Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Job shop scheduling with makespan objective: A heuristic approach

Mohsen Ziaee*

Department of Industrial Engineering, University of Bojnord, 94531-55111 Bojnord, Iran

C H R O N I C L E

Article history:
Received July 2 2013
Received in revised format
September 7 2013
Accepted November 23 2013
Available online
November 22 2013

Keywords:
Scheduling
Job shop
Makespan
Heuristic

A B S T R A C T

Job shop has been considered as one of the most challenging scheduling problems and there are literally tremendous efforts on reducing the complexity of solution procedure for solving job shop problem. This paper presents a heuristic method to minimize makespan for different jobs in a job shop scheduling. The proposed model is based on a constructive procedure to obtain good quality schedules, very quickly. The performance of the proposed model of this paper is examined on standard benchmarks from the literature in order to evaluate its performance. Computational results show that, despite its simplicity, the proposed heuristic is computationally efficient and practical approach for the problem.

1. Introduction

The job shop scheduling problem (JSP) is one of the most popular scheduling problems in the world (Jain & Meeran, 1998). It has attracted many researchers due to its wide applicability and inherent difficulty (Carlier & Pinson, 1989; Kolonko, 1999; Nowicki & Smutnicki, 1996; Yamada & Nakano, 1996; Ho et al., 2007). In the $n \times m$ classical JSP, 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 pre-specified processing order on the machines, which is fixed and known in advance, i.e., each operation needs to be executed on a given machine. Moreover, the processing times of all operations are also fixed and known in advance. Each machine is continuously available from time zero and can process at most one operation at a time. The operations are processed on all machines without interruption (Baker, 1974; Pinedo, 2002). A common objective function is to minimize the makespan, which is the time needed to complete all the jobs.

In this paper, we present a heuristic method based on a constructive procedure to solve the JSP with the objective of minimizing makespan (section 3). The primary objective is to produce reasonable and applicable schedules, very quickly. It can also be used to improve the quality of the initial feasible

* Corresponding author. Tel.: +98 584 2284611; fax: +98 584 2410700
E-mail: ziaee@iust.ac.ir (M. Ziaee)

doi: 10.5267/j.ijiec.2013.11.004

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 (Dell'Amico & Trubian, 1993; Matsuo et al., 1988; Van Laarhoven et al., 1992). In order to evaluate the performance of the proposed heuristic, The proposed mode has examined on several well-known benchmarks and the results of the computational experiments are presented (section 4). The results show that our novel method can obtain good solutions in 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) All jobs have equal priorities.
(5) 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; $i,z=1,...,n$,
***j*:** index of operations; $j=1,..., m$,
***k,y*:** index of machines; $k,y=1,...,m$,
t_{ij}: processing time of operation j of job i,
t'_{iy}**:** processing time of job i on machine y, that is the processing time of an operation of job i which is processed on machine y,
c_{ij}**:** completion time of operation j of job i.

2. Literature review

The JSP has been proven to be NP-hard (Garey et al., 1976). Therefore, only small size instances of the JSP can be solved optimally with good computational time using exact solution methods (Carlier & Pinson, 1989; Lenstra, 1976). When the problem size increases, the computational time of exact algorithms grows exponentially. Heuristic algorithms have generally acceptable time and memory requirements to obtain a near-optimal or optimal solution. During the past few decades, most researches on the JSP have been concentrated on developing heuristic algorithms (Jain & Meeran, 1998; Blazewicz et al., 1996; Vaessens et al., 1996).

Balas and Vazacopoulos (1998) developed a guided local search algorithm rooted in a specialized neighborhood tree for the JSP, which is considered as one of the most efficient neighborhood structures and algorithms for this problem. Pezzella and Merelli (2000) presented a heuristic algorithm based on a combination of tabu search (TS) method and shifting bottleneck (SB) procedure to solve the JSP. The SB was used to generate a good initial feasible solution, and a local re-optimization, based on the same procedure, was used to improve each current solution determined by the TS. Huang and Liao (2008) presented a hybrid approach combining ant colony optimization (ACO) algorithm with a TS algorithm to solve the JSP. In this hybrid algorithm, ACO was used to provide an appropriate initial schedule, and TS was applied to improve the solution quality. The proposed ACO algorithm employed a new decomposition method inspired by the SB procedure, and a mechanism of occasional re-optimizations of partial schedules.

Zhang et al. (2008) presented a hybrid TS-SA algorithm in which simulated annealing (SA) was used to find the promising elite solutions inside big valley (BV) and TS intensified search around these solutions. This hybridization can reduce the influence of the initial solution in the TS algorithm. Rego and Duarte (2009) presented a heuristic algorithm based on a filter-and-fan (F&F) procedure for the JSP, which uses the SB procedure as a constructive method to generate a starting solution and to enhance the best schedules produced and a dynamic and adaptive neighborhood search procedure. The F&F approach is a local search procedure that generates compound moves by an abbreviated form of tree search. Fıglalı et al. (2009) conducted a statistical experiment on the JSP. They presented an ACO-based software system to solve the problem and the parameters of this system were investigated on various sizes and randomly generated job shop scheduling problems by using design of experiments. The effects and interactions of the parameters were interpreted with the outputs of the experiments.

Luh and Chueh (2009) presented a multi-modal immune algorithm to solve the JSP emulating the features of a biological immune system. Operation-based antibody/schedule representation was adopted to guarantee feasible schedules. The exploration and exploitation of solutions within a search space were realized through the procedures, which resemble antibody molecule structure, antibody–antigen relationships in terms of specificity, clonal proliferation, germinal center, and the memory characteristics of adaptive immune responses.

Lin et al. (2010) presented a hybrid algorithm called MPSO consisting of particle swarm optimization (PSO), a multiple-type individual enhancement (MIE) scheme based on SA technique, and random-key (RK) encoding scheme for solving the JSP. The MPSO adopts continuous space as the search space called RK space and uses the RK encoding scheme to transform a position in RK continuous space to a discrete space, since the search space in the JSP is a discrete space. In RK space, a position of a particle composed of $n \times m$ real numbers can represent the permutation of all operations of all jobs by the encoding scheme. Asadzadeh and Zamanifar (2010) proposed an agent-based parallel genetic algorithm for the JSP. The agent-based parallel approach was used to parallelize the genetic algorithm and to accelerate the creation of the initial population of genetic algorithm. Some other recent studies on the problem are: (Gao et al., 2011; Kammer et al., 2011; Lochtefeld & Ciarallo, 2011; Mati et al., 2011; Sels et al., 2011; Ponsich & Coello, 2013; Zhang et al., 2013).

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, which considers simultaneously many factors influencing the solution quality and intelligently balances their effects, in the process of schedule generation, and the observation that it could lead to good results in some preliminary computational experiments on a wide range of complicated scheduling problems. 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. Some notations that will be used in the algorithm are defined as follows:

sj$_i$: total processing time of job i (i.e.,, $sj_i = \sum_{j=1}^{m} t_{ij}$),

sk$_y$: total processing time on machine y which is calculated as follows: $sk_y = \sum_{i=1}^{n} t'_{iy}$,

M: a large number.

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

until all operations of all jobs are scheduled, repeat
{

- Find i, j (such that: 1. j=1 or (j-1)th operation of job i is already scheduled, and 2. jth operation of job i is unscheduled) that minimizes *TC*.
- Schedule jth operation of job i on the last position of current partial sequence on machine k (the machine capable of processing operation j of job i).

}

Fig. 1. General outline of the proposed heuristic algorithm

The pseudocode of the proposed 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,

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

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,…,6) are constants and x_r (r=1,2,…,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 are 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 processing time of machines.

Other notations used in the pseudocode of the proposed heuristic are as follows:

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

As it can be seen in the pseudocode of the heuristic, the algorithm first sorts the jobs in decreasing order of their sj_i and then uses this order for evaluating their operations. 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_i is selected for scheduling sooner than the other operation. This sorting may lead to better solutions. x_r^* (r=1,2,…,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,…,6), 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^* (see Table 2). 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.

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.

Constructive Algorithm:

```
for x1:=L_x1 to U_x1 do
for x2:=L_x2 to U_x2 do
⋮
for x6:=L_x6 to U_x6 do
{
        % Beginning of a schedule generation
        until  all operations of all jobs are scheduled, repeat the following steps:
        {
                Set TC*:=M
                for j:= 1 to m do
                {
                        for i':=1 to n do
                        {
                                Set i:= i_sort(n-i'+1),
                                if (  1. j=1 or (j-1)th operation of job i is already  scheduled, and
                                      2. jth operation of job i is unscheduled) then
                                {
                                        Set TC:= Σ_{r=1}^{6} w_r.x_r.C_r
                                        if TC<TC* then
                                        {
                                                Set TC*:= TC
                                                Set z:=i
                                                Set j':=j
                                        }
                                }
                        }
                }
                if TC*<M then schedule j'th operation of job z on the last position of the current
                partial sequence on machine k (i.e. the machine capable of processing j'th operation of
                job z) to finish at time c_zj'.
        }

        % End of a schedule generation
        If the objective value of the obtained sequence (Cmax) is less than the best objective value
        obtained so far (Cmax*), then set Cmax*:=Cmax and xr*=xr (r=1,2,…, 6) corresponding to Cmax*.
}
```

Fig. 2. Pseudocode of the proposed heuristic method

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.

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	2	L_x_2	0	U_x_2	3
w_3	1	L_x_3	−3	U_x_3	0
w_4	1	L_x_4	−1	U_x_4	0
w_5	1	L_x_5	−2	U_x_5	0
w_6	1	L_x_6	−1	U_x_6	0

Table 2
Computational results for benchmark instances

Name	Size		BKS	PaGA	RPD	Heuristic								
	n	m				Cmax	Time(s)	x1	x2	x3	x4	x5	x6	RPD
LA01	10	5	666	666	0	694	0.09	1	0	-2	0	-2	-1	4.204
LA02	10	5	655	655	0	697	0.09	3	0	-1	0	-2	-1	6.412
LA03	10	5	597	617	3.35	640	0.09	3	1	0	-1	-2	-1	7.203
LA04	10	5	590	607	2.881	605	0.09	3	0	-3	-1	-2	-1	2.542
LA05	10	5	593	593	0	593	0.00	2	0	-3	-1	-2	-1	0
LA06	15	5	926	926	0	926	0.00	2	0	-3	-1	-2	-1	0
LA07	15	5	890	890	0	897	0.20	1	2	0	-1	-1	-1	0.787
LA08	15	5	863	863	0	869	0.20	2	3	-1	0	-1	-1	0.695
LA09	15	5	951	951	0	951	0.02	2	0	-3	-1	-2	-1	0
LA10	15	5	958	958	0	958	0.00	1	0	-3	-1	-2	-1	0
LA11	20	5	1222	1222	0	1222	0.02	1	0	-3	-1	-2	-1	0
LA12	20	5	1039	1039	0	1039	0.02	3	0	-3	-1	-2	-1	0
LA13	20	5	1150	1150	0	1150	0.02	1	0	-3	-1	-2	-1	0
LA14	20	5	1292	1292	0	1292	0.00	1	0	-3	-1	-2	-1	0
LA15	20	5	1207	1207	0	1266	0.34	2	3	0	-1	-2	-1	4.888
LA16	10	10	945	994	5.185	1027	0.27	4	3	-2	0	0	-1	8.677
LA17	10	10	784	793	1.148	822	0.27	2	2	0	-1	0	-1	4.847
LA18	10	10	848	860	1.415	871	0.27	4	2	0	-1	-2	-1	2.712
LA19	10	10	842	873	3.682	883	0.27	3	2	0	-1	0	-1	4.869
LA20	10	10	902	912	1.109	953	0.27	4	3	-2	-1	-1	-1	5.654
LA21	15	10	1046	1146	9.56	1150	0.59	2	2	0	0	0	-1	9.943
LA22	15	10	927	1007	8.63	999	0.59	3	2	-1	0	-2	-1	7.767
LA23	15	10	1032	1033	0.097	1077	0.61	4	1	-2	-1	-2	0	4.36
LA24	15	10	935	1012	8.235	1023	0.61	4	2	-1	0	-1	-1	9.412
LA25	15	10	977	1067	9.212	1108	0.59	2	1	-1	-1	-1	-1	13.41
LA26	20	10	1218	1323	8.621	1312	1.03	4	1	-3	-1	-2	-1	7.718
LA27	20	10	1235	1359	10.04	1378	1.03	3	0	-1	-1	-2	-1	11.58
LA28	20	10	1216	1369	12.58	1350	1.03	3	2	-3	-1	-1	-1	11.02
LA29	20	10	1152	1322	14.76	1348	1.02	2	3	-1	0	-1	-1	17.01
LA30	20	10	1355	1437	6.052	1473	1.05	4	2	-2	-1	-2	-1	8.708
LA31	30	10	1784	1844	3.363	1844	2.28	1	0	-2	0	0	0	3.363
LA32	30	10	1850	1907	3.081	1881	2.30	1	2	-3	-1	-1	-1	1.676
FT06	6	6	55	55	0	55	0.02	2	0	-3	-1	-1	-1	0
FT10	10	10	930	997	7.204	1028	0.27	3	3	0	0	0	-1	10.54
FT20	20	5	1165	1196	2.661	1244	0.34	4	2	0	0	-2	-1	6.781
ORB01	10	10	1059	1149	8.499	1160	0.27	1	0	0	-1	0	0	9.537
ORB02	10	10	888	929	4.617	929	0.27	2	2	0	-1	-1	-1	4.617
ORB03	10	10	1005	1129	12.34	1106	0.28	1	0	0	-1	-1	0	10.05
ORB04	10	10	1005	1062	5.672	1062	0.28	3	3	-1	-1	0	-1	5.672
ORB05	10	10	887	936	5.524	977	0.27	3	2	-1	0	-1	-1	10.15
ORB06	10	10	1010	1060	4.95	1102	0.27	3	2	0	-1	-2	0	9.109
ORB07	10	10	397	416	4.786	442	0.27	2	1	-1	0	-1	-1	11.34
ORB08	10	10	899	1010	12.35	991	0.28	3	0	0	-1	-2	-1	10.23
ORB09	10	10	934	994	6.424	1051	0.28	2	1	-3	0	-2	-1	12.53
Average					4.273		0.42	2.432	1.25	-1.45	-0.68	-1.34	-0.89	5.909

The algorithm is coded in C language and run on a Pentium IV, 2.2 GHz and 2.0 GB RAM PC. Three sets of benchmark problem instances are considered: 32 instances denoted as (LA01−LA32) presented by Lawrence (1984), 3 instances (FT6, FT10, FT20) of Fisher and Thompson (1963), and 9 instances (ORB01−ORB9) of Applegate and Cook (1991). All the instances can be downloaded from OR-Library web site (http://people.brunel.ac.uk/~mastjjb/jeb/info.html). Table 2 shows a comparison of the makespan results of our algorithm with those of the recently published algorithm: the parallel agent-based genetic algorithm (PaGA) proposed by Asadzadeh and Zamanifar (2010). The first four columns provide the problem name, its size in terms of the number of jobs (n) and machines (m), and the best known solution (BKS) for each instance. The results obtained by proposed algorithm are shown in the last nine columns. *Cmax* and *Time(s)* stand for the makespan and the computational time (in seconds), respectively. The best values of variables x_r (i.e. x_r^*), r=1,2,…,6; are also reported in Table 2. Average values of x_3, x_4, x_5 and x_6 are negative, that means they have adverse effect on C_{max}. *RPD* is the relative percentage deviation to *BKS* and calculated as follows:

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

where $Cmax_{alg}$ is the makespan obtained by the algorithm. PaGA and our algorithm have average RPD

values of 4.27 and 5.91, respectively. Herein, the heuristic is statistically compared with PaGA. A one-way analysis of variance (ANOVA) (Montgomery, 2000) is performed to test the null hypothesis that the means of the two methods are equal. The results of this ANOVA are presented in Table 3. As can be seen, the difference between the methods is not meaningful at a significance level of 5%. However, the heuristic is very fast, and needs less than 0.5 sec. on average for all instances.

Table 3
Results of one-way ANOVA for the methods: PaGA and proposed heuristic

Source	DF	SS	MS	F	P
Factor	1	58.9	58.9	3.01	0.086
Error	86	1683.9	19.6		
Total	87	1742.8			

5. Conclusion

This paper has investigated the job shop scheduling problem with the objective of minimizing the makespan. The main purpose was to produce reasonable schedules very quickly. A simple and easily extendable heuristic based on a constructive procedure has been presented. The algorithm has beed examined on benchmark instances from the literature in order to evaluate its performance. The computational results have shown that even the relatively straightforward implementation of the approach as presented here, could 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. However, the computational times show the interest of the heuristic, since in a fraction of a second on average, it produces very good solutions. Although the solutions produced by this simple heuristic are weakly dominated the solutions of the metaheuristic methods evaluated, 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 to adapt the approach to the flexible job shop scheduling problem.

References

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2) 149–156.

Asadzadeh, L., & Zamanifar, K. (2010). An agent-based parallel approach for the job shop scheduling problem with genetic algorithms. *Mathematical and Computer Modelling,* 52, 1957–1965.

Baker, K. (1974). *Introduction to sequencing and scheduling*. NewYork: Wiley.

Balas, E., & Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2), 262–275.

Blazewicz, J., Domschke, W., & Pesch, E. (1996), The job shop scheduling problem: conventional and new solution techniques. *European Journal of Operational Research,* 93, 1–33.

Carlier, J., & Pinson, E. (1989). An algorithm for solving the job-shop problem, *Management Science*, 35, 164–176.

Dell'Amico, M., & Trubian, M. (1993). Applying tabu-search to the job-shop scheduling problem. *Annals of Operations Research,* 4, 231–252.

Fıglalı, N., Ozkale, C., Engin, O., & Fıglalı, A. (2009). Investigation of ant system parameter interactions by using design of experiments for job-shop scheduling problems. *Computers & Industrial Engineering*, 56, 538–559.

Fisher, H., & Thompson, G.L. (1963). *Probabilistic learning combinations of local job shop scheduling rules*. J.F. Muth, G.L. Thompson (Editors), Industrial Scheduling, Prentice-Hall, Englewood Cliffs, New Jersey, pp. 225–251.

Gao, L., Zhang, G., Zhang, L., & Li, X. (2011). An efficient memetic algorithm for solving the job shop scheduling problem. *Computers & Industrial Engineering*, 60(4), 699–705.

Garey, M.R., Johnson, D.S., & Sethi, R. (1976). The complexity of flow shop and job-shop scheduling. *Mathematics of Operations Research*, 1(2), 117–129.

Ho, N.B., Tay, J.C., & Lai, E.M.-K. (2007). An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179, 316–333.

Huang, K.-L., & Liao, C.-J. (2008). Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & Operations Research*, 35, 1030–1046.

Jain, A.S., & Meeran, S. (1998). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2), 390–434.

Kammer, M., Akker, M., & Hoogeveen, H. (2011). Identifying and exploiting commonalities for the job-shop scheduling problem. *Computers & Operations Research*, 38(11), 1556–1561.

Kolonko, M. (1999). Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, 113, 123–136.

Lawrence, S. (1984). *Supplement to Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques*. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Lenstra, J.K. (1976). *Sequencing by enumerative methods*. Tech. Rep. Mathematical Centre Tract 69, Mathematisch Centrum, Amsterdam.

Lin, T.-L., Horng, S.-J., Kao, T.-W., Chen, Y.-H., Run, R.-S., Chen, R.-J., Lai, J.-L., & Kuo, I-H. (2010). An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications*, 37, 2629–2636.

Lochtefeld, D.F., & Ciarallo, F.W. (2011). Helper-objective optimization strategies for the Job-Shop Scheduling Problem. *Applied Soft Computing*, 11(6), 4161–4174.

Luh, G.-C., & Chueh, C.-H. (2009). A multi-modal immune algorithm for the job-shop scheduling problem. *Information Sciences*, 179, 1516–1532.

Mati, Y., Dauzère-Pérès, S., & Lahlou, C. (2011), A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research,* 212(1), 33–42.

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.

Montgomery, D.C. (2000). *Design and analysis of experiments*. 5th ed., NewYork: John Wiley & Sons.

Nowicki, E., & Smutnicki, C. (1996). A fast tabu search algorithm for the job shop problem. *Management Science*, 42(6), 797–813.

Pezzella, F., & Merelli, E. (2000). A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research*, 120, 297–310.

Pinedo, M. (2002). *Scheduling: theory, algorithms and systems*. Englewood cliffs, NJ: Prentice-Hall.

Ponsich, A., & Coello, C.A.C. (2013). A hybrid Differential Evolution—Tabu Search algorithm for the solution of Job-Shop Scheduling Problems. *Applied Soft Computing*, 13, 462–474.

Rego, C., & Duarte, R. (2009). A filter-and-fan approach to the job shop scheduling problem. *European Journal of Operational Research*, 194, 650–662.

Sels, V., Craeymeersch, K., & Vanhoucke, M. (2011), A hybrid single and dual population search procedure for the job shop scheduling problem. *European Journal of Operational Research*, 215 (3) 512–523.

Vaessens, R.J.M., Aarts, E.H.L., & Lenstra, J.K. (1996), Job shop scheduling by local search. *INFORMS Journal on Computing*, 8(3) 302–317.

Van Laarhoven, P., Aarts E., & Lenstra J. (1992). Job shop scheduling by simulated annealing. *Operations Research*, 40, 113–125.

Yamada, T., & Nakano, R. (1996), A fusion of crossover and local search, in: *Proceedings of the IEEE International Conference on Industrial Technology ICIT96*, Shangai, China, IEEE Press 426–430.

Zhang, C.Y., Li, P.G., Rao, Y.Q., & Guan Z.L. (2008), A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, 35, 282–294.

Zhang, R., Song, S., & Wu, C. (2013). A hybrid artificial bee colony algorithm for the job shop scheduling problem. *International Journal of Production Economics*, 141, 167–178.