Int J Adv Manuf Technol (2010) 49:263–279
DOI 10.1007/s00170-009-2388-x

ORIGINAL ARTICLE

A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups

Fantahun M. Defersha · Mingyuan Chen

Received: 18 November 2008 / Accepted: 16 October 2009 / Published online: 2 December 2009

Abstract The flexible job-shop scheduling problem is an extension of the classical job-shop scheduling problem by allowing an operation to be assigned to one of a set of eligible machines during scheduling. Thus, the problem is to simultaneously assign each operation to a machine (routing problem), prioritize the operations on the machines (sequencing problem), and determine their starting times. The minimization of the maximal completion time of all operations is a widely used objective function in solving this problem. This paper presents a mathematical model for a flexible job-shop scheduling problem incorporating sequence-dependent setup time, attached or detached setup time, machine release dates, and time lag requirements. In order to efficiently solve the developed model, we propose a parallel genetic algorithm that runs on a parallel computing platform. Numerical examples show that parallel computing can greatly improve the computational performance of the algorithm.

Keywords Flexible job-shop scheduling · Sequence dependent setups · Attached/detached setup · Time lag · Machine release date · Genetic algorithm · Parallel computing

F. M. Defersha · M. Chen (✉)
Department of Mechanical and Industrial Engineering, Concordia University, 1455 de Maisonneuve W., Montreal, Quebec, Canada, H3G 1M8
e-mail: mychen@encs.concordia.ca

1 Introduction

In today's marketplace, a high level of delivery performance has increasingly become a tool to secure competitive advantages. This makes scheduling play a major role in manufacturing processes and in overall supply chain management. Job-shop scheduling problem (JSP) is a branch of production scheduling and is among the hardest combinatorial optimization problems [14]. It is one of the most popular manufacturing optimization models used in practice [21]. The classical JSP consists of n jobs to be processed on m unrelated machines. For each job, process planners specify a complete routing that will be fixed and known before scheduling is performed. Thus, the JSP problem is to determine the sequence and the starting times of the jobs on each machine. Recent studies on this class of problem can be found in Guo et al. [18], Wang et al. [37], Zhang et al. [41], and Tsai et al. [36]. The flexible job-shop scheduling problem (FJSP) extends JSP by considering routing flexibility during scheduling to achieve a better schedule. This extension integrated the JSP problem with the routing problem of assigning each operation to one of its alternative machines. Bruker and Schlie [4] were the first to address FJSP. They developed a polynomial algorithm for solving FJSP with two jobs. Since then, there has been a growing literature in FJSP. Recent publications include those by Chen et al. [7, 8], Kacem et al. [23], Xia and Wu [39], Saidi and Fattahi [34], Gao et al. [11, 13], Pezzella et al. [29], and Xing et al. [40], among others. Some of these studies were targeted at providing models and solution approaches to more realistic FJSP. However, it seems that there is limited effort towards providing a comprehensive solution approach where several practical considerations are jointly addressed. A comprehensive solution approach consisting of different aspects of the system can help one to understand the problem better. It can minimize the possibility of certain important aspects of the system being overlooked, while other issues are being studied. To this end, in this paper, we propose a comprehensive mixed integer mathematical programming (MILP) model and a parallel genetic algorithm (PGA) for FJSP by incorporating several factors. These factors include sequence-dependent setup time, attached or detached nature of setups, machine release dates, and time lags. These attributes are discussed below.

Sequence-dependent setup In many real-life situations, a setup operation is often required between operations, and it strongly depends on the immediate preceding operation on the same machine [2, 9]. Panwalkar et al. [28] noticed that significant portion of jobs required sequence-dependent setups in job scheduling. Wortman [38] argued that inadequate treatment of sequence-dependent setup time will hinder competitive advantage. Flow shop problems with sequence-dependent setups are extensively covered in the current literature [19, 24, 27, 30, 31, 33]. However, as pointed out in Manikas and Chang [26], research on job-shops scheduling with sequence-dependent setups has been limited. This is more apparent in FJSP due to the complexity of the problem. In this paper, we considered sequence-dependent setup time in FJSP.

Attached or detached setup A setup of a particular operation is attached (non-anticipatory) if it is assumed that setting up the machine for this operation can be performed once the preceding operation of the job is completed and the next job arrives at the machine. When a setup is performed prior to the arrival of the job, the setup is called detached (or anticipatory). In this case, the setup time can overlap with the processing time of the preceding operation if these two consecutive operations are not assigned on the same machine. Allahverdi et al. [1] surveyed more than 300 papers published in recent years that consider setup. In their review paper, we noted that, with exception of a few of them, the majority of the authors unanimously assumed attached setup. However, Zhang and Gu [42] pointed out that the assumption of attached setup times may hinder maximal concurrency and, thus, adversely affect the solution quality in many occasions. In this paper, we propose a mathematical programming model and a solution procedure for FJSP where the setup of each operation can be treated as either attached or detached, depending on the actual requirements set by the process planner.

Machine release date Most scheduling research assumes that all machines are available at time zero. However, it is a common situation in industry that one may have to consider ongoing operations from previous schedules as production environments are seldom found empty [32]. When routing flexibility is considered during scheduling, the choice of an alternative machine can be affected by its release date since the machine that will be released soon may represent a better choice. In this research, we consider the FJSP problem where there can be alternative machines with different release dates to process an operation.

Time lag Time lag is the delay between the completion of an operation and the starting time of the next operation of the same job. Such positive time lag may occur when, for example, drying or cooling of products must be performed before further operations can take place. On the other hand, negative time lag may occur when not all items of a job need to be finished on a given machine before processing of the next operation starts on another machine [32]. This is particularly true if the batch size is very large and there is a need to transfer a portion of the batch to the next machine. In the model developed in this paper, time lag is allowed for each operation and can be either negative or positive depending on the actual requirements.

Solving classical JSP without the above attributes is known to be NP-hard [14]. This entails that the comprehensive model proposed in this paper, with all the above features, is even more NP-hard. Several authors used genetic algorithm to solve FJSP models with few of the above features (see [7, 11, 13, 29]). Using relatively smaller problems, these authors demonstrate that genetic algorithm is generally able to find good solutions in reasonable amounts of computing time for FJSP problems. Genetic algorithm has also been extensively applied on classical job shop, flow shop, and assembly line problems in several articles in the past and, most recently, in Jolai et al. [22], Guo et al. [15–18], Gao et al. [12], Wang et al. [37], Zhang et al. [41], and Tsai et al. [36]. However, as it is applied to larger and more complex problems, there is an increase in using CPU time and computer memory to find adequate solutions. Moreover, in such large and complex problems, there can be a larger probability for the search process being trapped in local optima. In such situations, the most promising choice is to use parallel implementations of the algorithms. From the early days of its development, the genetic algorithm's (GA's) potential for parallelization has been noted with all its attendant benefits of efficiency. Several authors have applied PGAs in diverse domains.

We also notice that, in addition to using expensive multiprocessor supercomputers, parallel computing can be implemented on a cluster of less expensive personal computers connected through a low-cost network and using public domain software. Also, access to high-performance parallel computing is nowadays available on pay-per-use basis, a business model called utility computing. It was as early as 1990 when Baxter [3] reported that commercial on-demand access to parallel computing is available at several government- and private-sector installations in the USA. In its November 2005 press release, IBM reported that it started commercial access to high-performance computing in which clients worldwide can have on-demand access to over 5,200 CPUs (http://www-03.ibm.com/press/us/en/pressrelease/7949.wss). Tsunamic Technology (http://www.clusterondemand.com/), Sun Microsystems (http://www.network.com/), and 3Tetra (http://www.3tera.com/) are also examples of commercial-access providers to parallel computing. Moreover, several universities and research institutes owe parallel computing facilities. Despite such availability of access to parallel computing, there are limited reports on the use of parallel computing in production research. For instance, out of the 178 papers reviewed in Chaudhry and Luo [6] on the application of GAs in production and operations management, only three of them reported the use of parallel computing. Literature on using parallel computing in job-shop scheduling in particular is also sparse despite its well-perceived computational difficulty. This paper contributes to the literature by providing a comprehensive model for FJSP and reporting the use of parallel computation in solving this difficult problem. The rest of this paper is organized as follows: In Section 2 we present the proposed MILP model for FJSP. The PGA is detailed in Section 3. Numerical examples are give in Section 4. Discussion and conclusion are in Section 5.

2 Mathematical formulation

In this section, we present the proposed MILP model for FJSP. The model can be used to optimally solve small-size problems using an off-the-shelf optimization package (e.g., CPLEX [20]). These optimal solutions can also be used to validate the computer implementation of the heuristic solution procedure by comparing them with those generated by the heuristic method.

2.1 Problem description and notations

Consider a job-shop consisting of M machines where certain machines are the same or have some common functionalities. The system is processing jobs of previous schedules and each machine has a release date D_m when it will be available for processing jobs of the current schedule. Consider also a set of J independent jobs to be scheduled in the system. Job j is formed by O_j number of operations in a fixed sequence, where each operation o has a set of eligible alterative machines. An operation o of job j can be processed on machine m with a processing time $B_j \times T_{o,j,m}$, where B_j is the batch size and $T_{o,j,m}$ is the unit processing time. This operation can be started on machine m after a delay (lag time) $L_{o,j}$ from the completion time of operation $o-1$, and after setup is performed on this machine. This setup time is sequence-dependent and denoted by $S_{o,j,m,o',j'}$, where operation o' of job j' is the last operation processed on machine m. If operation o of job j is the first operation to be processed on machine m, the setup time is represented by $S^*_{o,j,m}$. The setup time $S_{o,j,m,o',j'}$ (or $S^*_{o,j,m}$) can be overlapped with the processing time of operation $o-1$ if the setup is detached and machine m is available for setup. The problem is to assign each operation to one of its alternative machines and to determine the sequence and starting time of the operations on each machine. The objective is minimizing the makespan of the schedule. We next introduce some additional notations and then present the MILP formulation for FJSP.

Additional parameters:

R_m	Maximum number of production runs of machine m where production runs are indexed by r or $u = 1, 2, ..., R_m$. The assignment of operations to production runs determines their sequence.
$P_{o,j,m}$	A binary data equal to 1 if operation o of job j can be processed on machine m, 0 otherwise.
$A_{o,j}$	A binary data equal to 1 if the setup of operation o of job j is attached (non-anticipatory), or 0 if this setup is detached (anticipatory).
Ω	Large positive number.

Variables:

Continuous variables:

$c_{o,j,m}$	Completion time of operation o of job j on machine m
$\widehat{c}_{r,m}$	Completion time of the r^{th} run of machine m
$c_{\max}$	Makespan of the schedule

Binary variables:

$x_{r,m,o,j}$	Binary variable that takes the value 1 if the r^{th} run on machine m is for operation o of job j, 0 otherwise
$z_{r,m}$	A binary variable that is equal to 1 if the r^{th} potential run of machine m has been assigned to an operation, 0 otherwise

2.2 MILP model

Following the problem description and using the notation given above, the MILP model for the FJSP is presented below.

Objective
Minimize:

$$Z = c_{\max} \tag{1}$$

Subject to:

$$c_{\max} \geq c_{o,j,m} \;; \quad \forall(o, j, m) \tag{2}$$

$$\widehat{c}_{r,m} \geq c_{o,j,m} + \Omega \cdot x_{r,m,o,j} - \Omega \;; \quad \forall(r, m, o, j) \tag{3}$$

$$\widehat{c}_{r,m} \leq c_{o,j,m} - \Omega \cdot x_{r,m,o,j} + \Omega \;; \quad \forall(r, m, o, j) \tag{4}$$

$$\widehat{c}_{1,m} - B_j \cdot T_{o,j,m} - S^*_{o,j,m} - \Omega \cdot x_{1,m,o,j} + \Omega \geq D_m \;; \quad \forall(m, o, j) \tag{5}$$

$$\widehat{c}_{r,m} - B_j \cdot T_{o,j,m} - S_{o,j,m,o',j'} - \Omega \cdot (x_{r,m,o,j} + x_{r-1,m,o',j'}) + 2\Omega \geq \widehat{c}_{r-1,m} \;; \quad \forall(r, m, o, j, o', j')|\{(r > 1) \wedge ((o, j) \neq (o', j'))\} \tag{6}$$

$$\widehat{c}_{1,m} - B_j \cdot T_{o,j,m} - S^*_{o,j,m} \cdot A_{o,j} - \Omega \cdot (x_{1,m,o,j} + x_{r',m',o-1,j}) + 2\Omega \geq \widehat{c}_{r',m'} + L_{o,j} \;; \quad \forall(m, r', m', o, j)|\{\left((1, m) \neq (r', m')\right) \wedge (o > 1)\} \tag{7}$$

$$\widehat{c}_{r,m} - B_j \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \Omega \cdot (x_{r-1,m,o',j'} + x_{r,m,o,j} + x_{r',m',o-1,j}) + 3\Omega \geq \widehat{c}_{r',m'} + L_{o,j} \;; \quad \forall(r, m, r', m', o, j, o', j')|\{(r > 1) \wedge (o > 1) \wedge (r, m) \neq (r', m') \wedge (o, j) \neq (o', j')\} \tag{8}$$

$$x_{r,m,o,j} \leq P_{o,j,m} \;; \quad \forall(r, m, i) \tag{9}$$

$$\sum_{m=1}^{M} \sum_{r=1}^{R_m} x_{r,m,o,j} = 1 \;; \quad \forall(o, j) \tag{10}$$

$$\sum_{j=1}^{J} \sum_{o=1}^{O_j} x_{r,m,o,j} = z_{r,m} \;; \quad \forall(r, m, i) \tag{11}$$

$$z_{r+1,m} \leq z_{r,m} \;; \quad \forall(r, m, i) \tag{12}$$

$$x_{r',m,o',j} \leq 1 - x_{r,m,o,j} \;; \quad \forall(r, r'm, o, o'j)|\{(o' > o) \wedge (r' < r)\} \tag{13}$$

$$x_{r',m,o',j} \leq 1 - x_{r,m,o,j} \;; \quad \forall(r, r'm, o, o'j)|\{(o' < o) \wedge (r' > r)\} \tag{14}$$

$$x_{r,m,o,j}, \text{ and } z_{r,m} \text{ are binary} \tag{15}$$

The objective function in Eq. 1 is to minimize the makespan of the schedule. Constraint Eq. 2, along with the objective function, determines the makespan. Constraints Eqs. 3 and 4 together state that the completion time of the o^{th} operation of job j is equal to the completion time of the r^{th} run of machine m, if this production run is assigned to that particular operation. The staring time of the setup for the first run ($r = 1$) of machine m is given by $\widehat{c}_{1,m} - B_j \cdot T_{o,j,m} - S*_{o,j,m}$, if the o^{th} operation of job j is assigned to this first run. This starting time cannot be less than the release date D_m of the machine as enforced by the constraint in Eq. 5. Constraint Eq. 6 is to enforce that the setup of any production run $r > 1$ of a given machine cannot start before the completion time of run $r - 1$ of that machine. Constraint Eq. 7 states that, for any pair of machines (m, m'), the setup (if $A_{o,j} = 1$) or the actual processing (if $A_{o,j} = 0$) of the first run on machine m cannot start before the completion of run r' of machine m' plus the lag time $L_{o,j}$. This constraint is applied if run r' of machine m' is assigned to operation $o - 1$ of job j and the first run of machine m is assigned to operation o of this same job. Constraint in Eq. 8 is similar to Eq. 7, except that Eq. 8 is for run $r > 1$ of machine m where setup time depends on the operation assigned to run $r - 1$. Constraints Eqs. 9 and 10 are to restrict the assignment of an operation to only one production run of one of the elidible machine. Each production run of a given machine can be assigned to, at most, one operation (Eq. 11), and production run $r + 1$ can be assigned to an operation if and only if run r of that machine is already assigned (Eq. 12). Constraint Eq. 13 enforces that, if operation o of job j is assigned to a production run r of machine m, any upcoming operation o' of job j cannot be assigned to any earlier run r' of machine m. Constraint Eq. 14 is symmetric to constraint Eq. 13. It states that, if an operation of a job is assigned to a production run of a machine, any earlier operation of that job cannot be assigned to any upcoming production run of that machine. These two constraints are redundant as such requirements are indirectly imposed by constraints Eqs. 7 and 8. The purpose of including constraints Eqs. 13 and 14 in the model is to speed up the branch and bound search process if the model is to be solved by such methods. Integrality requirements on the variable $x_{r,m,o,j}$ and $z_{r,m}$ are given in Eq. 15. As the classical JSP problem without sequence pendent setup time and routing flexibility is NP-hard [14], solving the above MILP model for FJSP is also NP-hard. In the next section, a GA-based meta-heuristic is presented for solving the problem efficiently.

3 Genetic algorithm

A typical GA consists of an initial population, genetic operations, and fitness evaluation. These elements of the proposed GA and its parallel implementation are presented below.

3.1 Initial population

The initial population is generated based on the procedure presented in Pezzella et al. [29], which in turn is based on the localization approach of Kacem et al. [23]. This approach takes into account both the processing times and the workload of the machines, i.e., the sum of the processing times of the operations assigned to each machine. The procedure proceeds in finding, for each operation, the machine with the minimum workload. We further modify this approach to account for setup time and machine release date. Consider a small example problem with a four-machine job-shop to process three jobs where each operation can be processed by any one of the machines. The data of this example problem are given in Table 1. This table contains the average setup time and the processing time of each operation corresponding to each machine, and the release time of each machine. For initial assignment, we consider the average setup time for each operation on each machine since the actual setup time is sequence-dependant and not known in advance. The procedure in Pezzella et al. [29] begins by randomly permutating the jobs and the machines to generate an initial workload matrix as shown in the four columns of Table 2 under the heading "Initial data." This provides the initial workload of the machines. The procedure then proceeds in finding, for each operation, the machine with the minimum workload, fixing that assignment, and then adding the processing time to every subsequent entry in the same column, as shown in Table 2, where bold values correspond to workload updates. The last four columns of Table 2 show the final operations assignment. This assignment procedure assumes that there are no setup times and all machines are available at time zero. If there are considerable setup time variations among the alternative machines and not all machines are available at the same time, this procedure may not yield a good initial solution. We modify this procedure by using the sum of the average setup time, the processing time, and the machine release date as the initial workload matrix as shown in the four columns of Table 3 under the heading "Initial data." Then, we proceed in a similar fashion as the procedure outlined in Pezzella et al. [29] with the exception that we update the workload by adding the processing time, as well as the average setup time to every subsequent entry. In the last four columns of Table 3, it can be seen that machine M2 has been assigned the most number of operations as this machine is available from the previous schedule earlier than other machines. Moreover, machine M4, which is not immediately available than any other

Table 1 Average setup time, processing time, and machine release date

Job	Operation	Machine			
J	O	M1	M2	M3	M4
		(Average setup time, processing time)			
1	1	(10, 20)	(10, 50)	(15, 10)	(10, 30)
	2	(10, 40)	(15, 60)	(15, 80)	(10, 40)
	3	(15, 90)	(10, 70)	(15, 20)	(10, 20)
2	1	(25, 70)	(20, 60)	(20, 40)	(20, 50)
	2	(10, 40)	(10, 80)	(30, 50)	(20, 60)
	3	(15, 90)	(20, 50)	(15, 40)	(15, 70)
3	1	(10, 80)	(15, 60)	(25, 30)	(15, 50)
	2	(15, 30)	(15, 50)	(20, 80)	(15, 30)
Machine release date		70	0	90	130

Table 2 Initial assignments considering processing time (machine workload updates in bold)

J	O	Initial data				1st assignment				2nd assignment				···	Final assignment			
		M4	M1	M3	M2	M4	M1	M3	M2	M4	M1	M3	M2		M4	M1	M3	M2
3	1	50	80	30	60	50	80	$\boxed{30}$	60	50	80	30	60		50	80	$\boxed{30}$	60
	2	30	30	80	50	30	30	**110**	50	$\boxed{30}$	30	110	50		$\boxed{30}$	30	80	50
2	1	50	70	40	60	50	70	**70**	60	**80**	70	70	60		50	70	40	$\boxed{60}$
	2	60	40	50	80	60	40	**80**	80	**90**	40	80	80	···	60	$\boxed{40}$	50	80
	3	70	90	40	50	70	90	**70**	50	**100**	90	70	50		70	90	$\boxed{40}$	50
1	1	30	20	10	50	30	20	**40**	50	**60**	20	40	50		$\boxed{30}$	20	10	50
	2	40	40	80	60	40	40	**110**	60	**70**	40	110	60		40	$\boxed{40}$	80	60
	3	20	90	20	70	20	90	**50**	70	**50**	90	50	70		$\boxed{20}$	90	20	70

machine has been assigned the list number of operations. We refer to the above assignment rule as Assignment-Rule-1. Another assignment procedure, Assignment-Rule-2, that does not require the random permutation of the jobs and the machines used in Pezzella et al. [29] has also been used in our solution procedure. It looks for the global minimum of the workloads in the workload matrix in order to choose the operation to be assigned next. An assignment rule choice (ARC) parameter is used to decide how much proposition of the initial population is generated using Assignment-Rule-1, where $0 < \text{ARC} < 1$. The rest of the initial population is generated by using Assignment-Rule-2.

The next step is to determine the sequence of the operations on the machines. The sequencing of the initial assignments can be obtained by a mix of three known dispatching rules: (a) randomly select a job (RSJ), (b) most work remaining (MWR), and (c) most number of operations remaining (MOR). In using sequencing rule RSJ, one needs to respect the precedence constraints among operations of the same job if they are assigned to the same machine. In using the MWR rule, we consider both the average setup time and the processing time in calculating the remaining work for each operation. Based on the operation assignment in Table 3, the remaining work calculation and the operation sequence of an initial solution of the small example is given in Table 4.

3.2 Solution encoding and genetic operators

In solving the proposed FJSP model using PGA, we use the solution encoding and the genetic operators similar to those in Pezzella et al. [29], Kacem et al. [23], and Lee et al. [25].

3.2.1 Solution encoding

The solution representation technique presented in Pezzella et al. [29] is used in our paper. In this representation, a chromosome is composed of serval genes, one for each operation, and each gene is formed by a triplet (j, o, m), where m can assume the index of an alternative machine on which an operation o of job j can be assigned. The sequence of the genes in the

Table 3 Initial assignments considering processing time, average setup time, and machine release date (machine workload updates in bold)

J	O	Initial data				1st assignment				2nd assignment				···	Final assignment			
		M4	M1	M3	M2	M4	M1	M3	M2	M4	M1	M3	M2		M4	M1	M3	M2
3	1	195	160	145	75	195	160	145	$\boxed{75}$	195	160	145	75		195	160	145	$\boxed{75}$
	2	175	115	190	65	175	115	190	**140**	175	$\boxed{115}$	190	140		175	$\boxed{115}$	190	65
2	1	180	115	120	70	180	115	120	**145**	180	**160**	120	145		180	115	$\boxed{120}$	70
	2	190	120	200	70	190	120	200	**145**	190	**165**	200	145	···	190	120	200	$\boxed{70}$
	3	165	175	125	90	165	175	125	**165**	165	**220**	125	165		165	175	$\boxed{125}$	90
1	1	190	150	145	70	190	150	145	**145**	190	**195**	145	145		$\boxed{190}$	150	145	70
	2	200	120	155	95	200	120	155	**170**	200	**165**	155	170		200	$\boxed{120}$	155	95
	3	210	175	145	60	210	175	145	**135**	210	**220**	145	135		210	175	145	$\boxed{60}$

Table 4 Remaining work calculation and operations sequence for an initial solution based on the operation-machine assignment given in Table 3

Work remaining (WR) calculation			Operation sequence			
Job J	Operation O	WR	Machine	Run 1	Run 2	Run 3
1	1	$(10 + 50) + (10 + 40) + (10 + 50) = 170$	M1	J1,O2	J3,O2	
	2	$10 + 40) + (10 + 50) = 110$				
	3	$(10 + 50) = 60$	M2	J3,O1	J2,O2	J1,O3
2	1	$(20 + 10) + (10 + 60) + (15 + 20) = 135$	M3	J2,O1	J2,O3	
	2	$(10 + 60) + (15 + 20) = 105$				
	3	$(10 + 50) = 60$	M4	J1,O1		
3	1	$(15 + 60) + (15 + 30) = 120$				
	2	$(15 + 30) = 45$				

chromosome represents the sequence of the jobs in the machines. Figure 1, for example, illustrates the chromosomal encoding of the assignment and sequencing of the initial solution for the small example as indicated in the last three columns of Table 4.

3.2.2 Genetic operators

GAs use selection, crossover, and mutation operators. The selection is to choose the chromosomes for reproduction. In the proposed PGA, a k-way tournament selection method is used to first select k individuals randomly. Then, the individual representing the highest fitness (smallest makespan) is the winner. This process is repeated with replacement until the number of selected individuals is equal to the population size. Once the chromosomes are selected for reproduction, crossover and mutation operators are applied to produce the offsprings. These operators can be categorized as assignment operators and sequencing operators. Assignment operators only change the assignment property of the chromosomes. These operators are: (1) assignment crossover, (2) assignment mutation, and (3) intelligent mutation. Assignment crossover generates offsprings by exchanging the assignment of arbitrarily chosen operations between two parents as illustrated in Fig. 2. This operator is applied on each pair of parents with probability ρ_1. Assignment mutation alters the assignment of few operations of a given individual chromosome with a small probability ρ_2. In the intelligent mutation, we randomly select an operation on the machine with maximum workload and assign it to a compatible machine with minimum workload. An individual chromosome may undergo this intelligent mutation with probability ρ_3.

(j, o, m)							
(1, 1, 4)	(2, 1, 3)	(3, 1, 2)	(1, 2, 1)	(2, 2, 2)	(3, 2, 1)	(1, 3, 2)	(2, 3, 3)

Fig. 1 Solution representation

Sequencing operators only change the sequence of the operations in the parent chromosomes. We use two such operators in the proposed PGA. These are the precedence preserving order-based crossover (POX) of Kacem et al. [23] and precedence preserving shift mutation (PPS) operators of Lee et al. [25]. With probability ρ_4, POX generates two children from two parents as illustrated in Fig. 3. First, it randomly selects two operations, one from each parent, and copies to the respective child all the operations of the job to which the selected operation belongs. Then, it completes the new individual with the remaining operations, in the same order as they appear in the second parent, while maintaining the assignment property of the operations in the first parent. This operator respects the precedence constraint among operations of the same job. PPS selects an operation from a single parent chromosome and moves it into another position, taking care of the precedence constraints for that operation. This operator is applied with a small probability ρ_5.

3.3 Fitness evaluation

Based on assignment and sequencing information obtained from a chromosome, the corresponding staring and completion times of jobs and, hence, the makespan of the schedule can be calculated. The makespan is used as a measure of fitness of this individual. In calculating the makespan, we take into account: (1) the dependance of setup time on sequence, (2) the nature of the setup (attached or detached), (3) time lag requirement of certain operations, and (4) machine release dates. The procedure is outlined below.

Step 1. Set $l = 1$.

Step 2. Set the values of indices j, o, and m as obtained from the gene at location l of the chromosome under consideration.

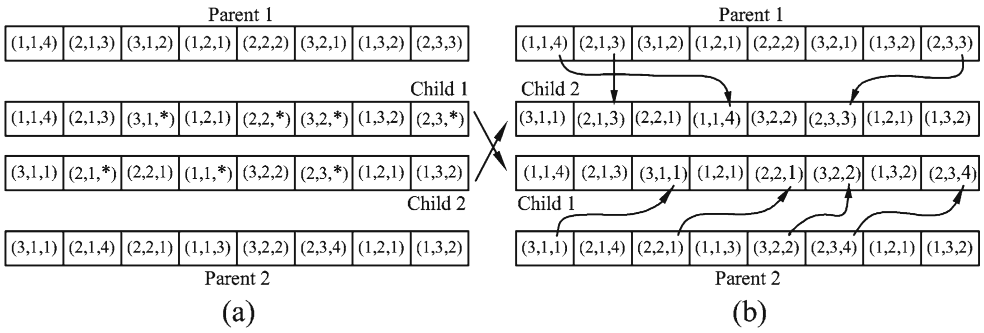

Fig. 2 Assignment crossover: **a** copy all the genetic material of the parents to their respective child except the assignment property of the randomly selected operations as indicated by *asterisks*, and then **b** get the assignment property of these randomly chosen operations from the other parent

Step 3. Calculate the completion time $c_{o,j,m}$.

- If (1) operation o of job j is the first operation assigned on machine m and (2) $o = 1$, then $c_{o,j,m} = D_m + S^*_{o,j,m} + B_j \cdot T_{o,j,m}$.
- If (1) operation o of job j is the first operation assigned on machine m, (2) $o > 1$, and (3) operation o-1 is assigned on machine m', then $c_{o,j,m} = \max\{D_m + (1 - A_{o,j}) \cdot S^*_{o,j,m}; c_{o-1,j,m'} + L_{o,j}\} + B_j \times T_{o,j,m} + A_{o,j} \cdot S^*_{o,j,m}$.
- If (1) operation o' of job j' is the operation to be processed immediately before operation o of job j on machine m and (2) $o = 1$, then $c_{o,j,m} = c_{o',j',m} + S_{o,j,m,o',j'} + B_j \cdot T_{o,j,m}$.
- If (1) operation o' of job j' is the operation to be processed immediately before operation o of job j on machine m, (2) $o > 1$, and (3) operation $o - 1$ is assigned on machine m', then $c_{o,j,m} = \max\{c_{o',j',m} + (1 - A_{o,j}) \cdot S_{o,j,m,o',j'}; c_{o-1,j,m'} + L_{o,j}\} + B_j \cdot T_{o,j,m} + A_{o,j} \cdot S_{o,j,m,o',j'}$.

Step 4. If l is less than the total number of operations, increase its value by 1 and go to step 2; otherwise, go to step 5

Step 5. Calculate the makespan of the schedule as $c_{\max} = \max\{c_{o,j,m};\ \forall(o, j, m)\}$ and set the fitness of the individual under consideration to $c_{\max}$.

3.4 Parallel implementation

Sequential GAs (SGAs) have been successful in many applications of different domains. However, in solving complex optimization problems of large-size problems, the population size needs to be large and a considerable amount of computer memory is required for data and information processing. In addition, SGAs are likely to be trapped in sub-optimal regions as the search is limited to a single population. The need to alleviate such difficulties has resulted in GAs being implemented on parallel computing platforms. Research in parallel GA (PGA) has led to several paradigms for how populations are evolved in parallel. One common method is the island-model approach. This approach suits well the problem considered in this research. The island-model PGA uses multiple sub-populations maintained by different processors. Each processor executes a SGA on its subpopulation. Periodic exchanges of individuals among the different sub-populations are performed through migration. In this parallelization technique, there are several known topologies in the literature for connecting the sub-populations (see, for example, [5]),

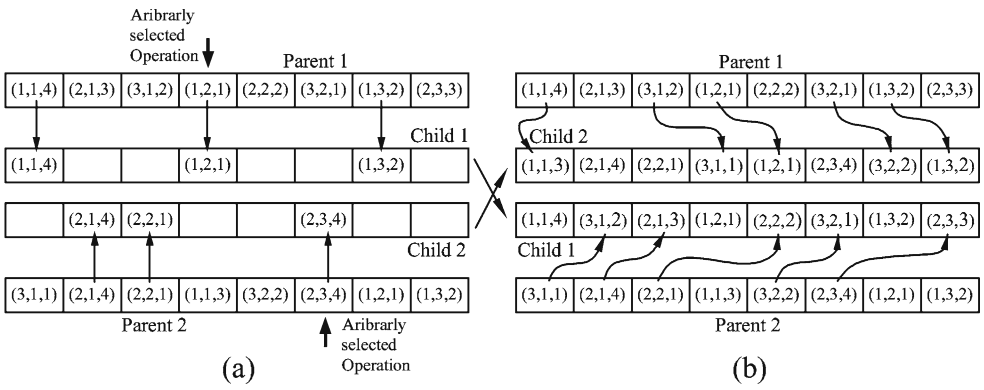

Fig. 3 POX: **a** arbitrarily select one operation from the each parent and copy to the respective child all the operations of the job to which the selected operation belong, and then **b** complete the new individual with the remaining operations, in the same order as they appear in the second parent while maintaining the assignment property of the operations in the first parent

such as ring, mesh, fully connected, and randomly connected topologies and their variations. Defersha and Chen [10] proposed a randomly connected topology having a parameter to control the degree of connectivity in solving a comprehensive manufacturing cell formation model. In this research, we adopt the same approach in parallelizing the proposed GA to solve the general FJSP model. This topology employs randomly generated migration routes for each communication epoch. One of the processors will coordinate the communication in addition to evolving its own subpopulation. This processor will randomly generate a communication matrix and broadcast it to all other processors before each communication epoch. The element $a_{i,j}$ in the communication matrix equals 1 if migrants are to be sent from processors i to j. The value of $a_{i,j}$ is determined using Eq. 16, where "rand()" is a random number generator. The parameter $\rho \in [0, 1]$ is used to control the density of the communication topology. Figure 4 shows examples of communication matrices and the corresponding topologies for two different values of ρ.

$$a_{i,j} = \begin{cases} 1 & \text{rand()} < \rho \text{ and } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

Another important factor of the island model PGA is the migration operator, which controls the migration of the individuals. This operator has a number of attributes, including: (1) the number of individuals undergoing migration, (2) the frequency of migration in numbers of generations, and (3) the migration policy directing the type of individuals (best, according to fitness, random, etc.) from one subpopulation to migrate to another. It also directs the type of individuals (worst, random, etc.) to be replaced. The incoming individuals are mixed with the destination subpopulation after selection but before the crossover and mutation operators are applied. In this research, we observed that the impact of the migration operator on the proposed PGA for FJSP are more or less similar to that observed in [10] in solving the cellular manufacturing problem.

4 Numerical examples

In this section, we present an example problem to illustrate the features of the proposed model and the computational performance of the developed PGA.

	0	1	2	3	4	5	6
0		0	0	0	0	0	1
1	0		1	0	0	1	0
2	1	0		1	0	0	0
3	1	0	1		0	1	0
4	0	0	0	1		0	0
5	1	0	0	0	0		0
6	0	0	0	0	0	1	

	0	1	2	3	4	5	6
0		1	1	1	0	1	0
1	1		1	0	1	1	1
2	0	1		1	1	0	1
3	1	1	1		1	0	1
4	1	1	1	0		1	1
5	1	0	1	0	1		1
6	1	1	1	1	1	0	

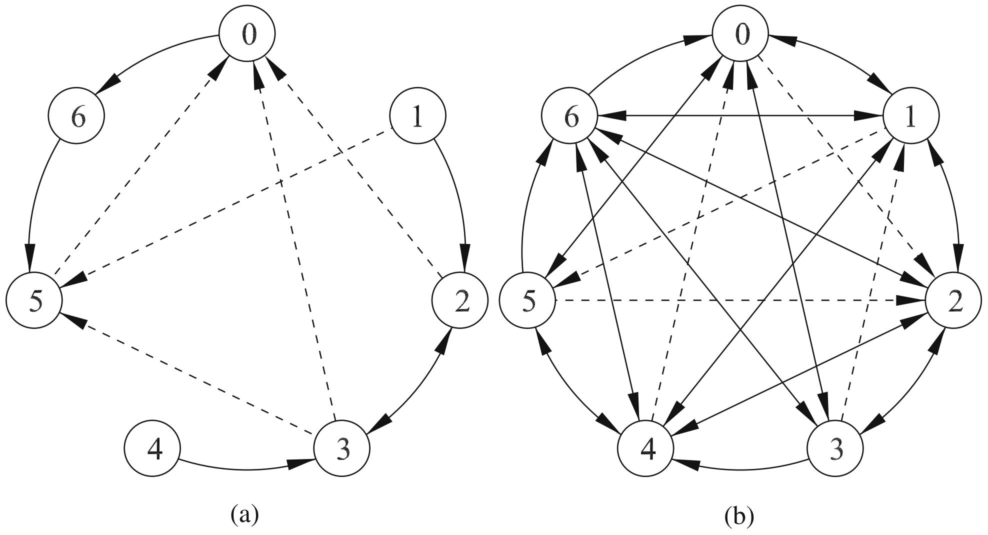

Fig. 4 Communication matrices and the corresponding topologies for different values ρ (adopted from [10]). **a** Generated for $\rho = 0.2$. **b** Generated for $\rho = 0.8$

4.1 Model illustration

A small problem instance consisting of the processing of five jobs in a four-machine flexible job-shop is considered. The batch size of each job is given in Table 5. This table also contains, for each operation, the nature of setup (attached or detached), lag time, indices of the alternative machines, and corresponding processing times. Table 6 is for the sequence-dependent setup time. This small example problem was solved to optimality using ILOG CPLEX for two different cases. In the first case, it was assumed that all the machines were available from time zero, while, in the second case, machine 2 was not available in the first 750 min. The Gantt charts of the resulting schedules are given in Fig. 5. The detail numerical values of the starting and the ending times of the setups and the operations are given in Table 7.

As shown in Table 5, job 1 has operations with detached setups, allowing overlapping of the setup and the processing of successive operations. Such overlapping can be observed in the Gantt charts in Fig. 5. In case 1, setup of the second operation of job 1 on machine 1 overlaps with the processing of the first operation on machine 3, and the setup of the third operation on machine 3 overlaps with the processing of the second operation on machine 1. For the rest of the jobs, attached setups were assumed, and hence, no such overlaps occur. Negative and positive time lags are also illustrated in this example. Negative time lag was assumed for the second operation of job 4 as the batch size of this job is very large and there may be a need to split the batch. In case 1, the setup of the second operation of this job started at 865 min. This implies that the first $(865 - 40)/2.5 = 330$ items of the batch were transferred from machine 4 to machine 1 before the whole batch is completed and the remaining 160 items were transferred immediately after the operation on machine 4 is completed. This presents an overlap between the second operation on machine 1 and the first operation on machine 4. A positive time lag was assumed for the third operation of job 3. As can be seen from the first Gantt chart in Fig. 5, setup and processing of the third operation of this job can start at 1,365 min without the positive time lag. However, because of the 300-min time lag requirement, the setup of the third operation began at time 1,665 min. The schedule shown in the second Gantt chart in Fig. 5 assumes that machine 2 is not available in the first 750 min. This results in a quite different schedule. For example, five operations were assigned to machine 2 in case 1, whereas, in case 2, only one operation is assigned to this machine. This clearly shows that, for flexible job-shops, assignment and operation sequence are greatly affected by machine release date.

Table 5 Processing data for jobs

j	B_j	o	$A_{o,j}$	$L_{o,j}$	Alternative routes, $(m, T_{o,j,m})$		
					1	2	3
1	70	1	na	na	(2, 1.25)	(3, 2.00)	
		2	0	0	(1, 3.00)	(3, 2.75)	
		3	0	0	(1, 3.50)	(2, 4.00)	(3, 3.75)
		4	0	0	(2, 3.50)	(4, 3.75)	
2	80	1	na	na	(1, 2.50)	(2, 2.75)	(4, 2.50)
		2	1	0	(1, 3.50)	(3, 3.25)	
		3	1	0	(3, 3.00)	(4, 2.50)	
3	60	1	1	0	(2, 2.75)	(4, 2.50)	
		2	1	0	(1, 2.75)	(2, 2.25)	(4, 2.75)
		3	1	300	(1, 2.50)	(2, 3.00)	
4	490	1	na	na	(1, 2.25)	(2, 2.75)	(4, 2.50)
		2	1	−400	(1, 2.25)	(2, 2.50)	(4, 2.50)
5	80	1	na	na	(1, 2.75)	(2, 2.50)	
		2	1	0	(2, 1.75)	(4, 1.75)	
		3	1	0	(1, 2.25)	(3, 2.50)	(4, 2.75)

na not applicable

4.2 Computational performance

The island-model PGA was coded in C++ using MPI for communication (for details of MPI, see [35]). The code was tested using up to 48 processors in a parallel computation environment with more than 800 computers and each having a P4 processor (3.2 GHz, 2 GB RAM). We generated and solved several medium- to large-size problem instances with different sets of genetic parameters. The genetic parameters given in Table 8 were generated randomly around those values with which the algorithm performed well. The initial sets of the parameters were chosen following the general guidelines provided in Pezzella et al. [29] and in most other published GAs. General features of the considered test problems, such as number of machines, number of products, total number of operations, and total number of alternative routes are given in Table 9. The sizes of these problems are considerably larger than most problem instances that appear in literature. Table 10 shows the time required by the PGA or SGA to advance the iteration by one generation in solving the test problems under different population sizes.

Table 6 Sequence-dependent setup time data

j	o	m	Setup time $(S^*_{o,j,m}) \cdots (j', o', S_{o,j,m,o',j'}) \cdots$
1	1	2	(220)(1,3,190)(1,4,160)(2,1,290)(3,1,200)(3,2,250)(3,3,280)(4,1,250)(4,2,200)(5,1,250)(5,2,240)
		3	(120)(1,2,230)(1,3,200)(2,2,280)(2,3,300)(5,3,240)
	2	1	(120)(1,3,250)(2,1,150)(2,2,150)(3,2,200)(3,3,250)(4,1,190)(4,2,250)(5,1,190)(5,3,200)
		3	(240)(1,1,260)(1,3,160)(2,2,150)(2,3,200)(5,3,100)
	3	1	(240)(1,2,260)(2,1,250)(2,2,150)(3,2,200)(3,3,150)(4,1,150)(4,2,200)(5,1,200)(5,3,150)
		2	(160)(1,1,200)(1,4,140)(2,1,220)(3,1,200)(3,2,150)(3,3,250)(4,1,200)(4,2,200)(5,1,250)(5,2,150)
		3	(120)(1,1,290)(1,2,190)(2,2,150)(2,3,200)(5,3,150)
	4	2	(260)(1,1,260)(1,3,120)(2,1,150)(3,1,200)(3,2,250)(3,3,200)(4,1,150)(4,2,200)(5,1,150)(5,2,200)
		4	(140)(2,1,250)(2,3,200)(3,1,250)(3,2,200)(4,1,150)(4,2,200)(5,2,150)(5,3,90)
2	1	1	(70)(1,2,200)(1,3,50)(2,2,200)(3,2,200)(3,3,250)(4,1,250)(4,2,250)(5,1,280)(5,3,200)
		2	(30)(1,1,200)(1,3,100)(1,4,150)(3,1,150)(3,2,200)(3,3,200)(4,1,200)(4,2,200)(5,1,200)(5,2,100)
		4	(200)(1,4,20)(2,3,150)(3,1,150)(3,2,100)(4,1,200)(4,2,100)(5,2,150)(5,3,60)
	2	1	(80)(1,2,200)(1,3,60)(2,1,20)(3,2,100)(3,3,150)(4,1,150)(4,2,150)(5,1,150)(5,3,100)
		3	(200)(1,1,200)(1,2,150)(1,3,100)(2,3,30)(5,3,100)
	3	3	(60)(1,1,150)(1,2,150)(1,3,150)(2,2,60)(5,3,100)
		4	(80)(1,4,150)(2,1,60)(3,1,150)(3,2,150)(4,1,100)(4,2,150)(5,2,200)(5,3,150)
3	1	2	(80)(1,1,200)(1,3,150)(1,4,150)(2,1,150)(3,2,60)(3,3,90)(4,1,150)(4,2,100)(5,1,150)(5,2,100)
		4	(80)(1,4,100)(2,1,150)(2,3,150)(3,2,30)(4,1,100)(4,2,200)(5,2,100)(5,3,200)
	2	1	(20)(1,2,150)(1,3,100)(2,1,100)(2,2,150)(3,3,20)(4,1,120)(4,2,200)(5,1,200)(5,3,100)
		2	(60)(1,1,150)(1,3,150)(1,4,200)(2,1,150)(3,1,150)(3,3,90)(4,1,150)(4,2,150)(5,1,150)(5,2,150)
		4	(60)(1,4,200)(2,1,100)(2,3,150)(3,1,90)(4,1,150)(4,2,150)(5,2,200)(5,3,150)
	3	1	(40)(1,2,100)(1,3,100)(2,1,100)(2,2,100)(3,2,120)(4,1,20)(4,2,100)(5,1,200)(5,3,150)
		2	(60)(1,1,150)(1,3,100)(1,4,150)(2,1,100)(3,1,100)(3,2,60)(4,1,200)(4,2,150)(5,1,100)(5,2,150)
4	1	1	(20)(1,2,100)(1,3,200)(2,1,200)(2,2,150)(3,2,150)(3,3,100)(4,2,60)(5,1,90)(5,3,100)
		2	(40)(1,1,150)(1,3,200)(1,4,150)(2,1,150)(3,1,100)(3,2,200)(3,3,150)(4,2,30)(5,1,150)(5,2,150)
		4	(40)(1,4,150)(2,1,150)(2,3,100)(3,1,100)(3,2,200)(4,2,90)(5,2,200)(5,3,200)
	2	1	(80)(1,2,150)(1,3,200)(2,1,150)(2,2,150)(3,2,100)(3,3,200)(4,1,100)(5,1,20)(5,3,100)
		2	(60)(1,1,200)(1,3,150)(1,4,200)(2,1,100)(3,1,150)(3,2,150)(3,3,150)(4,1,60)(5,1,150)(5,2,150)
		4	(80)(1,4,150)(2,1,150)(2,3,100)(3,1,150)(3,2,150)(4,1,120)(5,2,150)(5,3,100)
5	1	1	(60)(1,2,150)(1,3,100)(2,1,100)(2,2,150)(3,2,100)(3,3,100)(4,1,100)(4,2,150)(5,3,20)
		2	(60)(1,1,90)(1,3,100)(1,4,100)(2,1,150)(3,1,100)(3,2,200)(3,3,100)(4,1,200)(4,2,150)(5,2,30)
	2	2	(40)(1,1,150)(1,3,150)(1,4,150)(2,1,100)(3,1,200)(3,2,200)(3,3,100)(4,1,100)(4,2,60)(5,1,40)
		4	(40)(1,4,100)(2,1,150)(2,3,100)(3,1,100)(3,2,200)(4,1,100)(4,2,40)(5,3,60)
	3	1	(40)(1,2,150)(1,3,150)(2,1,150)(2,2,150)(3,2,150)(3,3,100)(4,1,200)(4,2,150)(5,1,90)
		3	(80)(1,1,100)(1,2,150)(1,3,150)(2,2,200)(2,3,150)
		4	(200)(1,4,200)(2,1,200)(2,3,150)(3,1,150)(3,2,150)(4,1,150)(4,2,150)(5,2,90)

We use this information to categorize the problems as medium- or large-size problems.

4.2.1 Medium-size problems

In small- and medium-size problems, the GA can run for large numbers of generations with shorter computing times and converge. In such cases, the main purpose of parallelizing the GA is to improve the solution quality that can be obtained after the same number of generations as that of the sequential GA. From Table 10, it can be seen that relatively shorter times are required to advance the GA by one generation in solving problems 2, 3, 4, and 6. Hence, we classified these problems as medium-size problems. Figure 6 shows the convergence graphs of the SGA and the 48-processor PGA in solving problem 2 under ten different sets of genetic parameters given in Table 8. A maximum number of generations of 10,000 was used as a termination criterion. For each test run, the size of the population in each individual processor of the PGA is the same as that of the SGA. This allows the PGA and the SGA to reach the maximum number of generations in almost the same amount of computing time. Thus, the graphs in Fig. 6 illustrate the solution quality improvement obtained by using PGA over that of the SGA within the same computational time. In this figure, it can be seen that the convergence behavior of the GA and, hence,

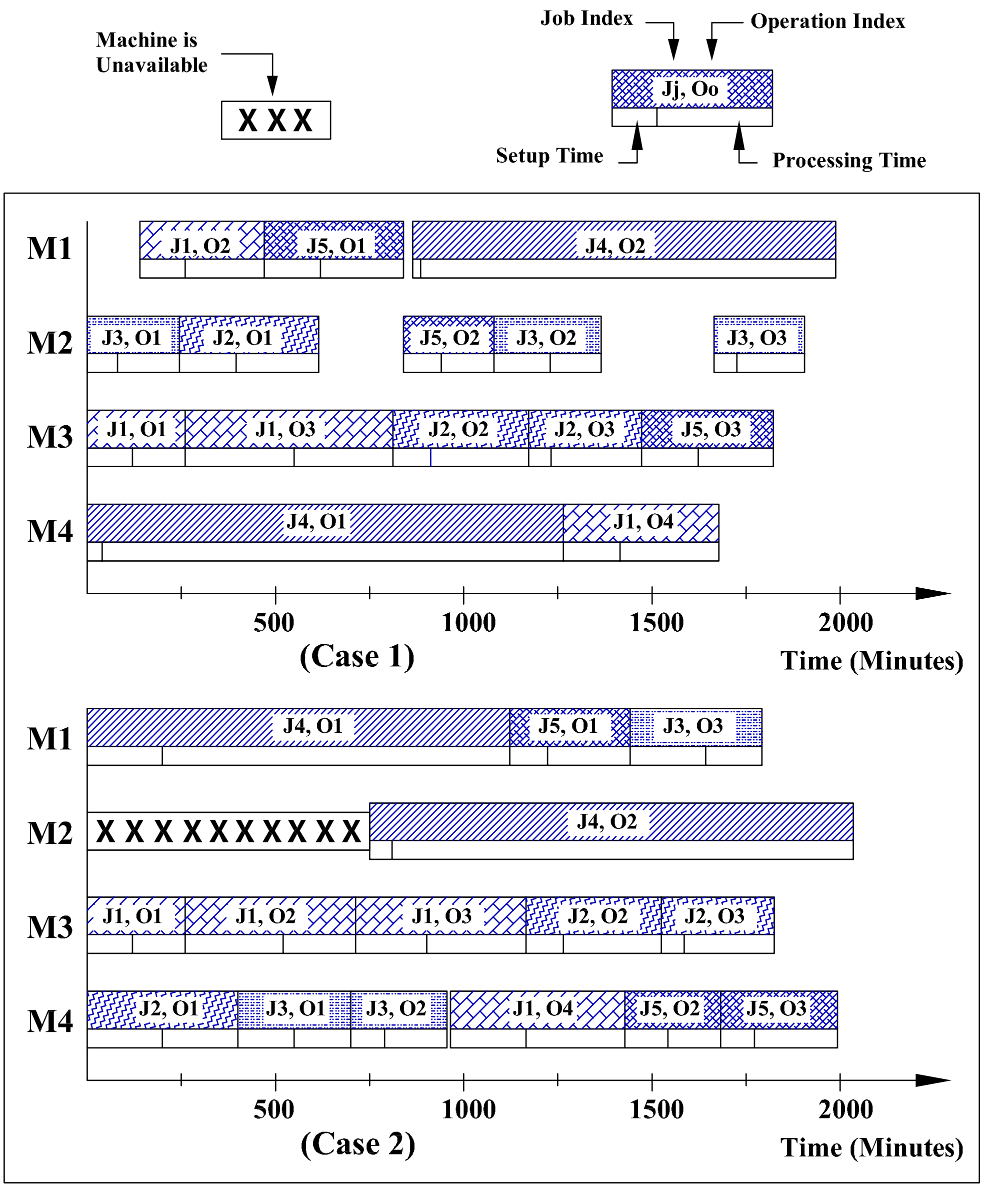

Fig. 5 Schedule of a small problem demonstrating model attributes. Case 1: All machines are available at time zero; case 2: machine 2 is not available during the first 450 min

the final solution quality are improved in all the test runs by using PGA. Figure 7 shows the improvement of the average convergence of ten test runs as the number of processors of the PGA is increased. The makespans of the final solutions for problems 2, 3, 4, and 6 obtained using SGA and PGA for the ten test runs for each problem are plotted in Fig. 8. It can be seen that, in each problem and in each test run, the makespan of the final solution is reduced using the PGA with the average reductions being 170, 88, 145, and 157 min, respectively. In Fig. 8a, we also observe that the patterns by which the final solution quality of the SGA is affected by the genetic parameters are similar to those of the PGA irrespective of the number of processors used. It indicates that the genetic parameter tuning on PGA can be done by using SGA. The other important observation is that the patters by which the final solutions quality from both SGA and PGA are affected by the change of the genetic parameters are similar for different problem instances as shown in Fig. 8a–c and d. This implies that the genetic parameters that worked well for one problem instance are likely to work well for others.

4.2.2 Large problems

In solving larger problem instances, the sequential GA requires longer computational time to advance a certain number of generations and may not converge

Table 7 The details of the schedules shown in Fig. 5

Machine	Run	Case 1					Case 2				
		Job	Opr.	SB	SE/PB	PE	Job	Opr.	SB	SE/PB	PE
1	1	1	2	140.0	260.0	470.0	4	1	0.0	20.0	1,122.5
	2	5	1	470.0	620.0	840.0	5	1	1,122.5	1,222.5	1,442.5
	3	4	2	865.0	885.0	1,987.5	3	3	1,442.5	1,642.5	1,792.5
2	1	3	1	0.0	80.0	245.0	4	2	750.0	810.0	2,035.0
	2	2	1	245.0	395.0	615.0					
	3	5	2	840.0	940.0	1,080.0					
	4	3	2	1,080.0	1,230.0	1,365.0					
	5	3	3	1,665.0	1,725.0	1,905.0					
3	1	1	1	0.0	120.0	260.0	1	1	0.0	120.0	260.0
	2	1	3	260.0	550.0	812.5	1	2	260.0	520.0	712.5
	3	2	2	812.5	912.5	1,172.5	1	3	712.5	902.5	1,165.0
	4	2	3	1,172.5	1,232.5	1,472.5	2	2	1,165.0	1,265.0	1,525.0
	5	5	3	1,472.5	1,622.5	1,822.5	2	3	1,525.0	1,585.0	1,825.0
4	1	4	1	0.0	40.0	1,265.0	2	1	0.0	200.0	400.0
	2	1	4	1,265.0	1,415.0	1,677.5	3	1	400.0	550.0	700.0
	3						3	2	700.0	790.0	955.0
	4						1	4	965.0	1,165.0	1,427.5
	5						5	2	1,442.5	1,542.5	1,682.5
	6						5	3	1,682.5	1,772.5	1,992.5
Makespan		1,987.5					2,035.0				

SB setup begins, *SE* setup ends, *PB* processing begins, *PE* processing ends

Table 8 Genetic parameters used for the test runs

Parameter	Test run									
	1	2	3	4	5	6	7	8	9	10
Population size	1,500	2,500	1,000	3,500	3,000	2,000	1,500	2,500	1,000	2,000
Tournament size factor	0.02	0.01	0.01	0.02	0.02	0.02	0.01	0.02	0.03	0.02
Initialization parameters:										
ARC	0.8	0.7	0.6	0.9	0.7	0.8	0.6	0.8	0.6	0.8
RSJ rule usage	0.1	0.05	0.1	0.08	0.06	0.05	0.1	0.12	0.13	0.08
MWR rule usage	0.15	0.12	0.08	0.1	0.12	0.06	0.08	0.07	0.1	0.11
MOR rule usage	0.9	0.8	0.7	0.7	0.8	0.9	0.8	0.8	0.6	0.8
Assignment operators probability:										
ρ_1	0.3	0.1	0.3	0.05	0.1	0.3	0.2	0.3	0.1	0.3
ρ_2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
ρ_3	0.1	0.1	0.3	0.3	0.1	0.1	0.15	0.2	0.15	0.25
Sequencing operators probability:										
ρ_4	0.5	0.4	0.4	0.3	0.5	0.3	0.35	0.4	0.45	0.4
ρ_5	0.4	0.5	0.3	0.4	0.4	0.6	0.5	0.4	0.4	0.35

Table 9 General descriptions of the example problems

Problem	No of			
	Machines	Products	Operations	Alternative routes[a]
2	20	80	238	506
3	12	40	94	231
4	15	50	201	418
5	40	250	1,583	3,906
6	30	70	311	807

[a]Total number of alternative routes is greater than total number of operations in flexible job shop, where each operation may have more than one alternative route

Table 10 Time required (in seconds) by the GA to move by one generation in solving the test problems under different population size

Problem	Population size			
	500	1,000	2,500	5,000
2	0.23	0.44	1.09	2.25
3	0.06	0.11	0.30	0.63
4	0.19	0.36	0.88	1.82
5	6.51	12.85	32.69	64.83
6	0.42	0.82	2.01	4.10

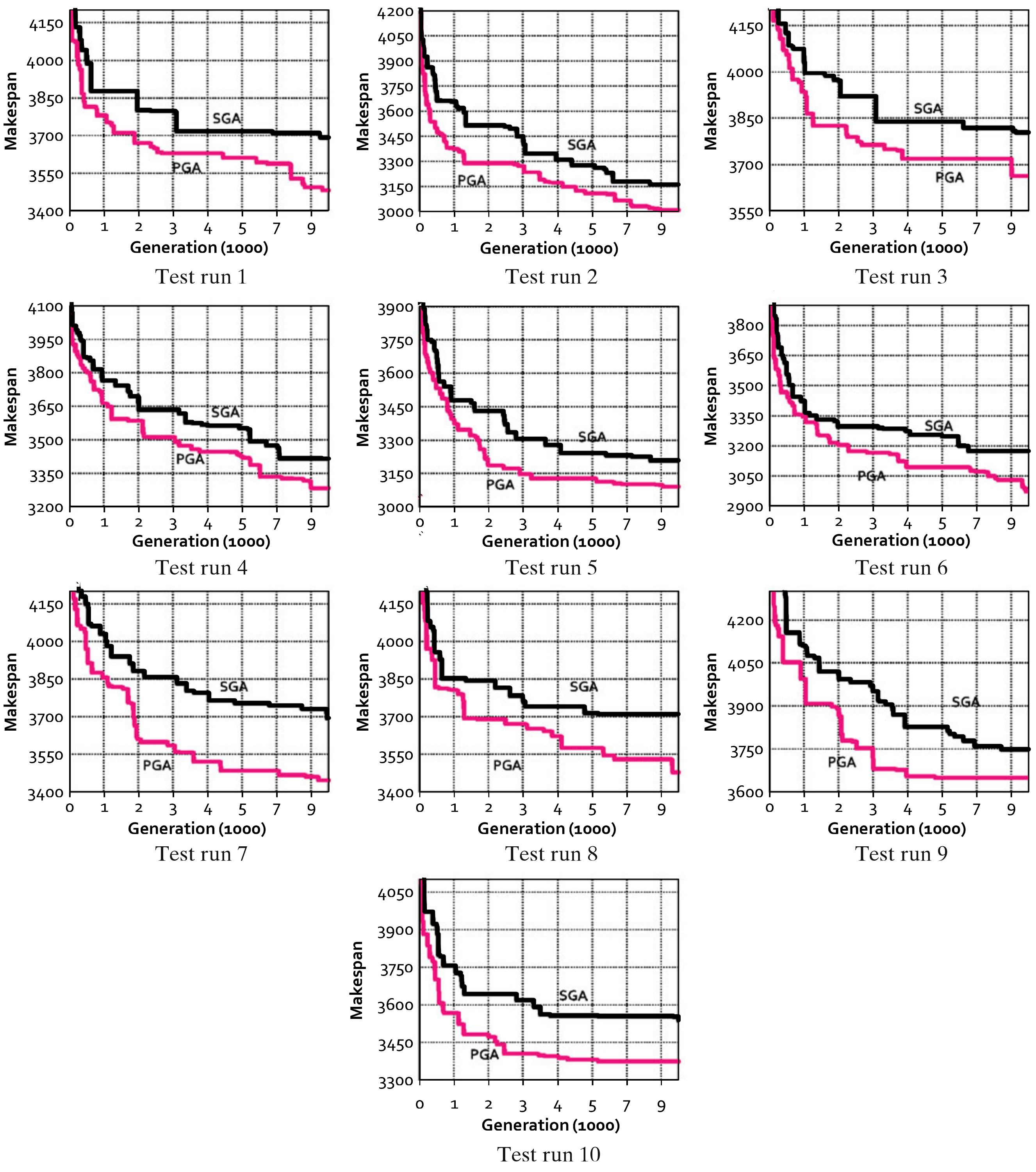

Fig. 6 Convergence graph of SGA and PGA in solving problem 2 under ten different sets of genetic parameters

within the acceptable time limit. As shown in Table 10, solving Problem 5 requires the GA to take much longer time to advance by one generation than other problems with the same population size. In such circumstances, the computational time may be unacceptable to let the GA run for a certain large number of generations until it converges. Hence, terminating the GA after a certain maximum computational time is inherently required. Figure 9 shows the convergence graphs of SGA with varying population sizes and the 48-processor PGA in

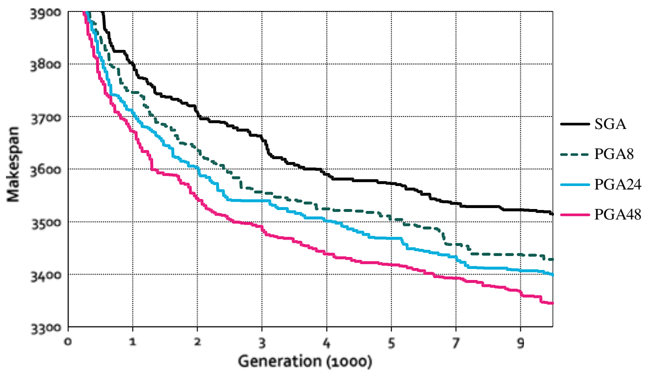

Fig. 7 Performance improvement through parallelization of the GA as the number of processor is increased from 1 to 8, to 24 and to 48

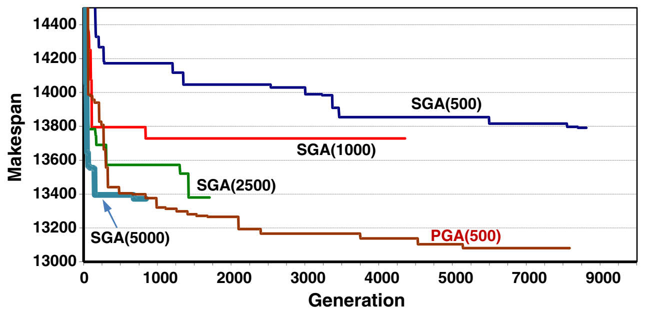

Fig. 9 Comparison of PGA and SGA for a larger problem instance under varying population size and a fixed computation time as a termination criterion (15 h in this example)

solving problem 5, terminated after 15 h of computation. From this figure, it can be seen that the SGA with a smaller population size can propagate a larger number of generations where the solution quality is unsatisfactory. In order to improve the solution quality that can be obtained by the SGA, one may consider using a large population size as it has been demonstrated in Fig. 9. However, as the population size is increased, the number of generations that the SGA could propagate within the allowable computational time is greatly reduced, and it is not able to converge. This difficulty can be alleviated by using PGA, where the large population can be distributed over multiple processors. In this figure, it can be seen that the PGA with a population size of 500 per processor propagates a larger number of generations and converges to a better solution within the allowable computational time. The makespan reductions by using the PGA were 710, 647, 298, and 291 min, compared to those obtained by SGAs using 500, 1,000, 2,500, and 5,000 population sizes, respectively.

5 Discussion and conclusions

In this paper, we developed a comprehensive model for FJSP by considering several factors in an integrated manner. We assume that the jobs have sequence-dependent setup time, where research considering this issue in job-shop scheduling is limited. The model also

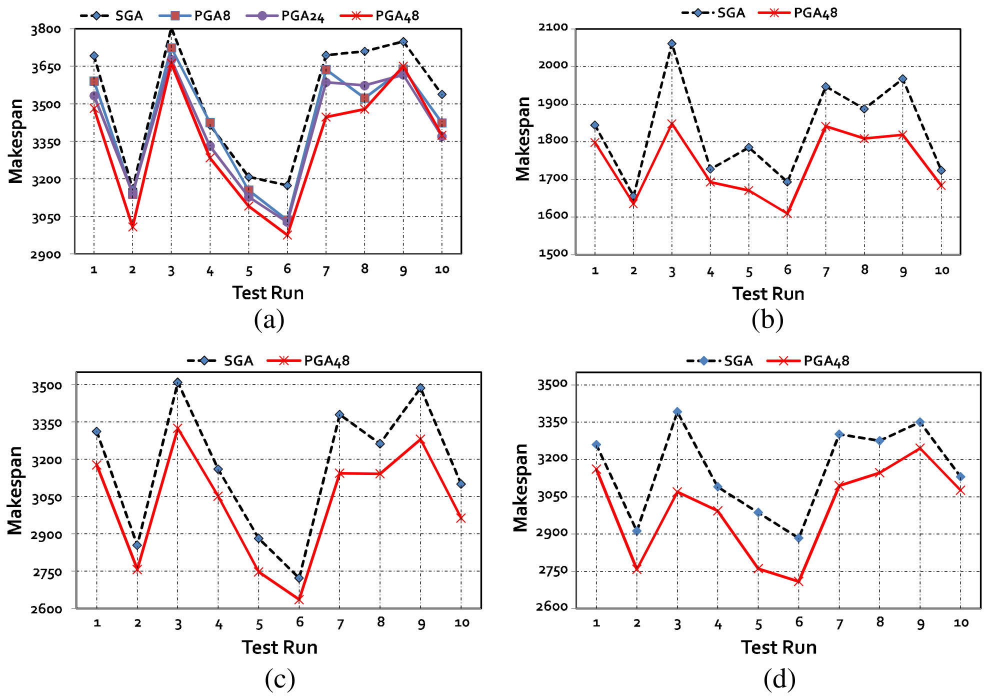

Fig. 8 Final solution qualities for problems 2, 3, 4, and 6 by using SGA and PGA under ten test runs. **a** Problem 2. **b** Problem 3. **c** Problem 4. **d** Problem 6

allows a given setup to be either attached or detached depending on the actual requirements. Other important factors incorporated in the proposed model are machine release date. In this research, we noted that this issue needs more attention in FJSP, where, for an operation, there can be alternative machines with different release dates. The last factor incorporated is the concept of time lag. In order to solve this NP-hard model efficiently, we proposed a PGA implemented using island model. In this parallelization model, subpopulations are separately evolved on several processors and periodically exchange individuals. The PGA was executed on a high-performance computing environment composed of multiple interconnected workstations. The developed model and solution procedure were extensively tested with medium and large problem instances. The results obtained using the PGA are very promising and encouraging compared to those obtained using the SGA. In our future research, we plan to extend the model and the solution procedure to consider multiple objectives such as workload balancing, due dates, and mean flow-time requirements, and others factors such as capacitated buffer and transportation time.

Acknowledgements This research is supported by Discovery Grant from NSERC and by Faculty Research Support Fund from the Faculty of Engineering and Computer Science, Concordia University, Montreal, Canada. We thank RQCHP (Réseau québécois en calcul de haute performance http://www.rqchp.qc.ca/) for its assistance in providing access to parallel computing facilities.

References

1. Allahverdi A, Ng C, Cheng T, Kovalyov M (2008) A survey of scheduling problems with setup times or costs. Eur J Oper Res 187:985–1032
2. Baker K (1974) Introduction to sequence and scheduling. Wiley, New York
3. Baxter F (1990) Information technology and global changing science. In: Conference on global change: economic issues in agriculture, forestry and natural resources, Washington, DC, 19–21 November 1990
4. Bruker P, Schlie R (1990) Job shop scheduling with multi-purpose machines. Computing 45:369–375
5. Cantú-Paz E (2000) Efficient and accurate parallel genetic algorithms. Kluwer Academic, Norwell
6. Chaudhry SS, Luo W (2005) Application of genetic algorithms in production and operations management: a review. Int J Prod Res 43:4083–4101
7. Chen H, Ihlow J, Lehmann C (1999) A genetic algorithm for flexible job-shop scheduling. In: The proceedings of the 1999 IEEE international conference on robotics & automation, Detroit, pp 1120–1125
8. Chen J, Chen K, Wu J, Chen C (2007) A study of the flexible job shop scheduling problem with parallel machines and reentrant process. Int J Adv Manuf Technol. doi:10.1007/s00170-007-1227-1
9. Conway R, Maxwell W (1967) Theory of scheduling. Addison-Wesley, Reading
10. Defersha FM, Chen M (2007) A parallel genetic algorithm for dynamic cell formation in cellular manufacturing systems. Int J Prod Res. doi:10.1080/00207540701441962
11. Gao J, Gen M, Sun L, Zhao X (2007) A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. Comput Ind Eng 53:149–162
12. Gao J, He G, Wang Y (2009) A new parallel genetic algorithm for solving multiobjective scheduling problems subjected to special process constraint. Int J Adv Manuf Technol 43:151–160
13. Gao J, Sun L, Gen M (2008) A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput Oper Res 35:2892–2907
14. Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and jobshop scheduling. Math Oper Res 1:117–129
15. Guo Z, Wong W, Leug S (2008) A genetic-algorithm-based optimization model for scheduling flexible assembly lines. Int J Adv Manuf Technol 36:156–168
16. Guo Z, Wong W, Leung S, Fan J, Chen S (2008) A genetic algorithm based optimization model for solving the flexible assembly line balancing problem with work-sharing and workstation revisiting. IEEE Trans Syst Man Cybern, Part C Appl Rev 38:218–228
17. Guo Z, Wong W, Leung S, Fan J, Chen S (2008) Genetic optimization of order scheduling with multiple uncertainties. Expert Syst Appl 35:1788–1801
18. Guo Z, Wong W, Leung S, Fan J, Chan SF (2006) Mathematical model and optimization for the job shop scheduling problem in a mixed- and multi-product assembly environment: a case study based on the apparel industry. Comput Ind Eng 50:202–219
19. Gupta J (1986) Flowshop schedules with sequence dependent setup times. J Oper Res Soc Jpn 29:206–219
20. ILOG Inc (2008) CPLEX 12.0 user's manual. 1080 Linda Vista Ave. Mountain View, CA 94043. http://www.ilog.com
21. Jain AS, Meeran S (1998) Deterministic job-shop scheduling: past, present and future. Eur J Oper Res 113:390–434
22. Jolai F, Sheikh S, Rabbani M, Karimi R (2009) A genetic algorithm for solving no-wait flexible flow lines with due window and job rejection. Int J Adv Manuf Technol 42:523–532
23. Kacem I, Hammadi S, Borne P (2002) Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans Syst Man Cybern 32:1–3
24. Kochhar S, Morris R (1987) Heuristic methods for flexible flow line scheduling. J Manuf Syst 6:299–314
25. Lee K, Yamakawa T, Lee K (1998) A genetic algorithm for general machine scheduling problems. International Journal of Knowledge-Based Electronic 2:60–66
26. Manikas A, Chang Y (2008) Multi-criteria sequence-dependent job shop scheduling using genetic algorithms. Comput Ind Eng 56:179–185
27. Osman I, Potts C (1989) Simulated annealing for permutation flow-shop scheduling. Omega 17:551–557
28. Panwalkar SS, Dudek RA, Smith ML (1973) Sequencing research and the industrial scheduling problem. In: Elmaghraby SE (ed) Symposium on the theory of scheduling and its applications. Springer, New York, p 29

29. Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 35:3202–3212
30. Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22:5–13
31. Rios-Mercado R, Bard J (1999) A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. IIE Trans 31:721–731
32. Ruiz R, Şerifoglub FS, Urlings T (2008) Modeling realistic hybrid flexible flowshop scheduling problems. Comput Oper Res 35:1151–1175
33. Ruiz R, Maroto C, Alcaraz J (2005) Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. Eur J Oper Res 165:34–54
34. Saidi M, Fattahi P (2007) Flexible job shop scheduling with tabu search algorithm. Int J Adv Manuf Technol 35:563–570
35. Snir M, Otto H-L, S S, Walker D, Dongarra J (1998) MPI-the complete reference, vol 1. MPI core, 2nd edn. Scientific and engineering computation series. MIT, Cambridge
36. Tsai J, Liu T, Ho W, Chou J (2008) An improved genetic algorithm for job-shop scheduling problems using taguchi-based crossover. Int J Adv Manuf Technol 38:987–994
37. Wang YM, Xiao N, Yin H, Hu E, Zhao C, Jiang Y (2008) A two-stage genetic algorithm for large size job shop scheduling problems. Int J Adv Manuf Technol 39:813–820
38. Wortman DB (1992) Managing capacity: getting the most from your company's asset. Ind Eng 24:47–49
39. Xia W, Wu Z (2005) An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 48:409–425
40. Xing L, Chen Y, Yang K (2008) Multi-objective flexible job shop schedule: design and evaluation by simulation modeling. Applied Soft Computing. doi:10.1016/j.asoc.2008.04.013
41. Zhang C, Rao Y, Li P (2008) An effective hybrid genetic algorithm for the job shop scheduling problem. Int J Adv Manuf Technol 39:965–974
42. Zhang H, Gu M (2008) Modeling job shop scheduling with batches and setup times by timed petri nets. Math Comput Model. doi:10.1016/j.mcm.2008.03.010