European Journal of Operational Research 200 (2010) 395–408

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing and Logistics

An Improved Genetic Algorithm for the Distributed and Flexible Job-shop Scheduling problem

L. De Giovanni [a,*], F. Pezzella [b]

[a] Dipartimento di Matematica Pura ed Applicata, Università degli studi di Padova, via Trieste, 63, 35121 Padova, Italy
[b] Dipartimento di Ingegneria Informatica, Gestionale e dell'Automazione, Università Politecnica delle Marche, via Brecce Bianche, 60131 Ancona, Italy

ARTICLE INFO

Article history:
Received 25 July 2007
Accepted 2 January 2009
Available online 14 January 2009

Keywords:
Genetic Algorithms
Flexible Manufacturing Systems
Distributed Job-shop Scheduling

ABSTRACT

The Distributed and Flexible Job-shop Scheduling problem (DFJS) considers the scheduling of distributed manufacturing environments, where jobs are processed by a system of several Flexible Manufacturing Units (FMUs). Distributed scheduling problems deal with the assignment of jobs to FMUs and with determining the scheduling of each FMU, in terms of assignment of each job operation to one of the machines able to work it (job-routing flexibility) and sequence of operations on each machine. The objective is to minimize the global makespan over all the FMUs. This paper proposes an Improved Genetic Algorithm to solve the Distributed and Flexible Job-shop Scheduling problem. With respect to the solution representation for non-distributed job-shop scheduling, gene encoding is extended to include information on job-to-FMU assignment, and a greedy decoding procedure exploits flexibility and determines the job routings. Besides traditional crossover and mutation operators, a new local search based operator is used to improve available solutions by refining the most promising individuals of each generation. The proposed approach has been compared with other algorithms for distributed scheduling and evaluated with satisfactory results on a large set of distributed-and-flexible scheduling problems derived from classical job-shop scheduling benchmarks.

1. Introduction

During the last decades, the manufacturing systems are evolving from traditional centralized environments to more flexible distributed settings, including multi-factory networks or multi-cell job shops. Multi-factory production takes place in several factories, which may be geographically distributed in different locations, in order to comply with and to take advantage from the trend of globalization [28,29]. Multi-cell environments include several independent manufacturing cells located in the same plant and they allow high-volume productions or multiple product types by exploiting all the available manufacturing resources. Systems of manufacturing cells may be found, for example, in the woodcutting industry, where a single plant may include more than one cell, each including a sawing machine and a trimming/boarding unit, able to perform all the operations needed to produce semi-finished parts for the furniture industry. A sample multi-cell setting is shown in Fig. 1. The system is composed of four Flexible Manufacturing Cell (FMCs) interconnected by a material handling system. Each FMC is made-up of several multi-purpose machines equipped with interchangeable tools to perform different operations during the same production cycle. The I/O center loads raw parts from a raw material storage unit, unloads finished parts and handles unfinished parts, so that the workload of a production cycle can be distributed among the different FMCs. Each FMC receives raw parts and material from the handling system and its set of flexible machines completes the subset of jobs assigned to the FMC itself.

Concerning distributed factory networks, each factory is configured as a flexible manufacturing system, with a set of multi-purpose machines. The I/O center and the handling system are replaced by a dispatching center assigning each job to one of the suitable factories and by a transportation system to move jobs between factories and customers [9].

The Distributed and Flexible Job-shop Scheduling problem (DFJS) considers the production scheduling problems emerging in distributed manufacturing environments, where jobs are carried out by a system of several, generally distributed, Flexible Manufacturing Units (FMUs), corresponding to the factories in a multi-factory environment or cells in a multi-cell setting. In the DFJS, the problem is to assign each job to one FMU and to solve a Flexible Job-shop Scheduling problem (FJS) for each cell, with the objective of minimizing the overall completion time of all the jobs on all the FMUs. In the DFJS, the optimization of the production cycle involves the following hierarchical problems that need to be solved sequentially or simultaneously:

* Corresponding author. Tel.: +39 049 827 1349; fax: +39 049 827 1392.
E-mail addresses: luigi@math.unipd.it (L. De Giovanni), pezzella@diiga.univpm.it (F. Pezzella).

0377-2217/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2009.01.008

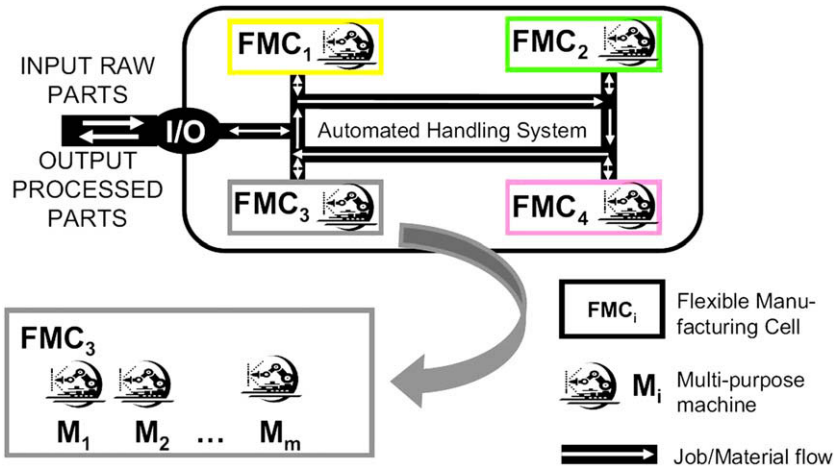

Fig. 1. A sample system of flexible manufacturing cells.

- determining the most suitable FMU (cell or factory) for each job (assignment problem);
- determining, for each operation of assigned jobs, the most suitable machine within a given FMU (routing problem);
- determining the assignment of operations to machines over time span (sequencing problem).

We consider that all the FMUs work simultaneously and cooperate to define an optimal production plan. Scheduling objectives have to be considered comprehensively. In particular, a measure related to the maximum job completion time (or makespan) is taken into account. Two types of makespan are defined:

- the *local* makespan related to a single FMU, which corresponds to the maximum completion time for jobs processed by the FMU itself; the completion time has to consider the job delivery time implied by the material handling system or by the transportation system, which may be different for different job assignments [9];
- the *global* makespan related to the whole system of FMUs, that is the maximum completion time of all jobs on all FMUs; this corresponds to the maximum over all the local makespans.

In the DFJS, the objective of minimizing the global makespan of the FMUs system is taken into account.

The DFJS extends the classic Flexible Job-shop Scheduling problem (FJS), as it allows several FMUs able to carry out different jobs of a manufacturing process. The DFJS extends also the Distributed Scheduling problem without machine route flexibility (DJS), where jobs can be assigned to several manufacturing units, but job routing within each unit is fixed, meaning that each job operation can be processed by an unique machine.

Production scheduling problems have been the subject of many research works in the last decades. Before concentrating on the literature concerning distributed scheduling, we note that, at the moment, research on scheduling of flexible manufacturing systems has been mostly devoted to non-distributed environments, that is to FJS. In particular, due to the complexity of the FJS, heuristic methods are generally proposed, mainly based on local search approaches ([2,6,13,18,24,32] among others) and genetic algorithms ([10,17,20,26,30] among others). Some interesting works from the methodological point of view propose the application of new meta-heuristics schemes to solve FJS. Xia and Wu [33] propose particle swarm optimization hybridized with simulated annealing to solve a multi-objective FJS. In [4] the FJS is represented as a language grammar by means of a syntactic model where Resource Elements are used to automatically consider all process planning options during the scheduling. This enables, on the one hand, to reduce the problem size and widen the scope of the problem to taking into account alternative process plans, and, on the other hand, to solve the problem by adopting an original linguistic based meta-heuristic, where existing job-shop scheduling algorithms are used in a simulated annealing framework. A mixed integer programming model along with a local search scheme is presented in [11] to solve the a generalized FJS where sequence-dependent set-up operations are involved. The works cited above directly address the scheduling of a single cell or factory. As we have seen, scheduling distributed manufacturing systems is more complex, as a further decision level is involved, related to the choice of the factory or cell to allocate each job. Literature mainly proposes heuristic approaches. Di Natale and Stankovic [14] present a general framework to apply to distributed static systems, based on simulated annealing. An agent-based approach is proposed in [12], where wasp-inspired autonomous agents implement an adaptive bidding mechanism for the dynamic assignment of jobs to manufacturing cells. Many other heuristic approaches have been proposed ([3,27,31], etc.). However, as already noted in [9], little attention has been devoted to the DFJS as described above.

Jia et al. [19] present a Modified Genetic Algorithm (MGA) to solve distributed scheduling problems in a multi-factory network, even if job delivery times are neglected. The algorithm is designed for fixed job routing and the flexibility issue is not taken into account. The proposed MGA is obtained by a straightforward extension of the chromosome encoding used for classic job-shop problems, where no information on operation routing is required: each gene specifies a job and a related factory, while the sequence of operations on machines is determined by the decoding procedure, according to the order in which the genes appear. One crossover operator is presented, based on exchanging partial strings, along with two mutation operators: a local one, to change randomly the order of the genes, and a global one, to randomly change the assignment of jobs to factories. Concerning computational results, one simple example of distributed multi-factory scheduling is presented and solved to optimality. MGA is also applied to some classic job-shop scheduling instances.

To the best of our knowledge, the first works addressing all the features of the DFJS are Chan et al. [8,9], where a multi-factory setting is considered and a Genetic Algorithm with Dominated Genes (GADG) is proposed. A more complex chromosome encoding is used, where genes specify job, operation, factory and machine. A

new crossover mechanism, the dominated gene crossover, is presented to enhance the performance of genetic search by recording and preserving the best genes, and to eliminate the problem of determining the optimal crossover rate. Also, similarity checking to prevent pre-mature convergence and mechanisms to adapt the value of some parameters to the evolution of the search are implemented. The GADG is tested on several DFJS instances proposed by the authors and some DJS and FJS cases presented in literature, showing the relevance of the new dominated gene crossover in obtaining better global makespans and improving the robustness of the genetic approach in terms of low standard deviation. The results are extended in [7], where a generalized version of the DFJS is studied, taking into account machine maintenance issues. A mathematical model for the problem is given and the results on some new test instances are presented, including the case with no maintenance (corresponding to the DFJS), confirming the good performance of the GADG approach.

The scope of this work is to propose a new optimization procedure combining genetic algorithms and local search heuristics to solve distributed scheduling problems in flexible manufacturing environments. In particular, we aim at improving the MGA proposed by Jia et al. [19] and extending the domain of its application to systems of FMUs (distributed factories or independent manufacturing cells in the same plant). The Improved Genetic Algorithm (IGA) presented in this paper introduces a new decoding mechanism, able to handle alternative job routings, and a refinement operator, acting as an "intelligent mutation" which exploits a local search procedure to refine the job routing and operations scheduling of most promising individuals.

This paper is organized as follows. Section 2 gives the specifications of the DFJS, introducing the basic notation to handle the scheduling of jobs and operations in systems of FMUs. Section 3 describes the IGA which combines a genetic algorithm, extension of the MGA, and a local search procedure applied to an elite subset of individuals of each generation to improve their schedules. Extended experiments on distributed and flexible scheduling problems have been run to verify the effectiveness of IGA with results discussed in Section 4. Section 5 concludes the paper and suggests possible lines for further research on the DFJS.

2. The Distributed and Flexible Job-shop Scheduling problem (DFJS)

The Distributed and Flexible Job-shop Scheduling problem (DFJS) can be stated as follows: a set $J = \{J_1, J_2, \ldots, J_i, \ldots, J_n\}$ of independent jobs is given, which have to be processed in a set $U = \{U^1, U^2, \ldots, U^l, \ldots, U^q\}$ of FMUs. We recall that a FMU may represent one factory of a geographically distributed network or one cell in a system of flexible manufacturing cells. For each pair of job i and FMU l a distance d_i^l is defined. In case of distributed factories networks, d_i^l is the time to deliver job i from the factory l it is assigned to, to its customer. In case of FMC systems, d_i^l is the time implied by the material handling system to move raw parts from the I/O center to the FMC l and the completed job from l back to the I/O center. Each FMU l is equipped with a set $M^l = \{M^{l1}, M^{l2}, \ldots, M^{lk}, \ldots, M^{lm^l}\}$ of machines. Each job i consists of an ordered set (sequence) of operations and can be processed on a set $\overline{U}_i \subseteq U$ of FMUs. Note that, once a job is assigned to a FMU, all the operations have to be processed on the same FMU. The set of operations depends, in general, on the FMU, so that different sequences may be defined for each job, one for each FMU enabled to process it. The sequence of operations for J_i to be processed on U^l is $(O_{i1}^l, O_{i2}^l, \ldots, O_{ij}^l, \ldots, O_{in_i^l}^l)$. Note that different FMUs may use different manufacturing technologies, and, as a consequence, each FMU may define a different number and/or sequence of operations to complete the same job. If J_i is assigned to U^l, each operation j (namely O_{ij}^l) has to be processed without interruption on one machine k: machine k has to be selected among a subset of the machines within U^l, namely the set $M_{ij}^l \subseteq M^l$. The processing time of each operation is related to both the FMU and the machine, and it is denoted by p_{ij}^{lk}. The completion time of a job is defined as the time at which the last operation of the job is completed, plus the distance d_i^l. Operations must be processed one after the other and without preemption, according to the sequence defined by the selected FMU. Machines can process only one operation on the same time. We assume that all FMUs, jobs and machines are available at time zero. The aim of the DFJS is to determine the assignment of jobs to FMUs, the routing of jobs through machines and the sequencing of operations on machines, with the objective of minimizing the global makespan.

Table 1
A sample DFJS instance.

		U^1				U^2				U^3		
		d_i^1	M^{11}	M^{12}	M^{13}	d_i^2	M^{21}	M^{22}	M^{23}	d_i^3	M^{31}	M^{32}
J_1	O_{11}	2	2	1	3	3	3	–	2	4	2	4
	O_{12}		3	5	–		3	3	3		3	–
	O_{13}		3	3	2		2	1	–		–	3
J_2	O_{21}	3	4	6	2	2	5	4	5	3	4	5
	O_{22}		3	2	7		5	4	3		4	3
J_3	O_{31}	3	3	1	4	5	3	6	4	4	4	3
	O_{32}		–	3	4		5	3	4		2	2
	O_{33}		4	4	2		–	–	–		2	3
J_4	O_{41}	4	5	4	5	3	6	3	5	5	3	4
J_5	O_{51}	3	–	5	8	–	–	–	–	3	5	4
	O_{52}		2	1	2		–	–	–		2	3

A sample instance of DFJS is given in Table 1. The related FMUs system is composed of three FMUs (U^1, U^2 and U^3) and five jobs (J_1, J_2, J_3, J_4 and J_5) have to be completed. The three FMUs are equipped with, respectively, three, three and two multi-purpose machines. For each FMU, the first column reports the distance of the related job to/from the marketplace or the I/O center. The remaining numerical entries represent the processing times p_{ij}^{lk}: for example, the time required to execute the first operation of J_2 on the third machine of U^1 is obtained by setting $i = 2$, $j = 1$, $l = 1$ and $k = 3$, that is, $p_{21}^{13} = 2$ time units. Note that only partial flexibility is considered in this example, as, for some FMU l, job i and operation j, $M_{ij}^l \subsetneq M^l$, that is, the operation cannot be performed by all the machines in the given FMU (this is the case for operation O_{32}^1 and machine M^{11}). Also, some FMUs cannot process some jobs, for example, J_5 and U^2. Different production technologies are also involved: for example, J_3 is performed by two operations in U^2, instead of the three required by the other FMUs.

The specifications of the DFJS given above allow the FJS and the JS to be derived as special cases: FJS is a DFJS where $|U| = 1$; JS is a DFJS where $|U| = 1$ and, for all operations O_{ij}^1, $|M_{ij}^1| = 1$. It directly follows that DFJS is $\mathcal{NP}$-*Hard*, given that JS is $\mathcal{NP}$-*Hard* [16].

3. An Improved Genetic Algorithm for DFJS

Genetic Algorithms (GA) are adaptive methods which may be used to solve optimization problems [5]. They are based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principle of natural selection, i.e. survival of the fittest. At each generation, every new individual (chromosome) corresponds to a solution, that is, a schedule for the given DFJS instance. Before a GA can be run, a suitable encoding (or representation) of the problem must be

devised. The essence of a GA is to encode a set of parameters (known as genes) and to join them together to form a string of values (chromosome). A fitness function is also required, which assigns a figure of merit to each encoded solution. The fitness of an individual depends on its chromosome and is evaluated by the fitness function. During the run, parents must be selected for reproduction and recombined to generate offspring. Parents are randomly selected from the population, using a scheme which favours fitter individuals. Having selected two parents, their chromosomes are combined, typically by using crossover and mutation mechanisms to generate better offspring, that means better solutions. The process is iterated until a stopping criterion is satisfied.

In the DFJS, where jobs can be dispatched to many FMUs and where job's operations can be dispatched to many machines within a given FMU, the encoding of the scheduling problem becomes more complex with respect to FJS or JS, since the chromosome has to comprise more information, including the selected FMU for every job and the related job routing, as well as operation sequence.

The overall structure of the Improved Genetic Algorithm (IGA) that we propose for the DFJS can be described as follows:

1. *Coding*: the genes of the chromosome describe the assignment of jobs to FMUs, and the order in which they appear in the chromosome describes the sequence of operations and determines the job routings. Each chromosome represents a potential solution for the problem.
2. *Initial population*: the initial chromosomes are obtained by a random permutation of jobs on FMUs and a random dispatching rule for sequencing operations on machines.
3. *Fitness evaluation*: the global makespan is computed for each chromosome of the current population.
4. *Selection*: at each iteration, the best chromosomes are chosen for reproduction by a linear ranking method.
5. *Offspring generation*: the new generation is obtained by two types of crossover operators (one-point crossover and two-points crossover) based on exchanging partial strings, and by four types of mutation operators: local mutation, global mutation, a new machine mutation and a new refinement mutation. The refinement mutation applies a local search to refine the job routing and the operation sequencing in the critical FMUs (that is, the ones determining the current global makespan) with the aim of improving the offspring generation. The genetic operators are designed to preserve feasibility of new individuals. New individuals are generated until a fixed maximum number of individuals is reached.
6. *Stop criterion*: the algorithm stops if a predefined number of iterations is reached or the best solution found so far has not changed during a predefined number of the last iterations. As the stop criterion is satisfied, the best chromosome, together with the corresponding schedule, is given as output. Otherwise the algorithm iterates through steps 3–5.

In the following, the different steps of IGA are detailed. We describe: in Section 3.1 the adopted coding and decoding schemes, in Section 3.2 the population initialization and the chromosome selection strategy, in Section 3.3 the crossover operators, in Section 3.4 the global mutation operator for changing the assignment of jobs, the local mutation operator for changing operations sequence and the new machine mutation operator for changing job routing, in Section 3.5 the new refinement mutation to improve job routing and machine sequencing by local search and finally, in Section 3.6, the stopping criterion and the overall structure of IGA.

3.1. Chromosome encoding and decoding

The information to be encoded into the chromosomes of a GA for the DFJS has to specify the allocation of each job to FMUs, the routing of each job through machines and the priority of each operation. In this work, we use a simple operation-based encoding method, which is basically the one proposed by Jia et al. [19] for distributed scheduling problems without routing flexibility, and we extend it to take into account the flexibility issues of the DFJS. The number of genes in a chromosome is equal to the total number of operations of all the jobs. As the number of operations for a given job may depend on the FMU the job is assigned to, the length of chromosomes would depend on job assignment. In order to keep the same number of genes in each chromosome, job production plans are completed, where necessary, with dummy operations to be executed on a dummy machine with processing time equal to zero. For example, a third dummy operation O_{33}^2 will be added in the sample instance of Table 1. Each gene represents an operation and consists of two parameters reporting, respectively, the FMU number and the job number. Note that all the operations of the same job are represented by the same allele (same FMU and same job) and then interpreted according to the order they occur in a given chromosome, given that the order for the operations of a job is fixed. As we keep the same simple representation as Jia et al. [19], no information about alternative machine routes is explicitly encoded into genes: as we will see hereafter, this information will be retrieved during the decoding phase. A sample individual is given in Fig. 2, where the DFJS instance of Table 1 is considered.

Jobs are composed of, respectively, three, two, three, one and two operations, so that a chromosome is a sequence of 11 genes. Genes are of five types: "1, 1", "2, 2", "1, 3", "2, 4" and "3, 5", meaning that jobs J_1 and J_3 are processed by U^1, J_2 and J_4 are processed by U^2 and J_5 by U^3.

The decoding process exploits the information provided by each chromosome to generate a schedule plan and evaluate the fitness of each individual. The objective of the DFJS is to minimize the global makespan of the FMUs system so that the fitness of an individual is inversely related to the global makespan.

As we have mentioned above, chromosomes explicitly represent information on job assignment to FMUs and the order of the genes is relevant to determine the priority of each operation, while no information on job routing is given. Instead of complicating gene encoding, we consider the flexibility issue in the decoding phase, that is able to dispatch job operations to one of the alternative machines of the selected FMU. The information on job routing is thus implicit in the decoding process, acting as follows. Operations are considered one by one, according to the order determined

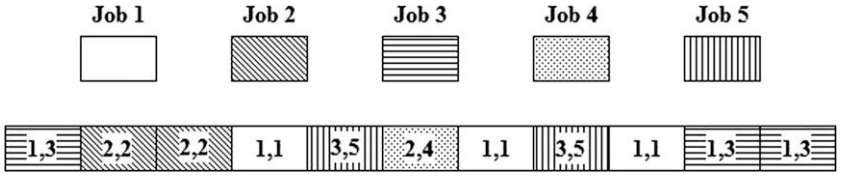

Fig. 2. A sample chromosome encoding.

Table 2
Decoding the chromosome of Fig. 2: job routing

		U^1			U^2			U^3	
		M^{11}	M^{12}	M^{13}	M^{21}	M^{22}	M^{23}	M^{31}	M^{32}
J_1	O_{11}	2	[1]	3	3	–	2	2	4
	O_{12}	[3]	5	–	3	3	3	3	–
	O_{13}	3	3	[2]	2	1	–	–	3
J_2	O_{21}	4	6	2	5	[4]	5	4	5
	O_{22}	3	2	7	5	4	[3]	4	3
J_3	O_{31}	3	[1]	4	3	6	4	4	3
	O_{32}	–	[3]	4	5	3	4	2	2
	O_{33}	4	4	[2]	–	–	–	2	3
J_4	O_{41}	5	4	5	[6]	3	5	3	4
J_5	O_{51}	–	5	8	–	–	–	5	[4]
	O_{52}	2	1	2	–	–	–	[2]	3

by the chromosome, and dispatched to a machine, with starting time equal to the completion time of the last operation assigned to the machine itself. In case one operation may be performed by more than one machine, the decoding process selects the routing that guarantees the lowest current local makespan, that means the one giving the lowest completion time for the operations assigned so far. If different routings lead to the same current makespan, the machine with the smallest processing time is chosen. In case more machines have the same smallest current makespan and processing time, one of them is selected at random, to give the optimization algorithm the opportunity to search different regions of the solution space. Once all the operations have been scheduled, the decoding process is completed by adding the delivery time (according to the FMU the job is assigned to), thus obtaining the local makespans and the global one. Table 2 and Fig. 3 show the results of the decoding process applied to the chromosome of Fig. 2. Table 2 reports the job routing in all the FMUs, squared entries denoting the operation-to-machine assignments. Fig. 3 reports the Gantt chart of the operation schedule. After considering the job-to-FMU distances, local makespans of 12 (9 + 3), 9 (7 + 2) and 9 (6 + 3) time units are obtained for U^1, U^2 and U^3, respectively, and the global makespan of the FMUs system is 12.

In order to show how the proposed simple encoding is able to represent implicitly alternative machine routes, we consider the individual of Fig. 4, obtained by swapping two genes of the chromosome of Fig. 2. The decoding process considers the second operation of J_3 before the first operation of J_1, so that O^1_{32} is scheduled one time unit in advance on machine M^{12} (instead of M^{13}), allowing like this the completion of J_3 at time 8 instead of 9. After including

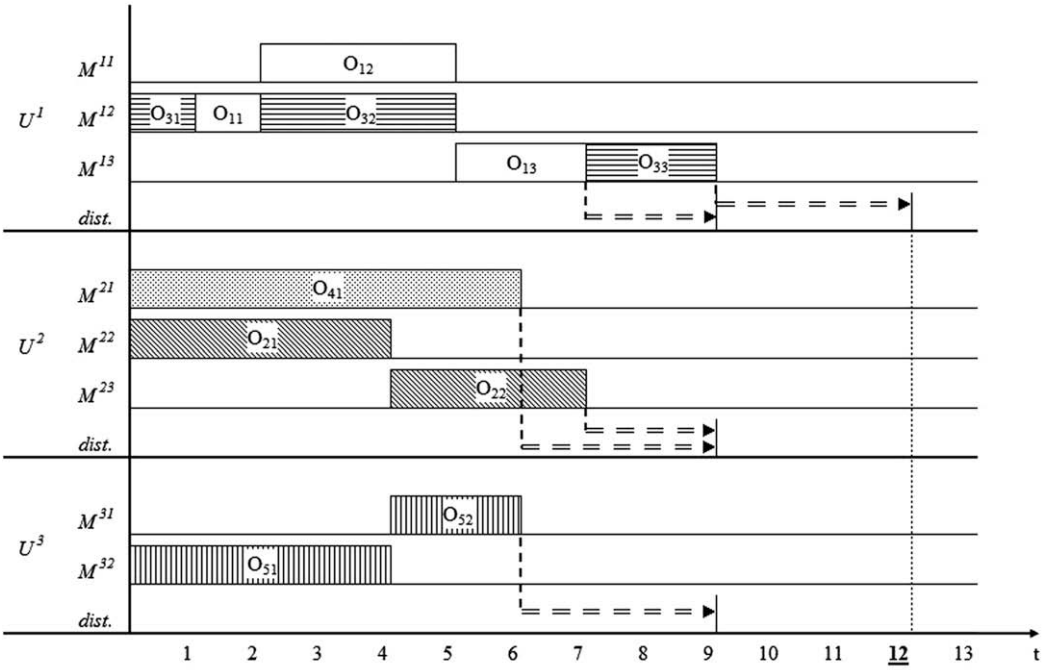

Fig. 3. Decoding the chromosome of Fig. 2: operation schedule.

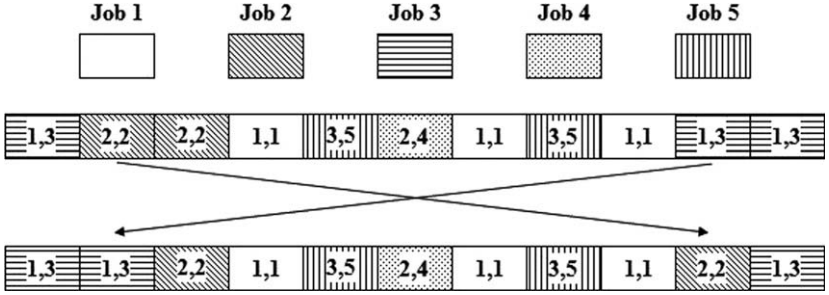

Fig. 4. A new chromosome obtained by swapping two genes.

Table 3
Decoding the chromosome of Fig. 4: new job routing.

		U^1			U^2			U^3	
		M^{11}	M^{12}	M^{13}	M^{21}	M^{22}	M^{23}	M^{31}	M^{32}
J_1	O_{11}	[2]	1	3	3	–	2	2	4
	O_{12}	[3]	5	–	3	3	3	3	–
	O_{13}	3	3	[2]	2	1	–	–	3
J_2	O_{21}	4	6	2	5	[4]	5	4	5
	O_{22}	3	2	7	5	4	[3]	4	3
J_3	O_{31}	3	[1]	4	3	6	4	4	3
	O_{32}	–	[3]	4	5	3	4	2	2
	O_{33}	4	[4]	2	–	–	–	2	3
J_4	O_{41}	5	4	5	[6]	3	5	3	4
J_5	O_{51}	–	5	8	–	–	–	5	[4]
	O_{52}	2	1	2	–	–	–	[2]	3

the job-to-FMU distances, the local makespan of U^1 is 11 (8 + 3, see Table 3 and Fig. 5), and the global makespan is reduced by one time unit.

3.2. Population initialization and chromosome selection

The initial population is determined in two phases: the first phase assigns jobs to FMUs at random and creates one gene for each operation with information on jobs and related FMU; the second phase combines genes in a random sequence and it is repeated until a predefined number of individuals (denoted by N) is generated. Note that just the second phase is iterated and all the chromosomes in the initial population have the same assignment of jobs to FMUs.

During the selection phase, a set of individuals from the current population is chosen in order to apply genetic operators and generate the offspring to include in the next generation. Among alternative strategies, linear ranking has been adopted [1], according to the results emerged during our computational tests. The linear ranking strategy sorts individuals by decreasing makespan (from the worst to the best individual). If $\sigma_i \in \{1, 2, \ldots, N\}$ is the position of individual i, the probability of selecting it is given by

$$p_i = \frac{2\sigma_i}{N(N+1)}, \quad i = 1 \ldots N.$$

For example, the probability of selecting the best chromosome ($\frac{2}{N+1}$) is twice the probability of selecting the median one ($\frac{1}{N+1}$).

The reason is to give good chromosomes more chances to transfer their genes, while allowing even to less promising individuals the opportunity of participating to the evolution: the search will move towards most promising regions while guaranteeing a certain diversity of the solution pool and preventing a premature convergence of the method.

3.3. Crossover operators

Crossover operators recombine the genes of two selected chromosomes to generate two new chromosomes to include in the next generation. We have tested two types of crossover based on exchanging partial strings: a one-point crossover and a two-points crossover.

The *one-point crossover* is illustrated in Fig. 6. It randomly divides the two parent chromosomes into two substrings and two new chromosomes are obtained by exchanging the first substring and maintaining the second one.

Once strings are exchanged, offspring may not represent a legal encoding, as they may have an uncorrect number of genes per job. It is thus necessary to legalize the offspring: starting from a random position of each chromosome (in the example we start from position 1), we delete the redundant genes (underlined in Fig. 6) and compensate the missing ones, in order each chromosome to comprise all the operations of all the jobs. We note that the crossover operator assumes that all the chromosomes of the same generation refer to the same job assignment, so that no further legalization process is necessary.

The *two-points crossover* is similar to the previous operator and is illustrated in Fig. 7. Two cut points are randomly chosen and three substrings are determined for each parent. The first and the third pair of substrings are exchanged. Also in this case, the legalization procedure is necessary to generate feasible offspring.

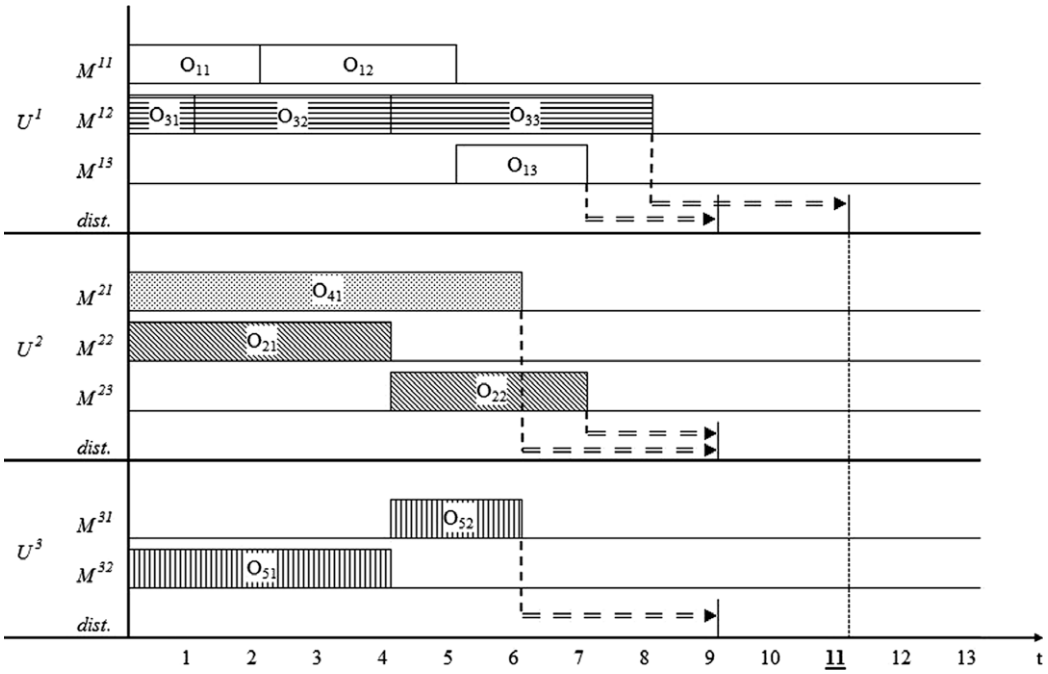

Fig. 5. Decoding the chromosome of Fig. 4: new operation schedule.

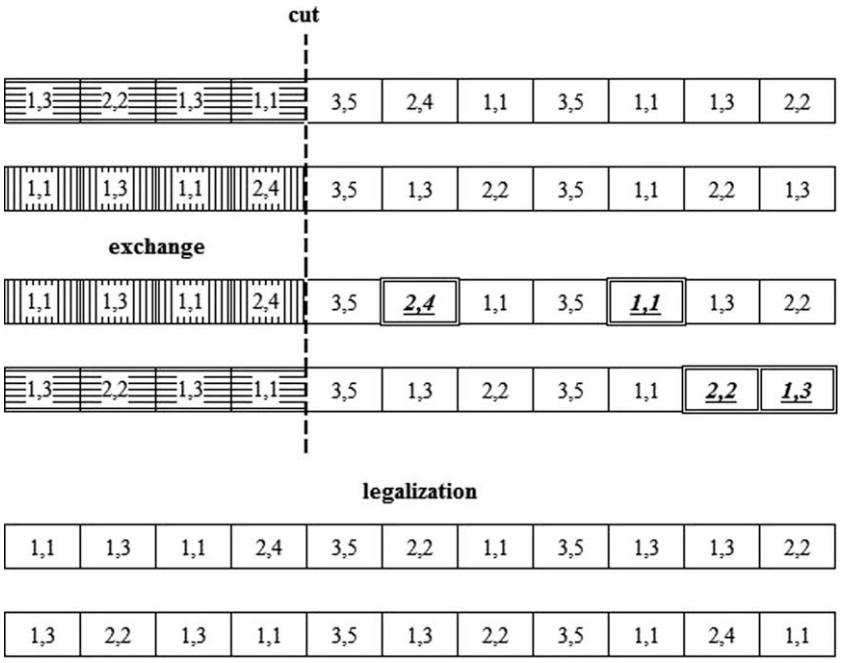

Fig. 6. One-point crossover.

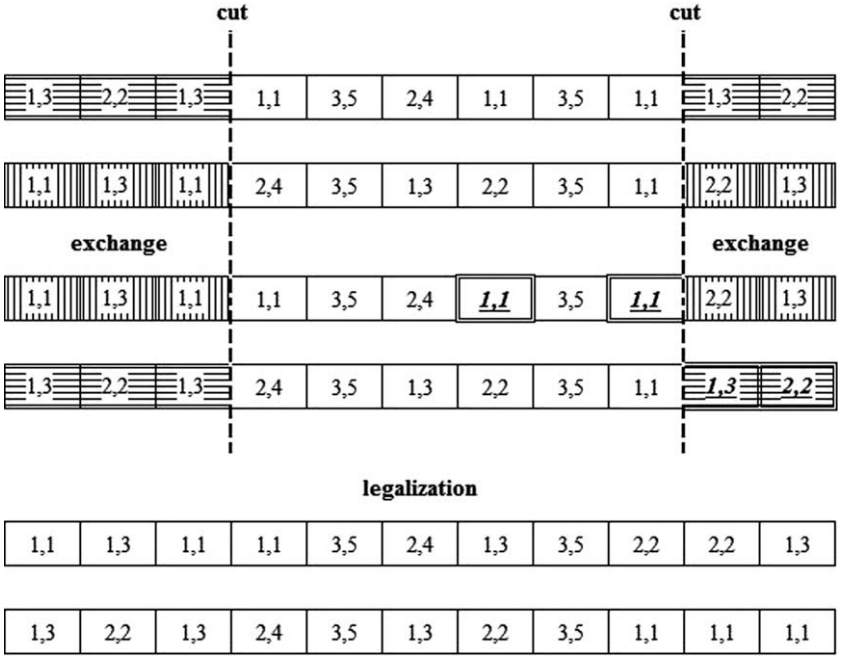

Fig. 7. Two-points crossover.

Crossover operators have the effect of changing the processing sequence of operations and job routes, according to the chromosome decoding scheme. Because all the chromosomes of the current generation refer to the same job assignment, the dispatching of jobs to FMUs does not change after the crossover, as single genes remain unaffected.

3.4. Mutation operators

Mutation is designed to let the genetic algorithm explore a wider region of the solution space, generally by introducing random gene or chromosome changes. Following the scheme in Jia et al. [19], we consider two operators: local and global mutation. We also introduce a third machine mutation operator, to take into account the job-routing flexibility, as from the definition of the DFJS.

Local mutation aims at preventing the genetic approach from premature convergence towards low-quality solutions. It is applied to the offspring generated by crossover and acts on one chromosome at a time (local effects). It consists in randomly choosing a pre-defined number of pairs of genes within the same chromosome and permuting their positions (Fig. 8). Two parameters are related to local mutation: the probability of being applied to a new

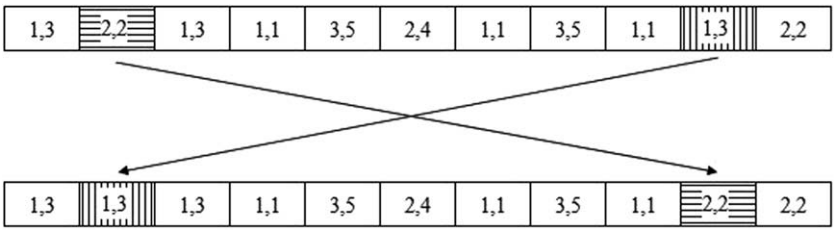

Fig. 8. Local mutation.

offspring chromosome (ρ_p^L) and the number of permutations to execute (ρ_q^L).

Global mutation takes into account the peculiarity of DFJS with respect to non-distributed scheduling problems, as it tries to explore solutions with different assignments of jobs to FMUs. Global mutation is applied at the end of each iteration yielding a new generation of individuals and consists of randomly changing the FMUs of a given number of jobs. Note that, in order to keep consistency with the remaining genetic operators, all the chromosomes have to reflect the new job assignments, that means that all the genes related to the selected job in all the chromosomes have to be updated (global effects). Global mutation has a significant impact on operation scheduling, so that it is not applied at every iteration, in order to let the algorithm explore solutions with a given job assignment before changing it. As a consequence, two parameters are defined: the probability of applying global mutation to the current generation (ρ_p^G) and the number of jobs undergoing a random FMU change (ρ_q^G).

We observe that, in general, the proposed simple encoding is not able to span all the solution space with relation to the job routing. Indeed, the decoding phase is guided by the minimum current makespan criterion and it is not possible to assign operations to an arbitrary machine. For this reason, the *machine mutation* operator is introduced: after a given number of not-improving iterations (ρ_q^M), an operation is randomly assigned to a machine able to process it, with a probability ρ_p^M. IGA is thus able to explore solutions with different job routings and to reduce the risk of premature convergence due to poor operations assignment.

3.5. Chromosome refinement by local search

A fourth mutation operator, the *refinement mutation*, is designed to have an intensive search strategy. Given an individual of the current generation, the refinement mutation applies a local search algorithm in order to explore efficiently different gene sequencing and determine a locally optimal one. Starting from a chromosome (center solution), new neighbor chromosomes are obtained by swapping one pair of genes, thus perturbing the priority of jobs and the related schedule. For each neighbor chromosome, the decoding process computes the global makespan. As far as a neighbor chromosome exists which has a better fitness than the current center one, it is selected as the center of a new neighborhood and the process is iterated until no neighbor chromosome gives a better fitness. The center of this last neighborhood represents a locally optimal individual, with respect to gene sequencing, and it replaces the original chromosome in the current generation. According to computational evidence, it is important to achieve a good trade-off between quality of individuals and running times. For this reason, besides considering perturbations as simple as the straightforward pairwise swap, the refinement mutation is applied to just a restricted number of most promising individuals, that means individuals with the smallest global makespan. This number is a parameter of the algorithm to be calibrated and is denoted by ρ_q^R. Also, just a subset of the possible gene swaps are considered, which reduces the number of neighbor solutions to be evaluated at each iteration. In particular, we determine the *critical* FMU, that is, the FMU with the maximum local makespan, and we consider only swaps between genes related to the critical FMU itself. Note that the critical FMU could dynamically change during the search: the local makespan of the current critical FMU may be reduced so that a new FMU becomes critical. In this case, the local search considers swap moves related to the new critical FMU, and the process is iterated until no further local improvements are possible. Finally, running times are further reduced by adopting a first improvement exploration strategy: the center for the next local search iteration is the first neighbor chromosome yielding a lower makespan (neighborhoods are explored in a random order). The pseudo-code of the refinement operator is shown in Fig. 9.

```
Compute local and global makespans
Select the critical FMU
Repeat (local search loop)
    Generate a new neighbor solution by swapping two genes related to the current critical FMU
    Compute the local makespan of the critical FMU
    If the local makespan is improved
        iterate_local_search = True
        If the current FMU is not critical, select the new critical FMU
    Else
        If further neighbor solutions exist
            iterate_local_search = True
        Else
            iterate_local_search = False
Until iterate_local_search = False
```

Fig. 9. The refinement mutation operator.

In the following example, the refinement operator is applied to the chromosome of Fig. 2, represented by the string

$$S = [(1,3),(2,2),(2,2),(1,1),(3,5),(2,4),(1,1),(3,5),(1,1),(1,3),(1,3)].$$

As from Fig. 3, the critical FMU is U^1 with a local makespan of 12 time units, so that we restrict to the string

$$S_1 = [(1,3),(1,1),(1,1),(1,1),(1,3),(1,3)].$$

By swapping the first two genes, we obtain

$$S_2 = [(1,1),(1,3),(1,1),(1,1),(1,3),(1,3)]$$

corresponding to the schedule of Fig. 10a, with a local makespan of 8 + 3 = 11 time units. A better makespan is obtained and the critical FMU does not change. Starting from S_2, a new swapping neighborhood is generated. In particular, when swapping the third and the sixth genes, the refinement operator obtains

$$S_3 = [(1,1),(1,3),(1,3),(1,1),(1,3),(1,1)].$$

The decoding process provides the schedule of Fig. 10b, with a makespan of 7 + 3 = 10 time units which cannot be further improved by the refinement operator. Indeed, we can show that 10 represents a lower bound for the makespan of U^1, given the current job assignments. As the lowest processing time for the first operation of both J_1 and J_3 is one time unit and it is given by the same machine M^{12}, the completion time of one among O^1_{11} and O^1_{31} is at least two. As the minimum processing time for the remaining operations of J_1 and J_3 is five, a lower bound for the completion time of both J_1 and J_3 on U^1 (excluding the delivery time) is seven. We recall that the refinement operator does not change the job assignment and the related job delivery time. As a consequence, if we add three (the delivery time of J_3 on U^1) we obtain 10 time units, which is a lower bound for the local makespan of U^1, given the current job assignment. Also, no new critical FMU emerges, as the makespans for both U^2 and U^3 is nine time units (see Fig. 3), and the local search stops providing a new locally optimal chromosome.

3.6. The Improved Genetic Algorithm for DFJS

The IGA is summarized in Fig. 11. After encoding job and operation information into genes, the population is initialized by combining this information into N chromosomes, each determined by a random gene sequence. The local and global makespans of individuals are then evaluated according to the decoding process and the algorithm enters two nested loops. The inner loop generates new individuals by crossover between selected pairs of chromosomes and by applying local and machine mutations to the offspring. Then, global and refinement mutations are applied within the outer loop, that controls the population evolution from a generation to the next one, until the termination conditions are satisfied. The IGA terminates if a predefined number ρ_I of iteration is reached or the best solution found has not changed within the last ρ_S iterations ($\rho_S < \rho_I$). Then, the best found schedule is output and the algorithm ends.

4. Numerical experiments

The performance of the IGA has been tested on several instances. We consider two groups of tests. The first group aims at comparing IGA with other algorithms designed for distributed scheduling and, in particular, Jia et al. [19], Chan et al. [7,9]. The second group of tests enlarges the benchmark to further instances of DFJS obtained from some FJS literature instances adapted to the distributed scheduling case. IGA has been implemented in C++ and

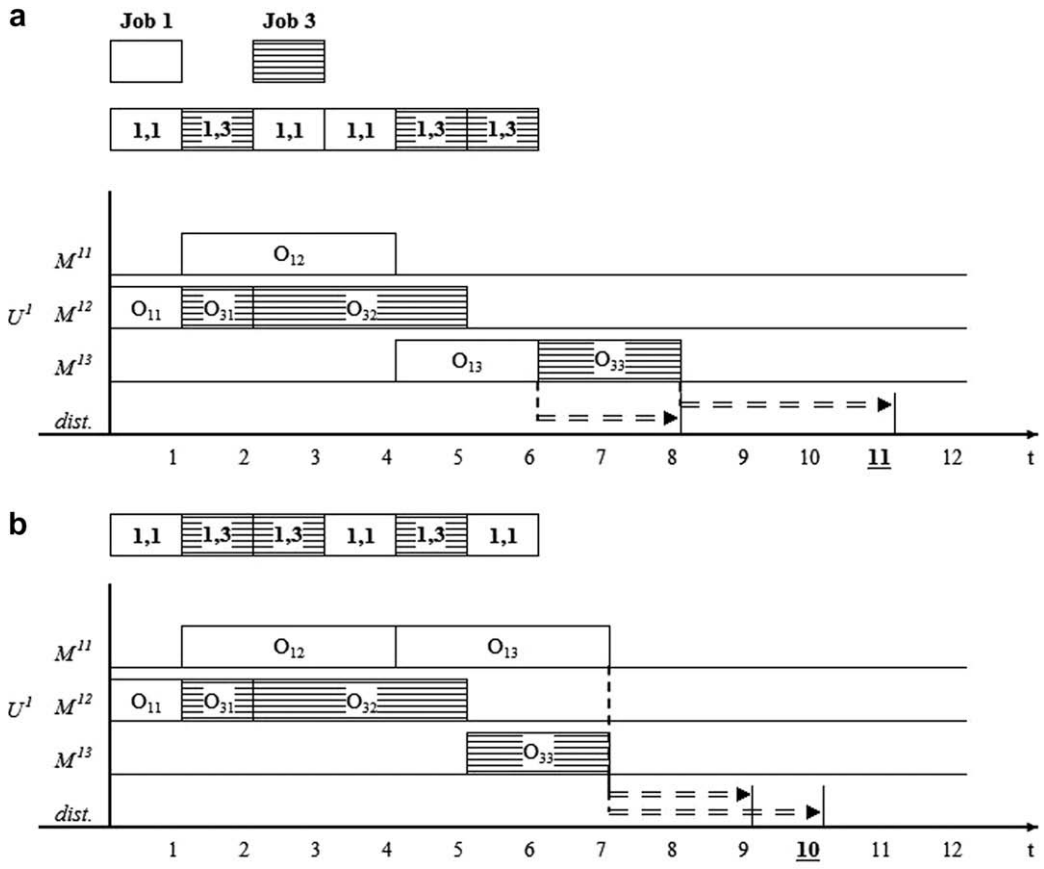

Fig. 10. Example of local search-based refinement mutation.

Read problem and parameters
For each job operation
    create a gene by encoding information on job and assigned FMU
Create the initial population of N chromosomes by randomly sequencing genes
Compute local and global makespans
Repeat (outer loop: evolve population)
    Repeat (inner loop: generate new individuals)
        Select two parent chromosomes and apply crossover
        Apply local mutation (change operation sequence and machine route)
        Apply machine mutation
    Until N new individuals are generated
    Apply global mutation (change job-to-FMU assignments)
    For each of the ρ^R_q best individuals
        Apply refinement mutation (local search on operation sequence and machine route)
    Select N elite members for the next generation
Until termination conditions are reached
Output the best solution

Fig. 11. The improved genetic algorithm.

compiled with Gnu compiler. The results described in the following sections have been obtained on a personal computer equipped with a 2.0 GHz Intel Core2 processor and 2 GB RAM.

4.1. Comparison with other algorithms designed for distributed scheduling

The instances used to compare IGA with other algorithms designed for distributed scheduling are summarized in Table 4. For each instance, columns report, in the order, the type of the problem they refer to, the name, the number of FMUs (1 for non-distributed problems), the maximum number of machines able to perform a given operation (1 if job routing is fixed), the number of jobs, the maximum number of operations per job and the reference where it was introduced and/or tested. Note that Chan et al. [7,9] provide results for further instances with up to 10 factories and 50 jobs, but it was not possible to retrieve a full description of them. Note also that the cited works test their algorithms on single FMU as well as fixed routing settings, that is, instances of the FJS and JS problems.

IGA parameters have been calibrated by preliminary tests on both the instances described above and the other instances derived from literature. The values used for different types of instances (columns DFJS, FJS, DJS and JS) are reported in Table 5.

IGA has a non-deterministic nature and hence, as for the computational tests described in [9], it has been run 50 times on each problem instance, to measure the average performance and the deviation of the obtained solutions. Results are shown in Table 6. We consider three cases: results reported in literature (as from column Reference), IGA with no refinement mutation and IGA with refinement mutation. For each instances, we report the best obtained makespan during the 50 runs (MK), the average makespan (Av.), the per-cent standard deviation (Dev%) and the average computational time in seconds (s, not available for literature results).

The results of IGA (with or without refinement mutation) dominate the other algorithms designed for distributed scheduling. Concerning the two DFJS instance, significant improvements are obtained in terms of best makespan as well as algorithm reliability, as the average makespan and the standard deviation are sensibly smaller. Also results for the FJS instance are improved both in terms of best schedule and average performance. For the DJS instance, IGA obtains the same minimum makespan as GADG or MGA (corresponding to the optimal value, as it is equal to the lower bound described in the next section) with a standard deviation under the 2%. Finally, concerning the Fisher–Thompson JS instances (mt06, mt10 and mt20), IGA sensibly improves over the makespan obtained by MGA.

Comparing the results of IGA with and without refinement, we observe that, thanks to the refinement operator, IGA is able to find always better solution and to improve the average performance. In particular, for the JS instances, IGA with refinement finds the opti-

Table 4
First group instances.

Type	Instance	$\|U\|$	m/o	$\|J\|$	ops/job	Reference
DFJS	dfjs01	2	3	6	4	[9]
	dfjs02	2	3	10	4	[7]
FJS	fjs01	1	3	5	4	[22]
DJS	djs01	3	1	6	4	[19]
JS	mt06	1	1	6	6	[15]
	mt10	1	1	10	10	[15]
	mt20	1	1	20	5	[15]

Table 5
First group instances: IGA parameters.

Parameter		DFJS	FJS	DJS	JS
Number of individuals per generation	N	100	100	100	30
Maximum number of generations	ρ_I	5000	800	5000	100
Number of generation without improvement (% referred to ρ_I)	ρ_S	75	75	75	75
Type of crossover ('one-' or 'two-' points)		Two	Two	Two	One
Probability of applying local mutation	ρ^L_p	0.9	0.9	0.75	0.25
Number of permutations in local mutation (% referred to N)	ρ^L_q	20	20	20	5
Probability of applying global mutation	ρ^G_p	0.5	–	0.4	–
Number of job-assignment changes in global mutation (% referred to the number of jobs)	ρ^G_q	20	20	20	–
Number of not improving iterations before applying machine mutation	ρ^M_q	200	200	–	–
Probability of changing the assignment of an operation in machine mutation	ρ^M_p	0.02	0.02	–	–
Number of individuals to refine	ρ^R_q	3	3	3	3

Table 6
First group instances: results (obtained optimal values in bold).

Inst.	Literature				IGA (no refinement)				IGA (refinement)			
	Ref.	MK	Av.	Dev%	MK	Av.	Dev%	s	MK	Av.	Dev%	s
dfjs01[a]	[9]	42	43.1	n.a.	39	40.7	2.1	8.5	37	38.6	1.9	156.0
dfjs02[a]	[7]	49	51.0	25.2	39	40.3	1.9	8.5	37	38.3	1.9	145.5
fjs01[a]	[9]	36	37.4	2.8	35	35.4	1.4	0.0	35	35.1	0.9	1.0
djs01[b]	[9]	**11**	11.0	n.a.	**11**	11.0	1.8	5.8	**11**	11.0	1.8	5.8
mt06[b]	[19]	**55**	n.a.	n.a.	**55**	57.6	2.3	0.0	**55**	55.0	0.0	0.9
mt10[b]	[19]	972	n.a.	n.a.	1139	1209.3	2.6	0.0	**930**	947.4	0.9	114.8
mt20[b]	[19]	1207	n.a.	n.a.	1435	1513.3	2.1	0.0	1172	1181.1	0.3	96.4

n.a.: not available.
[a] Results from GADG.
[b] Results from MGA.

mal solutions for mt06 and mt10 (as retrieved from [24]) and the gap for mt20 is 1% (1172 instead of 1160 [24]). The better results are obtained at the cost of sensibly increasing the running time. Concerning this last issues, we observe that running times can be sensibly reduced. Indeed, tests have considered the maximum number of iterations and the population size as from Chan et al. [9]. Computational evidence show that IGA normally converges in less than 1000 generations for DFJS and DJS instances (instead of 5000) and less than 100 iterations for FJS instance (instead of 800).

4.2. Test on further DFJS instances

In order to test IGA on a larger test set, we have considered several instances derived from classical scheduling benchmarks. In particular, we have obtained instances for the DFJS starting from 23 of the problems proposed by Hurink et al. [18] for the FJS (set rdata), which in turn are based on the well known Fisher and Thompson [15] problems (mt06, mt10 and mt20) and on the Lawrence [21] benchmark (la01-20) for the classical JS. Each DFJS instance has been obtained by supposing that all the FMUs share the same characteristics and by replicating the data describing the related flexible job shop for each FMU. Cases with two, three or four FMUs have been taken into account, leading to a total of 69 instances. The distance from FMUs to jobs has been considered negligible. All the basic JS and FJS instances are available in Mastrolilli [23].

IGA parameters have been calibrated by preliminary tests on the instances described above and the best estimated values are summarized in Table 7: column DFJS-2 refers to the parameters used for instances of the DFJS with 2 FMUs and column DFJS-3/4 refers to the parameters used for instances of the DFJS with 3 or 4 FMUs. Note that, in order to achieve a good trade-off between the quality of the results and the computational effort, we have reduced both the population size, the maximum number of generations and the interval for applying machine mutation with respect to those in Table 5, while the remaining parameters keep the same values. Indeed, as observed before, IGA normally converges in a number of generations smaller than shown in Table 5. Also we can reduce the population size thanks to the local-search-based refinement operator, able to let a set of chromosomes converge towards a local optimal one: if a larger population is considered, some of the individuals are likely to be redundant, being related to the same local optimum.

Table 8 shows the results obtained by the IGA for the DFJS instances with two FMUs. The first three columns give the name of the FJS instance used to derive the DFJS instance together with the number of jobs and operations per job. Columns MK, Av., Dev% and T report, respectively, the best global makespan obtained by IGA during five runs, the average makespan, the percent standard deviation and the average computational time in seconds. The best global makespan is compared with the lower bound reported in column LB. As no lower bound for DFJS is available from

Table 7
Parameter summary for DFJS instances.

Parameter		DFJS-2	DFJS-3/4
Number of individuals per generation	N	50	50
Maximum number of generations	ρ_I	300	250
Number of generation without improvement (% referred to ρ_I)	ρ_S	75	75
Type of crossover ('one-' or 'two-' points)		Two	Two
Probability of applying local mutation	ρ_p^L	0.9	0.9
Number of permutations in local mutation (% referred to N)	ρ_q^L	20	20
Probability of applying global mutation	ρ_p^G	0.5	0.5
Number of job-assignment changes in global mutation (% referred to the number of jobs)	ρ_q^G	20	20
Number of not improving iterations before applying machine mutation	ρ_q^M	40	40
Probability of changing the assignment of an operation in machine mutation	ρ_p^M	0.02	0.02
Number of individuals to refine	ρ_q^R	3	3

Table 8
IGA computational results: DFJS instances with two FMUs.

Inst.	Jobs	Ops.	MK	Av.	Dev%	T (s)	LB	Gap%
la01	10	5	413	413.0	0.0	12.0	413	0.0
la02	10	5	394	394.0	0.0	11.2	394	0.0
la03	10	5	349	349.0	0.0	10.8	349	0.0
la04	10	5	369	369.0	0.0	11.4	369	0.0
la05	10	5	380	380.0	0.0	8.0	380	0.0
la06	15	5	445	449.6	0.6	45.8	413	7.7
la07	15	5	412	419.2	1.1	50.2	376	9.6
la08	15	5	420	427.8	1.6	53.8	369	13.8
la09	15	5	469	474.6	0.8	45.2	382	22.8
la10	15	5	445	448.6	0.5	45.0	443	0.5
la11	20	5	570	571.6	0.3	126.0	413	38.0
la12	20	5	504	508.0	0.4	116.0	408	23.5
la13	20	5	542	552.2	1.0	125.4	382	41.9
la14	20	5	570	576.0	1.2	122.2	443	28.7
la15	20	5	584	588.8	0.7	119.6	378	54.5
la16	10	10	717	717.0	0.0	140.2	717	0.0
la17	10	10	646	646.0	0.0	112.6	646	0.0
la18	10	10	663	663.0	0.0	132.4	663	0.0
la19	10	10	617	617.2	0.1	147.2	617	0.0
la20	10	10	756	756.0	0.0	99.8	756	0.0
mt06	6	6	47	47.0	0.0	2.0	47	0.0
mt10	10	10	655	655.0	0.0	173.0	655	0.0
mt20	20	5	560	566.0	0.9	121.2	387	44.7
Average					0.4	79.6		12.4

literature, we compute it by keeping the precedence constraints for operations of the same job and by eliminating the constraint saying that a machine can process just one operation at a time. For each job, we determine the lowest possible time to complete it: for each FMU, we consider that all the operations are carried out by the machine with the smallest processing time, and we sum up these values and the delivery time due to the distance of between the job and the FMU obtaining the best completion time for a job within a given FMU; then we consider the minimum completion time among all FMUs. Finally, LB is obtained by considering the maximum value over all of these job completion times. This process is synthesized by the following formula:

$$LB = \max_{i \in J} \left\{ \min_{l \in \overline{U}_i} \left\{ \sum_{j=1}^{n_i^l} \min_{k \in M_{ij}^l} \left\{ p_{ij}^{lk} \right\} + d_i^l \right\} \right\}.$$

Note that the proposed lower bound may be very poor. As a matter of fact, just one single job determines LB, that means that LB is not influenced by the distribution of jobs among the different FMUs: according to LB hypothesis, machines are able to carry out all the operations simultaneously and the number of jobs assigned to a FMU does not influence the time required to process them. The last column of Table 8 reports a measure of the quality of the best global makespan, obtained as per-cent error related to LB:

$$\text{Gap}\% = \frac{MK - LB}{LB} \cdot 100$$

In many cases (13 out of 23), the global makespan obtained by IGA is proven to be optimal (MK = LB) or very close to the optimal one (the gap is less than 0.5%). For the remaining instances, the gap can be high, but we recall that the LB is very poor. Of course, given the lower bound definition, better gaps are obtained if we consider the distribution of jobs among more than two FMUs. IGA applied to instances with three FMUs (see Table 9) is able to find the optimal global makespan in 19 out of 23 instances, and just in two cases the gap is over 10%.

Finally, with four FMUs (see Table 10), IGA finds the optimal solution in all the tested instances but one with a 5% gap.

Table 9
IGA computational results: DFJS instances with three FMUs.

Inst.	Jobs	Ops.	MK	Av.	Dev%	T (s)	LB	Gap%
la01	10	5	413	413.0	0.0	4.6	413	0.0
la02	10	5	394	394.0	0.0	3.6	394	0.0
la03	10	5	349	349.0	0.0	3.8	349	0.0
la04	10	5	369	369.0	0.0	3.8	369	0.0
la05	10	5	380	380.0	0.0	2.6	380	0.0
la06	15	5	413	413.0	0.0	17.4	413	0.0
la07	15	5	376	376.0	0.0	18.2	376	0.0
la08	15	5	369	369.0	0.0	19.6	369	0.0
la09	15	5	382	387.4	1.3	17.8	382	0.0
la10	15	5	443	443.0	0.0	17.0	443	0.0
la11	20	5	425	436.8	1.7	50.6	413	2.9
la12	20	5	408	408.0	0.0	44.6	408	0.0
la13	20	5	419	430.2	1.6	45.8	382	9.7
la14	20	5	443	448.8	1.2	48.8	443	0.0
la15	20	5	451	456.0	1.6	42.2	378	19.3
la16	10	10	717	717.0	0.0	36.0	717	0.0
la17	10	10	646	646.0	0.0	31.6	646	0.0
la18	10	10	663	663.0	0.0	36.8	663	0.0
la19	10	10	617	617.0	0.0	62.4	617	0.0
la20	10	10	756	756.0	0.0	34.2	756	0.0
mt06	6	6	47	47.0	0.0	1.0	47	0.0
mt10	10	10	655	655.0	0.0	50.0	655	0.0
mt20	20	5	439	442.6	0.6	48.2	387	13.4
Average					0.4	27.9		2.0

Table 10
IGA computational results: DFJS instances with four FMUs.

Inst.	Jobs	Ops.	MK	Av.	Dev%	T (s)	LB	Gap%
la01	10	5	413	413.0	0.0	1.8	413	0.0
la02	10	5	394	394.0	0.0	1.8	394	0.0
la03	10	5	349	349.0	0.0	2.2	349	0.0
la04	10	5	369	369.0	0.0	2.0	369	0.0
la05	10	5	380	380.0	0.0	1.0	380	0.0
la06	15	5	413	413.0	0.0	9.0	413	0.0
la07	15	5	376	376.0	0.0	9.6	376	0.0
la08	15	5	369	369.0	0.0	12.6	369	0.0
la09	15	5	382	382.0	0.0	11.6	382	0.0
la10	15	5	443	443.0	0.0	7.8	443	0.0
la11	20	5	413	413.0	0.0	29.6	413	0.0
la12	20	5	408	408.0	0.0	26.6	408	0.0
la13	20	5	382	386.0	9.9	27.6	382	0.0
la14	20	5	443	443.0	0.0	29.8	443	0.0
la15	20	5	397	402.0	2.3	28.8	378	5.0
la16	10	10	717	717.0	0.0	20.2	717	0.0
la17	10	10	646	646.0	0.0	16.4	646	0.0
la18	10	10	663	663.0	0.0	24.4	663	0.0
la19	10	10	617	617.0	0.0	33.0	617	0.0
la20	10	10	756	756.0	0.0	18.0	756	0.0
mt06	6	6	47	47.0	0.0	0.2	47	0.0
mt10	10	10	655	655.0	0.0	31.2	655	0.0
mt20	20	5	387	388.4	2.0	27.0	387	0.0
Average					0.6	16.2		0.2

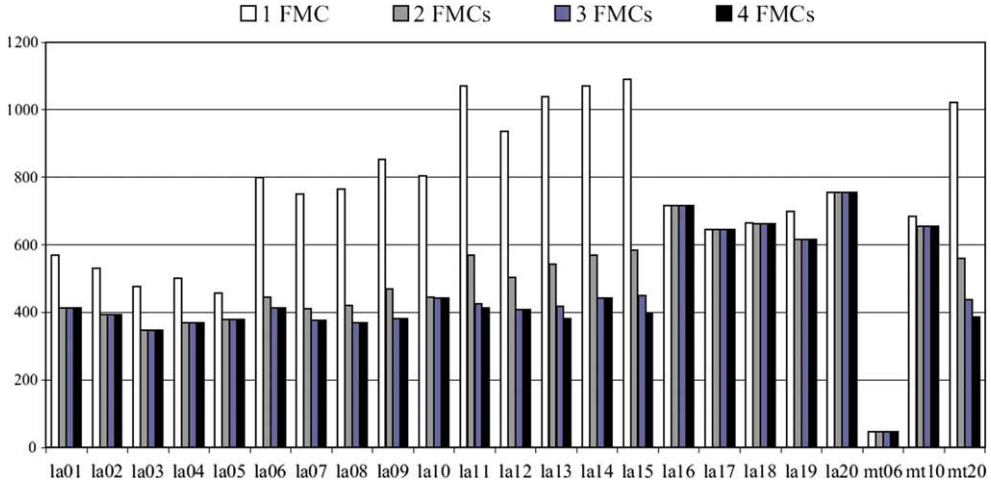

Fig. 12. IGA computational results: global makespans for DFJS instances with one two, three and four FMUs.

According to the tables above, IGA is very robust, as shown by the very small values of the standard deviation. This is mostly due to the refinement operator, able to make the algorithm converge towards a local minimum, if a near enough individual is generated. The IGA finds optimal or near-optimal solutions with reduced computational times: most of the instances are solved in a few seconds and in less than three minutes in the worst case. Note that computational times decrease as the number of FMUs increase: in fact, the average number of jobs per FMUs decrease and, as a consequence, the refinement operator (acting on just one critical FMU at a time) has to deal with a smaller number of genes, meaning that smaller swap-based neighborhoods have to be analyzed.

Fig. 12 summarizes the global makespans obtained by the IGA for the DFJS with two, three and four FMUs. For the special one-FMU case the best available results in literature are reported [25].

Results show that, in most of the tested instances, there is a relevant improvement of the global makespan when considering two FMUs instead of one, while the difference is not always appreciable when a third or a fourth FMU is considered. By providing similar reports, IGA could support the production management in deciding the number of FMUs to assign to each production lot.

5. Conclusions and future work

In this paper, the DFJS arising in systems of FMUs has been studied. Most of the literature about the scheduling of Flexible Manufacturing Systems at the operational level is concerned with the FJS, where a single FMU is considered and just two decisional levels are involved: choosing the most suitable multipurpose machine for each job operation (routing problem) and sequencing the operations assigned to each machine over time span (sequencing problem). FMUs systems give the opportunity of distributing the total workload of a production lot among several FMUs, so that a third decisional level has to be addressed by DFJS: choosing the FMU to process each job (job assignment problem).

The IGA presented in this paper is able to take the three decisional levels simultaneously into account and to provide a production scheduling aiming at minimizing the global makespan of FMUs systems. IGA starts from the same simple chromosome encoding of MGA [19] and extends it to the flexible case, thanks to a special decoding scheme able to choose among alternative machine routes. IGA exploits also a new mutation operator (refinement) to improve locally the best individuals of each generation by means of a neighborhood search based on gene swapping.

IGA has been compared with previous algorithm designed for distributed scheduling problems and tested on a large set of DFJS instances derived from well-known scheduling benchmarks, providing satisfactory results and proving the effectiveness of the new decoding scheme and refinement operator. The efficiency of the approach allows IGA to be used as the base for a tool to support FMUs systems scheduling.

The results provided by the genetic approach proposed in this paper are promising and further research on DFJS could extend it. For example, the greedy algorithm solving the job routing problem during the chromosome decoding phase may be improved, taking into account the trade-off with the efficiency of the decoding process in term of impact on the overall computational time. Another approach may decompose DFJS into two subproblems: the assignment of jobs to FMUs and the FJS related to each FMU. Based on this decomposition, a hybrid genetic procedure could be used where individuals would represent candidate job-to-FMU assignments. Genetic operators may be used to evolve the population and provide good hypothesis on job distribution among different FMUs. A neighborhood search procedure would evaluate the fitness of each individual by effectively solving the FJS related to each FMU. Finally, the definition of a better lower bound for DFJS is an open question. These lines are the object of ongoing research.

Acknowledgements

This work has been financed by EU Commission, within the cooperative research project SCOOP (Contract No. 032998) coordinated by Università Politecnica delle Marche – Ancona, Italy.

References

[1] J.E. Baker, Adaptive selection methods for genetic algorithms, in: Proceedings of the 1st International Conference on Genetic Algorithms (Mahwah, NJ, USA), Lawrence Erlbaum Associates, Inc., 1985, pp. 101–111.
[2] J.W. Barnes, J.B. Chambers, Flexible job shop scheduling by tabu search, Graduate Program in Operations Research and Industrial Engineering, Technical Report Series ORP96-09, The University of Texas at Austin, 1996.
[3] A.M. Barroso, J.R.A. Torreau, J.C.B. Leite, O.G. Loques, J.S. Fraga, A new technique for task allocation in real-time distributed systems, in: Proceedings of the 7th Brasilian Symposium of Fault Tolerant Computers, Campina Grande, Brazil, 1997, pp. 269–278.
[4] A. Baykasoglu, Linguistic-based meta-heuristic optimization model for flexible job shop scheduling, International Journal of Production Research 40 (17) (2002) 4523–4543.
[5] D. Beasley, D. Bull, R.R. Martin, An overview of genetic algorithms: Part 1, Fundamentals, University Computing 15 (2) (1993) 58–69.
[6] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research 22 (1993) 158–183.
[7] F.T.S. Chan, S.H. Chung, L.Y. Chan, G. Finke, M.K. Tiwari, Solving distributed FMS scheduling problems subject to maintenance: Genetic algorithms approach, Robotics and Computer-Integrated Manufacturing 22 (2006) 493–504.
[8] F.T.S. Chan, S.H. Chung, P.L.Y. Chan, An adaptive genetic algorithm with dominated genes for distributed scheduling problems, Expert Systems with Applications 29 (2005) 364–371.
[9] F.T.S. Chan, S.H. Chung, P.L.Y. Chan, Application of genetic algorithms with dominated genes in a distributed scheduling problem in flexible manufacturing, International Journal of Production Research 44 (3) (2006) 523–543.
[10] H. Chen, J. Ihlow, C. Lehmann, A genetic algorithm for flexible job-shop scheduling, in: IEEE International Conference on Robotics and Automation, Detroit, Michigan, 1999, pp. 1120–1125.
[11] I.C. Choi, D.S. Choi, A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups, Computers and Industrial Engineering Archive 42 (1) (2002) 43–58.
[12] V.A. Cicirello, F.S. Smith, Wasp-like agents for distributed factory coordination, Autonomous Agents and Multi-Agent Systems 8 (2004) 237–266.
[13] S. Dauzère-Pérès, J. Paulli, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, Annals of Operations Research 70 (1997) 281–306.
[14] M. Di Natale, J.A. Stankovic, Applicability of simulated annealing methods to real time scheduling and jitter control, in: Proceedings of the 16th IEEE Real-Time Systems Symposium, Pisa, Italy, 1995, pp. 190–199.
[15] H. Fisher, G.L. Thompson, Probabilistic learning combinations of local job shop scheduling rules, in: J.F. Muth, G.L. Thompson (Eds.), Industrial Scheduling, Prentice-Hall, Englewood Cliffs, NJ, 1963, pp. 225–251.
[16] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Mathematics of Operations Research 1 (1976) 117–129.
[17] N.B. Ho, J.C. Tay, GENACE: An efficient cultural algorithm for solving the flexible job-shop problem, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2004, pp. 1759–1766.
[18] E. Hurink, B. Jurisch, M. Thole, Tabu search for the job shop scheduling problem with multi-purpose machines, Operations Research Spektrum 15 (1994) 205–215.
[19] H.Z. Jia, J.Y.H. Fuh, A.Y.C. Nee, Y.F. Zhang, A modified genetic algorithm for distributed scheduling problems, Journal of Intelligent Manufacturing 15 (2003) 351–362.
[20] I. Kacem, S. Hammadi, P. Borne, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews 32 (1) (2002) 1–13.
[21] S. Lawrence, Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques, Tech. Report, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1984.
[22] D.Y. Lee, F. Di Cesare, Scheduling flexible manufacturing systems using Petri nets and heuristic search, IEEE Transactions on Robotics and Automation 10 (2) (1994) 123–132.
[23] M. Mastrolilli, <http://www.idsia.ch/~monaldo/fjsp.html>.
[24] M. Mastrolilli, L.M. Gambardella, Effective neighbourhood functions for the flexible job shop problem, Journal of Scheduling 3 (2000) 3–20.
[25] M. Mastrolilli, L.M. Gambardella, Effective neighbourhood functions for the flexible job shop problem: Appendix, Technical Report, IDSIA – Istituto Dalle

Molle di Studi sull'Intelligenza Artificiale, 2000. Electronic version available at: <http://www.idsia.ch/~monaldo/fjsp.html>.
[26] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Computers and Operations Research 35 (2008) 3202–3212.
[27] J. Santos, E. Ferro, J. Orozco, R. Cayssials, A realistic approach to the multitask–multiprocessor assignment problem using the empty-slots method and rate-monotonic scheduling, Journal of Real-time Systems 13 (1997) 167–199.
[28] M.J. Schniederjans, International Facility Acquisition and Location Analysis, Quorum Books, Westport, 1999.
[29] D.R. Sule, Logistics of Facility Location and Allocation, Marcel Dekker Inc., New York, NY, 1999.
[30] J.C. Tay, D. Wibowo, An effective chromosome representation for evolving flexible job shop schedules, in: GECCO, LNCS, vol. 3103, Springer Verlag, Berlin, 2004, pp. 210–221.
[31] K.W. Tindell, A. Burns, A.J. Wellings, Allocating hard realtime tasks: A NP-hard problem made easy, Journal of Real-time Systems 4 (1992) 145–165.
[32] R.J.M. Vaessens, E.H.L. Aarts, J. Lenstra, Job shop Scheduling by Local Search, COSOR Memorandum 94-05, Eindhoven University of Technology, 1994.
[33] W. Xia, Z. Wu, An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, Computers and Industrial Engineering Archive 48 (2) (2005) 409–425.