Int J Adv Manuf Technol (2014) 71:519–528
DOI 10.1007/s00170-013-5510-z

ORIGINAL ARTICLE

A heuristic algorithm for solving flexible job shop scheduling problem

Mohsen Ziaee

Received: 14 December 2011 / Accepted: 18 November 2013 / Published online: 1 December 2013

Abstract This paper deals with the flexible job shop scheduling problem with the objective of minimizing the makespan. An efficient heuristic based on a constructive procedure is developed to obtain high-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 can obtain effective solutions in very short and nearly zero time and is comparable with even metaheuristic algorithms and promising for practical problems.

Keywords Scheduling · Flexible job shop · Makespan · Heuristic

1 Introduction

In recent years, scheduling plays a vital role in manufacturing environments due to the growing consumer demand for variety, reduced product life cycles, changing markets with global competition, and the rapid development of new technologies. These economic and commercial market pressures have challenged the manufacturers to output products with low production cost and to deliver to customers on time. Scheduling, a decision making process which deals with the allocation of limited resources to tasks over time, plays an important role in achieving these goals. The Job shop Scheduling Problem (JSP) is one of the most popular scheduling problems existing in practice. It has attracted the attention of many researchers due to its wide applicability and inherent difficulty. The JSP has been proven to be NP-hard [10]. In the $n \times m$ classical JSP, a set of n jobs must be processed on a group of m machines, where the processing of each job i consists of J_i operations performed on these machines. Each job has a specified processing order on the machines which is fixed and known in advance, i.e., each operation has to be performed on a given machine. Moreover, the processing times of all operations are fixed and known. Each machine is continuously available from time zero and can process at most one operation at a time. The operations are processed on the machines without interruption [2]. A typical performance indicator for the JSP is the makespan, i.e., the time needed to complete all the jobs.

In modern manufacturing environments, a machine may have the flexible capability to be set up to process more than one type of operations. This capability leads to an extension of the classical JSP called the Flexible Job shop Scheduling Problem (FJSP). This problem (FJSP) is very important in both academic and application fields. In the FJSP, 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. The FJSP is more difficult than the classical JSP because it contains an additional problem, i.e., assigning operations to machines. Therefore, the FJSP is a problem of challenging complexity and high practical value. This problem is known to be strongly NP-hard even if each job has at most three operations and there are two machines [10].

In this paper, an efficient heuristic method based on a constructive procedure is presented to solve the FJSP with the objective of minimizing the makespan (Sect. 3). The main purpose is to produce good quality 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 [8, 22, 27]. In order

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

to evaluate the performance of the proposed heuristic, it is implemented using several well-known benchmark problems, and the results of the computational experiments are presented (Sect. 4). The results show that our novel method can produce good solutions in a very short time. Concluding remarks are given in the last section.

Other assumptions considered in this paper are as follows:

1. Jobs are independent of each other.
2. Machines are independent of each other.
3. Setup and transportation times are negligible.
4. An operation cannot be performed by more than one machine at the same time.
5. All jobs have equal priorities.
6. All jobs are available at time zero.

The notations used throughout the paper are as follows:

n	Number of jobs
m	Number of machines
i, z	Index of jobs; $i, z=1,\ldots,n$
J_i	Number of operations of job i
$maxJ$	Maximum number of operations per job (i.e., $maxJ=\max_i J_i$)
j	Index of operations; $j=1,\ldots, J_i$
k, y	Index of machines; $k, y=1,\ldots,m$
t_{ijy}	Processing time of operation j of job i on machine y
c_{ij}	Completion time of operation j of job i

2 Literature review

During the past two decades, the FJSP has captured the interest of many researchers. Exact methods based on a disjunctive graph representation of the problem have been developed, but they are not efficient for instances with more than 20 jobs and 10 machines [25]. However, many approximation algorithms, mainly metaheuristics, have been successfully applied to the FJSP. Dauzere-Peres and Paulli [7] presented a tabu search (TS) algorithm based on a new neighborhood structure for the FJSP. Mastrolilli and Gambardella [21] improved Dauzere-Peres' TS algorithm and developed two neighborhood functions. These two researches are well-known studies in the literature on the FJSP. Wang et al. [28] presented a bi-population based estimation of distribution algorithm (BEDA) to solve the FJSP. They investigated the influence of parameter setting on the performance of the algorithm by using the design of experiment based testing and used simulation tests to demonstrate the effectiveness of the BEDA in solving the FJSP. Wang et al. [29] proposed an artificial bee colony (ABC) algorithm for solving the FJSP. In this method, they developed a new local search based on critical path to perform local exploitation effectively. Ho et al. [14] presented an architecture for learning and evolving flexible job shop schedules called learnable genetic architecture in which the knowledge extracted from previous generation by its schemata learning module is used to influence the diversity and quality of offsprings, unlike the canonical evolution algorithm, where random elitist selection and mutational genetics are assumed. Zhang et al. [35] also developed a genetic algorithm (GA) to schedule the jobs in the flexible job shop. Bozejko et al. [3] solved the FJSP using two double-level parallel metaheuristic algorithms called meta2heuristics including two major modules: machine selection and operations scheduling. On the machine selection module, two metaheuristics, tabu search and population-based approach, are applied to determine an assignment of operations to machines. An insertion algorithm and tabu search is used as operations scheduling module to solve the problem after having assigned operations to machines. They also presented an approximation algorithm based on the tabu search metaheuristic which includes a new neighborhood structure called "golf neighborhood" to solve the problem [4]. Bagheri et al. [1] proposed an artificial immune algorithm (AIA) to solve the FJSP. An approach based on a combination of the ant colony optimization (ACO) and tabu search algorithms was presented by Liouane et al. [20]. Rossi and Dini [26] also presented a metaheuristic approach based on the ACO to solve the FJSP. Their method was designed for a real environment and is capable of solving more general cases of the problem where

Fig. 1 General outline of the proposed heuristic algorithm

```
for j:= 1 to maxJ do
{
    until jth operation of all jobs are scheduled, repeat
    {
        • Find i, k (such that: j ≤ J_i and jth operation of job i is an unscheduled
          operation and machine k is capable of processing this operation) that
          minimizes TC.

        • Schedule jth operation of job i on the last position of current partial
          sequence on machine k.
    }
}
```

```
Initialization:
•   Sort the jobs in increasing order of their sj_i and call the resulting set: i_sort.
    Let i_sort_z be zth job of the list i_sort.
•   Sort the machines in increasing order of their sk_y and call the resulting set:
    k_sort. Let k_sort_y be yth machine of the list k_sort.

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_6:=L_x_6 to U_x_6 do
for x_7:=0 to 1 do
for x_8:=0 to 1 do
{
        % Beginning of a schedule generation

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

                        for i':=1 to n do
                        {
                                Set i:=x_7.(i_sort_i')+(1-x_7).(i_sort_(n-i'+1)),

                                if (j ≤ J_i and jth operation of job i is an unscheduled
                                operation) then
                                {
                                        for k':=1 to m do
                                        {
                                                Set k:=x_8.(k_sort_k')+(1-x_8).(k_sort_(m-k'+1)),

                                                if (machine k is capable of processing jth
                                                operation of job i) then
                                                {
                                                        Set TC:= Σ_{r=1}^{6} w_r.x_r.C_r
                                                        if TC<TC* then
                                                        {
                                                                Set TC*:= TC
                                                                Set z:=i
                                                                Set y:=k
                                                        }
                                                }
                                        }
                                }
                        }

                        if TC*<M then schedule jth operation of job z on the last position
                        of the current partial sequence on machine y to finish at time
                        C_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,…, 8)
        corresponding to C_max*.
}
```

Fig. 2 Pseudo-code of the proposed heuristic method

there are sequence-dependent setup times and transportation times. Girish and Jawahar [11] presented two metaheuristic algorithms, a GA and an ACO to solve the problem. Ennigrou and Ghédira [9] developed two multi-agent approaches based on the tabu search algorithm for solving the problem. Yazdani et al. [32] suggested a parallel variable neighborhood search algorithm which uses multiple independent searches increasing the exploration in the search space. Li et al. [18] presented a hybridization of tabu search algorithm and a fast public critical block neighborhood structure (called TSPCB) to solve the problem. An approach based on an integration of ACO and knowledge model called knowledge-based ant colony optimization (KBACO) algorithm was presented by Xing et al. [30]. Pezzella et al. [24] presented a GA for the FJSP in which different strategies for generating the initial population, selecting the individuals for reproduction, and reproducing new individuals are integrated. They showed that the integration of more strategies in a genetic framework leads to better results, with respect to other GAs. Gutiérrez and García-Magariño [12] presented a hybrid method which combines GAs with repair heuristics. The algorithm first uses two GAs to find a non-optimal schedule, which does not satisfy all constraints of the problem. It then applies repair heuristics to refine the solution, instead of inserting this knowledge into the GAs. Hmida et al. [13] presented a new discrepancy-based method, called Climbing Depth-bounded Discrepancy Search to solve the problem, and presented various neighborhood structures related to assignment and sequencing problems using the concept of discrepancy to expand the search tree. Li et al. [19] presented a hybridization of particle swarm optimization (PSO) and TS algorithms to solve the problem. In the sequencing stage of the proposed hybrid algorithm, the PSO is used to produce a swarm of high-quality candidate solutions, and in the machine assignment stage of the algorithm, the TS is applied to find a near optimal solution around given good solutions. Chiang and Lin [6] investigated the multi-objective FJSP with the makespan, the total workload, and the maximum workload as objectives, and developed an evolutionary algorithm to generate set of Pareto solutions. Yuan et al. [34] proposed a hybrid harmony search (HHS) algorithm for solving the FJSP. They also presented hybrid differential evolution (HDE) algorithms to solve the problem [33]. They developed a new conversion mechanism to make the differential evolution algorithm that works on the continuous domain applicable to solve the discrete FJSP; and in the local search phase of the method, they presented a speed-up method for finding an acceptable schedule within the neighborhood more quickly. Huang et al. [15] consider the FJSP with the due window and the sequence-dependent setup times and developed a two pheromone ant colony optimization to solve the problem. Xiong et al. [31] address the FJSP with random machine breakdowns and present a multi-objective evolutionary algorithm to solve the problem.

Table 1 Parameter settings for the heuristic

Parameter	Value	Parameter	Value	Parameter	Value
w_1	2	L_x_1	1	U_x_1	4
w_2	1	L_x_2	0	U_x_2	1
w_3	1	L_x_3	−1	U_x_3	0
w_4	2	L_x_4	−3	U_x_4	−1
w_5	1	L_x_5	−1	U_x_5	0
w_6	1	L_x_6	−1	U_x_6	0

3 Proposed heuristic approach

In this section, we present an efficient heuristic algorithm to solve the FJSP. 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. This algorithm has a simple structure and great flexibility, 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. Some notations that will be used in the algorithm are defined as follows:

- A_{ij} Set of machines which are capable to execute operation j of job i
- N_{ij} Number of members of the set A_{ij}
- s'_{ij} Mean processing time of operation j of job i over the machines belonging to the set A_{ij} (i.e., $s'_{ij} = \left(\sum_{y \in A_{ij}} t_{ijy}\right)/N_{ij}$)
- sj_i total mean processing time of job i (i.e., $sj_i = \sum_{j=1}^{J_i} s'_{ij}$)
- sk_y Total weighted processing time on machine y which is calculated as follows: $sk_y = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ if\ y \in A_{ij}}}^{J_i} \frac{t_{ijy}}{s'_{ij}}$,
- M A large number

Table 2 Recent algorithms to solve the FJSP

Name	Reference	NumRuns
TSPCB	[18]	50
AIA	[1]	10
KBACO	[30]	10
HHS	[34]	30
TS3	[4]	–
HDE-N2	[33]	50
BEDA	[28]	50
ABC	[29]	50

Table 3 Computational results for the Brandimarte benchmark instances

TSPCB							AIA			KBACO				HHS				TS3			HDE-N2
Name	n×m	LB	BCmax	AV (Cmax)	Time	RPD	BCmax	Time	RPD	BCmax	AV (Cmax)	Time	RPD	BCmax	AV (Cmax)	Time	RPD	BCmax	Time	RPD	BCmax
MK01	10×6	36	40	40.3	140	11.11	40	972.1	11.11	39	39.8	4,692	8.33	40	40	2.1	11.111	41	47.87	13.889	40
MK02	10×6	24	26	26.5	965.5	8.33	26	1,034.6	8.33	29	29.1	5,922	20.83	26	26.63	22.2	8.3333	30	36.12	25	26
MK03	15×8	204	204	204	49	0.00	204	2,473.7	0.00	204	204	45,552	0.00	204	204	0.3	0	204	330.1	0	204
MK04	15×8	48	62	64.88	2,041	29.17	60	1,520.7	25.00	65	66.1	12,246	35.42	60	60.03	31.2	25	65	115.22	35.417	60
MK05	15×4	168	172	172.9	1,011.5	2.38	173	1,719.5	2.98	173	173.8	9,732	2.98	172	172.8	224.1	2.381	174	106.12	3.5714	172
MK06	10×15	33	65	67.38	1,359	96.97	63	2,456.2	90.91	67	69.1	24,126	103.03	58	59.13	1,821.9	75.758	71	2,119.53	115.15	57
MK07	20×5	133	140	142.21	1,764.5	5.26	140	1,619.2	5.26	144	145.4	17,118	8.27	139	139.57	317.7	4.5113	148	112.2	11.278	139
MK08	20×10	523	523	523	232.5	0.00	523	3,922.5	0.00	523	523	52,446	0.00	523	523	0.6	0	551	988.05	5.3537	523
MK09	20×10	299	310	311.29	3,519	3.68	312	3,897.1	4.35	311	312.2	47,016	4.01	307	307	11.7	2.6756	410	1,286.91	37.124	307
MK10	20×15	165	214	219.15	4,491.5	29.70	214	3,845.4	29.70	229	233.7	74,940	38.79	205	211.13	11,190.3	24.242	267	1,349.4	61.818	198
Average					1,557.35	18.66		2,346.10	17.76			29,379.00	22.17			1,362.21	15.40		649.15	30.86	

TSPCB	HDE-N2			BEDA				ABC				Heuristic										
Name	AV (Cmax)	Time	RPD	BCmax	AV (Cmax)	Time	RPD	BCmax	AV (Cmax)	Time	RPD	BCmax	Time	x1	x2	x3	x4	x5	x6	x7	x8	RPD
MK01	40	200.5	11.111	40	41.02	54.5	11.111	40	40	161.5	11.111	42	0.09	3	0	0	−3	0	0	0	1	16.67
MK02	26	304.5	8.3333	26	27.25	108	8.3333	26	26.5	1781.5	8.3333	28	0.17	2	0	−1	−1	−1	0	1	0	16.67
MK03	204	1,535	0	204	204	109	0	204	204	59.5	0	204	0.52	3	0	−1	−1	−1	−1	0	0	0.00
MK04	60	629	25	60	63.69	451	25	60	61.22	1,947	25	75	0.20	4	0	−1	−1	0	−1	0	0	56.25
MK05	172.82	1,894.5	2.381	172	173.38	355	2.381	172	172.98	964.5	2.381	179	0.20	4	0	0	−2	−1	0	0	1	6.55
MK06	58.64	4,916	72.727	60	62.83	1,510.5	81.818	60	64.48	3,330.5	81.818	69	0.45	3	1	-1	-1	-1	0	0	0	109.09
MK07	139.42	1,319	4.5113	139	141.55	853.5	4.5113	139	141.42	6,592	4.5113	149	0.39	2	0	−1	−1	−1	0	0	1	12.03
MK08	523	9,470.5	0	523	523	215	0	523	523	116.5	0	555	0.66	3	0	−1	−3	−1	−1	0	0	6.12
MK09	307	6,143.5	2.6756	307	310.35	4,599.5	2.6756	307	308.76	4,560.5	2.6756	342	0.94	4	0	0	−3	−1	0	0	0	14.38
MK10	201.52	13,290	20	206	211.92	9,505.5	24.848	208	212.84	11,855.5	26.061	242	1.20	2	0	−1	−1	0	0	1	0	46.67
Average		3,970.25	14.67			1,776.15	16.07			3,136.90	16.19		0.48	3	0.1	−0.7	−1.7	−0.7	−0.3	0.2	0.3	28.44

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

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

$$TC = \sum_{r=1}^{6} w_r.x_r.C_r$$

such that,

C_1 $max\ (Cmax_y, c_{i,j\text{-}1}) + t_{ijy}$
C_2 $max\ (0, (c_{i,j\text{-}1} - Cmax_y))$
C_3 $max\ (0, (Cmax_y - c_{i,j\text{-}1}))$
C_4 t_{ijy}
C_5 sj_i
C_6 sk_y

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. w_r $(r=1, 2,\ldots, 6)$ are constants and x_r $(r=1, 2,\ldots, 6)$ are integer variables used to increase the flexibility and effectiveness of criterion TC and 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. $Cmax_y$ is the maximum completion time across all the operations scheduled on machine y; that is, $Cmax_y$ is equal to the completion time of the operation situated just before operation j of job i on machine y. C_1, C_2, and C_3 are applied to decrease $Cmax_y$, idle times, and flowtime of jobs, respectively; clearly, all these three objectives affect the main objective function, i.e., C_{max}. For assigning operations to a machine, their processing time is also taken into account by C_4. According to C_5, the jobs with larger sj_i are scheduled sooner. C_6 is used for taking into account the total weighted processing time of machines.

Other notations used in the pseudo-code of the heuristic are as follows:

TC^*: denotes the best value of TC. After each operation is scheduled, TC^* is reset to M.
L_x_r $(r=1, 2,\ldots, 6)$: lower limit of x_r
U_x_r $(r=1, 2,\ldots, 6)$: upper limit of x_r

The algorithm starts by scheduling the first operation of all jobs, then the second operation of them, and so on. For each j $(j=1, 2,.., maxJ)$, the algorithm sorts the jobs in increasing (decreasing) order of their sj_i and, for each job i taken in this order, evaluates its jth operation (if $j \leq J_i$ and operation j of job i is an unscheduled operation).

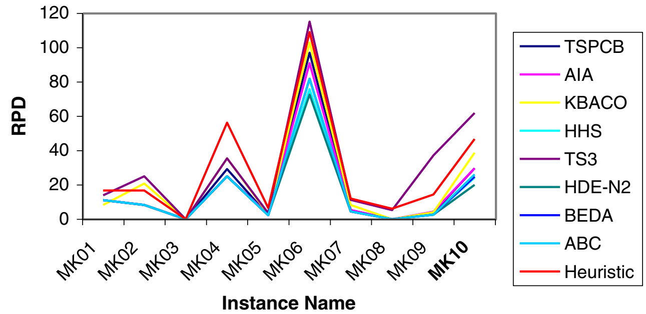

Fig 3 Comparison of the RPD values of the methods of Table 3 for different instances

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 smaller (greater) sj_i is selected for scheduling sooner than the other operation. Binary variable x_7 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. For evaluating operation j of job i, similarly the algorithm first sorts the machines in increasing (decreasing) order of their sk_y and, for each machine y taken in this order, evaluates this operation to be scheduled on machine y (if machine y is capable of processing this operation, i.e., $y \in A_{ij}$). Binary variable x_8 is applied for setting the order of the sorting; it takes a value of 1 for increasing order and 0 for decreasing one. Sorting the jobs and the machines, 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 jobs and machines instead of randomly selecting them, and interestingly observed that these sortings can lead to better solutions. x_r^* $(r=1, 2,\ldots, 8)$ are the best values of variables x_r (i.e., the values corresponding to the best solutions). Indeed, for various values of x_r $(r=1, 2,\ldots, 8)$, the algorithm of 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. The values of variables x_r for this best solution are also reported and denoted by x_r^*.

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

Table 4 Results of one-way ANOVA for the nine methods of Table 3

Source	DF	SS	MS	F	P
Factor	8	2,802	350	0.43	0.898
Error	81	65,510	809		
Total	89	68,312			

Fig. 4 Results of Tukey's pairwise comparisons test for the methods of Table 3

```
Tukey 95% Simultaneous Confidence Intervals
All Pairwise Comparisons

Individual confidence level = 99.80%


Proposed Heuristic subtracted from:

         Lower  Center  Upper  ---------+---------+---------+---------+
TSPCB   -50.34   -9.78  30.78    (-------------*------------)
AIA     -51.24  -10.68  29.88    (------------*-------------)
KBACO   -46.83   -6.28  34.28     (-------------*------------)
HHS     -53.60  -13.04  27.52   (-------------*------------)
TS3     -38.14    2.42  42.98        (-------------*------------)
HDE-N2  -54.33  -13.77  26.79   (------------*-------------)
BEDA    -52.93  -12.37  28.19   (-------------*------------)
ABC     -52.81  -12.25  28.31   (-------------*------------)
                               ---------+---------+---------+---------+
                                      -30         0        30        60
```

4 Computational results

This section describes the computational experiments performed 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 used the settings of Table 1 for benchmarking the presented algorithm. The algorithm was coded in C language and run on a Pentium IV, 2.2 GHz and 2.0 GB RAM PC. The benchmark problems used were the set of ten instances taken from Brandimarte [5] (BRdata) and five problem instances taken from Kacem et al. [16, 17]. These well-known test problems have been used by many papers in the literature to benchmark the proposed methods. We compared the results of our algorithm with those of some most recent algorithms in the literature listed in Table 2. The column "NumRuns" shows the number of runs of the algorithm to solve the benchmark instances. Table 3 shows comparison of the running time and makespan results obtained by our algorithm and the methods of Table 2. The first and second columns indicate the name and size of each problem instance, respectively. *LB* refers to the best-known lower bound [21]. *BCmax*, *AV*(*Cmax*), and *Time* stand for the best makespan, the average makespan, and the total computational time regarding the number of runs in seconds, respectively. The results obtained by our algorithm are shown in the last 11 columns. *BCmax* and *Time* represent the makespan and the computational time for each instance, respectively. The best values of variables x_r (i.e., x_r^*), $r=1, 2,\ldots, 8$ have also reported in Table 3. The average value of each variable x_r, $r=1, 2,\ldots, 6$ can be considered as the relative effect of the corresponding criterion on the quality of solutions. For example, all x_2 values but one are zero that means idle times have almost no effect on C_{max}. The values of other variables (i.e., x_r, $r=1, 3, 4, 5, 6$) have relatively high variance as it can be seen in the table, 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 values of x_3, x_4, x_5, and x_6 are negative that means they have adverse effect on C_{max}. Average values of variables x_7 and x_8 are nearer 0 than 1. It is because the jobs with larger sj_i 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. Similarly, the machines with larger sk_y which are firstly selected for scheduling have more sensibility and a determinative effect on C_{max}. Indeed, sorting the jobs and the machines before evaluating them for scheduling helps to keep balanced distribution of the operations among the machines. *RPD* is the relative percentage deviation to *LB* and calculated as follows:

$$RPD = \frac{\mathrm{Cmax}_{alg} - LB}{LB} \times 100,$$

Where $Cmax_{alg}$ is the best makespan obtained by the algorithm. As shown in the table, TSPCB, AIA, KBACO, HHS, TS3, HDE-N2, BEDA, ABC, and our algorithm have average RPD values of 18.67, 17.76, 22.17, 15.40, 30.86, 22.17, 14.67, 16.07, 16.19, and 28.44, respectively. To see the differences between the nine algorithms, we investigate them graphically. Figure 3 shows a graphical comparison of the RPD values of the methods for different instances.

Herein, the heuristic is statistically compared with the other eight methods. A one-way analysis of variance (ANOVA) [23] is performed to test the null hypothesis that the means of the nine methods are equal. The results of this ANOVA are presented in Table 4. As can be seen, the difference between the methods is not meaningful at a significance level of 5 %. The methods are also compared via Tukey's pairwise comparisons test [23]. The results presented in Fig. 4 show the interesting observation that there is no significant difference between the proposed algorithm and the other eight algorithms. However, as

Table 5 Computational results for the Kacem benchmark instances

TSPCB							HDE-N2				KBACO				BEDA		
Name	n×m	LB	BCmax	AV (Cmax)	Time	RPD	BCmax	AV (Cmax)	Time	RPD	BCmax	AV (Cmax)	Time	RPD	BCmax	AV (Cmax)	Time
Case 1	4×5	11	11	11	2.5	0	11	11	4.5	0	11	11	900	0	11	11	0.5
Case 2	8×8	14	14	14.2	234	0	14	14	15.5	0	14	14.3	3,882	0	14	14	11.5
Case 3	10×7	11	11	11	260.5	0	11	11	23	0	11	11	3,966	0	11	11	15
Case 4	10×10	7	7	7.1	86	0	7	7	18.5	0	7	7.4	6,642	0	7	7	21
Case 5	15×10	11	11	11.7	491	0	11	11	109.5	0	11	11.3	9,504	0	11	11	744
Average					214.80	0			34.20	0			4,978.80	0			158.40

TSPCB	BEDA	ABC				Heuristic						
Name	RPD	BCmax	AV (Cmax)	Time	RPD	BCmax	Time	x1	x6	x7	x8	RPD
Case 1	0	11	11	2	0	11	0.015	4	0	0	0	0.00
Case 2	0	14	14	14.5	0	15	0.094	3	0	0	0	7.14
Case 3	0	11	11	79	0	13	0.141	3	−1	0	1	18.18
Case 4	0	7	7	22.5	0	7	0.218	4	−1	0	0	0.00
Case 5	0	11	11	269.5	0	12	0.532	4	0	0	0	9.09
Average	0			77.50	0		0.20	3.6	−0.4	0	0.2	6.88

shown in Table 3, the average computational time for the heuristic is very low, and only 0.5 s (on a 2.2 GHz CPU and 2.0 GB RAM) compared to 1,557.35 s (on a 1.6 GHz CPU and 512 MB RAM) for TSPCB, 2,346.10 s (on a 2.0 GHz CPU and 256 MB RAM) for AIA, 29,379.00 s (on a 2.4 GHz CPU and 1 GB RAM) for KBACO, 1,362.21 s (on a 2.83 GHz CPU and 15.9 GB RAM) for HHS, 649.15 s (on a 1.0 GHz CPU and 8 GB RAM) for TS3, 3,970.25 s (on a 2.83 GHz CPU and 15.9 GB RAM) for HDE-N2, 1,776.15 s (on a 3.2 GHz CPU) for BEDA, and 3,136.90 s (on a 2.83 GHz CPU and 3.21 GB RAM) for ABC. Differences in the computers applied 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 the other eight algorithms.

Similarly, the computational experiments were performed on the Kacem benchmark instances and the results are reported in Table 5. The first and second columns indicate the name and size of each problem instance, respectively. *LB* is the best makespan value found so far. $BCmax$, $AV(Cmax)$, and *Time* stand for the best makespan, the average makespan, and the total computational time regarding the number of runs in seconds, respectively. The results obtained by our algorithm are shown in the last 11 columns. As shown in the table, the proposed heuristic algorithm has an average RPD value of 6.88 compared to zero for the other algorithms. We similarly performed ANOVA and Tukey's pairwise comparisons tests to statistically compare RPDs of the methods, and observed that the difference between the methods is not meaningful at a significance level of 5 %. However, as shown in Table 3, the average computational time for the heuristic is very low, and only 0.2 s compared to 214.80 s for TSPCB, 34.20 s for HDE-N2, 4,978.80 s for KBACO, 158.40 s for BEDA, and 77.50 s for ABC. Therefore, the proposed algorithm is an efficient method for the problem and can obtain good solutions in very short and nearly zero time showing that the method is very promising for practical problems.

5 Conclusion

This paper investigates the FJSP with the objective of minimizing makespan. A simple and easily extendable heuristic based on a constructive procedure is presented. This heuristic algorithm 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 (including 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 heuristic is tested on benchmark instances from the literature in order to evaluate its performance. The computational results show that the approach can yield good quality solutions with very little computational effort. Since the proposed method is a heuristic, its results cannot be compared in a meaningful way with those of the methods evaluated as they are metaheuristic-based algorithms. Nevertheless, the results of ANOVA tests show that the difference between the proposed heuristic and the metaheuristic methods evaluated is not meaningful at a significance level of 5 %. However, among the methods, the proposed heuristic is the most efficient method with the least average computational time, and it produces very good solutions in a fraction of a second on average. The procedure can be very 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 to adapt the approach to other objectives and process constraints. Moreover, the performance of the method proposed in this paper can be even better 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 that is left as a future research.

References

1. Bagheri A, Zandieh M, Mahdavi I, Yazdani M (2010) An artificial immune algorithm for the flexible job-shop scheduling problem. Futur Gener Comput Syst 26:533–541
2. Baker K (1974) Introduction to sequencing and scheduling. Wiley, NewYork
3. Bozejko W, Uchronski M, Wodecki M (2010) Parallel hybrid metaheuristics for the flexible job shop problem. Comput Ind Eng 59:323–333
4. Bozejko W, Uchronski M, Wodecki M (2010) The new golf neighborhood for the flexible job shop problem. Procedia Comput Sci 1: 289–296
5. Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 41:157–183
6. Chiang T-C, Lin H-J (2013) A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling. Int J Prod Econ 141:87–98
7. Dauzere-Peres S, Paulli J (1997) An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann Oper Res 70: 281–306
8. Dell'Amico M, Trubian M (1993) Applying tabu-search to the job-shop scheduling problem. Ann Oper Res 4:231–252
9. Ennigrou M, Ghédira K (2008) New local diversification techniques for flexible job shop scheduling problem with a multi-agent approach. Auton Agent Multi-Agent Syst 17:270–287
10. Garey MR, Johnson DS, Sethi R (1976) The complexity of flow shop and job-shop scheduling. Math Oper Res 1(2):117–129

11. Girish BS, Jawahar N (2009) Scheduling job shop associated with multiple routings with genetic and ant colony heuristics. Int J Prod Res 47(14):3891–3917
12. Gutiérrez C, García-Magariño I (2011) Modular design of a hybrid genetic algorithm for a flexible job-shop scheduling problem. Knowl-Based Syst 24(1):102–112
13. Hmida AB, Haouari M, Huguet M-J, Lopez P (2010) Discrepancy search for the flexible job shop scheduling problem. Comput Oper Res 37:2192–2201
14. Ho NB, Tay JC, Lai EM-K (2007) An effective architecture for learning and evolving flexible job-shop schedules. Eur J Oper Res 179:316–333
15. Huang R-H, Yang C-L, Cheng W-C (2013) Flexible job shop scheduling with due window: a two-pheromone ant colony approach. Int J Prod Econ 141:685–697
16. 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 Part C 32(1):1–13
17. Kacem I, Hammadi S, Borne P (2002) Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Math Comput Simul 60:245–276
18. Li J-Q, Pan Q-K, Suganthan PN, Chua TJ (2010) A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. Int J Adv Manuf Technol 52(5–8):683–697
19. Li J-Q, Pan Q-K, Xie S-X, Jia B-X, Wang Y-T (2010) A hybrid particle swarm optimization and tabu search algorithm for flexible job-shop scheduling problem, International Journal of Computer Theory and. Engineering 2(2):1793–8201
20. Liouane N, Saad I, Hammadi S, Borne P (2007) Ant systems and local search optimization for flexible job shop scheduling production. Int J Comput Commun Control 2(2):174–184
21. Mastrolilli M, Gambardella LM (2000) Effective neighborhood functions for the flexible job shop problem. J Sched 3(1):3–20
22. 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, Dept. of Management, The University of Texas, Austin
23. Montgomery DC (2000) Design and analysis of experiments, 5th edn. John Wiley & Sons, NewYork
24. Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 35: 3202–3212
25. Pinedo M (2002) Scheduling: theory, algorithms, and systems. Prentice-Hall, Englewood cliffs
26. Rossi A, Dini G (2007) Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. Robot Comput Integr Manuf 23:503–516
27. Van Laarhoven P, Aarts E, Lenstra J (1992) Job shop scheduling by simulated annealing. Oper Res 40:113–125
28. Wang L, Wang S, Xu Y, Zhou G, Liu M (2012) A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. Comput Ind Eng 62:917–926
29. Wang L, Zhou G, Xu Y, Wang S, Liu M (2012) An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. Int J Adv Manuf Technol 60:03–315
30. Xing L-N, Chen Y-W, Wang P, Zhao Q-S, Xiong J (2010) A knowledge-based ant colony optimization for flexible job shop scheduling problems. Appl Soft Comput 10:888–896
31. Xiong J, Xing L-N, Chen Y-W (2013) Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. Int J Prod Econ 141:112–126
32. Yazdani M, Amiri M, Zandieh M (2010) Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Expert Syst Appl 37:678–687
33. Yuan Y, Xu H (2013) Flexible job shop scheduling using hybrid differential evolution algorithms. Comput Ind Eng 65:246–260
34. Yuan Y, Xu H, Yang J (2013) A hybrid harmony search algorithm for the flexible job shop scheduling problem. Appl Soft Comput 13: 3259–3272
35. Zhang G, Gao L, Shi Y (2011) An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Syst Appl 38(4):3563–3573