1.23%	0.70%	0.24%	0.18%	1.87%	0.00%	4.30%	0.00%	2.39%
3	1	5	1.53%	1.20%	0.00%	6.47%	0.00%	5.32%	0.15%	1.39%	0.00%	2.06%	0.00%	3.78%	0.00%	3.40%
3	1	6	0.00%	5.39%	0.36%	0.12%	1.40%	0.00%	0.00%	2.26%	0.00%	2.34%	0.00%	4.35%	0.00%	1.86%
3	1	7	0.00%	1.28%	0.00%	0.62%	0.97%	2.57%	0.33%	1.57%	0.00%	0.58%	0.45%	1.04%	0.00%	0.53%
3	1	8	0.48%	0.21%	0.35%	0.87%	0.00%	0.85%	0.32%	0.11%	0.00%	0.91%	0.22%	2.69%	0.00%	0.24%
3	1	9	0.00%	1.03%	0.00%	1.40%	1.14%	1.70%	0.00%	1.33%	0.57%	0.87%	0.00%	0.50%	0.00%	1.51%
3	2	1	0.00%	2.87%	0.01%	0.65%	0.00%	1.67%	0.00%	1.58%	0.00%	4.41%	0.00%	1.26%	0.37%	2.00%
3	2	2	0.75%	0.00%	1.32%	0.96%	1.95%	0.93%	0.00%	0.46%	0.00%	2.14%	2.60%	1.80%	0.00%	3.28%
3	2	3	0.00%	2.30%	0.00%	2.40%	0.00%	1.98%	0.00%	1.30%	0.00%	3.81%	0.00%	2.67%	0.30%	1.35%
3	2	4	0.00%	2.06%	0.00%	3.05%	1.44%	0.64%	0.60%	0.40%	0.00%	2.97%	0.00%	0.35%	0.00%	1.28%
3	2	5	0.00%	2.16%	0.00%	1.50%	1.70%	0.00%	0.00%	1.97%	0.13%	0.18%	1.63%	3.76%	0.00%	3.75%
3	2	6	0.00%	2.54%	0.00%	2.04%	0.00%	3.29%	0.00%	2.88%	0.00%	3.15%	0.00%	2.58%	0.00%	3.33%
3	2	7	0.00%	4.53%	0.96%	0.00%	0.30%	1.99%	0.00%	1.60%	0.00%	2.09%	0.00%	1.93%	0.00%	3.34%
3	2	8	0.00%	2.42%	0.00%	3.24%	0.00%	3.26%	0.74%	2.17%	0.00%	2.44%	0.00%	2.24%	0.00%	5.01%
3	2	9	0.00%	2.28%	0.00%	2.70%	0.42%	0.18%	0.00%	1.28%	0.55%	2.17%	0.87%	0.48%	0.39%	0.23%
3	3	1	0.00%	1.40%	0.91%	0.31%	0.00%	1.64%	0.00%	1.20%	0.15%	0.87%	0.00%	2.20%	0.00%	2.23%
3	3	2	0.00%	3.02%	0.36%	1.24%	0.03%	2.11%	0.31%	1.98%	0.00%	0.91%	0.28%	0.90%	0.00%	3.48%
3	3	3	0.00%	2.23%	0.00%	2.79%	0.00%	1.54%	0.00%	2.41%	0.00%	2.85%	0.00%	2.34%	0.00%	2.59%
3	3	4	0.00%	3.86%	0.00%	0.81%	0.00%	1.81%	0.00%	2.89%	0.44%	1.41%	0.00%	2.87%	0.00%	3.38%
3	3	5	0.00%	1.96%	0.00%	1.67%	0.00%	1.39%	0.00%	0.60%	0.00%	3.95%	0.00%	3.70%	0.00%	3.60%
3	3	6	0.00%	0.89%	0.79%	0.00%	0.00%	1.51%	0.00%	2.94%	0.00%	0.94%	0.00%	0.55%	0.37%	1.12%
3	3	7	0.00%	2.14%	0.00%	2.15%	0.00%	2.23%	0.00%	1.88%	0.00%	1.64%	0.00%	0.80%	0.00%	2.18%
3	3	8	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
3	3	9	0.00%	2.08%	0.00%	0.00%	0.00%	0.00%	0.00%	0.99%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
grand average			***0.14%***	***0.87%***	***0.12%***	***0.72%***	***0.29%***	***0.75%***	***0.12%***	***0.67%***	***0.13%***	***0.82%***	***0.17%***	***0.78%***	***0.07%***	***0.90%***

Table 3.14. Comparison between GAI and HPSO for problems with 6 machines.

Level of g	Level of n	Level of a	$\overline{NWA}$ 87.5		75		62.5		50		37.5		25		12.5	
			RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}	RPD_{GAI}	RPD_{HPSO}
1	1	1	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1	1	2	0,00%	0,06%	0,00%	0,63%	0,44%	0,00%	0,00%	0,05%	0,28%	0,25%	0,00%	0,82%	0,00%	0,67%
1	1	3	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1	2	1	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1	2	2	0,00%	0,19%	0,00%	0,43%	0,10%	0,18%	0,18%	1,77%	0,00%	0,18%	0,00%	2,63%	0,00%	1,12%
1	2	3	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1	3	1	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1	3	2	0,00%	0,60%	0,00%	0,81%	0,00%	1,56%	0,00%	1,64%	0,00%	0,25%	0,40%	0,41%	0,00%	1,59%
1	3	3	0,07%	0,00%	0,10%	0,00%	0,00%	0,00%	0,22%	0,00%	0,09%	0,00%	0,00%	0,08%	0,00%	0,04%
2	1	1	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
2	1	2	0,00%	0,87%	0,44%	0,82%	0,57%	1,38%	0,00%	0,83%	0,00%	3,18%	0,00%	3,81%	0,12%	0,16%
2	1	3	0,00%	0,22%	0,09%	0,02%	0,00%	0,10%	0,09%	0,00%	0,09%	0,05%	0,00%	0,12%	0,00%	0,16%
2	2	1	0,00%	0,05%	0,00%	0,05%	0,00%	0,05%	0,30%	0,00%	0,00%	0,04%	0,23%	0,00%	0,00%	0,17%
2	2	2	0,00%	1,74%	0,00%	1,81%	0,48%	0,06%	1,59%	2,52%	0,36%	1,72%	0,76%	0,23%	0,00%	1,06%
2	2	3	0,04%	0,07%	0,03%	0,20%	0,05%	0,15%	0,00%	0,52%	0,22%	0,13%	0,00%	0,21%	0,03%	0,07%
2	3	1	0,00%	0,04%	0,18%	0,00%	0,20%	0,00%	0,10%	0,00%	0,01%	0,00%	0,00%	0,05%	0,00%	0,00%
2	3	2	0,00%	1,40%	0,00%	0,81%	0,00%	0,69%	0,00%	1,60%	0,00%	0,55%	0,89%	0,09%	0,00%	0,58%
2	3	3	0,10%	0,00%	0,36%	0,27%	0,00%	0,23%	0,18%	0,25%	0,00%	0,36%	0,08%	0,53%	0,06%	0,28%
3	1	1	0,00%	1,15%	0,00%	1,07%	0,00%	0,71%	0,00%	0,84%	0,00%	1,18%	0,00%	1,05%	0,00%	0,74%
3	1	2	0,00%	5,17%	0,00%	5,32%	0,00%	6,61%	0,00%	4,22%	0,00%	2,90%	0,00%	3,99%	0,00%	2,01%
3	1	3	0,49%	0,19%	0,00%	0,45%	0,00%	0,22%	0,00%	0,10%	0,00%	0,12%	0,01%	0,22%	0,00%	0,39%
3	2	1	0,00%	0,81%	0,00%	0,38%	0,00%	0,50%	0,01%	0,29%	0,00%	0,28%	0,09%	0,22%	0,00%	0,74%
3	2	2	0,00%	1,50%	0,00%	2,83%	0,00%	4,26%	0,00%	4,79%	0,00%	2,69%	0,00%	3,66%	0,00%	6,48%
3	2	3	0,00%	0,51%	0,00%	0,15%	0,55%	0,42%	0,00%	0,43%	0,03%	0,22%	0,41%	0,25%	0,12%	0,51%
3	3	1	0,00%	0,26%	0,00%	0,42%	0,00%	0,60%	0,00%	0,50%	0,00%	0,31%	0,14%	0,25%	0,02%	0,39%
3	3	2	0,00%	2,25%	0,00%	3,04%	0,00%	2,40%	0,00%	1,75%	0,00%	4,92%	0,00%	4,16%	0,00%	2,67%
3	3	3	0,00%	0,39%	0,00%	1,18%	0,00%	1,44%	0,00%	1,17%	0,00%	1,03%	0,00%	1,10%	0,00%	1,18%
grand average			***0.03%***	***0.65%***	***0.04%***	***0.77%***	***0.09%***	***0.80%***	***0.10%***	***0.86%***	***0.04%***	***0.75%***	***0.11%***	***0.88%***	***0.01%***	***0.78%***

As the reader can notice, GAI algorithm confirms its superior performances with respect to HPSO in tackling the FSDGS-SWA problem in hand regardless to the number of machines considered. In fact, the grand average RPDs obtained by the former procedure are always lower than the corresponding values yield by latter one. More in detail, the proposed genetic algorithm reports a grand average RPD ranging from 0.01% (6-machine problems with $\overline{NWA}$ = 12.5) to 0.29% (problems with 3 machines and $\overline{NWA}$ = 62.5), while the particle swarm optimization procedure arisen from literature gives a grand average deviation between 0.65% (6-machine problems with $\overline{NWA}$ = 12.5) and 1.16% (problems with 2 machines and $\overline{NWA}$ = 12.5). A further comparison between GAI and HPSO has been performed through an ANOVA analysis (Montgomery, 2007) executed on Stat-Ease® Design Expert 7.0.0. commercial tool. More in detail, LSD plots at 95% confidence level have been drawn for each

sub-benchmark considered, i.e., for 2-, 3- and 6-machines problems, as reported in Figures 3.5, 3.6 and 3.7, respectively.

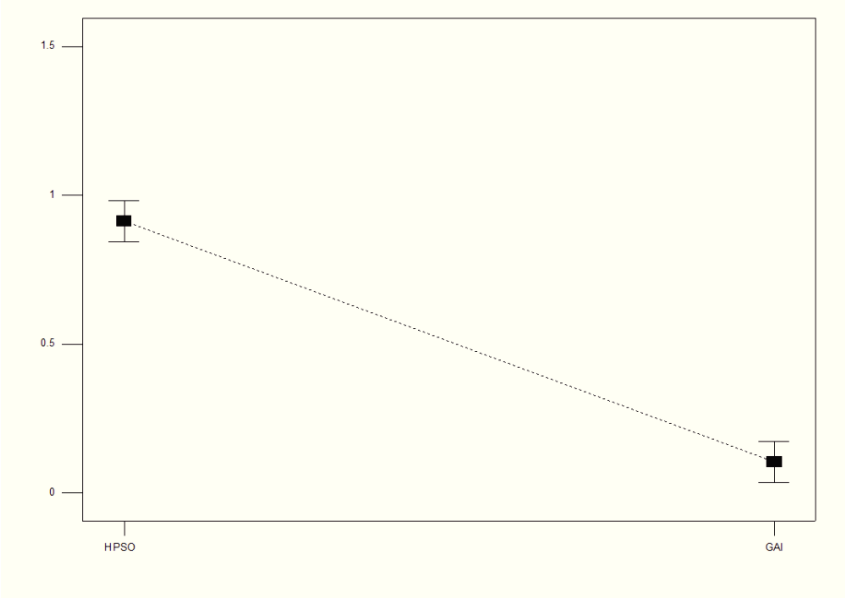

Figure 3.5. LSD plot for 2-machine problems.

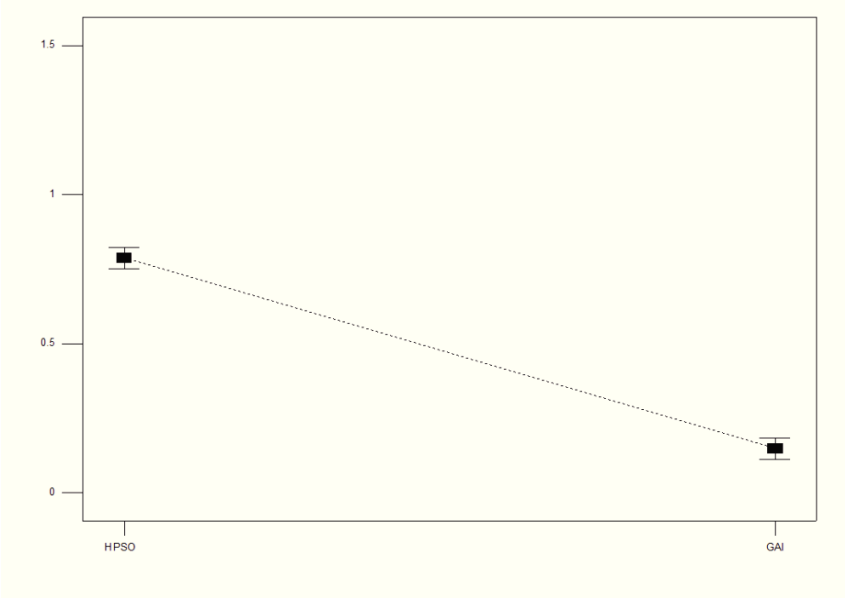

Figure 3.6. LSD plot for 3-machine problems.

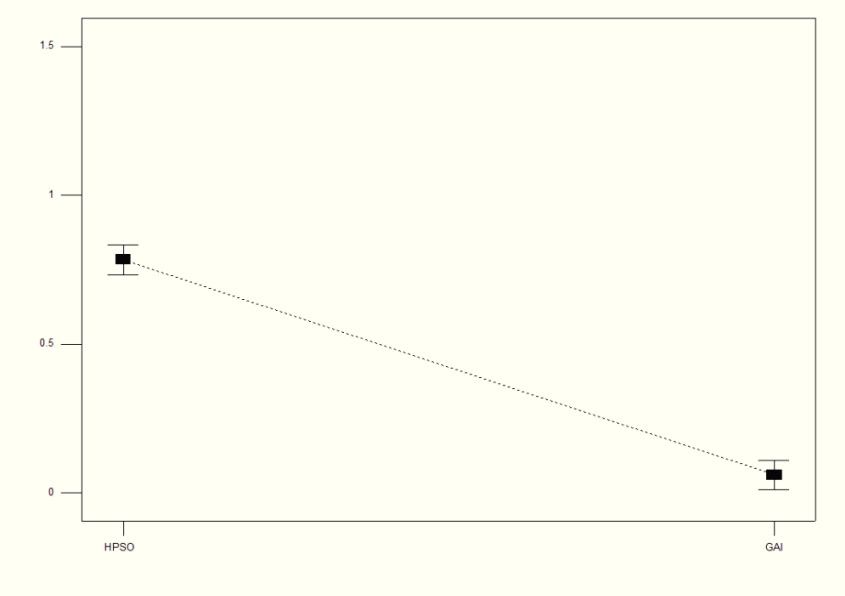

Figure 3.7. LSD plot for 6-machine problems.

The obtained graphs show how GAI systematically outperforms HPSO in a statically significant manner, as the LSD bar related to the proposed genetic algorithm always lies below the one regarding the particle swarm procedure without any overlapping. Therefore, on the basis of the multiple analysis performed, it can be concluded that the developed GAI is more effective than the HPSO method taken as reference for approaching the proposed FSDGS-SWA problem.

3.6 How manpower skills affect productivity

The aim of this section is to investigate the way different workforce teams featured by different average *NWA* may influence the performance of a given serial production system. In order to make this analysis as much comprehensive as possible, numerical results concerning both workforce configurations with $\overline{NWA} = 0$ and $\overline{NWA} = 100$ have here been included. Of course, as workforce teams with an average *NWA* equal to 0 or equal to 100 are composed only by identical workers, a simple GAI missing of the SWA encoding/decoding structure has been adopted for obtaining the corresponding near optimal solutions.

A trivial remark concerning the effect of the average skill level over the performances of the production system consists of the relation between makespan reduction and average *NWA* increase. In words, whenever the average *NWA* characterizing a give workforce team is improved the makespan decreases, despite the selected number of machines. Table 3.15 illustrates the average numerical results obtained by an indicator hereinafter named Scenario Makespan Percentage Reduction (*SMPR*). It refers to the percentage deviation between the average makespan related to a given $\overline{NWA}$ value and the reference value represented by the makespan obtained by $\overline{NWA} = 0$. For each test problem, index *SMPR* can be computed as:

$$SMPR = \frac{C_{\max}^{\overline{NWA}} - C_{\max}^{0}}{C_{\max}^{0}} \times 100 \tag{3.24}$$

Table 3.15. Average values for *SMPR* index.

Nr of machines	$\overline{NWA}$							
	12.5	25	37.5	50	62.5	75	87.5	100
2	-3.29%	-5.84%	-8.55%	-10.75%	-13.35%	-15.02%	-17.33%	-18.15%
3	-5.50%	-10.34%	-13.81%	-16.43%	-19.15%	-21.64%	-24.03%	-24.68%
6	-10.93%	-18.18%	-22.01%	-24.89%	-28.49%	-30.63%	-31.72%	-31.95%

What the reader should notice is that, for each sub-benchmark corresponding to a given number of machines, makespan reduction enlarges whenever the average *NWA* increase. Moreover, Figure 3.8 puts in evidence as the average makespan reduction due to the *NWA* increase is strongly influenced by the number of machines pertaining to the considered sub-benchmark. As higher is the number of machine, as stronger is the percentage makespan reduction.

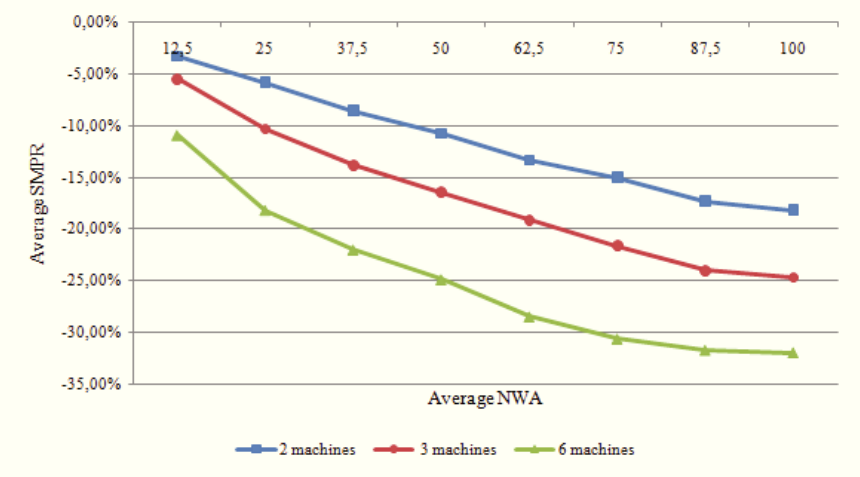

Figure 3.8. Average values for *SMPR* index.

With reference to the relationship between the average skill level and the global performance of the production system, the way a decision maker should take advantage by the aforementioned findings should be dealt with. Thus, in the following the answers to the following questions are provided:

a) How the decision maker can assess the *NWA* of the workforce currently employed within a given shop floor?

b) Once the $\overline{NWA}$ index of the working team has been established, how much the makespan of a flow-shop production system may be reduced by means of skills improvement?

c) Whether the decision maker knows how much the *NWA* of the workforce should be improved for achieving a given makespan reduction, how much the workforce skill upgrade would cost?

According to the procedure proposed in this research, evaluating the *NWA* of a given workforce team should arise from a cost analysis based on the difference between the maximum salary and the basic wage. Thus, for example, if a new inexperienced worker to be employed in a manufacturing company would receive a basic wage equal to 1,000$ per month, while the most experienced skilled worker would receive about 1,800$ per month, then the *NWA* concerning a worker paid 1,600$ should be equal to 75. Once the *NWA* has been computed for each worker, the average *NWA* characterizing a workforce team may be easily obtained. Table 3.16 reports the corresponding salary percentage increment with respect to the

basic wage as the average *NWA* changes. A Salary Step percentage increment (SSi%) equal to 10% arises by dividing by 8 the percentage difference between level nine related salary and level zero related salary.

Table 3.16. Relation *NWA* vs. salary.

Levels	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
NWA	0	12.5	25	37.5	50	62.5	75	87.5	100
Salary per month	1000$	1100$	1200$	1300$	1400$	1500$	1600$	1700$	1800$
Steady salary % increment (SSi% = 10%)	-	10%	20%	30%	40%	50%	60%	70%	80%

In order to relate the Makespan percentage reduction (*MKr*%) and the corresponding manpower percentage cost increment (*MPc*%) three decision making Tables have been properly designed. Each table refers to a given number of machines and makes the reader able to quantify the cost investment in terms of workforce skills upgrade to reach a given percentage makespan reduction. It is worth pointing out that each table should be read in the following way. Once the current average *NWA* of the workforce team is assessed, individuating the corresponding value on the left side of the Table should represent the first thing to do. Then, the decision maker should move to right until the target makespan reduction *MKr*% is met while the corresponding *NWA* value on the upper side of the Table denotes the required average *NWA* to ensure such a makespan improvement. Finally in order to quantify the manpower cost increment (*MPc*%) with respect to the current workforce configuration, a match between the SSi% value (four reference levels have been considered: 5%, 10%, 15% and 20%) and the *MPc*% value located in the same line must be detected. As for example, looking at Table 3.18 (3 machines), if the average skill level of the current workforce team is equal to 25 and the decision maker would reach a 13% makespan percentage reduction (*MKr*%), a workforce team with an average *NWA* equal to 75 should be arranged. The percentage cost increment to ensure such a makespan reduction depends on the so-called salary step increment (SSi%); thus, if the salary step increment is equal to 10% (i.e. the maximum salary is 80% higher than basic wage) then the manpower cost percentage increment (*MPc%*) is equal to 33%. On the other hand, if SSi% is equal to 5% (i.e., the maximum salary is 40% higher than basic wage) the cost increment to ensure a 13% makespan reduction is equal to 18%.

Table 3.17. Decision making table for a 2-machine production system.

		future ASL															
		100		87.5		75		62.5		50		37.5		25		12.5	
current ASL	SSi %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %
0	5%	-18%	40%	-17%	35%	-15%	30%	-13%	25%	-11%	20%	-8%	15%	-6%	10%	-3%	5%
	10%		80%		70%		60%		50%		40%		30%		20%		10%
	15%		120%		105%		90%		75%		60%		45%		30%		15%
	20%		160%		140%		120%		100%		80%		60%		40%		20%
12.5	5%	-15%	33%	-14%	29%	-12%	24%	-10%	19%	-8%	14%	-5%	10%	-3%	5%		
	10%		64%		55%		45%		36%		27%		18%		9%		
	15%		91%		78%		65%		52%		39%		26%		13%		
	20%		117%		100%		83%		67%		50%		33%		17%		
25	5%	-13%	27%	-12%	23%	-10%	18%	-8%	14%	-5%	9%	-3%	5%				
	10%		50%		42%		33%		25%		17%		8%				
	15%		69%		58%		46%		35%		23%		12%				
	20%		86%		71%		57%		43%		29%		14%				
37.5	5%	-10%	22%	-10%	17%	-7%	13%	-5%	9%	-2%	4%						
	10%		38%		31%		23%		15%		8%						
	15%		52%		41%		31%		21%		10%						
	20%		63%		50%		38%		25%		13%						
50	5%	-8%	17%	-7%	13%	-5%	8%	-3%	4%								
	10%		29%		21%		14%		7%								
	15%		38%		28%		19%		9%								
	20%		44%		33%		22%		11%								
62.5	5%	-6%	12%	-5%	8%	-2%	4%										
	10%		20%		13%		7%										
	15%		26%		17%		9%										
	20%		30%		20%		10%										
75	5%	-4%	8%	-3%	4%												
	10%		13%		6%												
	15%		16%		8%												
	20%		18%		9%												
87.5	5%	-1%	4%														
	10%		6%														
	15%		7%														
	20%		8%														

Table 3.18. Decision making table for a 3-machine production system.

		future ASL													
		100		87.5		75		62.5		50		37.5		25		12.5	
current ASL	SSi %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %
0	5%	-25%	40%	-24%	35%	-22%	30%	-19%	25%	-16%	20%	-14%	15%	-10%	10%	-6%	5%
	10%		80%		70%		60%		50%		40%		30%		20%		10%
	15%		120%		105%		90%		75%		60%		45%		30%		15%
	20%		160%		140%		120%		100%		80%		60%		40%		20%
12.5	5%	-20%	33%	-19%	29%	-17%	24%	-14%	19%	-12%	14%	-9%	10%	-5%	5%		
	10%		64%		55%		45%		36%		27%		18%		9%		
	15%		91%		78%		65%		52%		39%		26%		13%		
	20%		117%		100%		83%		67%		50%		33%		17%		
25	5%	-16%	27%	-15%	23%	-13%	18%	-10%	14%	-7%	9%	-4%	5%				
	10%		50%		42%		33%		25%		17%		8%				
	15%		69%		58%		46%		35%		23%		12%				
	20%		86%		71%		57%		43%		29%		14%				
37.5	5%	-13%	22%	-12%	17%	-9%	13%	-6%	9%	-3%	4%						
	10%		38%		31%		23%		15%		8%						
	15%		52%		41%		31%		21%		10%						
	20%		63%		50%		38%		25%		13%						
50	5%	-10%	17%	-9%	13%	-6%	8%	-3%	4%								
	10%		29%		21%		14%		7%								
	15%		38%		28%		19%		9%								
	20%		44%		33%		22%		11%								
62.5	5%	-7%	12%	-6%	8%	-3%	4%										
	10%		20%		13%		7%										
	15%		26%		17%		9%										
	20%		30%		20%		10%										
75	5%	-4%	8%	-3%	4%												
	10%		13%		6%												
	15%		16%		8%												
	20%		18%		9%												
87.5	5%	-1%	4%														
	10%		6%														
	15%		7%														
	20%		8%														

Table 3.19. Decision making table for a 6-machine production system.

		future ASL															
		100		87.5		75		62.5		50		37.5		25		12.5	
current ASL	SSi %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %	MKr %	MPc %
0	5%	-32%	40%	-32%	35%	-31%	30%	-29%	25%	-25%	20%	-22%	15%	-18%	10%	-11%	5%
	10%		80%		70%		60%		50%		40%		30%		20%		10%
	15%		120%		105%		90%		75%		60%		45%		30%		15%
	20%		160%		140%		120%		100%		80%		60%		40%		20%
12.5	5%	-24%	33%	-23%	29%	-22%	24%	-20%	19%	-16%	14%	-13%	10%	-8%	5%		
	10%		64%		55%		45%		36%		27%		18%		9%		
	15%		91%		78%		65%		52%		39%		26%		13%		
	20%		117%		100%		83%		67%		50%		33%		17%		
25	5%	-17%	27%	-17%	23%	-15%	18%	-13%	14%	-8%	9%	-5%	5%				
	10%		50%		42%		33%		25%		17%		8%				
	15%		69%		58%		46%		35%		23%		12%				
	20%		86%		71%		57%		43%		29%		14%				
37.5	5%	-13%	22%	-13%	17%	-11%	13%	-8%	9%	-4%	4%						
	10%		38%		31%		23%		15%		8%						
	15%		52%		41%		31%		21%		10%						
	20%		63%		50%		38%		25%		13%						
50	5%	-9%	17%	-9%	13%	-8%	8%	-5%	4%								
	10%		29%		21%		14%		7%								
	15%		38%		28%		19%		9%								
	20%		44%		33%		22%		11%								
62.5	5%	-5%	12%	-5%	8%	-3%	4%										
	10%		20%		13%		7%										
	15%		26%		17%		9%										
	20%		30%		20%		10%										
75	5%	-2%	8%	-1%	4%												
	10%		13%		6%												
	15%		16%		8%												
	20%		18%		9%												
87.5	5%	0%	4%														
	10%		6%														
	15%		7%														
	20%		8%														

Chapter 4

The hybrid flow shop scheduling problem with job overlapping and availability constraints

4.1 Preliminaries

In the last fifty years a huge amount of literature addressed the traditional scheduling problems as flow shop, flow lines, job shop, and parallel machines shop. This field of research has been investigated over the years by means of several optimization techniques entailing both exact approaches and heuristic techniques, also in relation to the degree of complexity of the problem in hand (Blazewicz, Ecker, Shmidt & Weglarz, 1992). However, the scheduling topic still represents a very active branch of research which recently got benefit from the study of several production environments strictly connected to the classical scheduling problems. As for example, scheduling optimization of hybrid or flexible configurations of the regular flow shop problem has been met with a great acceptance within the research community, basically due to the affinity between the way the theoretical problem is addressed and the real needs of the industrial practice. On the basis of what stated by Ruiz and Maroto (2006) there is a clear difference among Flexible Flow Line (FFL), Flow Shop with Multi-Processor (FSMP), Hybrid Flow Shop (HFS) and, finally, between Hybrid Flexible Flow Line (HFFL) and Hybrid Flexible Flow Shop (HFFS), even though all these production configurations may be considered as a variant of the traditional Flow Shop (FS) wherein one stage may hold more than one machine, at least. Both in the FFL and in the FSMP the machines available at each stage are identical. In addition, in the FSMP a given task of a job can be performed simultaneously by a set of parallel machines pertaining to a given stage. The HFS is significantly more complex than regular flow shop; each job must be processed by only one machine per stage, the production flow is unidirectional and machines within a given stage are unrelated. HFFS as well as HFFL are identical to the aforementioned HFS and FFL, the only difference being that jobs may skip some stages. An earlier research performed by Gupta

(1988) demonstrated as the FFL problem with just two stages is $\mathcal{NP}$Hard, though one of the two stages holds only one machine. One year later Gourgand, Grangeon and Norre (1999) stated that the HFS problem involves a total number of potential solutions equal to $n!\left(\prod_{i=1}^{m} m_i\right)^n$. Making full use of the criterion introduced by Linn and Zhang (1999) literary research on HFS can be roughly classified into three categories: (1) two-stage HFS; (2) three-stage HFS; (3) *m*-stage (*m*>3) HFS. Since the level of complexity considerably increases in relation to the number of stages, the earliest studies have addressed the two- and three-stages problems. Arthanary and Ramaswamy (1971) developed a pioneer research aiming to cope with a two-stage HFS scheduling problem through a Branch and Bound algorithm. A couple of years later, Salvador (1973) addressed a more complicated no-wait flowshop with multi-processors and *m*-stages by means of a dynamic programming algorithm.

More recently, several studies focused on the *m*-stage HFS problem have been elaborated. A comprehensive outline of the more relevant studies on the *m*-stage HFS has been provided by Linn and Zhang (1999) and Riane and Ariba (1999). Although optimal solutions of multi-stage HFS can be obtained via exact methods (Rajendran & Chaundhuri, 1992; Vignier, Billaut, Proust & T'Kindt, 1996; Vignier, Commandeur & Proust, 1997) when the problem size is small, the complexity connected to issues involving three or more stages justifies the employment of heuristic approaches (Ying & Lin, 2009; Jungwattanakit, Reodecha, Chaovalitwongse & Werner, 2008). Actually, heuristic methods able to address a HFS problem should be distinguished between constructive heuristics and metaheuristics. The former, traditionally characterized by the drawback of the non-robustness, are able to yield a fast response but the provided solution may result drastically far from the global optimum. Brah and Loo (1999) selected five better performing flow shop heuristics and evaluated their performances for a flow shop with multiple processors problem both in terms of makespan and flow time. They found that two well-known heuristics were comparable in performance in a flow shop with multi processors and that one of those was more consistent and robust than the other one for both optimization criteria. Ruiz, Serifoglu and Urlings (2008) proposed a mixed integer modelling and a set of constructive heuristics for the HFFS scheduling problem. Instances up to 15 jobs exhaustively have been solved by the MIP model while larger instances have been investigated by an adaptation of the NEH (Nawaz, Enscore & Ham, 1983)

algorithm as well as by means of a set of well-known despatching rules. Ying and Lin (2009) proposed an effective and efficient heuristic named Heuristic for the Multistage Hybrid Flowshop (HMHF) whose performance was properly compared with that of 10 heuristics and a tabu search based metaheuristic from the relevant literature. On the other hand, metaheuristic (ME) algorithms can be used for solving various NP-hard optimization problems, according to a proper adaptation to the structure of the problem to be tackled. MEs can reach near-optimal solutions of a large sized problem in a relatively narrow computational time than exact methods, but their implementation could result really sophisticated, also requiring arduous both coding and decoding tasks in relation to the kind of problem to be optimized. Tavakkoli-Moghddam, Sfaei and Sassoni (2009) introduced an efficient Memetic Algorithm (MA), i.e. a metaheuristic algorithm which mimics the cultural evolution, combined with a novel local search engine, named Nested Variable Neighbourhood Search (NVNS), for solving the flexible flow line scheduling problem with processor blocking and without intermediate buffers. It is worth pointing out that they adopted a structure of the chromosome for the problem representation composed by a matrix along with a vector, the former being used for job assignment to machines and the latter being used for the job permutation. The FSMP has been widely dealt with by the recent literature and several authors decided to use ME algorithms. Oguz, Zinder, Van Ha, Janiak and Lichtenstein (2004) demonstrated as the introduction of precedence constraints in the FSMP problem makes even the simplest version of this problem $\mathcal{NP}$ hard and, in addition, elaborated an approximation algorithm based on the idea of tabu search in order to address that kind of issue. Genetic Algorithms efficacy and efficiency for solving FSMP problems has been documented by Sivrikaya Serifoglu and Ulusoy (2004). Allaoui and Artiba (2004) deal with the hybrid flow shop scheduling problem under maintenance constraints to optimize several objectives based on flow time and due date. Setup, cleaning and transportation times have been take into account and a proper Simulated Annealing was implemented for optimizing the aforementioned problem. Ying and Lin (2006) developed a novel ant colony system for solving the hybrid flow shop with multiprocessor problem. To verify the proposed optimization technique, a thorough comparison with a genetic algorithm and a tabu search from the relevant literature has been carried out on the basis of two well-known benchmark problem sets. Tseng and Liao (2008) developed a Particle Swarm Optimization (PSO) algorithm, a novel metaheuristic inspired by the flocking behaviour of the

birds, powered by a new encoding scheme, namely a new way to represent in terms of string the studied problem, i.e. a flow shop with multiprocessor tasks. To assess the effectiveness of the proposed encoding embedded within the PSO, several computational experiments have been carried out, also to make a comparison with a genetic algorithm and an Ant Colony System proposed by the relevant literature. The leading role of the problem encoding has been emphasized in the research work of Gholami, Zandieh and Alem-Tabriz (2009) where a flow shop problem with multiprocessor, also including sequence-dependent setup times and machine breakdowns has been optimized. A parallel greedy algorithm approach to the hybrid flow shop with multiprocessor task scheduling problem recently has been carried out by Kaharaman, Engin, Kaya and Elif Öztürk (2010). They considered a set of 240 numerical examples divided into 24 groups and compared the performance of their technique with a tabu search and a genetic algorithm based metaheuristic, both arisen from the literature. A matching between simulation and optimization has been utilized by Rathinasamy and R (2010) for addressing the scheduling problem of an automotive vibration dampers manufacturing system arranged as a hybrid flow shop with multiprocessor. Recently a Particle Swarm Optimization PSO algorithm has been implemented for addressing a flexible flow shop with multi processors tasks (Singh & Mahapatra, 2012). Mutation, a commonly used operator in genetic algorithm, has been introduced in PSO so that trapping of solutions at local minima or premature convergence can be avoided.

With exception of the earliest researches, several papers recently dealt with the HFS scheduling problem through metaheuristics and a lot of them highlight the key role of the problem encoding for enhancing both the efficiency and the efficacy of such optimization algorithms. Ruiz and Maroto (2006) studied the makespan minimization of a HFS with sequence dependant setup times and machine eligibility scheduling problem through a genetic algorithm. They adopted a simple permutation problem encoding able to reach a good compromise between quality of solutions and computational time efficiency. A so-called rational problem encoding, basically a real number-based encoding different form the regular permutation one, has been embedded within an Immune Algorithm (IA) for solving a HFS scheduling problem with sequence dependent setup times (Zandieh, Fatemi Ghomi & Moattar Husseini, 2006). The obtained results have been compared with a random key genetic algorithm. The same kind of approach inspired by theoretical immunology has been adopted

by Engin and Döyen (2004) who showed as the Immune system technique is an effective and efficient method for solving HFS problems. A dual criteria optimization of a HFS with both machine and sequence dependent setup times scheduling problem has been performed by Jungwattanakit, Reodecha, Chaovalitwongse and Werner, 2008. Firstly, they developed a 0-1 mixed integer programming of the problem in hand and then a set of well-known constructive heuristics has been used to generate the initial populations of several genetic algorithms to be compared. The problem representation retrieved by the authors is concerned with the most popular encoding, i.e. simple permutation encoding. A similar issue involving a HFS scheduling problem with both availability and limited buffer constraints has been addressed by Yaurima, Burteseva and Tchemykh (2009). On the basis of a real printed circuit-board production environment, they extended and improved a well-known genetic algorithm equipped with a simple permutation encoding in order to enhance the scheduling strategy of the observed company. A novel Simulated Annealing (SA) algorithm to produce a reasonable manufacturing schedule in an acceptable computational time has been developed by Mirsanei, Zandieh, Moayed and Khabbazi, who used the SPTCH greedy heuristic proposed by Kurz and Askin (2004) for generating the initial solution. Hence, the SA evolutionary mechanism has been run by means of a regular permutation encoding and according to the following allocation rule: at each stage each job is assigned to the machine that allows it to be completed at the earliest time.

The hybrid flow shop can be found in many types of industries as the parallel machines at each stage can ensure both productivity and flexibility to the manufacturing functions. Adler, Fraiman, Kobacher, Pinedo, Plotnicoff and Wu (1993) used a set of priority rules to tackle a real manufacturing system for paper bags production wherein a set of unrelated parallel machines have been taken into account in some stages. Aghezzaf, Artiba, Moursli and Tahon (1995) integrated problem decomposition, heuristics and mixed-integer linear programming for solving a three stage and sequence dependent setup times HFS problem related to a carpet manufacturing firm. The most representative environment is concerned with the electronic industry, such as semiconductor wafer fabrication, printed circuit board manufacturing, thin film transistor-liquid crystal display (TFT-LCD) manufacturing, etc. With the aim to support what stated before, Choi, Kim and Lee (2011) studied the real-time scheduling problem of a

HFS with re-entrant product flow, called reentran hybrid flow shop, in a TFT-LCD manufacturing company.

The present chapter focuses on the optimization of a specific hybrid flow shop scheduling problem inspired to a real micro-electronics manufacturing environment. The aim of this chapter is presenting a new optimization approach based on a metaheuristic (ME) algorithm powered by a dual search mechanism based on two different problem encodings. The first part of the proposed MEs is arranged as a regular genetic algorithm which works by means of a regular problem representation scheme (i.e., the encoding). Once a given number of generations are reached, a properly designed random search based on a different problem encoding able to investigate a wider space of solutions is launched. A mixed-integer linear programming (MILP) model concerning the studied problem has been developed and a set of experimental analysis have been carried out in order to highlight the effectiveness of the proposed optimization approach.

The remainder of this chapter is organized as follows: in the next section a brief introduction to the problem is presented; in Section 4.3 the MILP model is reported; the problem encoding and its related decoding procedure are explained in Section 4.4, along with the help of an illustrative numerical example; in Section 4.5 the structure of the proposed metaheuristics are discussed while in Section 4.6 numerical results arisen from an extensive experimental campaign are discussed.

4.2 Problem description

A production problem characterized by a flow shop including a set of unrelated parallel machines for each stage, usually known as Hybrid Flow Shop scheduling (HFS) problem, is presented in this section. Conforming to most of the literature, the HFS problem here addressed consists of a set of independent jobs $i \in \{1,2,..,n\}$ which has to be processed through $j \in \{1,2,..,m\}$ stages; each stage entails a set of unrelated parallel machines $k \in \{1,2,..,k_j\}$ and between two subsequent stages exists a proper inter-stage buffer whose capacity, in terms of number of jobs to be stored, is unlimited.

The problem in hand should be considered as a both static and deterministic scheduling issue and, in addition, it is subject to the following further limitations. There are no precedence

relationships among jobs. Preemption is not allowed, thus avoiding that once an operation starts it could be subject to any interruption. Job splitting is not permitted. The product flow is unidirectional, i.e. a job starts at the first stage and finishes at the last stage, so that a given stage cannot be visited more than one time by the same job. Unlike the multi-processor flow shop problems, in a HFS a given job must be processed by only one machine per stage. The ready time of each job is zero, namely all the jobs are available at the beginning of the scheduling period. Machines within each stage are unrelated and set-up times have to be considered included into processing times and sequence independent.
Differently from other similar issues addressed by literature, and conforming to an observed real semi-conductor manufacturing problem, the proposed HFS problem is characterized by the following three restrictions.

1) *Overlap of jobs of the same type on a given machine is allowed*. Each machine is characterized by a manufacturing capacity according to which a limited number of "identical" jobs, i.e. jobs of the same type, can be processed at the same time on that machine, without leading to any machine processing time increase.
2) *Waiting time limit concerning the jobs staying within any inter-stage buffer is provided*. Whenever a job has been completed on a given stage, its waiting time on the inter-stage buffer before being processed on the subsequent stage must be lower than or equal to a provided time limit.
3) *Machine unavailability time intervals*. During the planning horizon each machine included within the HFS production system may be subject to one or more unavailability time intervals, similarly being done in case of machine maintenance inspections. It is worth pointing out as the unavailability time intervals are a-priori known by the decision maker for each machine pertaining to each stage.

Figure 4.1 shows the product flow concerning the aforementioned HFS scheduling problem. As the reader can notice, the job overlapping is allowed according to the machine manufacturing capacity, which is a-priori known. Then, job storage within each inter-stage buffer cannot exceed a provided time limit and each machine can get unavailable in relation to a well-known inspection time window.

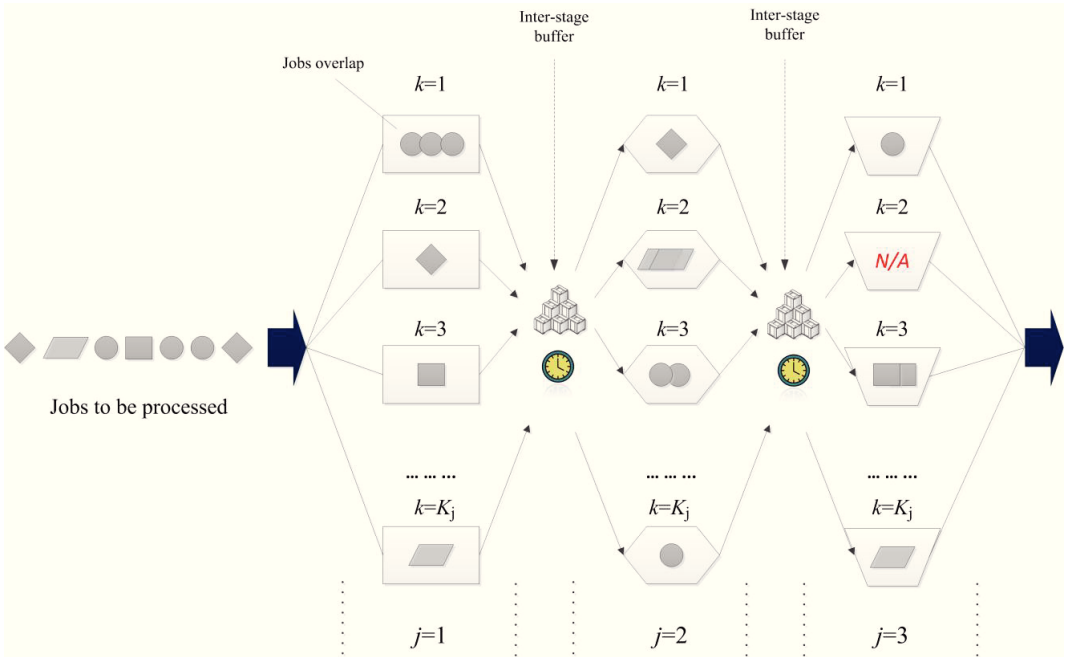

Figure 4.1. The product flow through the proposed HFS.

4.3 MILP formulation

In the present section the mathematical model of the proposed HFS problem is formalized.

Indices/parameters:

J	number of stages
K_j	number of machines at stage j
I	number of real jobs
U	number of dummy jobs corresponding to machines unavailability intervals
$R = I \cup U$	total amount of jobs including dummy jobs
S_i	number of jobs identical to job i
$T_{(j-1)j}$	maximum waiting time for every job between stage $j-1$ and stage j
SU_{ujk}	start time of u-th unavailability time interval of machine k at stage j
EU_{ujk}	completion time of u-th unavailability time interval of machine k at stage j

D_{ijk}	processing time of job i on machine k at stage j; if i is a dummy job which runs the u-th unavailability time interval of machine k at stage j $D_{ijk} = EU_{ujk} - SU_{ujk}$
N_{jk}	machine manufacturing capacity, i.e. maximum number of identical jobs that can be worked simultaneously on machine k at stage j
M	a big number

Decision variables:

Y_{ijk}	$\begin{cases} 1 \text{ if job } i \text{ is worked on machine } k \text{ at stage } j \\ 0 \text{ otherwise} \end{cases}$	$i \in R, \quad j \in J, \quad k \in K_j$
W_{iljk}	$\begin{cases} 1 \text{ if job } i \text{ and job } l \text{ are worked simultaneously on machine } k \text{ at stage } j \\ 0 \text{ otherwise} \end{cases}$	$i,l \in I, \quad l \in S_i, \quad l > i, \quad j \in J, \quad k \in K_j$
Z_{iljk}	$\begin{cases} 1 \text{ if job } i \text{ is completed before job } l \text{ is started on machine } k \text{ at stage } j \\ 0 \text{ if job } l \text{ is completed before job } i \text{ is started on machine } k \text{ at stage } j \end{cases}$	$i,l \in I, \quad l \notin S_i, \quad l > i, \quad j \in J, \quad k \in K_j$
H_{iljk}	$\begin{cases} 1 \text{ if job } i \text{ is completed before job } l \text{ is started on machine } k \text{ at stage } j \\ 0 \text{ if job } i \text{ starts at a time equal or greater than starting time of job } l \text{ on machine } k \text{ at stage } j \end{cases}$	$i,l \in I, \quad l \in S_i, \quad j \in J, \quad k \in K_j$
Q_{iljk}	auxiliary variable for *either-or* constraint	$i,l \in I, \quad l \in S_i, \quad l > i, \quad j \in J, \quad k \in K_j$
X_{ijk}	start time of job i on machine k at stage j	$i \in R, \quad j \in J, \quad k \in K_j$
$C_{\max}$	makespan	

Model

minimize $C_{\max}$

Subject to:

$$\sum_{k \in K_j} Y_{ijk} = 1 \qquad i \in I \quad j \in J \tag{4.1}$$

$$X_{ijk} \le M \cdot Y_{ijk} \qquad i \in I \quad j \in J \quad k \in K_j \tag{4.2}$$

$$\sum_{k\in K_j} X_{ijk} \geq \sum_{q\in K_{j-1}} (X_{i(j-1)q} + D_{i(j-1)q} \cdot Y_{i(j-1)q}) \qquad i \in I \quad j \in J \quad j \geq 2 \tag{4.3}$$

$$\sum_{k\in K_j} X_{ijk} - \sum_{q\in K_{j-1}} (X_{i(j-1)q} + D_{i(j-1)q} \cdot Y_{i(j-1)q}) \leq T_{(j-1)j} \qquad i \in I \quad j \in J \quad j \geq 2 \tag{4.4}$$

$$X_{ujk} = SU_{ujk} \qquad u \in U \quad j \in J \quad k \in K_j \tag{4.5}$$

$$\begin{cases} X_{ljk} + D_{ljk} \cdot Y_{ljk} \leq X_{ijk} + M \cdot Z_{iljk} \\ X_{ijk} + D_{ijk} \cdot Y_{ijk} \leq X_{ljk} + M \cdot (1 - Z_{iljk}) \end{cases} \qquad i,l \in R \quad l \notin S_i \quad l > i \quad j \in J \quad k \in K_j \tag{4.6}$$

$$\begin{cases} X_{ijk} \geq X_{ljk} - M \cdot H_{iljk} \\ X_{ijk} + D_{ijk} \cdot Y_{ijk} \leq X_{ljk} + M \cdot (1 - H_{iljk}) \end{cases} \qquad i,l \in I \quad l \in S_i \quad j \in J \quad k \in K_j \tag{4.7}$$

$$\begin{cases} (1 - W_{iljk}) - X_{ijk} + X_{ljk} + Y_{ijk} + Y_{ljk} \leq 2 + M \cdot Q_{iljk} \\ (1 - W_{iljk}) + X_{ijk} - X_{ljk} + Y_{ijk} + Y_{ljk} \leq 2 + M \cdot (1 - Q_{iljk}) \end{cases} \qquad i,l \in I \quad l \in S_i \quad l > i \quad j \in J \quad k \in K_j \tag{4.8}$$

$$\sum_{\substack{l\in S_i \\ l<i}} W_{lijk} + \sum_{\substack{l\in S_i \\ l>i}} W_{iljk} \leq N_{jk} - 1 \qquad i \in I \quad j \in J \quad k \in K_j \tag{4.9}$$

$$C_{\max} \geq \sum_{k\in K_j^-} (X_{i^-jk} + D_{i^-jk} \cdot Y_{i^-jk}) \qquad i \in I \tag{4.10}$$

$$X_{ijk} \in \mathbb{N}_0 \qquad i \in I \quad j \in J \quad k \in K_j \tag{4.11}$$

Constraint (4.1) ensures that a given job can be processed just by one machine. Constraint (4.2) states that the start time of job *i* on machine *k* of stage *j* must be set to zero ($X_{ijk} = 0$) whenever that job is not processed on that machine. Job processing on a given stage (different from stage 1) cannot start before the same job has been processed on the previous stage. Such a condition is determined by constraint (4.3). Constraint (4.4) implies that a given job in a given stage must be processed before the inter-stage buffer time limit is reached. Thus, the difference between a job start time on stage (*j*+1) and its completion time on stage *j* must be lower or equal to the inter-stage buffer time limit $T_{1,2}$. In order to manage each unavailability time of machine *k* on stage *j*, a proper dummy job is employed; thus, constraint (4.5) states that the starting time of a given machine unavailability time *u* coincides with start processing

time of a given dummy job *i*. Constraint (4.6) avoids any overlap between two non-identical jobs on machine *k* of stage *j*. Actually, this condition is ensured by a couple of distinct constraints which also work both in case only one job out of two is processed in machine *k*, and in case none of the two jobs is processed on machine *k*. Overlap of two identical jobs on the same machine *k* of stage *j* is run by constraint (4.7). This constraint operates as a combination of two distinct constraints which also manage the case in which only one out of two jobs is processed on machine *k* or, in alternative, both the two jobs are not processed by machine *k*. Constraint (4.8) is complementary to constraint (4.9), the former being necessary to manage the contemporary processing of several jobs on the same machine, the latter being useful for avoiding to exceed the maximum processing capacity of a given machine. Whenever job *j* and job *l* are worked together on machine *k*, W_{iljk} is equal to one; variable W_{iljk} makes constraint 4.9 able to control the total number of jobs simultaneously worked on machine *k* within stage *j*, in relation to the machine capacity limit. As constraint (4.8) consists of a combination of two constraints, it can manage also the following alternative conditions: a) only one job is processed on machine *k* of stage *j*; b) none of the two jobs are processed by machine *k* on stage *j*. The makespan computation is ensured by constraint (4.10) where the total completion time is equal to the maximum job completing time on a machine of the last stage.

4.4 The proposed problem encoding and the related decoding procedure

A new trend well-established in literature consists of investigating the impact of different problem representations, i.e. the so called problem encodings, in the performance of metaheuristic algorithms. Indeed, the performance of genetic algorithms as well as other evolutionary algorithms, both in terms of quality of solution and computational burden, can be strongly affected by the way a given problem can be formalized by means of a numerical string. On the other hand, every problem encoding needs a proper decoding procedure which aims to assign a given performance to a given string. In fact, once the sequence of jobs, i.e. the priority according to which they have to be processed, has been established, a proper decoding procedure can be run to compute a given KPI. Most of literature tackles the makespan minimization for the HFS problem by adopting a traditional permutation encoding wherein the problem representation string includes the overall set of jobs to be processed. Hence, in case

of makespan minimization, the most popular decoding procedure works stage by stage, by allocating a given job ready to be processed to the machine which can finish such a job at the earliest time. Though this "smart" encoding/decoding approach demonstrated its effectiveness in running simultaneously both the permutation and the assignation of jobs to the machines, it worth noticing as such technique could result lacking in exploring the overall space of feasible solutions. Due to the high number of constraints which characterize the proposed HFS problem and due to the inability of any smart encoding in investigating the entire domain of solutions, a proper two-encoding based metaheuristics has been developed. The proposed optimization technique exploits two different encodings in order to overcome the weakness arising from the employment of a single smart encoding approach. According to the high level of constraint which characterizes the proposed HFS problem, both penalty functions and repair algorithms have been properly embedded within the decoding procedures, aiming to probe into a as wide as possible space of feasible solutions. The following sub-sections make full use of a numerical example to deal with both the adopted encoding schemes and the related decoding procedures.

Problem encoding

Usually, the hybrid flowshop needs a problem encoding which separately runs both the job permutation and the assignment of jobs to machines at every stage. However, several approaches which used such a complex technique reached unsatisfying results. Recently, Ruiz and Maroto (2006) demonstrated the effectiveness of a simple permutation encoding linked to a proper decoding criterion wherein each job is assigned to the machine that can finish that job at the earliest time in a given stage.

As reported in the previous paragraph, the proposed optimization technique makes full use of a dual problem encoding. The first part of the evolutionary mechanism which consists of GA-based metaheuristics exploits the regular permutation encoding, i.e. the proposed HFS problem is managed by means of a permutation string containing a number of digits equal to the number of jobs to be scheduled, conforming to the regular flow shop. Then, in order to thoroughly investigate a wider space of solutions, a multi-stage encoding, hereinafter named *m*-stage encoding, has been adopted. While the permutation encoding forces the same sequencing of jobs for each distinct stage, the *m*-stage encoding allows to run a different job

sequencing for each stage, thus enhancing the capacity of meta-heuristics in exploring new solutions' domains. Of course, since the *m*-stage encoding should require a computational burden which increases with the number of stages, its employment should be confined to the final phase of the overall evolutionary mechanism.

Solution decoding

In order to explain the way the decoding algorithm works, a proper 2-stage (m = 2) HFS scheduling example, with 3 jobs (n = 3) and two machines per stage (K_1 = 2, K_2 = 2) here has been approached. Table 4.1 shows the computational data concerning processing times for each job (D_{ijk}) as well as machine unavailability starting times (SU_{ujk}) and completing times (EU_{ujk}) of each machine for each stage. The job waiting time limit within inter-stage buffer $T_{j,j+1}$ is $T_{1,2}$ = 2, and the solution, i.e. the sequence of jobs to be scheduled is π = {2, 3, 1}. Job 1 and 2 are identical, namely of the same type. Manufacturing capacity N_{jk} of all machines is two, with exception of machine 1 of stage 1 that has a capacity N_{11} equal to one (hereinafter see values in parenthesis on the Gantt diagrams).

Table 4.1. Set of data for the proposed numerical example.

D_{ijk}	j=1		j=2	
	k=1	k=2	k=1	k=2
i=1	4	2	2	10
i=2	4	2	2	10
i=3	3	2	1	5
SU_{ujk}	j=1		j=2	
	k=1	k=2	k=1	k=2
u=1	5	0	3	3
EU_{ujk}	j=1		j=2	
	k=1	k=2	k=1	k=2
u=1	6	3	8	5

The overall decoding procedure is composed by three distinct parts. First part entails the Scheduling and Job Allocation (SJA) procedure. Then, in order to handle the inter-stage buffer job waiting time constraint, a proper Penalty Function Computation procedure (PFC) has been introduced. Finally, with the aim to enhance the performance of the evolutionary computation, thus driving the search mechanism towards feasible solutions domains, a Solution Repairing (SR) algorithm has been embedded within the overall decoding procedure. With regard to the proposed example, Figure 4.2 shows the Gantt diagram obtained by applying the SJA procedure, while Table 4.2 makes the reader able to understand the proposed decoding strategy whose corresponding pseudo-code is reported below.

Procedure 4.1. Sequencing and job allocation (SJA)

Input

D_{ijk} : processing time of job *i* in machine $k \in \{1,...,K_j\}$ in stage *j*;

EU_{ujk} :completing time of *u*-th unavailability interval of machine *k* at stage *j*;

Output

Et_{ijk} : matrix of completing time of job *i* on machine *k* at stage *j*;

tr_{ij} : time at which job *i* leaves stage *j;*

trm_{jk} : time at which machine *k* at stage *j* is ready for job processing;

SST_{ij} : matrix of "entrance" time of job *i* in stage *j*;

ESt_{ij} : matrix "exit" time of job *i* from stage *j*;

Obj: objective function;

Step 0: Initialization

$Et_{ijk} = 0,\ \forall i \in \{1,...,n\},\ \forall k \in \{1,...,K_j\};$

$tr_{ij} = 0,\ \forall i \in \{1,...,n\},\ \forall j \in \{1,...,J\};$

$trm_{jk} = 0,\ \forall j \in \{1,...,J\},\ \forall k \in \{1,...,K_j\};$

Step 1:

j=1;

Step 2:

$i=1$;

Step 3:

selection of the job to be scheduled: γ=*i*-th job of the solution vector;

if $j \neq 1$ **then** compute the time when the job γ leaves the previous stage *j*-1: $tr_{\gamma,j-1} \leftarrow \max_{k \in 1,...,K_{j-1}} Et_{\gamma,j-1,k}$;

k=1;

Step 4:

$trm_{j,k} \leftarrow \max_{i \in 1,...,n} Et_{i,j,k}$: time at which machine *k* at stage *j* is free from the previous operations;

if the last job processed by machine *k* is identical to γ and machine *k* has a residual production capacity **then** $trm_{j,k} \leftarrow trm_{j,k} - D_{\gamma jk}$, machine *k* is ready again to process another job;

$pst_{\gamma jk} \leftarrow \max\{tr_{\gamma,j-1}; trm_{jk}\}$: expected initial working time weather job γ is processed on machine *k*;

if there is an overlap between the processing of job γ and the *u*-th machine *k* unavailability **then** $pst_{\gamma jk} \leftarrow EU_{ujk}$: expected initial working time of job g is equal to the end of the *u*-th unavailability.

$pet_{\gamma jk} \leftarrow pst_{\gamma jk} + D_{\gamma jk}$:expected finish working time;

if $k<K_j$ **then** $k=k+1$ goto Step 4;

$k^* \leftarrow \arg\min\{pet_{\gamma jk}, k = 1,..., K_j\}$: selection of the machine able to finish the job processing before the other machines;

$Et_{\gamma jk^*} \leftarrow pet_{\gamma jk^*}$: job γ is allocated to machine *k** and finish processing times are updated;

$SSt_{\gamma j} \leftarrow pet_{\gamma jk^*} - D_{\gamma jk^*}$: updating of the stage entrance time matrix;

$ESt_{\gamma j} \leftarrow pet_{\gamma jk^*}$: updating of the stage exit time matrix;

if $i < n$ **then** $i \leftarrow i+1$, goto Step 3;

if $j < J$ **then** $j \leftarrow j+1$, goto Step 2;

Step 6: Penalty evaluation

compute *pty*: evaluate the penalty, i.e. the level of unfeasibility, of a given solution;

Step 7:Objective function computation

$Obj = \max_{i=1,\ldots,n,\ k=1,\ldots,K_J} (Et_{i,J,k}) \cdot pty$: objective function computation.

Table 4.2. Step-by-step JSA decoding procedure

It	*j*	*i*	γ	$tr_{i,j}$	tmr_{jk}		$pst_{\gamma jk}$		$pet_{\gamma jk}$		*k**	$Et_{\gamma jk*}$		SSt_{ij}	ESt_{ij}
					k=1	*k*=2	*k*=1	*k*=2	*k*=1	*k*=2		Et_{i1k*}	Et_{i2k*}		
1	1	1	2	0	0	0	0	(0)3	4	(2)5	1	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 4 & 0 \\ 0 & 0 \end{bmatrix}$
2	1	2	3	0	4	0	(4)6	(0)3	(7)9	(2)5	2	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 3 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 4 & 0 \\ 5 & 0 \end{bmatrix}$
3	1	3	1	0	4	5	(4)6	5	(8)10	7	2	$\begin{bmatrix} 0 & 7 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 5 & 0 \\ 0 & 0 \\ 3 & 0 \end{bmatrix}$	$\begin{bmatrix} 7 & 0 \\ 4 & 0 \\ 5 & 0 \end{bmatrix}$
4	2	1	2	4	0	0	(4)8	(4)5	(6)10	(14)15	1	$\begin{bmatrix} 0 & 7 & 0 & 0 \\ 4 & 0 & 10 & 0 \\ 0 & 5 & 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 5 & 0 \\ 0 & 8 \\ 3 & 0 \end{bmatrix}$	$\begin{bmatrix} 7 & 0 \\ 4 & 10 \\ 5 & 0 \end{bmatrix}$
5	2	2	3	5	10	0	10	5	11	10	2	$\begin{bmatrix} 0 & 7 & 0 & 0 \\ 4 & 0 & 10 & 0 \\ 0 & 5 & 0 & 10 \end{bmatrix}$		$\begin{bmatrix} 5 & 0 \\ 0 & 8 \\ 3 & 5 \end{bmatrix}$	$\begin{bmatrix} 7 & 0 \\ 4 & 10 \\ 5 & 10 \end{bmatrix}$
6	2	3	1	7	(10)8	10	8	10	10	20	1	$\begin{bmatrix} 0 & 7 & 10 & 0 \\ 4 & 0 & 10 & 0 \\ 0 & 5 & 0 & 10 \end{bmatrix}$		$\begin{bmatrix} 5 & 8 \\ 0 & 8 \\ 3 & 5 \end{bmatrix}$	$\begin{bmatrix} 7 & 10 \\ 4 & 10 \\ 5 & 10 \end{bmatrix}$

It is worth pointing out that each value in parenthesis reported in the table refers to the temporary value assumed by a given variable during the pseudo-code step-by-step application. The basic idea which drives the proposed SJA decoding procedure concerns a priority-based criterion according to which the first job in the sequence is the first job to be scheduled, and so on for the other jobs. Once the job is selected, for each stage it takes priority to be assigned to the machine able to release it at the earliest time. Job overlapping may play a key role for makespan reduction while machine unavailability can affect the feasibility of the aforementioned priority-based decoding strategy. For that reason the SJA pseudo-code

includes several additional instructions necessary to run both the job overlapping and the machine unavailability issues.

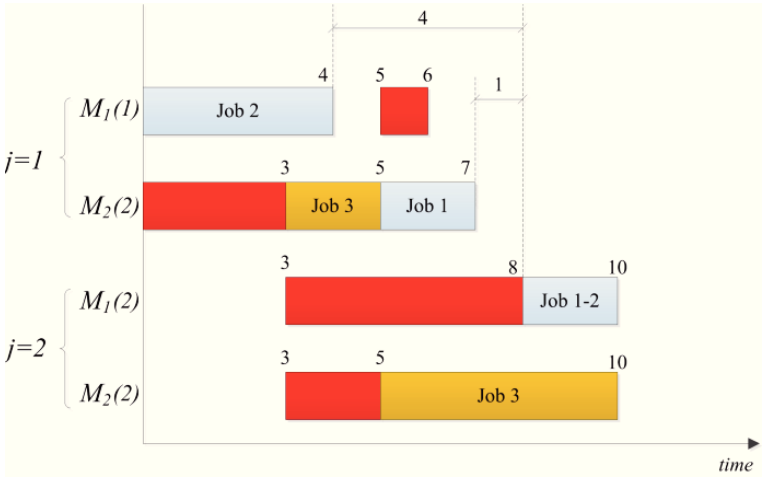

Figure 4.2. Gantt diagram obtained by SJA procedure.

As regards stage 1, numerical results concerning the first iteration in Table 4.2 show as job 2 profitably must be processed in machine 1 as it can be completed at time 4 ($pet_{211} = 4$). In fact, despite job 2 has a shorter processing time in machine 2 ($D_{112} = 2$), that machine is unavailable until time 3, thus constraining job 2 to be started at that time ($pst_{212} = 3$) and completed at time 5 ($pet_{212} = 5$). Job 3 can be processed on machine 2 after the provided unavailability time is finished; thus, job 3 can be completed at time 5 ($pet_{312} = 5$). Job 1 processing cannot be overlapped to job 2 on machine 1 as $N_{1,1} = 1$. Due to the unavailability time interval on machine 1, job 1 would start at time 6 ($pst_{111} = 6$) and it would be completed at time 10 ($pet_{111} = 10$). Instead, job 1 processing on machine 2 is more profitable under the completing time viewpoint ($pet_{112} = 7$). With regards to stage 2 (see iterations from 4 to 6 in Table 4.2), though a long unavailability time is provided for machine 1, the prior job to be scheduled, i.e. job 2, has to be processed on that machine, as its completing time on machine 2 would be ($pet_{222} = 15$). Job 3 processing on machine 2 ($pet_{322} = 10$) is better than machine 1 ($pet_{321} = 11$). Finally, as $N_{2,1} = 2$, processing of job 1 can be overlapped to that of job 2 on machine 1 and makespan is equal to 10.

After the makespan of a given solution is determined by the SJA algorithm, the feasibility of such a decoded solution with respect to the inter-stage waiting time limit must be evaluated.

Table 4.3 along with the PFC pseudo-code reported below can make the reader able to understand the way the penalty arising from the aforementioned constraint violation is computed.

Procedure 4.2. Penalty function computation (PFC).

Input

SST_{ij}: "entrance" time of job *i* within stage *j*;

ESt_{ij}: "exit" time of job *i* from stage *j*;

$T_{j-1,j}, \forall j \in 1,...,J \mid j \geq 2$: Maximum allowed waiting time between stage j-1 and stage j;

$c = 5$: penalty function exponent;

Output

pty: penalty coefficient;

for j=1 to *J*-1

for i=1 to n

compute percentage deviation between the actual waiting time and the maximum allowed waiting time of job *i* in stage *j*+1:

$$WPt_{ij} = \frac{(SST_{i,j+1} - ESt_{i,j}) - T_{j,j+1}}{T_{j,j+1}};$$

if $WPt_{ij} < 0$ **then** $WPt_{ij} = 0$: negative percentage differences are neglected;

next i

next j

$$pty = (1 + \sum_{i=1}^{n} \sum_{j=1}^{J-1} WPt_{ij})^{c}$$

As reported in Figure 4.2, the time interval between the completing time of job 2 in stage 1 and its starting time on stage 2 is equal to 4 time units, thus overstepping the provided limitation of $T_{1,2}$ = 2. Looking at both Table 4.3 and the PFC pseudo-code, the penalty function computation is equal to $pty = 2^5 = 32$, thus increasing the final solution evaluation to 10*32 = 320 time units.

Table 4.3. Steb-by-step PFC procedure.

Iter	*j*	*i*	$SSt_{i,j+1}$	ESt_{ij}	$SSt_{i,j+1} - ESt_{ij}$	$T_{j,(j+1)}$	WPt_{ij}
1	1	1	8	7	1	2	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
2	1	2	8	4	4	2	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$
3	1	3	5	5	0	2	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$

Whenever the unfeasibility of a given solution has been recognized by the PFC algorithm, the third part of the overall decoding procedure, i.e. the repairing algorithm, has to be employed. The SR procedure (see pseudo-code below) makes full use of a late-start scheduling criterion and tries to repair the current unfeasible solution, also at the expense of the total completion time. In particular, a maximum number of 21 repairing attempts (p = 0,..., 20) can be provided by the SR algorithm. With exception of the first attempt (p = 0), each attempt yields a makespan increment equal to 1% with respect to the current value. As a consequence, the maximum makespan increment (*inc*) performed by the SR algorithm may be equal to 20%. Table 4.4 and Figure 4.3 report both the step-by-step numerical trace and the Gantt diagram corresponding to a repairing attempt whose provided makespan increment is equal to zero (p = 0). Conforming to the provided SR pseudo-code, in Table 4.4 a backward scheduling repair has been performed. Since the makespan increment is equal to zero, the completion time as well the time scheduling of the jobs on stage two remains the same, ensuring a makespan equal to 10 (compare Figure 4.3 with Figure 4.2). On the basis of a backward investigation of the stages, on stage one both job 1 and job 2 have been shifted according to a late-start approach. Of course, due to the unavailability on machine 1, job 2 cannot be further shifted forward without generating a delay of the overall production. The result of the analyzed repair attempt generates a reduction of the inter-stage time interval of one time unit, going from four to three time units. Nevertheless, as the repaired schedule is not still able to fulfil the required

limit on the inter-stage waiting time, the related penalty function can be computed by means of the PFC procedure (see Table 4.5). In particular, the *pty* value for this solution is equal to 7.59 and the corresponding solution yields an increased makespan equal to 7.59*10 = 75.9 time units. Since the current solution is not yet feasible the SR procedure should try again a repair attempt by means of an additional 1% makespan increasing.

Table 4.4. Step-by-step SR procedure for $p = 0$ ($inc = 0$).

Iter	j	i	γ	k^*	tss_{ij}	tsp_{j,k^*}	$pet_{\gamma jk^*}$	$pst_{\gamma jk^*}$	$Et_{\gamma jk^*}$	$St_{\gamma jk^*}$	SSt_{ij}	ESt_{ij}
									Et_{i1k} Et_{i2k}	St_{i1k} St_{i2k}		
1	1	3	1	2	8	∞	8	6	$\begin{bmatrix} 0 & 8 & 10 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$	$\begin{bmatrix} \infty & 6 & 8 & \infty \\ \infty & \infty & 8 & \infty \\ \infty & \infty & \infty & 5 \end{bmatrix}$	$\begin{bmatrix} 6 & 8 \\ 0 & 8 \\ 0 & 5 \end{bmatrix}$	$\begin{bmatrix} 8 & 10 \\ 0 & 10 \\ 0 & 10 \end{bmatrix}$
2	1	2	3	2	5	6	5	3	$\begin{bmatrix} 0 & 8 & 10 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 5 & 0 & 10 \end{bmatrix}$	$\begin{bmatrix} \infty & 6 & 8 & \infty \\ \infty & \infty & 8 & \infty \\ \infty & 3 & \infty & 5 \end{bmatrix}$	$\begin{bmatrix} 6 & 8 \\ 0 & 8 \\ 3 & 5 \end{bmatrix}$	$\begin{bmatrix} 8 & 10 \\ 0 & 10 \\ 5 & 10 \end{bmatrix}$
3	1	1	2	1	8	∞	(8)5	(4)1	$\begin{bmatrix} 0 & 8 & 10 & 0 \\ 5 & 0 & 10 & 0 \\ 0 & 5 & 0 & 10 \end{bmatrix}$	$\begin{bmatrix} \infty & 6 & 8 & \infty \\ 1 & \infty & 8 & \infty \\ \infty & 3 & \infty & 5 \end{bmatrix}$	$\begin{bmatrix} 6 & 8 \\ 1 & 8 \\ 3 & 5 \end{bmatrix}$	$\begin{bmatrix} 8 & 10 \\ 5 & 10 \\ 5 & 10 \end{bmatrix}$

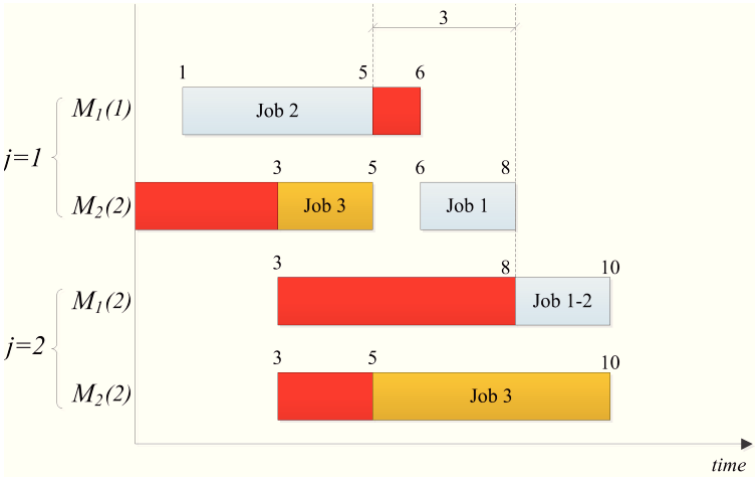

Figure 4.3. Gantt diagram after the SR procedure execution ($p = 0$).

Procedure 4.3. Solution repair (SR) algorithm.

Input

D_{ijk} : processing time of job *i* in machine $k \in \{1,...,K_j\}$ in stage *j*;

SU_{ujk} : starting time of *u*-th unavailability interval of machine *k* at stage *j*;

St_{ijk} : matrix of starting time of job *i* on machine *k* at stage *j*;

Et_{ijk} : matrix of completing time of job *i* on machine *k* at stage *j*;

Output

Obj : new objective function value after the repairing attempt;

Step 0: initialization*;*

p=0;

j=*J*;

Step 1:

$inc = p\left(\max_{i=1,...,n,\, j=1,...,J,\, k=1,...,K_j}\left(Et_{ijk}\right)\right)/100$ percentage makespan increase computation;

$St_{ijk} = \infty \;\; \forall i = 1,...,n,\; j = 1,...,J,\; k = 1,...,K_j$: start processing time initialization;

Step 2:

if *p* = 0 **then** no repairs will be performed on the last stage *J*, goto step 4;

if $p \neq 0$ **then** for each machine of the last stage all jobs are shifted ahead, i.e.,,a late scheduling of jobs according to an increase of makespan equal to *inc* is performed, thus:

$ET_{ijk} \leftarrow Et_{ijk} + inc$: finish processing time increase of job *i* on machine *k* of stage *j*;

$St_{ijk} \leftarrow Et_{ijk} - D_{ijk}$: start processing time update;

Step 3:

if the job shifting caused an overlap with any machine unavailability interval **then** *p* = *p* + 1, a new repairing attempt is needed;

if *p* < 20 **then** goto *Step 1*;

Step 4:

$Et_{ijk} \leftarrow 0 \;\; \forall i = 1,...,n,\; j = 1,...,(J-1),\; k = 1,...,K_j$: finish processing times initialization of earlier stages;

Step 5:

$j \leftarrow j$ - 1;

Step 6:

$i = n$;

Step7:

selection of the job to be scheduled: γ = *i*-th job of the solution vector;

if γ is the selected job to be scheduled and *k* is* the machine of stage *j* wherein it was allocated **then** $tss_{\gamma j} \leftarrow \min_{k=1,..,K_{j+1}} (St_{\gamma, j+1, k})$: computation of the start processing time of job γ on the subsequent stage *j*+1;

$tsp_{jk^*} \leftarrow \min_{w=(i+1),...,n} (St_{w, j, k^*})$: computation of the start processing time of job λ which follows job γ on machine *k**;

if λ is identical to γ and machine *k** has enough capacity **then** job λ may be overlapped to job γ;

$pet_{\gamma jk^*} = \min\{tss_{\gamma j},\ tsp_{jk^*}\}$: expected finish processing time computation;

$pst_{\gamma jk^*} = pet_{\gamma jk^*} - D_{\gamma jk^*}$: expected start processing time computation;

endif

if there is an overlap between the processing of job γ and the *u*-th machine *k** unavailability **then**

$pet_{\gamma jk^*} \leftarrow SU_{ujk^*}$:finish job processing time turns to the start of the *u*-th machine unavailability;

$pst_{\gamma jk^*} \leftarrow pet_{\gamma jk^*} - D_{\gamma jk^*}$: new expected starting processing time computation;

endif

$Et_{\gamma jk^*} \leftarrow pet_{\gamma jk^*},\ St_{\gamma jk^*} \leftarrow pst_{\gamma jk^*},\ ESt_{\gamma j} \leftarrow pet_{\gamma jk^*},\ SSt_{\gamma j} \leftarrow pst_{\gamma jk^*}$;

endif

if $i > 1$ **then** $i \leftarrow i$ - 1, goto Step 7;

if $j > 1$ **then** $j \leftarrow j$ - 1, goto Step 5;

Step 8: penalty computation;

If there is no penalty (*pty* = 1) **OR** *p* = 20 **then**

$Obj = \max_{i=1,\ldots,n,\ k=1,\ldots,K_J} Et_{i,J,k}$: new objective function computation;

else

$p = p + 1$ goto Step 1;

endif

Table 4.5. Step-by-step PFC procedure.

Iter	j	i	$SSt_{i,j+1}$	ESt_{ij}	$SSt_{i,j+1} - ESt_{ij}$	$T_{j,(j+1)}$	$WPt_{i,j}$
1	1	1	8	8	0	2	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
2	1	2	8	5	3	2	$\begin{bmatrix} 0 & 0 \\ 0.5 & 0 \\ 0 & 0 \end{bmatrix}$
3	1	3	5	5	0	2	$\begin{bmatrix} 0 & 0 \\ 0.5 & 0 \\ 0 & 0 \end{bmatrix}$

For sake of simplicity a 20% makespan increment, which corresponds to the last attempt, has been considered as decisive. Hence, whether makespan is increased of two times units (which correspond to a 20% increase) by means of a forward shift of the jobs processed on stage two (as shown in Figure 4.4), the inter-stage time interval of job 2 turns to zero. The step-by-step SR procedure for p = 20 is reported on Table 4.6, while by Table 4.7 the PFC procedure applied to the repaired schedule does not generate any penalty, thus ensuring a feasible makespan equal to 12 time units. Switching to the *m*-stage encoding does not involve any change in the decoding strategy, the only difference being that both SJA and SR procedures are applied several times to a number of sequences equal to the number of stages. PFC procedure is job dependent and is not influenced by the type of encoding. The advantage arising from the use of a so called *m*-stage encoding easily can be explained by the diagram reported in Figure 4.5 which refers to the following 2-stage solution: (3,2,1|2,3,1). As the reader can verify, both (3,2,1) and (2,3,1) generate a makespan equal to 12 units. if managed as single encoding-based solutions. Instead, the aforementioned *m*-stage string, working

separately for each stage, can explore an outperforming feasible solution that the single permutation encoding is not able to reach.

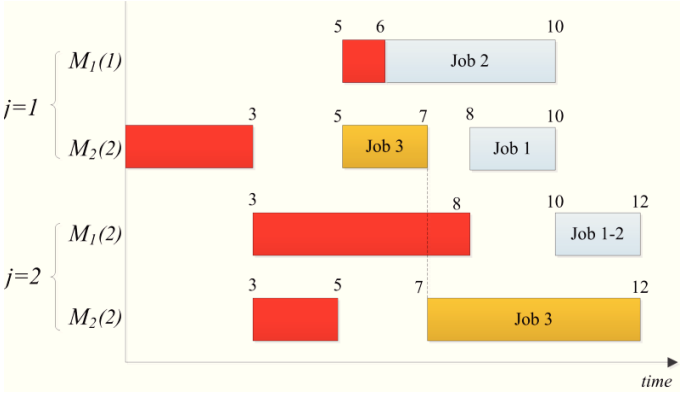

Figure 4.4 Gantt diagram after the execution of the SR procedure (p = 20).

Table 4.6 Table 4.4. Step-by-step SR procedure for p = 20 (inc = 20%).

Iter	j	i	γ	k^*	tss_{ij}	tsp_{j,k^*}	$pet_{\gamma jk^*}$	$pst_{\gamma jk^*}$	$Et_{\gamma jk^*}$	$St_{\gamma jk^*}$	SSt_{ij}	ESt_{ij}
									Et_{i1k} Et_{i2k}	St_{i1k} St_{i2k}		
1	1	3	1	2	10	∞	10	8	$\begin{bmatrix} 0 & 10 & 12 & 0 \\ 0 & 0 & 12 & 0 \\ 0 & 0 & 0 & 12 \end{bmatrix}$	$\begin{bmatrix} \infty & 8 & 10 & \infty \\ \infty & \infty & 10 & \infty \\ \infty & \infty & \infty & 7 \end{bmatrix}$	$\begin{bmatrix} 8 & 10 \\ 0 & 10 \\ 0 & 7 \end{bmatrix}$	$\begin{bmatrix} 10 & 12 \\ 0 & 12 \\ 0 & 12 \end{bmatrix}$
2	1	2	3	2	7	8	7	5	$\begin{bmatrix} 0 & 10 & 12 & 0 \\ 0 & 0 & 12 & 0 \\ 0 & 7 & 0 & 12 \end{bmatrix}$	$\begin{bmatrix} \infty & 8 & 10 & \infty \\ \infty & \infty & 10 & \infty \\ \infty & 5 & \infty & 7 \end{bmatrix}$	$\begin{bmatrix} 8 & 10 \\ 0 & 10 \\ 5 & 7 \end{bmatrix}$	$\begin{bmatrix} 10 & 12 \\ 0 & 12 \\ 7 & 12 \end{bmatrix}$
3	1	1	2	1	10	∞	10	6	$\begin{bmatrix} 0 & 10 & 12 & 0 \\ 10 & 0 & 12 & 0 \\ 0 & 7 & 0 & 12 \end{bmatrix}$	$\begin{bmatrix} \infty & 8 & 10 & \infty \\ 6 & \infty & 10 & \infty \\ \infty & 5 & \infty & 7 \end{bmatrix}$	$\begin{bmatrix} 8 & 10 \\ 6 & 10 \\ 5 & 7 \end{bmatrix}$	$\begin{bmatrix} 10 & 12 \\ 10 & 12 \\ 7 & 12 \end{bmatrix}$

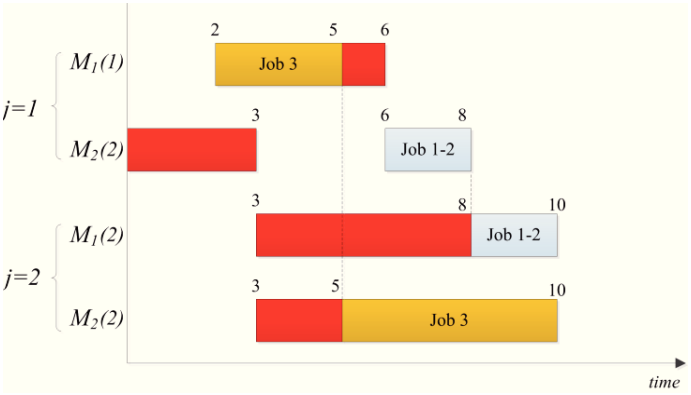

Figure 4.5. Gantt diagram concerning the m-stage encoding solution.

Table 4.7. Step-by-step PFC procedure for repaired solution.

Iter	*j*	*i*	$SSt_{i,j+1}$	ESt_{ij}	$SSt_{i,j+1} - ESt_{ij}$	$T_{j,(j+1)}$	$WPt_{i,j}$
1	1	1	10	10	0	2	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
2	1	2	10	10	0	2	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
3	1	3	7	7	0	2	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

4.5 The proposed metaheuristic algorithm

In computer science, metaheuristics (MEs) consists of stochastic computational methods able to optimize a given problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Metaheuristics disregard the specific nature of the problem to be optimized and can search very large spaces of candidate solutions also over a discrete search space. However, metaheuristics do not ensure an optimal solution is ever found. In this paper a metaheuristic optimization framework based on a Genetic Algorithm (GA) has been implemented for addressing the proposed HFS problem. In particular, two metaheuristics have been considered, the former being a regular Single Encoding Genetic Algorithm (SEGA) and the latter being a Dual Encoding Metaheuristic (DEME) consisting of a single permutation encoding-based GA that, once a given threshold is reached, switches to an *m*-stage encoding-based local search. It is worth pointing out as both SEGA and DEME algorithms are equipped with the same SJS, PFC, SR decoding procedures the only difference being that DEMEs exploit an *m*-stage encoding form a certain threshold on. At each generation a population having dimension equal to P_{size} = 100 chromosomes is evolved. Initial population is randomly generated. A position-based crossover has been adopted for exchanging the genetic material

between two chromosomes; according to a random criterion the number of genes subject to the crossover operator may vary from two to $n/2$; probability of crossover $p_{cross} = 0.5$ has been considered for all the experiments. Mutation is based on a regular pairwise interchange mechanism whose probability is $p_m = 0.05$. Selection mechanism is a regular roulette based method while, for avoiding a premature convergence, an elitism strategy working on the best two individuals of each population has been embedded within the proposed GA. The total number of fitness evaluations $N_{ev} = 10{,}000$ has been taken into account as exit criterion of the proposed SEGA optimization technique. Since a parameter $t < 100\%$ represents the threshold value according to which the single permutation encoding switches to the m-stage encoding, $DEME_t$ identifies a dual encoding metaheuristic algorithm wherein $t^* N_{ev}$ is the exit criterion of the single encoding-based optimization phase, while $(1-t)^* N_{ev}$ represents the number of evaluations carried out by the subsequent dual encoding-based local search. Both probabilities of crossover and mutation, as well as population size, were selected after a preliminarily tuning analysis not here reported for sake of brevity.

Actually, the local search consists of a random search applied on the best individual generated by the previous optimization phase. A Smart Shift Insertion (SSI) method has been adopted as perturbation operator to explore the neighborhood of a given individual. Such a method includes a mechanism aiming to avoid any breaking of groups of identical jobs, thus preserving the benefits which may result by overlapping jobs of the same type. An example of SSI application for three type of jobs is reported in Figure 6 where *pos1* and *pos2* are two randomly selected positions. A regular shift insertion procedure would yield the new chromosome 3-4-1-6-2-5 while the SSI, through a check on the neighborhood of the selected positions, aims to preserve block of identical jobs (i.e., 1-6-5), thus getting more chances of maskespan reduction by exploiting overlap of identical jobs on machines.

Figure 5.6. Smart shift insertion.

4.6 Computational experiments and results

To test the efficiency of the proposed dual encoding-based optimization procedure, an extensive numerical analysis has been performed on a large number of randomly generated HFS scheduling problems arranged within several scenarios where the number of jobs to be worked n, as well as the number of stages m, number of machine per stage K_j, machine overlapping capacity N_{jk}, maximum waiting time within inter-stage buffers $T_{(j\text{-}1)j}$, and the number of machine unavailability intervals U_{jk} have been assumed as environmental influencing factors. With exception of the number of jobs, all the other factors were varied between two levels, according to a low level and a high level respectively, as reported in Table 8. In fact, both the number of stages and the number of machines per stages vary on two levels: 3 and 5; the maximum overlapping capacity can assume a "low" value if it is extracted from a uniform distribution in the range [1, 3] or, in alternative, a "high" value whether it arises from a uniform distribution in the range [3, 5]. At the same time, further two factors, i.e. maximum waiting time within the inter-stage buffers and number of machine unavailability intervals, were varied on two levels with uniform distributions: U [400, 500], U [500, 600] and U [0, 2], U [2, 4], respectively. Totally, $5*2^5*10=1600$ runs have been carried out for each optimization technique.

Table 4.8. Scenario Problems

Parameter	Notation	Value	
Number of jobs	n	(10, 30, 50, 70, 100)	
Parameter	Notation	Value	
		Low level	High level
Number of stages	J	3	5
Number of machines per stage	K_j	3	5
Machine overlapping capacity	N_{jk}	U [1, 3]	U [3, 5]
Maximum waiting time within inter-stage buffers	$T_{(j\text{-}1)j}$	U [400, 500]	U [500, 600]
Number of machine unavailability intervals	U_{jk}	U [0, 2]	U [2, 4]

The effect of the aforementioned parameters varying between two levels, also including their interactions, were statistically investigated through a two-factor design and a successive multiple comparison analyses (Montgomery, 2007), performed on the unique response

variable represented by the makespan. Five classes of problems have been considered, according to the five levels of number of jobs adopted for this study. Each class includes 2^5 scenario problems generated by the permutation of the five factors reported in Table 4.8, each one varying between two levels. For each scenario a set of 10 randomly generated instances has been considered for the proposed experimental analysis. Since the performances of a regular SEGA along with 3 distinct DEMEs have been investigated, a total number of $5*2^5*10*4 = 6400$ instances have been investigated in this study. For each problem, processing times have been randomly extracted by a uniform distribution U(1,99). In particular, the three proposed DEME algorithms ($DEME_{70}$, $DEME_{80}$, $DEME_{90}$) are characterized by three different thresholds according to which the single permutation encoding switches to a m-stage encoding. For each unavailability time interval u, two values have been drawn out of a uniform distribution U(1,99*n/K_j), being the lower SU_{ujk} and the higher EU_{ujk}, respectively. To get more insight about the proposed MEs efficacy in detecting high quality near-optimal solutions, a comparison of these solutions with the true global optima has been performed. This step of numerical analysis has been carried out by running within ILOG CPlex® 12.1 64bit version software environment a mathematical linear programming model for optimizing the class I subset of problems which involves 10 jobs. Looking at Table 4.9, for each scenario the MK_opt column reports the average makespan computed on the 10 instances pertaining to that scenario problem. Columns whose name is $\Delta\%\_DE_{XX}$ show the average percentage difference between each DEME algorithm and the corresponding MK_opt. Columns coded as opt_DE_{XX} show the number of instances out of ten whose makespan is equal to the global optimum. $DEME_{70}$ and $DEME_{80}$ confirm their effectiveness since they are able to reach the same results obtained by the exact procedure for about 80% of the overall set of instances, also ensuring an average percentage deviation from the true optima slightly higher than 1%. Similarly, $DEME_{90}$ is able to match the global optima 228 times out of 320 with an average makespan increase equal to 1,81%. Finally, the regular GA with single permutation encoding, namely SEGA, matches the global optima results for about one out of every two instances.

Table 4.9. Comparison with the global optima obtained by ILOG CPLEX® 12.1.

Sc.	J	K_j	N_{jk}	$T_{(j-1)j}$	U_{jk}	MK_opt	Δ%_DE$_{70}$	*opt_DE$_{70}$*	Δ%_DE$_{80}$	*opt_DE$_{80}$*	Δ%_DE$_{90}$	*opt_DE$_{90}$*	Δ%_ SEGA	*Opt_SEGA*
1	3	3	1	1	1	234,1	2,67%	6	2,67%	6	4,58%	7	5,46%	6
2	3	3	1	1	2	400,8	0,20%	9	0,20%	9	0,20%	9	0,27%	8
3	3	3	1	2	1	222,7	1,11%	4	1,33%	5	3,42%	4	4,82%	4
4	3	3	1	2	2	405,3	1,02%	9	1,02%	9	1,13%	8	1,75%	7
5	3	3	2	1	1	165,9	1,05%	7	1,05%	7	2,68%	6	2,82%	6
6	3	3	2	1	2	352,5	0,00%	10	0,00%	10	0,00%	10	0,31%	8
7	3	3	2	2	1	144,0	2,61%	6	2,61%	6	2,61%	6	5,22%	3
8	3	3	2	2	2	314,2	0,35%	8	0,35%	8	0,35%	8	2,36%	5
9	3	5	1	1	1	112,4	0,43%	8	0,66%	7	0,87%	8	4,01%	5
10	3	5	1	1	2	166,4	1,66%	8	1,66%	8	4,44%	6	9,29%	4
11	3	5	1	2	1	112,0	0,52%	9	0,52%	9	0,83%	8	4,06%	3
12	3	5	1	2	2	147,8	0,32%	9	0,74%	8	2,60%	4	8,06%	3
13	3	5	2	1	1	106,3	0,86%	9	1,61%	8	0,86%	9	4,11%	4
14	3	5	2	1	2	102,3	0,00%	10	0,00%	10	0,00%	10	1,08%	8
15	3	5	2	2	1	99,4	2,39%	9	2,39%	9	2,39%	9	3,06%	8
16	3	5	2	2	2	156,6	0,00%	10	0,00%	10	0,36%	9	2,73%	5
17	5	3	1	1	1	268,8	6,67%	2	6,52%	2	9,63%	1	10,58%	1
18	5	3	1	1	2	432,1	1,05%	6	1,14%	7	0,44%	7	1,75%	6
19	5	3	1	2	1	333,0	6,51%	4	5,62%	4	7,78%	4	9,68%	4
20	5	3	1	2	2	420,6	0,81%	7	0,76%	7	0,81%	7	0,81%	7
21	5	3	2	1	1	239,5	0,98%	9	0,98%	9	1,69%	8	4,09%	4
22	5	3	2	1	2	310,6	0,95%	8	0,66%	9	0,95%	8	2,56%	5
23	5	3	2	2	1	239,5	0,00%	10	0,53%	8	1,27%	6	3,86%	5
24	5	3	2	2	2	348,1	0,98%	7	0,98%	7	1,56%	5	1,78%	4
25	5	5	1	1	1	161,1	0,74%	8	0,05%	9	2,42%	3	7,25%	**0**
26	5	5	1	1	2	256,3	0,50%	9	0,50%	9	0,50%	9	1,86%	6
27	5	5	1	2	1	165,7	0,16%	9	0,16%	9	1,07%	6	5,36%	2
28	5	5	1	2	2	314,8	0,09%	9	0,09%	9	0,97%	8	3,45%	6
29	5	5	2	1	1	146,3	0,00%	10	0,00%	10	0,93%	7	4,86%	3
30	5	5	2	1	2	180,4	0,00%	10	0,00%	10	0,32%	9	1,88%	5
31	5	5	2	2	1	149,4	0,00%	10	0,00%	10	0,33%	9	3,83%	4
32	5	5	2	2	2	245,2	0,00%	10	0,00%	10	0,00%	10	4,19%	7
*ave/**tot***							*1,08%*	***259***	*1,09%*	***258***	*1,81%*	***228***	*3,98%*	***156***

In order to highlight the results of the comparison between each configuration of DEME with respect to the simple SEGA, for each class of problems a thorough numerical analysis has been reported in the following tables. Table 4.10 illustrates the numerical performance yielded by each DEME algorithm along with the SEGA algorithm for class I of problems. In

particular, for each scenario, is reported the average makespan and the average percentage deviation ($\Delta MK_{\%}$) of each DEME from the regular SEGA. With exception of a single scenario where $DEME_{70}$ and $DEME_{90}$ do not lead to any average improvement, all the other results confirm the efficacy of the DEMEs in terms of quality of solutions.

Table 4.10. Class I: average makespan solutions and comparisons.

Sc.	n	J	K_j	N_{jk}	$T_{(j-1)j}$	U_{jk}	$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	SEGA	$\Delta MK_{\%}$ $DEME_{70}$/SEGA	$\Delta MK_{\%}$ $DEME_{80}$/SEGA	$\Delta MK_{\%}$ $DEME_{90}$/SEGA
1		3	3	1	1	1	239,5	239,5	243,4	245,4	-2,40%	-2,40%	-0,81%
2		3	3	1	1	2	401,6	401,6	401,6	401,8	-0,05%	-0,05%	-0,05%
3		3	3	1	2	1	224,6	225,5	229,0	231,9	-3,15%	-2,76%	-1,25%
4		3	3	1	2	2	409,1	409,1	409,6	412,5	-0,82%	-0,82%	-0,70%
5		3	3	2	1	1	167,8	167,8	170,2	170,5	-1,58%	-1,58%	-0,18%
6		3	3	2	1	2	352,5	352,5	352,5	353,4	-0,25%	-0,25%	-0,25%
7		3	3	2	2	1	148,2	148,2	148,2	151,0	-1,85%	-1,85%	-1,85%
8		3	3	2	2	2	315,3	315,3	315,3	321,3	-1,87%	-1,87%	-1,87%
9		3	5	1	1	1	112,9	113,2	113,4	117,6	-4,00%	-3,74%	-3,57%
10		3	5	1	1	2	168,2	168,2	171,9	179,2	-6,14%	-6,14%	-4,07%
11		3	5	1	2	1	112,5	112,5	112,8	117,3	-4,09%	-4,09%	-3,84%
12		3	5	1	2	2	148,2	149,0	152,1	160,8	-7,84%	-7,34%	-5,41%
13		3	5	2	1	1	107,3	108,0	107,3	111,1	-3,42%	-2,79%	-3,42%
14		3	5	2	1	2	102,3	102,3	102,3	103,5	-1,16%	-1,16%	-1,16%
15		3	5	2	2	1	101,6	101,6	101,6	102,1	-0,49%	-0,49%	-0,49%
16	10	3	5	2	2	2	156,6	156,6	156,9	159,8	-2,00%	-2,00%	-1,81%
17		5	3	1	1	1	285,9	285,6	293,1	295,2	-3,15%	-3,25%	-0,71%
18		5	3	1	1	2	436,3	436,6	434,1	439,0	-0,62%	-0,55%	-1,12%
19		5	3	1	2	1	349,8	347,9	352,7	357,2	-2,07%	-2,60%	-1,26%
20		5	3	1	2	2	423,8	423,6	423,8	423,8	**0,00%**	-0,05%	**0,00%**
21		5	3	2	1	1	241,4	241,4	242,8	248,2	-2,74%	-2,74%	-2,18%
22		5	3	2	1	2	314,1	313,2	314,1	320,1	-1,87%	-2,16%	-1,87%
23		5	3	2	2	1	239,5	240,8	242,5	248,5	-3,62%	-3,10%	-2,41%
24		5	3	2	2	2	350,9	350,9	352,8	353,6	-0,76%	-0,76%	-0,23%
25		5	5	1	1	1	161,9	161,2	164,7	173,8	-6,85%	-7,25%	-5,24%
26		5	5	1	1	2	257,7	257,7	257,7	260,2	-0,96%	-0,96%	-0,96%
27		5	5	1	2	1	165,9	165,9	167,2	173,1	-4,16%	-4,16%	-3,41%
28		5	5	1	2	2	315,2	315,2	318,3	326,4	-3,43%	-3,43%	-2,48%
29		5	5	2	1	1	146,3	146,3	147,5	152,5	-4,07%	-4,07%	-3,28%
30		5	5	2	1	2	180,4	180,4	181,4	185,7	-2,85%	-2,85%	-2,32%
31		5	5	2	2	1	149,4	149,4	149,8	154,9	-3,55%	-3,55%	-3,29%
32		5	5	2	2	2	245,2	245,2	245,2	256,2	-4,29%	-4,29%	-4,29%
						Ave	235,4	235,4	236,7	240,9	-2,69%	-2,66%	-2,06%
						St.dev.					1,93%	1,89%	1,54%

Similarly to what provided by class I numerical analysis, class II related numerical results reported in Table 4.11 confirm again the quality of solutions of the DEME algorithms. All the average results obtained for $DEME_{90}$ outperform those yielded by the SEGA algorithm. Only three scenarios out of 32 for $DEME_{70}$, and two scenarios out of 32 for $DEME_{80}$ result

unsatisfactory on the average. Despite of a different standard deviation, the grand averages at the end of the table confirm again the effectiveness of the proposed DEMEs.

Table 4.11. Class II: average makespan solutions and comparisons.

Sc.	n	J	K_j	N_{jk}	$T_{(j-1)j}$	U_{jk}	$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	SEGA	$\Delta MK_{\%}$ $DEME_{70}$/SEGA	$\Delta MK_{\%}$ $SEME_{80}$/SEGA	$\Delta MK_{\%}$ $DEME_{90}$/SEGA
1		3	3	1	1	1	366,2	357,6	376,2	380,6	-3,78%	-6,04%	-1,16%
2		3	3	1	1	2	813,3	803,5	811,2	812,2	**0,14%**	-1,07%	-0,12%
3		3	3	1	2	1	371,7	396,5	398,9	404,3	-8,06%	-1,93%	-1,34%
4		3	3	1	2	2	790,2	786,7	793,9	797,1	-0,87%	-1,30%	-0,40%
5		3	3	2	1	1	239,1	245,7	253,6	261	-8,39%	-5,86%	-2,84%
6		3	3	2	1	2	496,1	486,7	526,8	537,2	-7,65%	-9,40%	-1,94%
7		3	3	2	2	1	234,7	245,4	239,8	252,6	-7,09%	-2,85%	-5,07%
8		3	3	2	2	2	488,5	513,0	526,9	545,6	-10,47%	-5,98%	-3,43%
9		3	5	1	1	1	193,8	196,3	197,3	199,7	-2,95%	-1,70%	-1,20%
10		3	5	1	1	2	205,8	210,3	213,0	215,1	-4,32%	-2,23%	-0,98%
11		3	5	1	2	1	187,4	184,0	186,5	189,5	-1,11%	-2,90%	-1,58%
12		3	5	1	2	2	195,0	198,5	200,9	202,6	-3,75%	-2,02%	-0,84%
13		3	5	2	1	1	133,6	134,1	135,0	137,2	-2,62%	-2,26%	-1,60%
14		3	5	2	1	2	151,7	153,0	155,8	159,6	-4,95%	-4,14%	-2,38%
15		3	5	2	2	1	125,5	125,2	126,4	127,9	-1,88%	-2,11%	-1,17%
16	30	3	5	2	2	2	158,8	160,3	169,2	174,8	-9,15%	-8,30%	-3,20%
17		5	3	1	1	1	548,3	552,3	550,2	552,3	-0,72%	0,00%	-0,38%
18		5	3	1	1	2	941,1	923,8	919,8	923,1	**1,95%**	**0,08%**	-0,36%
19		5	3	1	2	1	573,3	571,4	566,5	569,9	**0,60%**	**0,26%**	-0,60%
20		5	3	1	2	2	882,8	890,1	886,4	892,7	-1,11%	-0,29%	-0,71%
21		5	3	2	1	1	402,8	410,1	421,1	428,1	-5,91%	-4,20%	-1,64%
22		5	3	2	1	2	655,4	663,7	669,6	679,6	-3,56%	-2,34%	-1,47%
23		5	3	2	2	1	438,7	434,1	450,1	458,5	-4,32%	-5,32%	-1,83%
24		5	3	2	2	2	647,8	660,1	661,0	676,6	-4,26%	-2,44%	-2,31%
25		5	5	1	1	1	240,4	241,2	247,4	248,7	-3,34%	-3,02%	-0,52%
26		5	5	1	1	2	305,9	303,4	305,2	306,8	-0,29%	-1,11%	-0,52%
27		5	5	1	2	1	223,7	228,9	229,9	231,4	-3,33%	-1,08%	-0,65%
28		5	5	1	2	2	343,2	341,2	344,2	347,3	-1,18%	-1,76%	-0,89%
29		5	5	2	1	1	179,9	177,0	183,0	184,5	-2,49%	-4,07%	-0,81%
30		5	5	2	1	2	203,0	208,8	211,0	214,7	-5,45%	-2,75%	-1,72%
31		5	5	2	2	1	179,4	179,3	182,1	185,3	-3,18%	-3,24%	-1,73%
32		5	5	2	2	2	231,0	232,3	233,6	232,9	-0,82%	-0,26%	0,30%
						Ave	379,6	381,7	386,6	391,5	-3,57%	-2,86%	-1,41%
						St.dev.					3,02%	2,34%	1,10%

With regards to class III scenario problems, looking at Table 4.12 it can be observed as some scenarios penalize the DEME algorithms (see bold values) while other scenarios provide an

average percentage makespan reduction higher than 11%. $DEME_{70}$ and $DEME_{80}$ ensure a higher grand average makespan reduction than $DEME_{90}$ and the same trend is confirmed by analyzing class IV and class V scenario problems reported in Table 4.13 and Table 4.14, respectively.

Table 4.12. Class III: average makespan solutions and comparisons.

Sc.	n	J	K_j	N_{jk}	$T_{(j-1)j}$	U_{jk}	$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	SEGA	$\Delta MK_{\%}$ $DEME_{70}$/SEGA	$\Delta MK_{\%}$ $DEME_{80}$/SEGA	$\Delta MK_{\%}$ $DEME_{90}$/SEGA
1		3	3	1	1	1	601,6	631,1	643,0	650,7	-7,55%	-3,01%	-1,18%
2		3	3	1	1	2	1369,9	1373,7	1377,9	1374,9	-0,36%	-0,09%	**0,22%**
3		3	3	1	2	1	674,3	669,4	704,4	712,9	-5,41%	-6,10%	-1,19%
4		3	3	1	2	2	1370,5	1379,2	1374,1	1378,4	-0,57%	**0,06%**	-0,31%
5		3	3	2	1	1	470,1	473,4	488,1	511,6	-8,11%	-7,47%	-4,59%
6		3	3	2	1	2	1087,5	1069,2	1091,9	1107,4	-1,80%	-3,45%	-1,40%
7		3	3	2	2	1	424,0	448,6	456,2	485,8	-12,72%	-7,66%	-6,09%
8		3	3	2	2	2	1124,9	1110,1	1149,2	1173,1	-4,11%	-5,37%	-2,04%
9		3	5	1	1	1	320,0	319,0	320,4	328,3	-2,53%	-2,83%	-2,41%
10		3	5	1	1	2	324,4	318,9	322,2	330,9	-1,96%	-3,63%	-2,63%
11		3	5	1	2	1	304,0	300,4	307,1	312,0	-2,56%	-3,72%	-1,57%
12		3	5	1	2	2	322,3	317,6	328,0	331,0	-2,63%	-4,05%	-0,91%
13		3	5	2	1	1	186,3	182,1	190,5	197,7	-5,77%	-7,89%	-3,64%
14		3	5	2	1	2	245,6	244,3	247,6	259,1	-5,21%	-5,71%	-4,44%
15		3	5	2	2	1	189,1	186,6	196,3	198,0	-4,49%	-5,76%	-0,86%
16	50	3	5	2	2	2	244,6	257,5	264,4	277,1	-11,73%	-7,07%	-4,58%
17		5	3	1	1	1	867,6	865,3	871,6	873,3	-0,65%	-0,92%	-0,19%
18		5	3	1	1	2	1474,6	1456,8	1469,3	1481,6	-0,47%	-1,67%	-0,83%
19		5	3	1	2	1	905,2	894,8	896,1	895,6	**1,07%**	-0,09%	**0,06%**
20		5	3	1	2	2	1434,4	1437,3	1437,5	1460,4	-1,78%	-1,58%	-1,57%
21		5	3	2	1	1	621,2	615,7	638,7	652,6	-4,81%	-5,65%	-2,13%
22		5	3	2	1	2	1100,1	1109,3	1097,0	1117,3	-1,54%	-0,72%	-1,82%
23		5	3	2	2	1	735,1	736,4	736,1	744,5	-1,26%	-1,09%	-1,13%
24		5	3	2	2	2	1095,1	1068,7	1075,4	1094,4	**0,06%**	-2,35%	-1,74%
25		5	5	1	1	1	350,9	349,6	350,3	353,9	-0,85%	-1,22%	-1,02%
26		5	5	1	1	2	500,2	495,4	499,5	517,9	-3,42%	-4,34%	-3,55%
27		5	5	1	2	1	346,4	340,5	339,5	344,4	**0,58%**	-1,13%	-1,42%
28		5	5	1	2	2	523,7	518,2	525,5	532,4	-1,63%	-2,67%	-1,30%
29		5	5	2	1	1	272,8	279,7	279,6	289,1	-5,64%	-3,25%	-3,29%
30		5	5	2	1	2	333,0	336,6	335,2	339,9	-2,03%	-0,97%	-1,38%
31		5	5	2	2	1	280,0	279,3	288,8	294,5	-4,92%	-5,16%	-1,94%
32		5	5	2	2	2	367,7	358,2	362,8	368,4	-0,19%	-2,77%	-1,52%
						Ave	639,6	638,2	645,8	655,9	-3,28%	-3,42%	-1,95%
						St.dev.					3,31%	2,39%	1,48%

Table 4.13. Class IV: average makespan solutions and comparisons.

Sc.	n	J	K_j	N_{jk}	$T_{(j\text{-}1)j}$	U_{jk}	$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	SEGA	$\Delta MK_{\%}$ $DEME_{70}$/SEGA	$\Delta MK_{\%}$ $DEME_{80}$/SEGA	$\Delta MK_{\%}$ $DEME_{90}$/SEGA
1		3	3	1	1	1	921,9	925,7	933,6	952,4	-3,20%	-2,80%	-1,97%
2		3	3	1	1	2	1945,2	1958,6	1960,4	1971,0	-1,31%	-0,63%	-0,54%
3		3	3	1	2	1	1005,8	1002,3	1015,5	1035,6	-2,88%	-3,22%	-1,94%
4		3	3	1	2	2	1960,0	1958,3	1943,8	1961,0	-0,05%	-0,14%	-0,88%
5		3	3	2	1	1	838,1	824,4	857,2	885,0	-5,30%	-6,85%	-3,14%
6		3	3	2	1	2	1622,5	1632,3	1624,7	1655,4	-1,99%	-1,40%	-1,85%
7		3	3	2	2	1	787,6	788,2	819,3	846,4	-6,95%	-6,88%	-3,20%
8		3	3	2	2	2	1873,9	1855,4	1842	1878,4	-0,24%	-1,22%	-1,94%
9		3	5	1	1	1	462,0	457,7	472,4	482,8	-4,31%	-5,20%	-2,15%
10		3	5	1	1	2	439,8	440,1	446,3	451,2	-2,53%	-2,46%	-1,09%
11		3	5	1	2	1	462,6	453	455,4	466,2	-0,77%	-2,83%	-2,32%
12		3	5	1	2	2	457,3	456,5	461,2	468,3	-2,35%	-2,52%	-1,52%
13		3	5	2	1	1	295,2	305,1	314,3	325,0	-9,17%	-6,12%	-3,29%
14		3	5	2	1	2	367,1	355,3	366,3	378,1	-2,91%	-6,03%	-3,12%
15		3	5	2	2	1	300,1	295	303,5	314,6	-4,61%	-6,23%	-3,53%
16	70	3	5	2	2	2	358,5	367,4	380,4	390,8	-8,27%	-5,99%	-2,66%
17		5	3	1	1	1	1276,9	1273,9	1351,8	1357,2	-5,92%	-6,14%	-0,40%
18		5	3	1	1	2	2056,6	2049,7	2034,7	2035,1	**1,06%**	0,72%	-0,02%
19		5	3	1	2	1	1396,8	1386,7	1340,5	1351,3	**3,37%**	2,62%	-0,80%
20		5	3	1	2	2	2031,2	2019,7	2044,6	2051,3	-0,98%	-1,54%	-0,33%
21		5	3	2	1	1	931,3	939,8	942	963,5	-3,34%	-2,46%	-2,23%
22		5	3	2	1	2	1772,7	1754	1779	1786,9	-0,80%	-1,84%	-0,44%
23		5	3	2	2	1	1015,8	1010,8	1013,1	1040,8	-2,40%	-2,88%	-2,66%
24		5	3	2	2	2	1639,8	1645	1628,1	1648,8	-0,55%	-0,23%	-1,26%
25		5	5	1	1	1	460,1	457,2	458,9	467,2	-1,52%	-2,14%	-1,78%
26		5	5	1	1	2	735,1	752	730,8	758,7	-3,11%	-0,88%	-3,68%
27		5	5	1	2	1	446,2	444,1	447,2	453,5	-1,61%	-2,07%	-1,39%
28		5	5	1	2	2	728,5	733,1	730,1	749,4	-2,79%	-2,18%	-2,58%
29		5	5	2	1	1	401,2	396,5	406,7	417,8	-3,97%	-5,10%	-2,66%
30		5	5	2	1	2	527,3	524,3	525,7	537,3	-1,86%	-2,42%	-2,16%
31		5	5	2	2	1	402,6	403,6	408,7	413,3	-2,59%	-2,35%	-1,11%
32		5	5	2	2	2	507,5	507,1	513,1	520,2	-2,44%	-2,52%	-1,36%
						Ave	950,9	949,2	954,7	969,2	-2,70%	-2,87%	-1,87%
						St.dev.					2,55%	2,35%	1,02%

Table 4.14. Class V: average makespan solutions and comparisons.

Sc	n	J	K_j	N_{jk}	$T_{(j-1)j}$	U_{jk}	$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	SEGA	$\Delta MK_{\%}$ $DEME_{70}$/SEGA	$\Delta MK_{\%}$ $DEME_{80}$/SEGA	$\Delta MK_{\%}$ $DEME_{90}$/SEGA
1		3	3	1	1	1	1442,2	1444,7	1442,8	1476,1	-2,30%	-2,13%	-2,26%
2		3	3	1	1	2	3285,5	3285,4	3290,5	3293,2	-0,23%	-0,24%	-0,08%
3		3	3	1	2	1	1445,4	1457,9	1486,1	1519,6	-4,88%	-4,06%	-2,20%
4		3	3	1	2	2	2750,3	2760,3	2746,2	2756,7	-0,23%	0,13%	-0,38%
5		3	3	2	1	1	1266,8	1261,5	1256,8	1329,0	-4,68%	-5,08%	-5,43%
6		3	3	2	1	2	2725	2723,7	2740,2	2738,2	-0,48%	-0,53%	0,07%
7		3	3	2	2	1	1219,8	1210,2	1253,7	1292,1	-5,60%	-6,34%	-2,97%
8		3	3	2	2	2	2764,8	2774,8	2770,7	2802,8	-1,36%	-1,00%	-1,15%
9		3	5	1	1	1	666,4	671,1	684,5	694,1	-3,99%	-3,31%	-1,38%
10		3	5	1	1	2	629,8	635,3	641,6	656,2	-4,02%	-3,19%	-2,22%
11		3	5	1	2	1	660,3	671,3	672,9	683,7	-3,42%	-1,81%	-1,58%
12		3	5	1	2	2	627,2	629,1	633,2	649,6	-3,45%	-3,16%	-2,52%
13		3	5	2	1	1	489,2	482,7	497,3	507,8	-3,66%	-4,94%	-2,07%
14		3	5	2	1	2	524,0	529,3	544,9	561,2	-6,63%	-5,68%	-2,90%
15		3	5	2	2	1	488,3	492,1	513,5	531,0	-8,04%	-7,33%	-3,30%
16	100	3	5	2	2	2	565,9	567,1	569,1	586,1	-3,45%	-3,24%	-2,90%
17		5	3	1	1	1	2437,6	2316,2	2437,6	2460,1	-0,91%	-5,85%	-0,91%
18		5	3	1	1	2	3435,2	3441,7	3422,9	3437,9	-0,08%	**0,11%**	-0,44%
19		5	3	1	2	1	2635,1	2614,5	2608,1	2620,9	**0,54%**	-0,24%	-0,49%
20		5	3	1	2	2	3114,4	3112,5	3098,4	3116,3	-0,06%	-0,12%	-0,57%
21		5	3	2	1	1	1521,0	1545,2	1534,2	1543,9	-1,48%	0,08%	-0,63%
22		5	3	2	1	2	3344,2	3359,8	3354,6	3360,6	-0,49%	-0,02%	-0,19%
23		5	3	2	2	1	1938,4	1920,4	1914,1	1930,9	**0,39%**	-0,54%	-0,87%
24		5	3	2	2	2	2983,3	3003,9	3006,2	3028,1	-1,48%	-0,80%	-0,72%
25		5	5	1	1	1	671,3	673,9	676,9	682,5	-1,64%	-1,26%	-0,82%
26		5	5	1	1	2	1115,3	1097,5	1101,2	1112,8	**0,22%**	-1,37%	-1,04%
27		5	5	1	2	1	630,3	629,7	634,6	642,4	-1,88%	-1,98%	-1,21%
28		5	5	1	2	2	1108,5	1102,6	1107,7	1112,8	-0,39%	-0,92%	-0,46%
29		5	5	2	1	1	556,7	551,4	558,3	569,8	-2,30%	-3,23%	-2,02%
30		5	5	2	1	2	777,0	775,9	778,3	802,1	-3,13%	-3,27%	-2,97%
31		5	5	2	2	1	568,4	569,1	577,3	587,7	-3,28%	-3,16%	-1,77%
32		5	5	2	2	2	778,0	775,8	782,2	801,0	-2,87%	-3,15%	-2,35%
						Ave	1536,4	1534,0	1541,8	1559,0	-2,35%	-2,43%	-1,59%
						St.dev.					2,15%	2,12%	1,21%

The final step of the experimental study is concerned with an ANOVA-based statistical analysis which aims to confirm a significant efficacy of the proposed dual-encoding metaheuristic algorithm. In particular a full factorial design has been carried out wherein the response variable coincides with the makespan C_{max}; the total amount of runs needed for performing the overall experimental plan is equal to 6,400. Table 4.15 represents the summary of the full-factorial ANOVA table obtained by means of Design Expert® 7.0.0 version commercial tool. For sake of simplicity, only the second level interactions among factors have been included in Table 4.15. The random blocking factor named as "Block" in Table 4.15 coincides with the problem and is varied at 10 levels since ten different instances have been generated for each scenario problem. In the full plan blocking was necessary to eliminate the variability which otherwise could altered the statistical analysis. The fixed factor A coincides with the DEME threshold, i.e. the percentage of fitness evaluations corresponding to which both the encoding switch and the local search take place; it is varied at 4 levels as follows: $DEME_{70}$, $DEME_{80}$, $DEME_{90}$, SEGA. As stated before, according to the role of the threshold, the SEGA algorithm could be considered as a $DEME_{100}$ algorithm. The obtained results from the experimental plan show that:

- The Model F-value of 203.055 implies the model is significant.
- The "Pred R-Squared" of 0.9412 is in reasonable agreement with the "Adj R-Squared" of 0.9465. "Adeq Precision" measures the signal to noise ratio. A ratio greater than 4 is desirable. Ratio equal to 66.087 indicates an adequate signal. This model can be used to navigate the design space.
- Factor A depending on the algorithms' threshold configuration is always statistically influent on makespan response variable as confirmed by the associated *p*-value lower than 0.05; thus, this means that the threshold factor is a key factor for improving the optimization performance of the dual encoding-based MEs.
- With the exception of influencing factor C, namely the limited waiting time of jobs within the inter-stage buffers, all the other factor result statistically influent. On the other hand, the interactions between factors C-E and C-F, as well as interaction B-D, are not statistically significant.

Table 4.15. ANOVA results.

Source	Response variable		Makespan	
	df	*Mean square*	*F*	*p*-Value
Block	9	279935,166		
A-Threshold	3	2,732	2,732	**0.0422**
B-UnavIntervals	1	7026,262	7026,262	**< 0.0001**
C-BufferWaiting	1	0,069	0,069	0.7923
D-Capacity	1	1394,453	1394,453	**< 0.0001**
E-Ws/Stage	1	31063,163	31063,163	**< 0.0001**
F-Stages	1	1320,262	1320,262	**< 0.0001**
G-Job	4	11349,975	11349,975	**< 0.0001**
AB	3	0,099	0,099	0.9603
AC	3	0,020	0,020	0.9963
AD	3	0,155	0,155	0.9264
AE	3	0,146	0,146	0.9325
AF	3	0,233	0,233	0.8731
AG	12	0,134	0,134	0.9998
BC	1	24,876	24,876	**< 0.0001**
BD	1	3,331	3,331	0.0680
BE	1	3877,842	3877,842	**< 0.0001**
BF	1	11,651	11,651	**0.0006**
BG	4	681,846	681,846	**< 0.0001**
CD	1	7,669	7,669	**0.0056**
CE	1	0,910	0,910	0.3403
CF	1	1,404	1,404	0.2361
CG	4	4,625	4,625	**0.0010**
DE	1	210,145	210,145	**< 0.0001**
DF	1	44,028	44,028	**< 0.0001**
DG	4	56,343	56,343	**< 0.0001**
EF	1	107,297	107,297	**< 0.0001**
EG	4	3998,266	3998,266	**< 0.0001**
FG	4	151,743	151,743	**< 0.0001**
Pred R-Sq	94,12%			
Adj R-Sq	94,65%			
Model	560	6173925,055	203,055	< 0.0001

Though the multi-stage encoding involves a higher number of digits to be used for the problem representation, computational time evaluations reported in Table 4.16 show as DEME algorithms still keep a satisfying efficiency, thus avoiding an excessive increase of computational burden. As a result of a preliminary analysis on the computational times it has been noticed that the only influencing factors affecting the time of convergence are confined to both the number of stages and the number of machines per stage, along with the number of jobs. As a consequence, for sake of simplicity, in Table

4.16 for each class of problems and for each couple of influencing factors the average computational times needed by each ME algorithm has been reported. Without loss of generality, with regard to $DEME_{90}$ it could be said that the multi-stage encoding based local search, on the average, yields a slight computational time saving with respect to SEGA. The local search threshold along with the number of machines per stage may affect the computational times performance of the proposed DEMEs if compared with the SEGA algorithm. In fact, with exception of class I problems, the average percentage deviations involving $DEME_{70}$ (ΔMK% $DEME_{70}$/SEGA) result higher than the corresponding values obtained by $DEME_{80}$ (ΔMK% $DEME_{80}$/SEGA), especially for the scenario problems characterized by a lower (i.e., equal to three) number of machines per stage.

Table 4.16. Computational time evaluations.

Job	Stages	WS/Stage	*DEME$_{70}$*	*DEME$_{80}$*	*DEME$_{90}$*	SEGA	ΔMK$_{\%}$ DEME$_{70}$/SEGA	ΔMK$_{\%}$ DEME$_{80}$/SEGA	ΔMK$_{\%}$ DEME$_{90}$/SEGA
10	3	3	42,8	43,3	42,4	42,4	*0,87%*	*2,06%*	*-0,06%*
	3	5	54,2	54,9	53,7	53,8	*0,77%*	*2,18%*	*-0,08%*
	5	3	68,3	68,9	67,5	67,5	*1,14%*	*2,09%*	*-0,06%*
	5	5	87,6	88,2	86,8	86,8	*0,93%*	*1,67%*	*-0,03%*
30	3	3	131,3	131,1	125,8	125,4	*4,71%*	*4,52%*	*0,32%*
	3	5	156,1	158,6	155,1	155,3	*0,55%*	*2,14%*	*-0,10%*
	5	3	223,3	216,6	213,4	212,7	*5,00%*	*1,82%*	*0,35%*
	5	5	257,6	254,5	255,8	255,3	*0,90%*	*-0,28%*	*0,20%*
50	3	3	239,1	231,8	227,8	227,3	*5,21%*	*2,01%*	*0,26%*
	3	5	260,0	257,7	258,0	258,3	*0,67%*	*-0,23%*	*-0,11%*
	5	3	442,0	435,0	426,3	426,7	*3,58%*	*1,96%*	*-0,09%*
	5	5	443,8	439,2	439,4	439,7	*0,95%*	*-0,12%*	*-0,07%*
70	3	3	383,3	359,2	357,8	357,8	*7,12%*	*0,39%*	*-0,01%*
	3	5	381,2	359,8	367,6	367,9	*3,62%*	*-2,19%*	*-0,08%*
	5	3	739,1	685,9	689,8	692,1	*6,80%*	*-0,90%*	*-0,33%*
	5	5	679,4	662,8	662,4	664,2	*2,29%*	*-0,20%*	*-0,27%*
100	3	3	578,7	576,4	575,0	577,0	*0,29%*	*-0,10%*	*-0,35%*
	3	5	585,7	579,5	587,1	589,3	*-0,62%*	*-1,66%*	*-0,38%*
	5	3	1114,3	1097,6	1092,1	1094,4	*1,81%*	*0,29%*	*-0,21%*
	5	5	1011,8	1007,3	1008,5	1033,2	*-2,06%*	*-2,50%*	*-2,38%*

Finally, it worth remembering as the MILP numerical examples have been solved by ILOG CPLEX® 12.0 64bit version installed within a workstation powered by two quad-core 2,39 GHz processors with 24 GB RAM. Metaheuristics have been launched within a virtual machine embedded within the same workstation with 2 GB RAM.

4.7. Comparison with other metaheuristics

Despite the proposed HFS problem including the specific constraints discussed in section 4.2 has never been studied by literature, in order to evaluate the effectiveness of the proposed dual encoding metaheuristics an extended comparison with a set of alternative metaheuristics has been carried out. In particular, performances of the so-called DEME and SEGA algorithms have been evaluated in relation to the following alternative metaheuristics per formed by literature. A brief description is reported for each one.

- Genetic Algorithm hereinafter coded as GAR developed by Ruiz and Maroto (2006), for minimizing the total completion time (makespan) of 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 suggested by the authors, similar block 2-point cross over and shift insertion have been employed as cross-over and mutation operator respectively.
- Artificial Immune System algorithm named AIE (Engine & Döyen, 2004), used by authors for solving a FFL scheduling problem as all the machines pertaining to each stage are identical. Encoding and decoding strategies used for makespan minimization are similar to what discussed before for GAR.
- Particle Swarm Optimization from now on defined as PSOS developed by Singh and Mahapatra (2012) with the aim of minimizing the makespan of a HFS scheduling problem with unrelated machines. It worth pointing out as this optimization technique adopts a string with n^*J digits and a real-number based encoding/decoding strategy according to which the integer par of each digit runs the job allocation to machines.
- Genetic Algorithm here coded as GAK (Kurz & Askin, 2004) devised for FFL optimization problems with sequence dependent setup times and allowed stage skipping thus further reducing the makespan. Similarly being done by PSOS authors

adopted a real-number based encoding for the first stage job allocation to machines. Then, for the subsequent stages, an early finish machine job allocation policy has been adopted.

- Simulated Annealing elaborated by Allaoui and Artiba (2004), hereinafter defined as SAA, used for a HFS with sequence dependent setup times and machine unavailability. Adopted encoding is based on a regular single permutation string while the decoding strategy make full use of a commercial simulation tool. For such a reason, in the present research the SAA was equipped with a regular early finish machine decoding strategy, similarly being done by the most techniques here evaluated.

Conforming to what discussed in Section 4.6, Tables 4.17 and 4.18 refers to Class I problems, i.e. HFS problem with 10 jobs. For each scenario problem composed by ten instances the average values of makespan (*ave)*, its percentage difference with respect to the average global optimum obtained by ILOG CPLEX (*Δ%*) as well as *opt*, i.e. the number of instances out of ten whose makepsan is equal to the exact solution, are reported. At the end of each table are located the grand average concerned with the percentage difference and the number of found optimal solution, along with the number of times (*wins*) the *ave* performance indicator of a given metaheuristics reaches the minimum value among the other techniques (see also underling values in Table 4.17 and 4.18). As the reader can notice all the three DEMEs strongly outperforms the other optimization techniques. In Table 4.17 $DEME_{80}$ reaches an average deviation equal to 1.02% from the average optimal solution provided by ILOG CPLEX. On the contrary, the best competitor, with exception of SEGA is the AIE algorithm with a 4.20% deviation. Table 4.18 shows as $DEME_{70}$ ensures a deviation of the average makespan lower than 1%. While both SEGA and AIE obtain similar performances with an average deviation equal to 3.59% and 3.55%, respectively. Also in terms of *opt_ave* and *wins* the provided DEMEs confirm their leadership with respect to the other optimization procedures.

As far as the other classes of problems are concerned, the PSOS algorithm has been excluded from the comparison due to the extremely weak results obtained for the Class I test cases. Remaining sets of comparison involving Classes from II to V have been addressed in Tables from 4.19 to 4.22, respectively. Average values of makespan (*AVE*) and its own percentage

deviation (Δ%) from the best result among the other competitors has been reported for each scenario problem. At the end of each Table the grand average concerning the aforementioned deviations has been included. Then, the number of times (wins) a given metaheuristics gets the best result for a given scenario has been retrieved. It is worth pointing out that, especially in case of a large number jobs to be scheduled, some metaheuristics may not find any feasible solution, thus reaching a penalized solution. Therefore, the following Tables refer only to the provided feasible solutions and reports a bold number in parenthesis aiming to put in evidence the number of instances out of ten for which a feasible solution has been reached. As for example, for one instance of scenario problem 18 in Table 4.21, both GAR and SAA do not get any feasible solution; thus, their respective Δ% have been computed on the basis of the provided feasible solutions. Finally, the following superscript symbol (*) on a given *Δ%_AVE* result indicates a kind of average performance disregarding any unfeasible instance.

Table 4.19 shows as the proposed DEMEs ensure an average deviation ranging from 0.74% of $DEME_{70}$ to 3.06% of $DEME_{90}$. $DEME_{70}$ gets 16 best results out of 32 while AIE and SEGA reach a number of wins equal to 3 and an average deviation equal to 4.55% respectively.

In Table 4.20 $DEME_{80}$ has the best average deviation equal to 0.98%. The other DEMEs do not exceed a 3% deviation while, in terms of wins, both AIE and GAK achieve the best AVE 6 and two times respectively, in spite of the weak results in terms of average deviation.

Tables 4.21 and 4.22 highlight the leading performance of the proposed dual encoding metaheuristics. Both in terms of average deviation and wins. Table 4.21 shows as scenario problems 18, 20, 22 result critical for Gar and SAA as several instances have not met a feasible solution. In Table 22 several scenario problems affected the capability of both GAR and SAA in finding feasible solutions. Even no feasible solutions (NFS) have been obtained by GAR for each instance of scenario problem 22 (see Table 4.22).

Table 4.17. Metaheuristics comparison - Class I, 10 jobs, 3 stations problems.

Sc	J	K_j	N_{jk}	T_{ij}	U_{jk}		DEME$_{70}$	DEME$_{80}$	DEME$_{90}$	GAR	AIE	PSOS	GAK	SAA	SEGA	CPLEX
						ave	239.5	239.5	239.7	262.0	247.2	439.4	258.8	248.8	245.4	234.1
1	3	3	1	1	1	*Δ%*	2.31%	2.31%	2.39%	11.92%	5.60%	87.70%	10.55%	6.28%	4.83%	
						opt	6	6	6	3	6	0	2	3	0	
						ave	401.6	401.6	401.6	414.2	401.8	784.2	403.4	401.8	401.8	400.8
2	3	3	1	1	2	*Δ%*	0.20%	0.20%	0.20%	3.34%	0.25%	95.66%	0.65%	0.25%	0.25%	
						opt	9	9	9	7	8	0	7	8	0	
						ave	224.6	225.5	231.7	255.5	229.9	416.4	262.5	236.8	231.9	222.7
3	3	3	1	2	1	*Δ%*	0.85%	1.26%	4.04%	14.73%	3.23%	86.98%	17.87%	6.33%	4.13%	
						opt	4	5	4	1	6	0	0	4	0	
						ave	409.1	409.1	409.1	418.7	412.5	736.3	413.4	413.2	412.5	405.3
4	3	3	1	2	2	*Δ%*	0.94%	0.94%	0.94%	3.31%	1.78%	81.67%	2.00%	1.95%	1.78%	
						opt	9	9	9	6	7	0	6	7	0	
						ave	167.8	167.8	167.8	183.2	170.5	368.6	186.6	178.0	170.5	165.9
5	3	3	2	1	1	*Δ%*	1.15%	1.15%	1.15%	10.43%	2.77%	122.18%	12.48%	7.29%	2.77%	
						opt	7	7	7	2	6	0	5	4	0	
						ave	352.5	352.5	352.5	366.3	353.4	641.8	357.2	357.9	353.4	352.5
6	3	3	2	1	2	*Δ%*	0.0%	0.0%	0.0%	3.9%	0.3%	82.1%	1.3%	1.5%	0.3%	
						opt	10	10	10	7	8	0	7	8	0	
						ave	148.2	148.2	148.2	158.5	150.8	378.2	153.4	151.8	151.0	144.0
7	3	3	2	2	1	*Δ%*	2.9%	2.9%	2.9%	10.1%	4.7%	162.6%	6.5%	5.4%	4.9%	
						opt	6	6	6	1	4	0	4	3	0	
						ave	315.3	315.3	315.2	324.7	321.3	643.5	330.1	321.3	321.3	314.2
8	3	3	2	2	2	*Δ%*	0.4%	0.4%	0.3%	3.3%	2.3%	104.8%	5.1%	2.3%	2.3%	
						opt	8	8	9	5	5	0	4	5	0	
						ave	112.9	113.2	113.0	118.0	117.6	266.2	122.8	117.6	117.6	112.4
9	3	5	1	1	1	*Δ%*	0.4%	0.7%	0.5%	5.0%	4.6%	136.8%	9.3%	4.6%	4.6%	
						opt	8	7	8	5	5	0	2	5	0	
						ave	168.2	168.2	171.2	188.6	179.2	488.4	205.7	189.0	179.2	166.4
10	3	5	1	1	2	*Δ%*	1.1%	1.1%	2.9%	13.3%	7.7%	193.5%	23.6%	13.6%	7.7%	
						opt	8	8	6	3	4	0	1	2	0	
						ave	112.5	112.5	112.5	119.7	117.3	265.7	124.1	117.3	117.3	112.0
11	3	5	1	2	1	*Δ%*	0.4%	0.4%	0.4%	6.9%	4.7%	137.2%	10.8%	4.7%	4.7%	
						opt	9	9	9	1	3	0	0	3	0	
						ave	148.2	149.0	151.0	168.0	160.3	512.6	170.9	160.3	160.8	147.8
12	3	5	1	2	2	*Δ%*	0.3%	0.8%	2.2%	13.7%	8.5%	246.8%	15.6%	8.5%	8.8%	
						opt	9	8	7	1	3	0	1	3	0	
						ave	107.3	108.0	108.0	111.6	111.1	228.9	115.7	111.1	111.1	106.3
13	3	5	2	1	1	*Δ%*	0.9%	1.6%	1.6%	5.0%	4.5%	115.3%	8.8%	4.5%	4.5%	
						opt	9	8	8	4	4	0	3	4	0	
						ave	102.3	102.3	102.3	104.8	103.5	471.7	115.2	103.5	103.5	102.3
14	3	5	2	1	2	*Δ%*	0.0%	0.0%	0.0%	2.4%	1.2%	361.1%	12.6%	1.2%	1.2%	
						opt	10	10	10	6	8	0	3	8	0	
						ave	101.6	101.6	101.6	102.8	102.1	227.6	108.6	102.2	102.1	99.4
15	3	5	2	2	1	*Δ%*	2.2%	2.2%	2.2%	3.4%	2.7%	129.0%	9.3%	2.8%	2.7%	
						opt	9	9	9	8	8	0	3	8	0	
						ave	156.6	156.6	157.3	159.8	159.8	504.9	163.6	159.8	159.8	156.6
16	3	5	2	2	2	*Δ%*	0.0%	0.0%	0.4%	2.0%	2.0%	222.4%	4.5%	2.0%	2.0%	
						opt	10	10	9	5	5	0	3	5	0	
						Δ%_ave	***1.07%***	***1.02%***	***1.48%***	***7.33%***	***4.20%***	***187.7%***	***8.72%***	***5.18%***	***4.02%***	
						opt_ave	***8.0***	***8.1***	***6.9***	***3.1***	***4.3***	***0***	***2.7***	***3.3***	***0***	
						wins	***15***	***11***	***9***	***0***	***0***	***0***	***0***	***0***	***0***	

Table 4.18. Metaheuristics comparison - Class I, 10 jobs, 5 stations problems.

Sc	J	K_j	N_{jk}	T_{ij}	U_{jk}		DEME$_{70}$	DEME$_{80}$	DEME$_{90}$	GAR	AIE	PSOS	GAK	SAA	SEGA	CPLEX
						ave	285.9	285.6	285.9	309.6	300.8	825.2	317.1	309.5	295.2	268.8
17	5	3	1	1	1	*Δ%*	6.4%	6.3%	6.4%	15.2%	11.9%	207.0%	18.0%	15.1%	9.8%	
						opt	2	2	1	1	1	0	1	1	0	
						ave	436.3	436.6	437.1	456.5	440.6	879.3	463.4	446.7	439.0	432.1
18	5	3	1	1	2	*Δ%*	1.0%	1.0%	1.2%	5.6%	2.0%	103.5%	7.2%	3.4%	1.6%	
						opt	6	7	6	2	5	0	1	2	0	
						ave	349.8	347.9	350.4	395.7	353.5	802.1	361.4	366.4	357.2	333.0
19	5	3	1	2	1	*Δ%*	5.0%	4.5%	5.2%	18.8%	6.2%	140.9%	8.5%	10.0%	7.3%	
						opt	4	4	4	2	5	0	2	1	0	
						ave	423.8	423.6	423.8	437.3	427.6	832.4	435.7	425.2	423.8	420.6
20	5	3	1	2	2	*Δ%*	0.8%	0.7%	0.8%	4.0%	1.7%	97.9%	3.6%	1.1%	0.8%	
						opt	7	7	7	3	5	0	5	5	0	
						ave	241.4	241.4	241.7	258.4	248.2	836.9	263.1	248.2	248.2	239.5
21	5	3	2	1	1	*Δ%*	0.8%	0.8%	0.9%	7.9%	3.6%	249.4%	9.9%	3.6%	3.6%	
						opt	9	9	8	3	4	0	3	4	0	
						ave	314.1	313.2	314.4	337.6	322.8	707.5	330.6	326.4	320.1	310.6
22	5	3	2	1	2	*Δ%*	1.1%	0.8%	1.2%	8.7%	3.9%	127.8%	6.4%	5.1%	3.1%	
						opt	8	9	8	3	5	0	5	3	0	
						ave	239.5	240.8	243.7	255.2	247.3	641.1	262.7	249.4	248.5	239.5
23	5	3	2	2	1	*Δ%*	0.0%	0.5%	1.8%	6.6%	3.3%	167.7%	9.7%	4.1%	3.8%	
						opt	10	8	7	4	6	0	3	3	0	
						ave	350.9	350.9	352.8	388.1	354.7	800.8	373.9	367.2	353.6	348.1
24	5	3	2	2	2	*Δ%*	0.8%	0.8%	1.4%	11.5%	1.9%	130.0%	7.4%	5.5%	1.6%	
						opt	7	7	5	2	4	0	3	2	0	
						ave	161.9	161.2	162.9	174.6	174.0	593.1	175.9	174.2	173.8	161.1
25	5	5	1	1	1	*Δ%*	0.5%	0.1%	1.1%	8.4%	8.0%	268.2%	9.2%	8.1%	7.9%	
						opt	8	9	5	0	0	0	0	0	0	
						ave	257.7	257.7	257.7	260.2	260.2	682.1	263.0	260.2	260.2	256.3
26	5	5	1	1	2	*Δ%*	0.5%	0.5%	0.5%	1.5%	1.5%	166.1%	2.6%	1.5%	1.5%	
						opt	9	9	9	6	6	0	4	6	0	
						ave	165.9	165.9	166.6	178.8	173.1	557.0	178.8	173.5	173.1	165.7
27	5	5	1	2	1	*Δ%*	0.1%	0.1%	0.5%	7.9%	4.5%	236.1%	7.9%	4.7%	4.5%	
						opt	9	9	7	1	2	0	2	2	0	
						ave	315.2	315.2	319.1	328.8	326.4	694.4	334.8	326.4	326.4	314.8
28	5	5	1	2	2	*Δ%*	0.1%	0.1%	1.4%	4.4%	3.7%	120.6%	6.4%	3.7%	3.7%	
						opt	9	9	7	5	6	0	5	6	0	
						ave	146.3	146.3	148.2	153.0	152.2	551.8	160.3	153.0	152.5	146.3
29	5	5	2	1	1	*Δ%*	0.0%	0.0%	1.3%	4.6%	4.0%	277.2%	9.6%	4.6%	4.2%	
						opt	10	10	8	3	3	0	1	3	0	
						ave	180.4	180.4	180.4	186.8	185.7	669.4	200.3	187.9	185.7	180.4
30	5	5	2	1	2	*Δ%*	0.0%	0.0%	0.0%	3.5%	2.9%	271.1%	11.0%	4.2%	2.9%	
						opt	10	10	10	5	5	0	3	4	0	
						ave	149.4	149.4	149.5	155.5	154.9	547.0	169.4	154.9	154.9	149.4
31	5	5	2	2	1	*Δ%*	0.0%	0.0%	0.1%	4.1%	3.7%	266.1%	13.4%	3.7%	3.7%	
						opt	10	10	9	4	4	0	2	4	0	
						ave	245.2	245.2	245.2	256.5	256.2	671.8	266.5	256.2	256.2	245.2
32	5	5	2	2	2	*Δ%*	0.0%	0.0%	0.0%	4.6%	4.5%	174.0%	8.7%	4.5%	4.5%	
						opt	10	10	10	6	7	0	3	7	0	
						Δ%_ave	***0.88%***	***1.00%***	***1.39%***	***7.05%***	***3.55%***	***147.87%***	***9.43%***	***4.58%***	***3.59%***	
						opt_ave	***8.2***	***8.1***	***7.9***	***4.1***	***5.6***	***0.0***	***3.2***	***5.0***	***0.0***	
						wins	***10***	***16***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	

Table 4.19. Metaheuristics comparison - Class II, 30 jobs.

Sc	*J*	K_j	N_{ik}	T_{ri}	U_{ik}		$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	GAR	AIE	GAK	SAA	SEGA
1	3	3	1	1	1	*AVE*	366.2	357.6	376.2	488.9	389.9	455.3	405.2	380.6
						Δ%	2.4%	0.0%	5.2%	36.7%	9.0%	27.3%	13.3%	6.4%
2	3	3	1	1	2	*AVE*	813.3	803.5	811.2	1013.6	819.6	891	823.1	812.2
						Δ%	1.2%	0.0%	1.0%	26.1%	2.0%	10.9%	2.4%	1.1%
3	3	3	1	2	1	*AVE*	371.7	396.5	398.9	552.7	388.4	482.3	431.3	404.3
						Δ%	0.0%	6.7%	7.3%	48.7%	4.5%	29.8%	16.0%	8.8%
4	3	3	1	2	2	*AVE*	790.2	786.7	793.9	968	806.6	854.8	818.2	797.1
						Δ%	0.4%	0.0%	0.9%	23.0%	2.5%	8.7%	4.0%	1.3%
5	3	3	2	1	1	*AVE*	239.1	245.7	253.6	364.6	270.8	398	278.9	261
						Δ%	0.0%	2.8%	6.1%	52.5%	13.3%	66.5%	16.6%	9.2%
6	3	3	2	1	1	*AVE*	496.1	486.7	526.8	735.1	555.6	782.8	506.6	537.2
						Δ%	1.9%	0.0%	8.2%	51.0%	14.2%	60.8%	4.1%	10.4%
7	3	3	2	2	1	*AVE*	234.7	245.4	239.8	358.1	254.5	431.6	255.7	252.6
						Δ%	0.0%	4.6%	2.2%	52.6%	8.4%	83.9%	8.9%	7.6%
8	3	3	2	2	2	*AVE*	488.5	513	526.9	743.9	540.6	772.9	507.2	545.6
						Δ%	0.0%	5.0%	7.9%	52.3%	10.7%	58.2%	3.8%	11.7%
9	3	5	1	1	1	*AVE*	193.8	196.3	197.3	228.9	198	271.5	215.4	199.7
						Δ%	0.0%	1.3%	1.8%	18.1%	2.2%	40.1%	11.1%	3.0%
10	3	5	1	1	2	*AVE*	205.8	210.3	213	247.8	217.7	244.4	229.7	215.1
						Δ%	0.0%	2.2%	3.5%	20.4%	5.8%	18.8%	11.6%	4.5%
11	3	5	1	2	1	*AVE*	187.4	184	186.5	225.8	193.5	270.4	203.9	189.5
						Δ%	1.8%	0.0%	1.4%	22.7%	5.2%	47.0%	10.8%	3.0%
12	3	5	1	2	2	*AVE*	195	198.5	200.9	241	208.5	240.9	221.5	202.6
						Δ%	0.0%	1.8%	3.0%	23.6%	6.9%	23.5%	13.6%	3.9%
13	3	5	2	1	1	*AVE*	133.6	134.1	135	155.7	137.4	229.5	145.5	137.2
						Δ%	0.0%	0.4%	1.0%	16.5%	2.8%	71.8%	8.9%	2.7%
14	3	5	2	1	2	*AVE*	151.7	153	155.8	184	165.7	179	172.7	159.6
						Δ%	0.0%	0.9%	2.7%	21.3%	9.2%	18.0%	13.8%	5.2%
15	3	5	2	2	1	*AVE*	125.5	125.2	126.4	160	138.3	260.6	138.3	127.9
						Δ%	0.2%	0.0%	1.0%	27.8%	10.5%	108.1%	10.5%	2.2%
16	3	5	2	2	2	*AVE*	158.8	160.3	169.2	204.2	176.6	199.5	183.6	174.8
						Δ%	0.0%	0.9%	6.5%	28.6%	11.2%	25.6%	15.6%	10.1%
17	5	3	1	1	1	*AVE*	548.3	552.3	550.2	636.5	579.3	563.8	588.5	552.3
						Δ%	0.0%	0.7%	0.3%	16.1%	5.7%	2.8%	7.3%	0.7%
18	5	3	1	1	2	*AVE*	941.1	923.8	919.8	1123.1	935.3	932.8	963.8	923.1
						Δ%	2.3%	0.4%	0.0%	22.1%	1.7%	1.4%	4.8%	0.4%
19	5	3	1	2	1	*AVE*	573.3	571.4	566.5	738	592.3	649.4	627.4	569.9
						Δ%	1.2%	0.9%	0.0%	30.3%	4.6%	14.6%	10.8%	0.6%
20	5	3	1	2	2	*AVE*	882.8	890.1	886.4	1077.5	855.5	881.3	897.3	892.7
						Δ%	3.2%	4.0%	3.6%	25.9%	0.0%	3.0%	4.9%	4.3%
21	5	3	2	1	1	*AVE*	402.8	410.1	421.1	528.8	440.2	423.1	446.6	428.1
						Δ%	0.0%	1.8%	4.5%	31.3%	9.3%	5.0%	10.9%	6.3%
22	5	3	2	1	2	*AVE*	655.4	663.7	669.6	883.9	658.2	675.8	678.7	679.6
						Δ%	0.0%	1.3%	2.2%	34.9%	0.4%	3.1%	3.6%	3.7%
23	5	3	2	2	1	*AVE*	438.7	434.1	450.1	562.5	469.3	455.1	465.1	458.5
						Δ%	1.1%	0.0%	3.7%	29.6%	8.1%	4.8%	7.1%	5.6%
24	5	3	2	2	2	*AVE*	647.8	660.1	661	935.7	647.4	675.2	732.8	676.6
						Δ%	0.1%	2.0%	2.1%	44.5%	0.0%	4.3%	13.2%	4.5%
25	5	5	1	1	1	*AVE*	240.4	241.2	247.4	277.3	241.2	301	257.6	248.7
						Δ%	0.0%	0.3%	2.9%	15.3%	0.3%	25.2%	7.2%	3.5%
26	5	5	1	1	2	*AVE*	305.9	303.4	305.2	372	312	442.3	331.5	306.8
						Δ%	0.8%	0.0%	0.6%	22.6%	2.8%	45.8%	9.3%	1.1%
27	5	5	1	2	1	*AVE*	223.7	228.9	229.9	253.1	229.3	284	248.8	231.4
						Δ%	0.0%	2.3%	2.8%	13.1%	2.5%	27.0%	11.2%	3.4%
28	5	5	1	2	2	*AVE*	343.2	341.2	344.2	411.0	349.5	491.4	376	347.3
						Δ%	0.6%	0.0%	0.9%	20.5%	2.4%	44.0%	10.2%	1.8%
29	5	5	2	1	1	*AVE*	179.9	177	183	224.4	189.5	243	189.1	184.5
						Δ%	1.6%	0.0%	3.4%	26.8%	7.1%	37.3%	6.8%	4.2%
30	5	5	2	1	2	*AVE*	203	208.8	211	236.7	207.8	337.1	216.6	214.7
						Δ%	0.0%	2.9%	3.9%	16.6%	2.4%	66.1%	6.7%	5.8%
31	5	5	2	2	1	*AVE*	179.4	179.3	182.1	215.6	188.4	245.1	202.6	185.3
						Δ%	0.1%	0.0%	1.6%	20.2%	5.1%	36.7%	13.0%	3.3%
32	5	5	2	2	2	*AVE*	231	232.3	233.6	271.9	221	422	228.6	232.9
						Δ%	4.5%	5.1%	5.7%	23.0%	0.0%	91.0%	3.4%	5.4%
						Δ%_ave	***0.74%***	***1.51%***	***3.06%***	***28.59%***	***5.33%***	***34.72%***	***9.24%***	***4.55%***
						wins	***16***	***11***	***2***	***0***	***3***	***0***	***0***	***0***

Table 4.20. Metaheuristics comparison - Class III, 50 jobs.

Sc	*J*	K_j	N_{jk}	$T_{(j-1)j}$	U_{jk}		$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	GAR	AIE	RKGA	SAA	SEGA
1	3	3	1	1	1	*AVE*	601.6	631.1	643	790.2	641.6	754.5	697.2	650.7
						Δ%	0.0%	4.9%	6.9%	31.3%	6.6%	25.4%	15.9%	8.2%
2	3	3	1	1	2	*AVE*	1369.9	1373.7	1377.9	1626.6	1421.2	1386.6	1430.4	1374.9
						Δ%	0.0%	0.3%	0.6%	18.7%	3.7%	1.2%	4.4%	0.4%
3	3	3	1	2	1	*AVE*	674.3	669.4	704.4	845.8	670.3	810.8	735	712.9
						Δ%	0.7%	0.0%	5.2%	26.4%	0.1%	21.1%	9.8%	6.5%
4	3	3	1	2	2	*AVE*	1370.5	1379.2	1374.1	1588.3	1359.5	1373.5	1400.1	1378.4
						Δ%	0.8%	1.4%	1.1%	16.8%	0.0%	1.0%	3.0%	1.4%
5	3	3	2	1	1	*AVE*	470.1	473.4	488.1	736.5	562.5	741.4	522.6	511.6
						Δ%	0.0%	0.7%	3.8%	56.7%	19.7%	57.7%	11.2%	8.8%
6	3	3	2	1	1	*AVE*	1087.5	1069.2	1091.9	1387.5	1197.4	1331.3	1193.2	1107.4
						Δ%	1.7%	0.0%	2.1%	29.8%	12.0%	24.5%	11.6%	3.6%
7	3	3	2	2	1	*AVE*	424	448.6	456.2	616.8	421.2	693.9	509.4	485.8
						Δ%	0.7%	6.5%	8.3%	46.4%	0.0%	64.7%	20.9%	15.3%
8	3	3	2	2	2	*AVE*	1124.9	1110.1	1149.2	1448.2	1221.1	1396.3	1200.4	1173.1
						Δ%	1.3%	0.0%	3.5%	30.5%	10.0%	25.8%	8.1%	5.7%
9	3	5	1	1	1	*AVE*	320	319	320.4	363.3	314.9	463	339.3	328.3
						Δ%	1.6%	1.3%	1.7%	15.4%	0.0%	47.0%	7.7%	4.3%
10	3	5	1	1	2	*AVE*	324.4	318.9	322.2	383.9	326.4	399.4	341.3	330.9
						Δ%	1.7%	0.0%	1.0%	20.4%	2.4%	25.2%	7.0%	3.8%
11	3	5	1	2	1	*AVE*	304	300.4	307.1	362.1	312.8	444.3	345.6	312
						Δ%	1.2%	0.0%	2.2%	20.5%	4.1%	47.9%	15.0%	3.9%
12	3	5	1	2	2	*AVE*	322.3	317.6	328	365.3	334.7	382.9	359.4	331
						Δ%	1.5%	0.0%	3.3%	15.0%	5.4%	20.6%	13.2%	4.2%
13	3	5	2	1	1	*AVE*	186.3	182.1	190.5	241.9	199.8	398	224.4	197.7
						Δ%	2.3%	0.0%	4.6%	32.8%	9.7%	118.6%	23.2%	8.6%
14	3	5	2	1	2	*AVE*	245.6	244.3	247.6	301.9	248.5	298.6	279.1	259.1
						Δ%	0.5%	0.0%	1.4%	23.6%	1.7%	22.2%	14.2%	6.1%
15	3	5	2	2	1	*AVE*	189.1	186.6	196.3	252.8	202	437.1	227.5	198
						Δ%	1.3%	0.0%	5.2%	35.5%	8.3%	134.2%	21.9%	6.1%
16	3	5	2	2	2	*AVE*	244.6	257.5	264.4	329.2	253.4	311.3	299.1	277.1
						Δ%	0.0%	5.3%	8.1%	34.6%	3.6%	27.3%	22.3%	13.3%
17	5	3	1	1	1	*AVE*	867.6	865.3	871.6	1082.5	928.3	1363.3	976.4	873.3
						Δ%	0.3%	0.0%	0.7%	25.1%	7.3%	57.6%	12.8%	0.9%
18	5	3	1	1	2	*AVE*	1474.6	1456.8	1469.3	1939.5	1629.7	1463.7	1579.7	1481.6
						Δ%	1.2%	0.0%	0.9%	33.1%	11.9%	0.5%	8.4%	1.7%
19	5	3	1	2	1	*AVE*	905.2	894.8	896.1	1287.401	939.7	1341.2	1076.2	895.6
						Δ%	1.2%	0.0%	0.1%	43.9%	5.0%	49.9%	20.3%	0.1%
20	5	3	1	2	2	*AVE*	1434.4	1437.3	1437.5	1819.6	1584.2	1436.7	1509.2	1460.4
						Δ%	0.0%	0.2%	0.2%	26.9%	10.4%	0.2%	5.2%	1.8%
21	5	3	2	1	1	*AVE*	621.2	615.7	638.7	804.8	699.3	896.5	705.8	652.6
						Δ%	0.9%	0.0%	3.7%	30.7%	13.6%	45.6%	14.6%	6.0%
22	5	3	2	1	2	*AVE*	1100.1	1109.3	1097	1507.6	1210.4	1073.8	1227.9	1117.3
						Δ%	2.4%	3.3%	2.2%	40.4%	12.7%	0.0%	14.4%	4.1%
23	5	3	2	2	1	*AVE*	735.1	736.4	736.1	892.8	783.9	1177.1	763.2	744.5
						Δ%	0.0%	0.2%	0.1%	21.5%	6.6%	60.1%	3.8%	1.3%
24	5	3	2	2	2	*AVE*	1095.1	1068.7	1075.4	1464.4	1223.3	1050.5	1176.4	1094.4
						Δ%	4.2%	1.7%	2.4%	39.4%	16.4%	0.0%	12.0%	4.2%
25	5	5	1	1	1	*AVE*	350.9	349.6	350.3	386.6	347.7	467	365.6	353.9
						Δ%	0.9%	0.5%	0.7%	11.2%	0.0%	34.3%	5.1%	1.8%
26	5	5	1	1	2	*AVE*	500.2	495.4	499.5	635.4	505.8	749.6	538.1	517.9
						Δ%	1.0%	0.0%	0.8%	28.3%	2.1%	51.3%	8.6%	4.5%
27	5	5	1	2	1	*AVE*	346.4	340.5	339.5	391.5	346.2	462.3	371.5	344.4
						Δ%	2.0%	0.3%	0.0%	15.3%	2.0%	36.2%	9.4%	1.4%
28	5	5	1	2	2	*AVE*	523.7	518.2	525.5	651.7	534.2	781.3	562.3	532.4
						Δ%	1.1%	0.0%	1.4%	25.8%	3.1%	50.8%	8.5%	2.7%
29	5	5	2	1	1	*AVE*	272.8	279.7	279.6	327.6	270.4	368	299.2	289.1
						Δ%	0.9%	3.4%	3.4%	21.2%	0.0%	36.1%	10.7%	6.9%
30	5	5	2	1	2	*AVE*	333	336.6	335.2	426.7	349.4	618.7	378.1	339.9
						Δ%	0.0%	1.1%	0.7%	28.1%	4.9%	85.8%	13.5%	2.1%
31	5	5	2	2	1	*AVE*	280	279.3	288.8	336.9	278.7	381.2	299.7	294.5
						Δ%	0.5%	0.2%	3.6%	20.9%	0.0%	36.8%	7.5%	5.7%
32	5	5	2	2	2	*AVE*	367.7	358.2	362.8	473.6	358.9	708.8	400.1	368.4
						Δ%	2.7%	0.0%	1.3%	32.2%	0.2%	97.9%	11.7%	2.8%
						Δ%_ave	***1.08%***	***0.98%***	***2.54%***	***28.07%***	***5.74%***	***40.86%***	***11.76%***	***4.62%***
						wins	***7***	***16***	***1***	***0***	***6***	***2***	***0***	***0***

Table 4.21. Metaheuristics comparison - Class IV, 70 jobs.

Sc	*J*	K_j	N_{jk}	$T_{(j-1)j}$	U_{jk}		$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	GAR	AIE	RKGA	SAA	SEGA
1	3	3	1	1	1	*AVE*	921.9	925.7	933.6	1221.5	1047.7	1108.9	1090.5	952.4
						Δ%	0.0%	0.4%	1.3%	32.5%	13.6%	20.3%	18.3%	3.3%
2	3	3	1	1	2	*AVE*	1945.2	1958.6	1960.4	2376	2030.8	2029.7	2057.4	1971
						Δ%	0.0%	0.7%	0.8%	22.1%	4.4%	4.3%	5.8%	1.3%
3	3	3	1	2	1	*AVE*	1005.8	1002.3	1015.5	1276.1	1067.8	1115	1084.9	1035.6
						Δ%	0.3%	0.0%	1.3%	27.3%	6.5%	11.2%	8.2%	3.3%
4	3	3	1	2	2	*AVE*	1960	1958.3	1943.8	2195.9	2053.9	1985.5	2023.3	1961
						Δ%	0.8%	0.7%	0.0%	13.0%	5.7%	2.1%	4.1%	0.9%
5	3	3	2	1	1	*AVE*	838.1	824.4	857.2	1200.2	892.9	1067.9	1030.4	885
						Δ%	1.7%	0.0%	4.0%	45.6%	8.3%	29.5%	25.0%	7.4%
6	3	3	2	1	1	*AVE*	1622.5	1632.3	1624.7	2044.5	1785.5	1799.6	1892.9	1655.4
						Δ%	0.0%	0.6%	0.1%	26.0%	10.0%	10.9%	16.7%	2.0%
7	3	3	2	2	1	*AVE*	787.6	788.2	819.3	1028.3	873.9	1112.5	850.4	846.4
						Δ%	0.0%	0.1%	4.0%	30.6%	11.0%	41.3%	8.0%	7.5%
8	3	3	2	2	2	*AVE*	1873.9	1855.4	1842	2140.4	1971.2	1902.7	1914.1	1878.4
						Δ%	1.7%	0.7%	0.0%	16.2%	7.0%	3.3%	3.9%	2.0%
9	3	5	1	1	1	*AVE*	462	457.7	472.4	529.2	437.7	653.6	492.4	482.8
						Δ%	5.6%	4.6%	7.9%	20.9%	0.0%	49.3%	12.5%	10.3%
10	3	5	1	1	2	*AVE*	439.8	440.1	446.3	517.5	450.6	533.1	485.7	451.2
						Δ%	0.0%	0.1%	1.5%	17.7%	2.5%	21.2%	10.4%	2.6%
11	3	5	1	2	1	*AVE*	462.6	453	455.4	524.1	441.1	632.8	493.7	466.2
						Δ%	4.9%	2.7%	3.2%	18.8%	0.0%	43.5%	11.9%	5.7%
12	3	5	1	2	2	*AVE*	457.3	456.5	461.2	512.1	443	532.9	476.8	468.3
						Δ%	3.2%	3.0%	4.1%	15.6%	0.0%	20.3%	7.6%	5.7%
13	3	5	2	1	1	*AVE*	295.2	305.1	314.3	378.8	283.6	604.5	324.7	325
						Δ%	4.1%	7.6%	10.8%	33.6%	0.0%	113.2%	14.5%	14.6%
14	3	5	2	1	2	*AVE*	367.1	355.3	366.3	443	363.1	437.4	413.3	378.1
						Δ%	3.3%	0.0%	3.1%	24.7%	2.2%	23.1%	16.3%	6.4%
15	3	5	2	2	1	*AVE*	300.1	295	303.5	384.8	280.4	626.5	341.7	314.6
						Δ%	7.0%	5.2%	8.2%	37.2%	0.0%	123.4%	21.9%	12.2%
16	3	5	2	2	2	*AVE*	358.5	367.4	380.4	445.5	381.7	469.8	424.6	390.8
						Δ%	0.0%	2.5%	6.1%	24.3%	6.5%	31.0%	18.4%	9.0%
17	5	3	1	1	1	*AVE*	1276.9	1273.9	1351.8	1786.7	1271.1	2063.5	1537.4	1357.2
						Δ%	0.5%	0.2%	6.3%	40.6%	0.0%	62.3%	21.0%	6.8%
18	5	3	1	1	2	*AVE*	2056.6	2049.7	2034.7	2729.8**(8)**	2315.2	1992	2479.3**(9)**	2035.1
						Δ%	3.2%	2.9%	2.1%	37.0%	16.2%	0.0%	24.5%	2.2%
19	5	3	1	2	1	*AVE*	1396.8	1386.7	1340.5	1662.8	1274.5	2224.8	1624.2	1351.3
						Δ%	9.6%	8.8%	5.2%	30.5%	0.0%	74.6%	27.4%	6.0%
20	5	3	1	2	2	*AVE*	2031.2	2019.7	2044.6	2497.6**(9)**	2246	2049.7	2213.0**(9)**	2051.3
						Δ%	0.6%	0.0%	1.2%	23.7%	11.2%	1.5%	9.6%	1.6%
21	5	3	2	1	1	*AVE*	931.3	939.8	942	1187.8	1056.5	1526.4	1032.1	963.5
						Δ%	0.0%	0.9%	1.1%	27.5%	13.4%	63.9%	10.8%	3.5%
22	5	3	2	1	2	*AVE*	1772.7	1754	1779	2512.9**(7)**	2107	1670.6	2035.4**(9)**	1786.9
						Δ%	6.1%	5.0%	6.5%	50.4%	26.1%	0.0%	21.8%	7.0%
23	5	3	2	2	1	*AVE*	1015.8	1010.8	1013.1	1407.7	1100.4	1721.4	1080.3	1040.8
						Δ%	0.5%	0.0%	0.2%	39.3%	8.9%	70.3%	6.9%	3.0%
24	5	3	2	2	2	*AVE*	1639.8	1645	1628.1	2229.4	1920.4	1573.1	1900.8	1648.8
						Δ%	4.2%	4.6%	3.5%	41.7%	22.1%	0.0%	20.8%	4.8%
25	5	5	1	1	1	*AVE*	460.1	457.2	458.9	515.8	462.5	704.9	485.3	467.2
						Δ%	0.6%	0.0%	0.4%	12.8%	1.2%	54.2%	6.1%	2.2%
26	5	5	1	1	2	*AVE*	735.1	752	730.8	953.3	763.7	1085.8	829.3	758.7
						Δ%	0.6%	2.9%	0.0%	30.4%	4.5%	48.6%	13.5%	3.8%
27	5	5	1	2	1	*AVE*	446.2	444.1	447.2	501.5	444.9	678.1	465.9	453.5
						Δ%	0.5%	0.0%	0.7%	12.9%	0.2%	52.7%	4.9%	2.1%
28	5	5	1	2	2	*AVE*	728.5	733.1	730.1	911	782.2	1074.2	826.8	749.4
						Δ%	0.0%	0.6%	0.2%	25.1%	7.4%	47.5%	13.5%	2.9%
29	5	5	2	1	1	*AVE*	401.2	396.5	406.7	445.5	386.4	615	427.3	417.8
						Δ%	3.8%	2.6%	5.3%	15.3%	0.0%	59.2%	10.6%	8.1%
30	5	5	2	1	2	*AVE*	527.3	524.3	525.7	692.2	568.5	979.7	607.1	537.3
						Δ%	0.6%	0.0%	0.3%	32.0%	8.4%	86.9%	15.8%	2.5%
31	5	5	2	2	1	*AVE*	402.6	403.6	408.7	463.2	391	677.6	440.5	413.3
						Δ%	3.0%	3.2%	4.5%	18.5%	0.0%	73.3%	12.7%	5.7%
32	5	5	2	2	2	*AVE*	507.5	507.1	513.1	751.9	557.5	1032.9	623.7	520.2
						Δ%	0.1%	0.0%	1.2%	48.3%	9.9%	103.7%	23.0%	2.6%
						Δ%_ave	***2.08%***	***1.90%***	***2.98%***	***27.87%(*)***	***6.48%***	***42.08%***	***13.95%(*)***	***4.96%***
						wins	***8***	***9***	***3***	***0***	***9***	***3***	***0***	***0***

Table 4.22. Metaheuristics comparison - Class V, 100 jobs.

Sc	*J*	K_j	N_{jk}	$T_{(j-1)j}$	U_{jk}		$DEME_{70}$	$DEME_{80}$	$DEME_{90}$	GAR	AIE	RKGA	SAA	SEGA
1	3	3	1	1	1	*AVE*	1442.2	1444.7	1442.8	1891.8	1599.9	1744.4	1593.9	1476.1
						Δ%	0.0%	0.2%	0.0%	31.2%	10.9%	21.0%	10.5%	2.4%
2	3	3	1	1	2	*AVE*	3285.5	3285.4	3290.5	3357.0**(4)**	3139.9	3060.5	3305.8**(8)**	3293.2
						Δ%	7.4%	7.3%	7.5%	9.7%	2.6%	0.0%	8.0%	7.6%
3	3	3	1	2	1	*AVE*	1445.4	1457.9	1486.1	1871.4	1622.7	1719.6	1684.2	1519.6
						Δ%	0.0%	0.9%	2.8%	29.5%	12.3%	19.0%	16.5%	5.1%
4	3	3	1	2	2	*AVE*	2750.3	2760.3	2746.2	3473.2	2956.7	2809.9	3136.1	2756.7
						Δ%	0.1%	0.5%	0.0%	26.5%	7.7%	2.3%	14.2%	0.4%
5	3	3	2	1	1	*AVE*	1266.8	1261.5	1256.8	1676.9	1437.4	1614.3	1574.7	1329.0
						Δ%	0.8%	0.4%	0.0%	33.4%	14.4%	28.4%	25.3%	5.7%
6	3	3	2	1	1	*AVE*	2725.0	2723.7	2740.2	3159.1**(9)**	2933.4	2801.6	2979.6**(9)**	2738.2
						Δ%	0.0%	0.0%	0.6%	16.6%	7.7%	2.9%	9.4%	0.5%
7	3	3	2	2	1	*AVE*	1219.8	1210.2	1253.7	1616.5	1407.4	1651.2	1375.8	1292.1
						Δ%	0.8%	0.0%	3.6%	33.6%	16.3%	36.4%	13.7%	6.8%
8	3	3	2	2	2	*AVE*	2764.8	2774.8	2770.7	3271.3	2997.5	2861.8	3148.8	2802.8
						Δ%	0.0%	0.4%	0.2%	18.3%	8.4%	3.5%	13.9%	1.4%
9	3	5	1	1	1	*AVE*	666.4	671.1	684.5	765.4	702.7	949.1	721.5	694.1
						Δ%	0.0%	0.7%	2.7%	14.9%	5.4%	42.4%	8.3%	4.2%
10	3	5	1	1	2	*AVE*	629.8	635.3	641.6	705.4	660.5	814.0	681.0	656.2
						Δ%	0.0%	0.9%	1.9%	12.0%	4.9%	29.2%	8.1%	4.2%
11	3	5	1	2	1	*AVE*	660.3	671.3	672.9	741.3	656.0	937.9	715.7	683.7
						Δ%	0.7%	2.3%	2.6%	13.0%	0.0%	43.0%	9.1%	4.2%
12	3	5	1	2	2	*AVE*	627.2	629.1	633.2	709.8	642.2	756.5	661.2	649.6
						Δ%	0.0%	0.3%	1.0%	13.2%	2.4%	20.6%	5.4%	3.6%
13	3	5	2	1	1	*AVE*	489.2	482.7	497.3	626.6	513.9	895.0	533.4	507.8
						Δ%	1.3%	0.0%	3.0%	29.8%	6.5%	85.4%	10.5%	5.2%
14	3	5	2	1	2	*AVE*	524.0	529.3	544.9	640.5	568.4	709.5	591.9	561.2
						Δ%	0.0%	1.0%	4.0%	22.2%	8.5%	35.4%	13.0%	7.1%
15	3	5	2	2	1	*AVE*	488.3	492.1	513.5	621.8	509.4	904.2	544.3	531.0
						Δ%	0.0%	0.8%	5.2%	27.3%	4.3%	85.2%	11.5%	8.7%
16	3	5	2	2	2	*AVE*	565.9	567.1	569.1	656.1	572.9	764.4	610.3	586.1
						Δ%	0.0%	0.2%	0.6%	15.9%	1.2%	35.1%	7.8%	3.6%
17	5	3	1	1	1	*AVE*	2437.6	2316.2	2437.6	2448.3**(6)**	2283.7	3435.0	2531.2**(8)**	2460.1
						Δ%	6.7%	1.4%	6.7%	7.2%	0.0%	50.4%	10.8%	7.7%
18	5	3	1	1	2	*AVE*	3435.2	3441.7	3422.9	3868.0**(1)**	9796.9	3119.1	3923.3**(3)**	3437.9
						Δ%	10.1%	10.3%	9.7%	24%	214.1%	0.0%	25.8%	10.2%
19	5	3	1	2	1	*AVE*	2635.1	2614.5	2608.1	2618.9**(8)**	2309.7	3364.8	2490.1**(9)**	2620.9
						Δ%	14.1%	13.2%	12.9%	13.4%	0.0%	45.7%	7.8%	13.5%
20	5	3	1	2	2	*AVE*	3114.4	3112.5	3098.4	3887.6**(7)**	3455.1	3015.2	3294.1**(8)**	3116.3
						Δ%	3.3%	3.2%	2.8%	28.9%	14.6%	0.0%	9.3%	3.4%
21	5	3	2	1	1	*AVE*	1521.0	1545.2	1534.2	1819.2	1623.9	2744.9	1672.6	1543.9
						Δ%	0.0%	1.6%	0.9%	19.6%	6.8%	80.5%	10.0%	1.5%
22	5	3	2	1	2	*AVE*	3344.2	3359.8	3354.3	***NFS*(0)**	3749.7	2896.1	3387.8**(4)**	3360.6
						Δ%	15.5%	16.0%	15.8%	-	29.5%	0.0%	17%	16.0%
23	5	3	2	2	1	*AVE*	1938.4	1920.4	1914.1	2158.1	1949.0	3170.7	2170.4	1930.9
						Δ%	1.3%	0.3%	0.0%	12.7%	1.8%	65.6%	13.4%	0.9%
24	5	3	2	2	2	*AVE*	2983.3	3003.9	3006.2	3568.0**(7)**	3315.8	2614.9	3427.3**(9)**	3028.1
						Δ%	14.1%	14.9%	15.0%	36.5%	26.8%	0.0%	31.1%	15.8%
25	5	5	1	1	1	*AVE*	671.3	673.9	676.9	743.7	678.2	1194.1	711.2	682.5
						Δ%	0.0%	0.4%	0.8%	10.8%	1.0%	77.9%	5.9%	1.7%
26	5	5	1	1	2	*AVE*	1115.3	1097.5	1101.2	1443.7	1255.6	1627.1	1312	1112.8
						Δ%	1.6%	0.0%	0.3%	31.5%	14.4%	48.3%	19.5%	1.4%
27	5	5	1	2	1	*AVE*	630.3	629.7	634.6	687.9	644.3	1217.9	667.2	642.4
						Δ%	0.1%	0.0%	0.8%	9.2%	2.3%	93.4%	6.0%	2.0%
28	5	5	1	2	2	*AVE*	1108.5	1102.6	1107.7	1407.7	1243.4	1607.4	1336.0	1112.8
						Δ%	0.5%	0.0%	0.5%	27.7%	12.8%	45.8%	21.2%	0.9%
29	5	5	2	1	1	*AVE*	556.7	551.4	558.3	627.3	569.2	946.2	593.7	569.8
						Δ%	1.0%	0.0%	1.3%	13.8%	3.2%	71.6%	7.7%	3.3%
30	5	5	2	1	2	*AVE*	777.0	775.9	778.3	1094.6	953.3	1553.8	971.4	802.1
						Δ%	0.1%	0.0%	0.3%	41.1%	22.9%	100.3%	25.2%	3.4%
31	5	5	2	2	1	*AVE*	568.4	569.1	577.3	628.7	560.4	1044.6	594.0	587.7
						Δ%	1.4%	1.6%	3.0%	12.2%	0.0%	86.4%	6.0%	4.9%
32	5	5	2	2	2	*AVE*	778.0	775.8	782.2	1231.4	1010.6	1527.4	1129.5	801.0
						Δ%	0.3%	0.0%	0.8%	58.7%	30.3%	96.9%	45.6%	3.2%
						Δ%_ave	***2.54%***	***2.46%***	***3.35%***	***21.68%(*)***	***15.43%***	***42.27%***	***13.98%(*)***	***5.02%***
						wins	***12***	***9***	***2***	***0***	***4***	***5***	***0***	***0***

Conclusions

The aim of the present Thesis was to investigate the use of exact and heuristic approaches for solving three novel scheduling problems arising from real-world manufacturing processes. Each problem has been firstly formulated by means of an MILP model, so to optimally solve small-instances within a reasonable time through the use of commercial solvers. The, in order to effectively manage larger instances of the problems addressed, heuristic optimization algorithms, mainly based on GAs have been adopted.

With reference to the unrelated parallel machine scheduling problem with limited and multi-skilled human resources, three different GA-based metaheuristic procedures 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 optimization 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-sized 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. Results obtained have shown the superiority of performance of the algorithm called HGA_{25}, which starts with the single-stage encoding and moves to the multi-stage encoding after 25% of total makespan evaluations.

After the best optimization procedure was selected, obtained results have been compared with those deriving from production scenarios in which workforce is not assumed to be multi-skilled. It has been shown that the differences of technical abilities among workers may yield better results than the case in which operators are all identically featured by an average skill level.

As far as the flow shop sequence dependent group scheduling problem with skilled workforce is concerned, a new metaheuristic based on a genetic algorithm, namely GAI, has been developed to properly address both the sequencing and the worker allocation problem under an integrated viewpoint. Such procedure has been compared with two distinct genetic

algorithms, each one exploiting a different way to manage the worker allocation issue. After the numerical results confirmed the efficacy of the provided GAI, this optimization strategy has been assessed against the best available metaheuristic in literature for the FDSGS problem, i.e., the HPSO proposed by Hajinejad, Salmasi and Mokhtaria (2011). A multiple analysis confirmed the superiority of the proposed genetic approach for tackling the problem in hand.

A subsequent study performed on the numerical results achieved by the GAI highlighted as the higher is the problem size (in terms of number of machines) and the higher is impact of the worker skill under the makespan reduction viewpoint. Of course, a makespan reduction may be determined by an increase of the average worker ability of a given workforce team but, at the same time, that skill improvement surely would lead to a cost investment in terms of training or manpower replacement. As a consequence of the aforementioned remarks, a set of tables has been developed in this paper whose ultimate objective is providing specific guidelines for decision makers in case a trade-off between makespan reduction and manpower improvement must be fulfilled.

With regards to the constrained hybrid flow shop scheduling problem, a GA-based metaheuristic combining two different problem encodings for improving both exploration and exploitation within the overall discrete space of solutions has been proposed. The first phase of the proposed dual encoding metaheuristics, called DEME, uses a simple permutation encoding while, from a provided threshold on, an encoding switch is performed and a specific local search equipped with a multiple stage encoding is launched in order to investigate solutions that the previous encoding is not able to evaluate. The dual encoding metaheuristic algorithms featured by different values of threshold have been compared with a regular genetic algorithm-based metaheuristics named SEGA equipped with a regular permutation encoding. An extensive experimental analysis has been carried out for demonstrating the superiority of the proposed optimization approach. Then, in order to emphasize the effectiveness of the proposed approach, an extensive comparison with other metaheuristics arisen from literature has been carried out.

Further research perspectives should include the study of the aforementioned problems under performance objectives different from the makespan (i.e., total weighted completion time, total tardiness, etc.). Bi-objective optimization analysis could also be carried out through the use of theoretical framework such as Pareto fronts. With respect to the GA-based algorithms

adopted, their effectiveness could be tested with reference to well-known problems already addressed in literature, in order to explore their performances against the best procedures available. Of course the two-step approach here adopted (i.e., the MILP formulation followed by a properly developed metaheuristic procedure able to cope with larger-sized instances) could be replicate to investigate other scheduling problems arising from real-world manufacturing environments.

References

1. Adler L, Fraiman N, Kobacher E, Pinedo M, Plotnicoff J, Wu TP. BSS: a scheduling support system for the packaging industry. *Operations Research* 1993; **41**(4): 641–648.
2. Allahverdi A, Gupta JND, Aldowaisian T. A review of scheduling research involving setup considerations. *OMEGA, The International Journal of Management and Science* 1999; **27**: 219-239.
3. Anghinolfi D, Paolucci M. Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. *Computers & Operations Research* 2007; **34**: 3471-3490.
4. Aghezzaf E, Artiba A, Moursli O, Tahon C. Hybrid flowshop problems, a decomposition based heuristic approach. In: *Proceedings of the International Conference of Industrial Engineering and Production Management*, *IEPM'95*. Marrakech: FUCAM-INRIA; 1985.
5. Allaoui H, Artiba A. Integrating simulation and optimization to schedule a hybrid flow shop with maintenance constraints. *Computers & Industrial Engineering* 2004; **47**: 431–450.
6. Arthanary L, Ramaswamy K. An extension of two machine sequencing problem. *OPSEARCH. Journal of the Operational Research Society of India* 1971; **8**(4): 10–22.
7. Askin RG, Huang RY. Forming effective work teams for cellular manufacturing. *International Journal of Production Research* 2001; **39**(11): 2431-2451.
8. Balin S. Non-identical parallel machine scheduling using genetic algorithm. *Expert Systems with Applications* 2011; **38**: 6814-6821.
9. Baker KR, Trietsch D. *Principles of Sequencing and Scheduling* Hoboken: John-Wiley & Sons; 2007.
10. Blazewicz J, Ecker KH, Pesch E, Schmidt G, Weglarz J. *Scheduling in computer and manufacturing systems*. Second edition. Berlin: Springer; 1992
11. Brah SA, Loo L. Heuristic for scheduling in a flow shop with multiple processors. *European Journal of Operational Research* 1999; **113**: 113–122.
12. Campbell HG, Dudek RA, Smith ML. A heuristic algorithm for the *n* job *m* machine sequencing problem. *Management Science* 1970; **16**: B630-B637
13. Celano G, Costa A, Fichera S. Constrained scheduling of the inspection activities on semiconductor wafers grouped in families with sequence-dependent set-up times. *The international Journal of Advanced Manufacturing Technology* 2010; **46**(5-8): 695-705.

14. Celano G, Costa A, Fichera S. Scheduling of unrelated parallel manufacturing cells with limited human resources. *International Journal of Production Research* 2008; **46**(2): 405-427.
15. Celano G, Costa A, Fichera S, Perrone G. Human factor policy testing in the sequencing of manual mixed model assembly lines. *Computers & Operations Research* 2004; **31**(1): 39-59.
16. Cheng TCE, Gupta JND, Wang G. A review of flowshop scheduling research with setup times. *Production and Operations Management* 2000; **9**(3): 262-282.
17. Choi HS, Kim JS, Lee DH. Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line. *Expert Systems with Applications* 2011; **38:** 3514–3521.
18. Corominas A, Olivella J and Pastor R. A model for the assignment a set of tasks when work performance depends on experience of all tasks involved. *International Journal of Production Economics* 2010; **126**: 335-340.
19. Costa A, Celano G, Fichera S, Trovato E. A new efficient encoding/decoding procedure for the design of a supply chain network with genetic algorithms. *Computers & Industrial Engineering* 2010; **59**(4): 986-999.
20. Chaudhry IA. Minimizing flow time for the worker assignment problem in identical parallel machine models using GA. *International Journal of Advanced Manufacturing Technology* 2010; **48**: 747-760.
21. Chaudhry IA, Drake PR. Minimizing total tardiness for the machine scheduling and worker assignment problems in identical parallel machines using genetic algorithms. *International Journal of Advanced Manufacturing Technology* 2009; **42**: 581-594.
22. Cheng R, Gen M. Parallel machine scheduling problems using memetic algorithms. *Computers & Industrial Engineering* 1997; **33**(3-4): 761-764.
23. Cochran JK, Horng SM, Fowler JW. A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers & Operations Research* 2003; **30**: 1087-1102.
24. ElMaraghy H, Patel V, Ben Abdallah I. Scheduling of manufacturing systems under dual-resource constraints using genetic algorithms. *Journal of Manufacturing Systems* 2000: **19**(3): 186-198.
25. Engin O, Döyen A. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems* 2004; **20**: 1083–1095.
26. Fanjul-Peyro L, Ruiz R. Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers & Operations Research* 2011; **38**: 301-309.

27. Fanjul-Peyro L, Ruiz R. Scheduling unrelated parallel machines with optional machines and jobs selection. *Computers & Operations Research* 2012; **39**: 1745-1753.
28. Fitzpatrick E, Askin L, Ronald G. Forming effective worker teams with multi-functional skill requirements. *Computers & Industrial Engineering* 2005; **48:** 593-608.
29. França PM, Gupta JND, Mendes AS, Moscato P, Veltink KJ. Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups. *Computers & Industrial Engineering* 2005; **48**(3): 491–506.
30. Garey MR, Johnson DS. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. New York: Freeman & Co.; 1979.
31. Gendreau M, Potvin JY, *Handbook of Metaheuristics*. Second edition. New York: Springer; 2010.
32. Gholami M, Zandieh M, Alem-Tabriz A. Scheduling hybrid shop with sequence-dependent setup times and machines with random breakdowns. *The International Journal of Advanced Manufacturing Technology* 2009; **42**: 189–201.
33. Glover F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* 1986; **13**(5): 533-549.
34. Gokhale R, Mathirajan M. Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flowtime in automobile gear manufacturing. *International Journal of Advanced Manufacturing Technology* 2012; **60**: 1099-1110.
35. Gourgand M, Grangeon N, Norre S. Metaheuristics for the deterministic hybrid flow shop problem. In: *Proceedings of the international conference on industrial engineering and production management*, *IEPM'99*. Glasgow: FUCAM-INRIA; 1999.
36. Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* 1979; **5**: 287-326.
37. Gupta, JND. Two-stage, hybrid flow shop scheduling problem. *Journal of the Operational Research Society* 1988; **39**(4): 359–364.
38. Hajinejad D, Salmasi B, Mokhtari R. A fast hybrid particle swarm optimization algorithm for flow shop sequence dependent group scheduling problem. *Scientia Iranica* 2011; **18**(3): 759–764.
39. Hendizadeh H, Faramarzi H, Mansouri SA, Gupta JND, Elmekkawy TY. Meta-heuristics for scheduling a flowshop manufacturing cell with sequence dependent family setup times. *International Journal of Production Economics* 2008; **111**: 593–605.

40. Holland JH. *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press; 1975.
41. Hottenstein MP, Bowman SA. *Cross-training and worker flexibility: a review of DRC system research*. The Journal of High Technology Management Research 1998; **9**(2): 157-174.
42. Hu PC. Minimising total tardiness for the worker assignment scheduling problem in identical parallel-machine models. *International Journal of Advanced Manufacturing Technology* 2004; **23**: 383-388.
43. Hu PC. Minimizing total flow time for the worker assignment scheduling problem in the identical parallel-machines models. *International Journal of Advanced Manufacturing Technology* 2005; **25**: 1046-1052.
44. Hu PC. Further study of minimizing total tardiness for the worker assignment scheduling problem in the identical parallel-machines models. *International Journal of Advanced Manufacturing Technology* 2006; **29**: 165-169.
45. Hunter JE. Cognitive, ability, cognitive aptitudes, job knowledge, and job performance. *Journal of Vocational. Behavior* 1986; **29:** 340-362.
46. Jarboui P, Siarry P, Teghem J. *Metaheuristics for Production Scheduling*. London: ISTE; Hoboken: John-Wiley & Sons; 2013.
47. Jungwattanakit J, Reodecha M, Chaovalitwongse P, Werner F. Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. *The International Journal of Advanced Manufacturing Technology* 2008; **37**: 354–370.
48. Kaharaman C, Engin O, Kaya I, Elif Öztürk R. Multiprocessor task scheduling in multistage hybrid flow-shops: A parallel greedy algorithm approach. *Applied Soft Computing* 2010; **10**: 1293–1300.
49. Karmakar N. A new polynomial time algorithm for linear programming. *Combinatorica* 1984; **4**(4): 373-395.
50. Kim SC, Bobrowsky PM. Scheduling jobs with uncertain setup times and sequence dependency. *Omega, the International Journal of Management and Science* 1997; **25**(4): 437-447.
51. Kim DW, Na DG, Chen FF. Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer Integrated Manufacturing* 2003; **19**: 173-181.
52. Kher HV, Fry TD. Labour flexibility and assignment policies in a job shop having incommensurable objectives. *International Journal of Production Research* 2001; **39**(11): 2295–2311.

53. Kirkpatrick S, Gelatt CDJ, Vecchi MP. Optimization by simulated annealing. *Science* 1983; **220**(4598): 671-680.
54. Kurz ME, Askin RG. Scheduling flexible flow lines with sequence dependent setup times. *European Journal of Operational Research* 2004; **159**: 66–82.
55. Linn R, Zhang W. Hybrid flow shop scheduling: A survey. *Computers & Industrial Engineering* 1999; **37**(1-2): 57–61.
56. Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1977; **1**: 342-362.
57. Leung JYT. *Handbook of Scheduling*. Boca Raton: CRC Press; 2004.
58. Lodree EJJ, Christofer D, Geiger W, Xiaochun J. Taxonomy for integrating scheduling theory and human factors: Review and research opportunities. *International Journal of Industrial Erogonomics* 2009; **39**: 39-51.
59. Lopez P, Roubellat F. *Production Scheduling*. London: ISTE; Hoboken: John-Wiley & Sons; 2008.
60. McDonald T, Ellis KP, Van Aken EM, Koelling CP. Development and application of a worker assignment model to evaluate a lean manufacturing cell. *International Journal of Production Research* 2009; **47**: 2427-2447.
61. Michalewicz Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Second edition. Berlin: Springer-Verlag; 1994.
62. Mirsanei HS, Zandieh M, Moayed MJ, Khabbazi MR. A simulated annealing approach to hybrid flow shop scheduling with sequence-dependent setup times. *Journal of Intelligent Manufacturing* 2011; **22**: 965–978.
63. Montgomery D. *Design and Analysis of Experiments*. Fifth edition. New York: John-Wiley & Sons; 2007.
64. Naderi B, Salmasi N. Permutation flowshops in group scheduling with sequence-dependent setup times. *European Journal of Industrial Engineering* 2012; **6**: 177-198.
65. Nawaz M, Enscore EEJ, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management and Science* 1983; **11**(1): 91–105.
66. Nelson RT. Labor and machine limited production systems. *Management Science* 1967; **13**(9): 648-671.
67. Nikolaev AG, Jacobson SH. Simulated annealing. In: Gendreau M, Potvin JY, *Handbook of Metaheuristics*. Second edition. New York: Springer; 2010.

68. Norman BA, Tharmmaphornphilas W, Lascola Needy K, Bidanda B, Colosimo Warner N. Worker assignment in cellular manufacturing considering technical and human skills. *International Journal of Production Research* 2002; **40**(6): 1479-1492.
69. Oguz C, Zinder Y, Van Ha D, Janiak A, Lichtenstein M. Hybrid flowshop scheduling problems with multiprocessor task systems. *European Journal of Operational Research* 2004; **152**: 115–131.
70. Oliver IM, Smith DJ, Holland JRC. A study of permutation crossover operators on the TSP. In: Grefenstette JJ, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference*. Hillsdale: Lawrence Erlbaum; 1987.
71. Palmer DS. Sequencing jobs through a multi-stage process in the minimum total time – A quick method of obtaining a near optimum. *Operational Research Quarterly* 1965; **16**: 101-107.
72. Pan QK, Suganthan PN, Chua TJ, Cai TX. Solving manpower scheduling problem in manufacturing using mixed-integer programming with a two-stage heuristic algorithm. *International Journal of Advanced Manufacturing Technology* 2010; **46**: 1229-1237.
73. Pereira Lopes MJ, Valério de Carvalho JM. A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research* 2007; **176**: 1508-1527.
74. Perron L. Planning and Scheduling teams of skilled workers. *Journal of Intelligent Manufacturing* 2010; **21**: 155-164.
75. Pinedo M. *Scheduling: Theory, Algorithms and Systems*. Fourth edition. New Jersey: Prentice Hall; 2012.
76. Rajendran C, Chaundhuri D. A multi-stage parallel processor flowshop problem with minimum flowtime. *European Journal of Operational Research* 1992; **57**: 111–122.
77. Rathinasamy R, R R. Sequencing and scheduling of nonuniform flow pattern in parallel hybrid flow shop. *The International Journal of Advanced Manufacturing Technology* 2010; **49**: 213–225.
78. Riane F, Ariba A. Scheduling multistage flowshop problem: a brief review In: *Proceedings of the international conference on industrial engineering and production management*, *IEPM'99*. Glasgow: FUCAM-INRIA; 1999
79. Ruiz R, Maroto C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research* 2006; **169:** 781–800.
80. Ruiz R, Serifoglu FS, Urilings T. Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research* 2008; **35**: 1151–1175.

81. Salmasi N, Logendran R. A heuristic approach for multi-stage sequence-dependent group scheduling problems. *Journal of Industrial Engineering International* 2008; **4**(7): 48–58.
82. Salmasi N, Logendran R, Skandari MR. Total flow time minimization in a flowhop sequence-dependent group scheduling problem. *Computers & Operations Research* 2010; **37**: 199-212.
83. Salmasi N, Logendran R, Skandari MR. Makespan minimization of a flowshop sequence-dependent group scheduling problem. *The International Jouornal of Advanced Manufacturing Technology* 2011; **56**: 699-710.
84. Salvador MS. A solution to a special case of flow shop scheduling problems. In ElMaghraby SE, *Symposium of the Theory of Scheduling and Applications*. New York: Springer-Verlag; 1973.
85. Schaller JE, Gupta JND, Vakharia AJ. Scheduling a flowline manufacturing cell with sequence dependent family set-up times. *European Journal of Operational Research* 2000; **125:** 324-339.
86. Singh MR, Mahapatra SS. A swarm optimization approach for flexible flow shop scheduling with multi-processor tasks. *The International Journal of Advanced Manufacturing Technology* 2012; **62:** 267–277.
87. Sivrikaya Serifoglu F, Ulusoy G. Multiprocessor task scheduling in multistage hybrid flow-shops: A genetic algorithm approach. *Journal of the Operational Research Society* 2004; **55**: 504–513.
88. Sule DR. *Production Planning and Industrial Scheduling: Examples, Case Studies and Applications*. Second edition. Boca Raton: CRC Press; 2008.
89. Syswerda G. Uniform crossover in genetic algorithms. In: Schaffer JD, *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo: Morgan Kaufmann Publishers; 1989.
90. Syswerda G. Schedule optimization using genetic algorithms. In: Davis L, *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold; 1991.
91. Talbi EG. *Metaheuristics. From Design to Implementation.* Hoboken: John-Wiley & Sons; 2009.
92. Tavakkoli-Moghddam R, Sfaei N, Sassoni F. A memetic algorithm for the flexible flow line scheduling problem with processor blocking. *Computers & Operations Research* 2009; **36**: 402–414.
93. Tavakkoli-Moghaddam R, Taheri F, Bazzazi M, Izadi M, Sassani F. Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup times and precedence constraints. *Computers & Operations Research* 2009; **36**: 3224-3230.

94. Treleven M. A review of the dual resource constrained system research. *IIE Transactions* 1989; **21**(3): 279-287.
95. Tseng CT, Liao CJ. A particle swarm optimization for hybrid flow-shop scheduling with multiprocessor tasks. *International Journal of Production Research* 2008; **46**(17(1)): 4655–4670.
96. Vallada E, Ruiz R. *A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times*. European Journal of Operational Research 2011; **211**: 612-622.
97. Valls V, Perez A, Quintanilla S. Skilled workforce scheduling in Service Centres. *European Journal of Operational Research* 2009; **193**: 791-804.
98. Vignier A, Billaut JC, Proust C, T'Kindt V. Resolution of some two stage hybrid flowshop scheduling problems. In: *Proceedings of IEEE international conference on systems, man and cybernetics*. Piscataway: IEEE Press; 1996.
99. Vignier A, Commandeur P, Proust C. New lower bound for the hybrid flowshop scheduling problem. In: *Proceedings of the sixth international conference on emerging technologies and factory automation*, *ETFA'97*. Piscataway: IEEE Press; 1997.
100. Xu J, Xu X, Xie SQ. Recent developments in Dual Resource Constrained (DRC) system research. *European Journal of Operational Research* 2011; **215**: 309-318.
101. Yaurima V, Burteseva L, Tchernykh A. Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. *Computers & Industrial Engineering* 2009; **56**: 1452–1463.
102. Ying KC, Lin SW. Multiprocessor task scheduling in multistage hybrid flowshops: an ant colony system approach. *International Journal of Production Research* 2006, **44**(16): 3161–3177.
103. Ying KC, Lin SW. Scheduling multistage flowshops with multiprocessor tasks by an effective heuristic. *International Journal of Production Research* 2009, **47**(13): 3525–3538.
104. Zandieh M, Fatemi Ghomi SMT, Moattar Husseini, SM. An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. *Applied Mathematics and Computation* 2006; **180**: 111–127.
105. Zhu X, Wilhelm WE. Scheduling and lot sizing with sequence-dependent setup: a literature review. *IIE Transactions* 2006; **38:** 987-1007.
106. Zoubaa M, Baptistea P, Rebaineb D. Scheduling identical parallel machines and operators within a period based changing mode. *Computers & Operations Research* 2009; **36**: 3231-3239.