J Supercomput (2014) 67:69–83
DOI 10.1007/s11227-013-0986-8

A heuristic algorithm for the distributed and flexible job-shop scheduling problem

Mohsen Ziaee

Published online: 18 July 2013

Abstract The distributed manufacturing takes place in a multi-factory environment including several factories, which may be geographically distributed in different locations, or in a multi-cell environment including several independent manufacturing cells located in the same plant. Each factory/cell is capable of manufacturing a variety of product types. An important issue in dealing with the production in this decentralized manner is the scheduling of manufacturing operations of products (jobs) in the distributed manufacturing system. In this paper, we study the distributed and flexible job-shop scheduling problem (DFJSP) which involves the scheduling of jobs (products) in a distributed manufacturing environment, under the assumption that the shop floor of each factory/cell is configured as a flexible job shop. A fast heuristic algorithm based on a constructive procedure is developed to obtain good quality schedules very quickly. The algorithm is tested on benchmark instances from the literature in order to evaluate its performance. Computational results show that, despite its simplicity, the proposed heuristic is computationally efficient and promising for practical problems.

Keywords Distributed scheduling · Flexible job shop · Makespan · Heuristic

1 Introduction

The significance of distributed manufacturing has been recognized by many researchers and industrialists in recent years due to the changes in the mode of today's production environment. Single factory or centralized production environment in traditional manufacturing systems has been gradually replaced by more flexible distributed settings, including multi-factory networks or multi-cell job shops due to

M. Ziaee (✉)
Department of Industrial Engineering, University of Bojnord, 94531-55111 Bojnord, Iran
e-mail: ziaee@iust.ac.ir

the trend of globalization [1]. The distributed manufacturing enables the enterprises to be closer to their customers and suppliers, to produce and market their products more effectively, to be responsive to market changes more quickly, to achieve better product quality, lower production cost, reduced management risk, and better utilization of production resources [1–4].

The distributed manufacturing takes place in a multi-factory environment including several factories, which may be geographically distributed in different locations or in a multi-cell environment including several independent manufacturing cells located in the same plant [5]. Each factory/cell is capable of manufacturing a variety of product types. In addition, the factories or cells have different production efficiencies, operating costs, production lead times, constraints, etc. [1]. An important issue in dealing with the production in this decentralized manner is the scheduling of manufacturing operations of products (jobs) in the distributed manufacturing system [4, 6].

In this paper, we study the distributed and flexible job-shop scheduling problem (DFJSP) which involves the scheduling of jobs in a distributed manufacturing environment described above, under the assumption that the shop floor of each factory/cell is configured as a flexible job shop (therefore, each factory/cell will be hereafter called flexible manufacturing unit (FMU) [5]). The flexible job-shop scheduling problem (FJSP) is an extension of the classical job-shop scheduling problem (JSP), where each operation is allowed to be processed on any among set of available machines; and thus, the scheduling problem is to choose, for each operation, a machine and a starting time at which the operation must be processed [7, 8]. The DFJSP consists of the following two subproblems which can be solved sequentially or simultaneously: (1) the assignment of each job to exactly one FMU (which corresponds to one cell in a multi-cell environment or one factory in a multi-factory setting), and (2) the scheduling of the jobs in each FMU, i.e. solving an FJSP for each FMU. Once a job is assigned to an FMU and started its processing (but not finished), it is usually not possible or uneconomical to transfer this unfinished job to another FMU for the remaining operations [1]. Therefore, we assume that once a job is allocated to an FMU, all its operations have to be processed in the same FMU. We also suppose that all the FMUs belong to the same company. Accordingly, they work cooperatively to define an optimal production plan maximizing the revenue of the company as a whole. The objective is to minimize the makespan, i.e. the overall completion time of all the jobs on all the FMUs. The DFJSP is more complicated than the traditional FJSP in single FMU, because it involves not only the FJSP in each FMU but also the problem in an upper level that is the assignment of the jobs to the FMUs. This problem is strongly NP-hard, since the problem with one FMU, i.e. the FJSP, is already strongly NP-hard [9]. This justifies the need for developing efficient heuristic algorithms to obtain approximate solutions of good quality at little computational cost. These heuristic algorithms are very fast in comparison with the exact methods and the metaheuristic algorithms, and if they have been proved to produce solutions with adequate accuracy, they can be the best approach. Therefore, this paper proposes a powerful heuristic algorithm to solve the problem.

Almost all existing studies in the field of production scheduling deal with the centralized or non-distributed production environments, and only few papers investigate the distributed scheduling problems, especially those involving the DFJSP [6]. Due to the complexity of the distributed scheduling problems mentioned above, approximate algorithms, mainly metaheuristics, have been often used to solve the problem. A review of heuristics developed to solve these problems can be found in references [5, 6]. Wang et al. [10] consider the distributed permutation flow-shop scheduling problem and present an effective estimation of distribution algorithm (EDA) to solve the problem. Just this problem is also examined in [11] and solved by a tabu search algorithm. Jia et al. [4, 12] present a modified genetic algorithm to solve the distributed scheduling problem in a multi-factory network in which each factory is configured as a job shop. Chan et al. [13, 14] consider the DFJSP and present a genetic algorithm with dominated genes (GADG) to solve the problem and demonstrate the performance of the method by using several new DFJSP instances. Also, Chan et al. [1] study a generalized version of the DFJSP in which machine maintenance constraints are considered and it is assumed that the maintenance time is related to the machine age. A mathematical model and a genetic algorithm with dominant genes (GADG) are developed for the problem. For both with and without maintenance constraints, the performance of the presented genetic algorithm is evaluated using some new test instances. Giovanni and Pezzella [5] present an improved genetic algorithm to solve the DFJSP. The proposed approach is compared with other algorithms for distributed scheduling and tested on a large set of DFJSP instances derived from well-known FJSP benchmarks, providing good results. As it can be seen in the above review of literature on the distributed scheduling problems, studies on methods for solving these problems are in the early stage. Almost all the proposed methods are metaheuristic algorithms and very time-consuming in comparison with the heuristic algorithms, because they perform search procedure in a large part of the solution space.

In this study, we present a heuristic method based on a constructive procedure to solve the DFJSP with the objective of minimizing makespan (Sect. 3). The main purpose is to produce reasonable and applicable schedules very quickly. It can also be used to improve the quality of the initial feasible solution of metaheuristics applied to solve the problem, since the choice of a good initial solution is an important aspect of the performance of the algorithms in terms of computation time and solution quality [15–17]. In order to evaluate the performance of the proposed heuristic, we implement it using several benchmark problems and present the results of the computational experiments (Sect. 4). The results show that our novel method can obtain good solutions in very short time. Concluding remarks are given in the last section.

2 Assumptions and notation

The assumptions considered in this paper are as follows:

(1) Each FMU can produce all jobs with different efficiencies.
(2) For each job, all FMUs have the same number of operations.
(3) Jobs are independent of each other.
(4) Setup and transportation times are negligible.
(5) Preemption is not allowed, i.e. a started operation cannot be interrupted during its processing.
(6) Each machine can process at most one operation at the same time.

(7) All jobs have equal priorities.
(8) Machines never break down and are available throughout the scheduling period.
(9) All FMUs, jobs and machines are available at time zero.

The notation used throughout the paper is as follows:

l: number of FMUs,
n: number of jobs,
f: index of FMUs; $f = 1, \ldots, l$,
i, z: index of jobs; $i, z = 1, \ldots, n$,
m_f: number of machines in FMU f,
J_i: number of operations of job i,
$\max J$: maximum number of operations per job (i.e., $\max J = \max_i J_i$),
j: index of operations; $j = 1, \ldots, J_i$,
k, y: index of machines; $k, y = 1, \ldots, m_f$,
t_{fijy}: processing time (duration) of operation j of job i on machine y of FMU f,
c_{fij}: completion time of operation j of job i on FMU f.
A_{fij}: set of machines in FMU f which are capable to execute operation j of job i,
N_{fij}: number of members of the set A_{fij},
s'_{fij}: mean processing time of operation j of job i over the machines belonging to the set A_{fij} (i.e., $s'_{fij} = (\sum_{y \in A_{fij}} t_{fijy})/N_{fij}$),
sj_{fi}: total mean processing time of job i in FMU f (i.e., $sj_{fi} = \sum_{j=1}^{J_i} s'_{fij}$),
sk_{fy}: total weighted processing time on machine y of FMU f which is calculated as follows: $sk_{fy} = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ if\, y \in A_{fij}}}^{J_i} \frac{t_{fijy}}{N_{fij}}$,
IF_f: number of jobs assigned to FMU f,
M: a large number.

3 Proposed heuristic approach

In this section, a heuristic method is presented to solve the problem. This approach is motivated by the idea of developing a constructive heuristic that considers simultaneously many factors affecting the solution quality and intelligently balances their effects in the process of schedule generation, and the observation that it can lead to good results in some preliminary computational experiments on a wide range of difficult scheduling problems.

An outline of the proposed heuristic algorithm is given in Fig. 1.

The pseudocode of the proposed heuristic is shown in Fig. 2. In this algorithm, each unscheduled operation (i, j) (i.e. operation j of job i) to be scheduled on machine y of FMU f is evaluated by the following criterion, and the unscheduled operation with minimum TC is selected for scheduling.

$$TC = \sum_{r=1}^{5} w_r \times x_r \times C_r$$

such that,

```
for $j := 1$ to $\max J$ do
{
    until the $j$th operation of all jobs are scheduled, repeat
    {
        • Find $i$, $k$, $f$ (such that: 1. ($j = 1$), or ($j > 1$ and job $i$ is assigned to FMU $f$); 2.
          $j \le J_i$; and 3. $j$th operation of job $i$ is an unscheduled operation and machine $k$ of
          FMU $f$ is capable of processing this operation) that minimizes $TC$.
        • If job $i$ is an unassigned job (i.e. a job not assigned to any FMU) to any FMU
          so far, then assign it to FMU $f$; and schedule $j$th operation of this job at the last
          position of current partial sequence on machine $k$ of FMU $f$.
    }
}
```

Fig. 1 General outline of the proposed heuristic algorithm

$$
\begin{aligned}
&C_1 = \max(C_{\max_{fy}}, c_{f,i,j-1}) + t_{fijy}\\
&C_2 = \max(0, (c_{f,i,j-1} - C_{\max_{fy}}))\\
&C_3 = t_{fijy}\\
&C_4 = sk_{fy}/(n + \max J)\\
&C_5 = sj_{fi}
\end{aligned}
$$

TC is weighted sum of some criteria which are established based on the factors affecting the objective function value. Minimization of TC in the process of schedule generation leads to improvement in solution quality. The w_r $(r = 1, 2, \ldots, 5)$ are constants and the x_r $(r = 1, 2, \ldots, 5)$ are integer variables used to increase the flexibility and efficiency of criterion *TC* and they have a significant impact on the performance of the algorithm. The constant weights (w_r) are preliminary estimated weights assigned to criteria according to their importance, and the coefficients x_r are variables bounded in a given range and used to refine the TC. $C_{\max_{fy}}$ is the maximum completion time across all the operations scheduled on machine y of FMU f; that is, $C_{\max_{fy}}$ is equal to the completion time of the operation situated just before operation j of job i on machine y of FMU f. C_1 and C_2 are applied to decrease $C_{\max_{fy}}$ and idle times, respectively; clearly, both these objectives affect the main objective function, i.e. $C_{\max}$. For assigning operations to a machine, their processing times are also taken into account by C_3. C_4 and C_5 are used for taking into account the total weighted processing time of machines, and the total mean processing time of jobs, respectively.

Another notation used in the pseudocode of the heuristic is as follows:

TC^*: denotes the best value of *TC*. After each operation is scheduled, TC^* is reset to M.

L_x_r $(r = 1, 2, \ldots, 5)$: lower limit of x_r.

U_x_r $(r = 1, 2, \ldots, 5)$: upper limit of x_r.

To prevent assigning too many jobs to a single FMU and to achieve balanced distribution of the jobs among the FMUs, we use a parameter called *MIF* which is defined as the maximum number of jobs assigned to a single FMU and calculated as follows:

$$MIF = n/l.$$

Thus, IF_f (for $f = 1, 2, \ldots, l$) must be less than *MIF* $(IF_f < MIF)$.

```
Initialization:
•  For FMU $f$ ($f$=1,2,…,$l$), sort the jobs in increasing order of their $sj_{fi}$ and call the
   resulting set: $i\_sort_f$. Let $i\_sort_{fz}$ be $z$th job of the list $i\_sort_f$.
•  For FMU $f$ ($f$=1,2,…,$l$), sort the machines in increasing order of their $sk_{fy}$ and call
   the resulting set: $k\_sort_f$. Let $k\_sort_{fy}$ be $y$th machine of the list $k\_sort_f$.

Constructive Algorithm:
for $x_1$:=$L\_x_1$ to $U\_x_1$ do
for $x_2$:=$L\_x_2$ to $U\_x_2$ do
⋮
for $x_5$:=$L\_x_5$ to $U\_x_5$ do
for $x_6$:=0 to 1 do
{
        % Beginning of a schedule generation

        for $f$:=1 to $l$ do
                Set $IF_f$:=0

        for $j$:= 1 to $maxJ$ do
        {
                until $j$th operation of all jobs are scheduled, repeat the following
                steps:
                {
                        Set $TC^*$:=$M$

                        Sort the FMUs in increasing order of their $IF_f$ and call the
                        resulting set: $f\_sort$. Let $f\_sort_f$ be $f$th FMU of the list $f\_sort$.

                        for $f''$:=1 to $l$ do
                        {
                                Set $f$:=$f\_sort_{f''}$,

                                for $i'$:=1 to $n$ do
                                {
                                        Set $i$:=$i\_sort_{f,(n-i'+1)}$,

                                        if (    1. job $i$ is an unassigned job, and $IF_f$ <
                                                $MIF$; or
                                                2. job $i$ is assigned to FMU $f$
                                           ) then
                                        {
                                                if ($j \le J_i$ and $j$th operation of job $i$ is an
                                                unscheduled operation) then
                                                {
                                                        for $k''$:=1 to $m_f$ do
                                                        {
                                                                Set $k$:=$x_6$.($k\_sort_{f,k''}$)+(1-
                                                                $x_6$).($k\_sort_{f,(m_f-k''+1)}$),

                                                                if (machine $k$ of FMU $f$ is
                                                                capable of processing $j$th
                                                                operation of job $i$) then
                                                                {
                                                                        Set $TC:=\sum_{r=1}^{5} w_r.x_r.C_r$
                                                                        if $TC<TC^*$ then
                                                                        {
                                                                                Set $TC^*$:= $TC$
                                                                                Set $z$:=$i$
                                                                                Set $y$:=$k$
                                                                                Set $f'$:=$f$
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }
                        }

                        if ($TC^*<M$) then
                        {
                                1. if job $z$ is an unassigned job, then assign it to FMU
                                   $f'$ and set $IF_{f'}$= $IF_{f'}$+1,
                                2. schedule $j$th operation of job $z$ at the last position
                                   of the current partial sequence on machine $y$ of FMU $f'$
                                   to finish at time $c_{f'zj}$.
                        }
                }
        }

        % End of a schedule generation

        If the objective value of the obtained sequence ($C_{max}$) is less than the best
        objective value obtained so far ($C_{max}$*), then set $C_{max}$*:=$C_{max}$ and $x_r^*$=$x_r$ ($r$=1,2,…, 6)
        corresponding to $C_{max}$*.
}
```

Fig. 2 Pseudocode of the proposed heuristic method

The algorithm starts by scheduling the first operation of all jobs, then their second operation, and so on (F2, L19 (i.e. Line 31 of Fig. 2)). For each j ($j = 1, 2, \ldots, \max J$), the algorithm sorts the FMUs in increasing order of their IF_f (F2, L26) and, for each FMU f taken in this order (F2, L31), the algorithm sorts the jobs in decreasing order of their sj_{fi} and takes a job i in this order (F2, L35). Therefore, if two unscheduled operations belonging to two different jobs have the same value of TC, then according to this sorting of the jobs, the operation of job with greater sj_{fi} is selected for scheduling sooner than the other operation. Next, if job i is an unassigned job and $IF_f < MIF$, the algorithm evaluates its first operation to be assigned to and scheduled on FMU f. If job i is already assigned to FMU f, then the algorithm evaluates its jth ($j > 1$) operation (if $j \leq J_i$ and operation j of job i is an unscheduled operation). For evaluating operation j of job i on FMU f, similarly the algorithm first sorts the machines of FMU f in increasing (decreasing) order of their sk_{fy} and, for each machine y taken in this order (F2, L47), evaluates this operation (F2, L54 to L60) to be scheduled on machine y (if machine y is capable of processing this operation, i.e. $y \in A_{fij}$). Binary variable x_6 is applied for setting the order of the sorting (i.e. either increasing order or decreasing order): it takes a value of 1 for increasing order and 0 for decreasing one. Sorting the jobs, the machines and the FMUs, described above and done before evaluating them for scheduling, may lead to better solutions. Indeed, in our preliminary computational experiments, we used these sortings of the FMUs, jobs and machines instead of randomly selecting them, and observed that these sortings can lead to better solutions. Specially, the results showed that in most cases, sorting the jobs in decreasing order of their sj_{fi} leads to better solutions in comparison with increasing order. It is because the jobs with larger sj_{fi} which are firstly selected for scheduling have more sensibility and effect on the objective value. In other words, the schedule of these jobs determines the performance of overall schedule of the problem. Therefore, we have used only their decreasing order in the computational experiments. Similarly, the results showed that in most cases, sorting the FMUs in increasing order of their IF_f leads to better solutions in comparison with decreasing order. This is because this sorting leads to keep balanced distribution of the jobs among the FMUs. Therefore, we have used only their increasing order in the computational experiments. For the machines however we could not definitely determine their best order (i.e. either increasing or decreasing one), and so the algorithm itself selects the best order for each problem instance. The x_r^* ($r = 1, 2, \ldots, 6$) are the best values of variables x_r (i.e. the values corresponding to the best solutions). Indeed, for various values of x_r ($r = 1, 2, \ldots, 6$) (F2, L8 to L12), the algorithm in Fig. 1 is run and a complete schedule is generated. Among all these schedules, the one with minimum makespan is reported as the final solution (F2, L82 to L84). The values of variables x_r for this best solution are also reported and denoted by x_r^* (see Table 2).

As mentioned earlier, the evaluation of the operations for scheduling them is done using the criterion TC (F2, L54), i.e. the unscheduled operation with minimum TC is selected for scheduling.

4 Computational results

This section describes the computational experiments conducted in order to evaluate the performance of the proposed heuristic method. First, some preliminary experiments have been conducted for the parameter settings. Regarding the test on various values for the parameters of the algorithm and considering the computational results, we use the settings of Table 1 for benchmarking the presented algorithm.

The algorithm is coded in C language and run on a Pentium IV, 2.2 GHz and 2.0 GB RAM PC. The computational results of the proposed algorithm are compared to the results of the improved genetic algorithm (IGA) developed by Giovanni and Pezzella [5] which, to the best of our knowledge, is the only contribution containing explicit computational results for the DFJSP. The benchmark problems used are the set of 69 DFJSP instances presented in [5]. They are classified into three categories: DFJSP instances with two, three and four FMUs, where the computational results of them are shown in Tables 2, 3 and 4, respectively. The first three columns refer to the name of the FJSP instance used to derive the DFJSP instance together with the number of jobs and machines. LB stands for the lower bounds computed by [5]. $BC_{\max}$, *Av*. ($C_{\max}$) and *Av. time* indicate the best makespan, the average makespan and the average computational time, respectively. The results obtained by the heuristic are shown in the last nine columns. $C_{\max}$ and *Time* represent the makespan value and computational time (in seconds), respectively. The best values of variables x_r (i.e. x_r^*), $r = 1, 2, \ldots, 6$, are also reported in the tables. The average value of each variable x_r, $r = 1, 2, \ldots, 5$, can be considered as the relative effect of the corresponding criterion on the quality of solutions. For example, the average value of x_4 is near zero in all three tables which means that the total weighted processing time of machines has little effect on $C_{\max}$. Of course, as it can be seen, the values of each variable x_r, $r = 1, 2, \ldots, 5$, have relatively high variance in all three tables, meaning that they are strongly dependent on the specifications of problem instance under consideration and on the values of other variables x_r. The proposed algorithm selects for each instance the best combination of x_r values leading to the best result. Average weight of C_5 is negative in all three instance sets, i.e. $w_5 \times x_5 < 0$, which means that it has adverse effect on $C_{\max}$. Average value of x_6 in the tables is nearer to 0 than 1. It is because the machines with larger sk_{fy} which are firstly selected for scheduling have more sensibility and effect on the objective value. *RPD* is the relative percentage deviation to *LB* and is calculated as follows:

$$RPD = \frac{C\max_{\text{alg}} - LB}{LB} \times 100,$$

where $C_{\max_{\text{alg}}}$ is the best makespan obtained by the algorithm. The results show that the heuristic finds the optimal solution ($C^*_{\max} = LB$) for 46 out of 69 problems, which represents 66.7 % of the problems tested. For some instances (specially those with two FMUs), the gap is large, but it is notable that the lower bounds computed by [5] are poor as stated in this reference. Herein, the heuristic is statistically compared with the method IGA. For each of the three instance sets, a one-way analysis of variance (ANOVA) [18] is performed to test the null hypothesis that the means of the two methods are equal. The results of these ANOVA for DFJSP instances with two, three and four FMUs are presented in Tables 5, 6 and 7, respectively. As it can be seen,

Table 1 Parameter settings for the heuristic

Instances with 2 FMUs						Instances with 3 FMUs						Instances with 4 FMUs					
Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
w_1	2	L_x_1	2	U_x_1	5	w_1	4	L_x_1	0	U_x_1	4	w_1	1	L_x_1	0	U_x_1	3
w_2	2	L_x_2	0	U_x_2	5	w_2	2	L_x_2	0	U_x_2	4	w_2	1	L_x_2	0	U_x_2	2
w_3	1	L_x_3	0	U_x_3	2	w_3	1	L_x_3	0	U_x_3	2	w_3	1	L_x_3	0	U_x_3	3
w_4	2	L_x_4	−2	U_x_4	2	w_4	1	L_x_4	−1	U_x_4	1	w_4	1	L_x_4	0	U_x_4	2
w_5	−1	L_x_5	0	U_x_5	3	w_5	−1	L_x_5	0	U_x_5	2	w_5	1	L_x_5	−1	U_x_5	1

Table 2 Computational results for DFJS instances with two FMUs

Name	Size		LB	IGA				Heuristic								
	n	m		$BC_{\max}$	Av. ($C_{\max}$)	Av. time (second)	RPD	$C_{\max}$	Time (second)	x_1	x_2	x_3	x_4	x_5	x_6	RPD
la01	10	5	413	413	413	12	0	413	0.38	2	1	0	0	0	0	0
la02	10	5	394	394	394	11.2	0	394	0.36	2	0	0	0	1	1	0
la03	10	5	349	349	349	10.8	0	349	0.38	2	0	0	0	0	1	0
la04	10	5	369	369	369	11.4	0	369	0.39	5	1	0	−1	2	0	0
la05	10	5	380	380	380	8	0	380	0.38	2	0	0	0	0	1	0
la06	15	5	413	445	449.6	45.8	7.7	463	0.78	2	1	2	2	1	0	12.11
la07	15	5	376	412	419.2	50.2	9.6	417	0.80	2	0	0	0	2	0	10.9
la08	15	5	369	420	427.8	53.8	13.8	436	0.78	2	2	1	1	1	0	18.16
la09	15	5	382	469	474.6	45.2	22.8	485	0.81	2	1	1	2	2	0	26.96
la10	15	5	443	445	448.6	45	0.5	500	0.80	3	0	0	2	1	0	12.87
la11	20	5	413	570	571.6	126	38	583	1.41	3	5	0	0	2	0	41.16
la12	20	5	408	504	508	116	23.5	508	1.36	2	4	2	2	2	0	24.51
la13	20	5	382	542	552.2	125.4	41.9	572	1.41	2	2	1	0	1	0	49.74
la14	20	5	443	570	576	122.2	28.7	589	1.39	4	5	2	−1	0	0	32.96
la15	20	5	378	584	588.8	119.6	54.5	595	1.39	2	4	0	−2	3	0	57.41
la16	10	10	717	717	717	140.2	0	717	0.83	2	0	0	−2	0	0	0
la17	10	10	646	646	646	112.6	0	646	0.83	3	0	0	−2	0	0	0
la18	10	10	663	663	663	132.4	0	663	0.84	5	1	1	0	1	0	0
la19	10	10	617	617	617.2	147.2	0	682	0.81	4	0	0	0	1	1	10.53
la20	10	10	756	756	756	99.8	0	756	0.83	2	0	0	0	0	1	0
mt06	6	6	47	47	47	2	0	47	0.19	2	0	0	−2	0	0	0
mt10	10	10	655	655	655	173	0	655	0.83	2	0	0	0	0	1	0
mt20	20	5	387	560	566	121.2	44.7	596	1.36	4	1	0	−2	1	0	54.01
Average						79.6087	12.42174		0.84	2.652	1.217	0.435	−0.13	0.913	0.261	15.27

Table 3 Computational results for DFJS instances with three FMUs

Name	Size		LB	IGA				Heuristic								
	n	m		$BC_{\max}$	Av. ($C_{\max}$)	Av. time (second)	RPD	$C_{\max}$	Time (second)	x_1	x_2	x_3	x_4	x_5	x_6	RPD
la01	10	5	413	413	413	4.6	0	413	0.20	1	1	0	−1	0	0	0
la02	10	5	394	394	394	3.6	0	394	0.20	2	1	2	0	0	1	0
la03	10	5	349	349	349	3.8	0	349	0.23	1	0	0	0	0	1	0
la04	10	5	369	369	369	3.8	0	369	0.22	3	3	1	−1	1	0	0
la05	10	5	380	380	380	2.6	0	380	0.22	2	0	0	−1	0	0	0
la06	15	5	413	413	413	17.4	0	413	0.44	3	1	0	0	1	1	0
la07	15	5	376	376	376	18.2	0	376	0.44	2	1	0	0	0	1	0
la08	15	5	369	369	369	19.6	0	370	0.44	3	2	0	0	1	1	0.271
la09	15	5	382	382	387.4	17.8	0	417	0.47	2	1	1	0	2	0	9.162
la10	15	5	443	443	443	17	0	443	0.45	2	1	0	−1	1	0	0
la11	20	5	413	425	436.8	50.6	2.9	469	0.78	4	2	1	0	1	0	13.56
la12	20	5	408	408	408	44.6	0	417	0.78	3	1	0	0	1	0	2.206
la13	20	5	382	419	430.2	45.8	9.7	454	0.83	2	1	2	1	2	1	18.85
la14	20	5	443	443	448.8	48.8	0	445	0.80	3	2	0	0	2	1	0.451
la15	20	5	378	451	456	42.2	19.3	477	0.80	1	4	0	1	1	1	26.19
la16	10	10	717	717	717	36	0	717	0.44	1	0	0	−1	0	0	0
la17	10	10	646	646	646	31.6	0	646	0.47	1	0	0	1	0	0	0
la18	10	10	663	663	663	36.8	0	663	0.44	1	1	0	−1	0	0	0
la19	10	10	617	617	617	62.4	0	617	0.45	2	0	0	−1	1	0	0
la20	10	10	756	756	756	34.2	0	756	0.45	2	0	0	−1	0	0	0
mt06	6	6	47	47	47	1	0	47	0.09	1	0	0	−1	0	0	0
mt10	10	10	655	655	655	50	0	655	0.47	1	0	0	1	0	0	0
mt20	20	5	387	439	442.6	48.2	13.4	450	0.78	2	1	1	1	1	0	16.28
Average						27.85217	1.969565		0.47	1.957	1	0.348	−0.174	0.652	0.348	3.781

Table 4 Computational results for DFJS instances with four FMUs

Name	Size		LB	IGA				Heuristic								
	n	m		$BC_{\max}$	Av. ($C_{\max}$)	Av. time (second)	RPD	$C_{\max}$	Time (second)	x_1	x_2	x_3	x_4	x_5	x_6	RPD
la01	10	5	413	413	413	1.8	0	413	0.16	1	0	0	0	−1	0	0
la02	10	5	394	394	394	1.8	0	394	0.14	1	0	0	0	−1	0	0
la03	10	5	349	349	349	2.2	0	349	0.16	1	0	0	0	−1	0	0
la04	10	5	369	369	369	2	0	369	0.17	2	0	0	0	−1	0	0
la05	10	5	380	380	380	1	0	380	0.16	1	0	0	0	−1	0	0
la06	15	5	413	413	413	9	0	413	0.33	2	0	0	0	−1	0	0
la07	15	5	376	376	376	9.6	0	376	0.34	3	1	0	0	−1	0	0
la08	15	5	369	369	369	12.6	0	369	0.33	2	0	0	0	−1	1	0
la09	15	5	382	382	382	11.6	0	382	0.34	3	0	1	0	−1	1	0
la10	15	5	443	443	443	7.8	0	443	0.34	3	0	3	0	−1	0	0
la11	20	5	413	413	413	29.6	0	413	0.59	1	0	0	2	1	0	0
la12	20	5	408	408	408	26.6	0	408	0.58	3	1	1	0	−1	0	0
la13	20	5	382	382	386	27.6	0	395	0.63	2	1	2	1	−1	0	3.403
la14	20	5	443	443	443	29.8	0	443	0.59	2	1	1	1	−1	0	0
la15	20	5	378	397	402	28.8	5	408	0.61	3	1	1	0	1	0	7.937
la16	10	10	717	717	717	20.2	0	717	0.33	1	0	0	0	−1	0	0
la17	10	10	646	646	646	16.4	0	646	0.34	2	0	0	0	−1	0	0
la18	10	10	663	663	663	24.4	0	663	0.33	1	0	0	0	−1	0	0
la19	10	10	617	617	617	33	0	617	0.33	1	0	0	0	−1	0	0
la20	10	10	756	756	756	18	0	756	0.33	2	0	0	0	−1	0	0
mt06	6	6	47	47	47	0.2	0	47	0.08	3	0	0	0	−1	0	0
mt10	10	10	655	655	655	31.2	0	655	0.34	1	0	0	0	−1	0	0
mt20	20	5	387	387	388.4	27	0	399	0.59	3	1	3	0	−1	1	3.101
Average						16.18261	0.21739		0.35	1.913	0.261	0.522	0.174	−0.83	0.13	0.628

Table 5 Results of one-way ANOVA for the methods: IGA and proposed heuristic, and for instances with two FMUs

Source	DF	SS	MS	F	P
Factor	1	94	94	0.27	0.605
Error	44	15157	344		
Total	45	15250			

Table 6 Results of one-way ANOVA for the methods: IGA and proposed heuristic, and for instances with three FMUs

Source	DF	SS	MS	F	P
Factor	1	37.7	37.7	0.91	0.344
Error	44	1815.6	41.3		
Total	45	1853.4			

Table 7 Results of one-way ANOVA for the methods: IGA and proposed heuristic, and for instances with four FMUs

Source	DF	SS	MS	F	P
Factor	1	1.94	1.94	0.86	0.359
Error	44	99.03	2.25		
Total	45	100.97			

in all three instance sets, the difference between the methods is not meaningful at a significance level of 5 %. However, the average computational time for the heuristic over all 69 instances is very low, and only $(0.4 + 0.5 + 0.8)/3 = 0.57$ seconds (on a Pentium IV, 2.2 GHz) compared to $(16.2 + 27.9 + 79.6)/3 = 41.2$ seconds (on a 2.0 GHz Intel Core2 processor) for IGA. Differences in the computers used for running the programs make the direct comparison among the running times difficult. However, even accounting for relative differences in the speed between the processors involved, the heuristic is significantly faster than IGA.

5 Conclusion

This paper investigates the distributed and flexible job-shop scheduling problem (DFJSP) with the objective of minimizing the overall completion time (makespan). The main purpose is to produce reasonable schedules very quickly. A simple and easily extendable heuristic based on a constructive procedure is presented. This heuristic uses an accurate, relatively comprehensive and flexible criterion for scheduling job operations and constructing a feasible high-quality solution. In this criterion, several factors affecting the quality of solutions are used, and to each of these factors, two weights (i.e. a constant weight and a variable weight) are assigned. By setting different values to the variable weights, different solutions are generated and evaluated. The proposed algorithm is tested on some problem instances from the literature in order to evaluate its performance. Since the proposed method is a heuristic, its results cannot be compared in a meaningful way with those of the method evaluated (IGA) as it is a metaheuristic based algorithm. Nevertheless, the heuristic is statistically compared with the IGA. For each of the three instance sets (i.e. DFJSP instances with two, three and four FMUs), a one-way ANOVA is performed to test the null hypothesis that the means of the two methods are equal. The results show that, in all three instance sets, the difference between the methods is not meaningful at a significance level of 5 %. However, the solutions of the heuristic are weakly dominated the solutions of the IGA in terms of average RPD. The computational times show the interest of the heuristic, since in a fraction of a second on average, it produces very good solutions. This algorithm has a simple structure, is easy to implement, and requires very little computational effort which makes it preferable over other more complex and time-consuming approaches, even if its results for benchmark instances are so weakly dominated the lower bounds in the literature. The procedure is useful in applications that deal with real-time systems and that involve the generation of initial schedules for local search and metaheuristic algorithms. Further research needs to be conducted in applying other criteria in the *TC* in order to improve the solution quality and adapt the approach to other objectives and process constraints. Moreover, the performance of the method proposed in this paper can be improved by doing a detailed study on the impact of different values of L_x_r, U_x_r and w_r on the quality of solutions and considering other combinations of values of these variables, which is left as a future research.

References

1. Chan FTS, Chung SH, Chan LY, Finke G, Tiwari MK (2006) Solving distributed FMS scheduling problems subject to maintenance: genetic algorithms approach. Robot Comput-Integr Manuf 22:493–504
2. Rosenau MD (1996) The PDMA handbook of new product development. Wiley, New York
3. Wang B (1997) Integrated product, process and enterprise design. Chapman & Hall, London
4. Jia HZ, Fuh JYH, Nee AYC, Zhang YF (2003) A modified genetic algorithm for distributed scheduling problems. J Intell Manuf 15:351–362
5. Giovanni LD, Pezzella F (2010) An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. Eur J Oper Res 200:395–408
6. Wang L, Shen W (eds) (2007) Process planning and scheduling for distributed manufacturing. Springer, London
7. Baker K (1974) Introduction to sequencing and scheduling. Wiley, NewYork
8. Pinedo M (2002) Scheduling: theory, algorithms and systems. Prentice-Hall, Englewood Cliffs
9. Garey MR, Johnson DS, Sethi R (1976) The complexity of flow shop and job-shop scheduling. Math Oper Res 1(2):117–129
10. Wang S-Y, Wang L, Liu M, Xu Y (2013) An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. Int J Prod Econ (in press)
11. Gao J, Chen R, Deng W (2013) An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. Int J Prod Res 51(3):641–651
12. Jia HZ, Fuh JYH, Nee AYC, Zheng YF (2002) Web-based multi-functional scheduling system for a distributed manufacturing environment. Concurr Eng Res Appl 10(1):27–39
13. Chan FTS, Chung SH, Chan PLY (2005) An adaptive genetic algorithm with dominated genes for distributed scheduling problems. Expert Syst Appl 29:364–371
14. Chan FTS, Chung SH, Chan PLY (2006) Application of genetic algorithms with dominated genes in a distributed scheduling problem in flexible manufacturing. Int J Prod Res 44(3):523–543
15. Dell'Amico M, Trubian M (1993) Applying tabu-search to the job-shop scheduling problem. Ann Oper Res 4:231–252

16. Matsuo H, Suh C, Sullivan R (1988) A controlled search simulated annealing method for the general job-shop scheduling problem. Tech rep 03-04-88, Department of Management, The University of Texas, Austin
17. Van Laarhoven P, Aarts E, Lenstra J (1992) Job shop scheduling by simulated annealing. Oper Res 40:113–125
18. Montgomery DC (2000) Design and analysis of experiments, 5th edn. Wiley, New York