A Hybrid Imperialist Competitive Algorithm for the Flexible Job Shop Problem

Behrooz Ghasemishabankareh[1(✉)], Nasser Shahsavari-Pour[2], Mohammad-Ali Basiri[3], and Xiaodong Li[1]

[1] School of Computer Science and IT, RMIT University, Melbourne, Australia
{behrooz.ghasemishabankareh,xiaodong.li}@rmit.edu.au
[2] Department of Industrial Management, Vali-e-Asr University, Rafsanjan, Iran
shahsavari_n@alum.sharif.edu
[3] Department of Industrial Engineering, Science and Research Branch, Islamic Azad University, Kerman, Iran
mohammadali.basiri@yahoo.com

Abstract. Flexible job shop scheduling problem (FJSP) is one of the hardest combinatorial optimization problems known to be NP-hard. This paper proposes a novel hybrid imperialist competitive algorithm with simulated annealing (HICASA) for solving the FJSP. HICASA explores the search space by using imperial competitive algorithm (ICA) and use a simulated annealing (SA) algorithm for exploitation in the search space. In order to obtain reliable results from HICASA algorithm, a robust parameter design is applied. HICASA is compared with the widely-used genetic algorithm (GA) and the relatively new imperialist competitive algorithm (ICA). Experimental results suggest that HICASA algorithm is superior to GA and ICA on the FJSP.

Keywords: Flexible job shop scheduling problem · Imperialist competitive algorithm · Genetic algorithm · Simulated annealing algorithm · Taguchi parameter design

1 Introduction

The classical job shop problem (CJSP) deals with scheduling n jobs on m machines, which is known as a NP-hard Problem [1]. Each job involves a set of operations with their pre-specified sequences as well as processing times. However, in today's competitive businesses, companies often need to apply more flexible and efficient production systems in order to satisfy their requirements. More specifically, not only automation and flexible machines need to be used, but also a flexible scheduling should be designed as well.

Flexible job shop scheduling problem (FJSP) extends CJSP which does not restrict the operations to be processed on pre-specified machines [2, 3]. Flexibility allows the problem to be modeled in a more realistic manner, however, exact methods are unable to solve the problem efficiently. The FJSP scheduling encompasses two sub-problems: assigning an operation to a machine through existing machines and specifying the sequence of the jobs' operations. Brucker and Schlie [4] studied the FJSP for the first time.

T. Ray et al. (Eds.): ACALCI 2016, LNAI 9592, pp. 221–233, 2016.
DOI: 10.1007/978-3-319-28270-1_19

They introduced a polynomial algorithm for the problem with two jobs. Although in some cases the exact methods can in theory find the optimal solution for the problems, computational time is so long that it is not practical to use them. Researchers have been trying to find ways in which optimal or near optimal solutions can be obtained in reasonable computational time. In recent years, some heuristic and meta-heuristic methods have shown to be promising in achieving this goal, including tabu search (TS), simulated annealing (SA), ant colony optimization (ACO), genetic algorithm (GA) [5–8].

A new evolutionary algorithm named imperialist competitive algorithm (ICA), has been proposed recently by Atashpaz and Lucas [9]. This meta-heuristic algorithm has shown promising results on several engineering problems and industrial engineering field [10–15]. Combining two or more meta-heuristic methods seem to help achieve good efficiency that is not possible by applying each one alone. Here, TS, SA and variable neighborhood search (VNS) play an important role. Tavakkoli-Moghaddam et al. [16] and Naderi et al. [17] presented a hybridization of electromagnetic-like mechanism and SA. Some other hybrid meta-heuristics for solving the abovementioned problem are also available [18–22]. Furthermore, Shahsavari-pour and Ghasemishabankareh [23] presented a novel hybrid GA and SA algorithm to solve the FJSP, where for the first time, an efficient hybrid ICA and SA has been applied for solving the FJSP.

As mentioned earlier, since the FJSP is well-known to be NP-hard, meta-heuristic algorithms have significant advantages to solve the problem over exact methods. Hybridization of meta-heuristic methods has attracted much attention of many researchers. This paper proposes a new hybridized algorithm named as hybrid imperialist competitive algorithm with Simulated Annealing (HICASA), where SA is applied as a local search algorithm, while ICA does global search in the solution space. In this study the FJSP is considered as a single-objective problem and the proposed algorithm is applied to minimize the makespan. The robust parameter setting procedure is applied to set all parameters for HICASA, GA and ICA. By solving the same benchmarks, our results show that HICASA is superior to GA and ICA.

The remaining sections of the paper are organized as follow: Sect. 2 gives problem representation. Section 3 describes solution methodologies for solving the FJSP. The experimental design and computational results are provided in Sect. 4. Finally the conclusions are presented in Sect. 5.

2 Problem Representation

The FJSP includes n jobs which are scheduled on m machines. The jobs are represented by the set $J = \{1, 2, \ldots, n\}$ and the set $M = \{1, 2, \ldots, m\}$ indicates the machines. The purpose of the optimization task is to generate a feasible schedule consistent with minimization of the objective function and satisfying problem constraints at the same time. In this FJSP problem, all machines are assumed to be available at time zero, all jobs can be processed at time zero, each machine can have only one operation at a time, each job can be processed by only one machine at a time and transportation times are not considered. Notations and variables of the FJSP are presented as follows:

J	Indices of jobs, $j = 1, 2, ...,n$
i, p	Indices of machines, $i, p = 1, 2, ..., m$
e	Indices of jobs which operate exactly before job j on the same machine, $e = 1, 2, ..., n$
K	The set of numbers of each job's operations. For example $K(j) = L$ means that the *jth* job has L operations.
l, q	Indices of numbers of operations, $l,q = 1, 2, ..., K(j)$
C_{lji}	Completion time of the *lth* operation of job j on machine i
P_{lji}	Processing time of the *lth* operation of job j on machine i

The mathematical model of the FJSP is given as follows:

$$Z = Min\{\max\{C_{K(j)ji}\}\}; K(j) \in K; j \in J; i \in M \tag{1}$$

s.t:

$$C_{lji} - P_{lji} \geq C_{l-1jp} \quad l = 1, 2, \ldots, K(j) \quad j = 1, 2, \ldots, n \quad i, p = 1, 2, .., m \tag{2}$$

$$C_{lji} - P_{lji} \geq C_{qei} \quad q, l = 1, 2, \ldots, K(j) \quad e, j = 1, 2, \ldots, n \quad i = 1, 2, \ldots, m \tag{3}$$

$$C_{lji} \geq 0 \quad l = 1, 2, \ldots, K(j) \quad j = 1, 2, \ldots, n \quad i = 1, 2, \ldots, m, \tag{4}$$

where Eq. (1) implies the objective function (makespan), which should be minimized. As noted above, the problem contains two basic restrictions: the first one is precedence constraint belonging to the operations of a job. It means that the operation l of job j, cannot be started until the whole previous operations (operation 1 to $l - 1$) to be completed (Eq. (2)). The second restriction is non-overlapping constraint of the operations on a machine which is specified by Eq. (3). It means that the machine does not start to process the next operation until the current operation is finished completely.

3 Solution Procedure

3.1 Proposed *HICASA* Algorithm

The imperialist competitive algorithm is one of the efficient evolutionary algorithms in solving discrete optimization problems [9]. In this algorithm, there are some countries (or colonies) which are divided into two categories, imperialists and colonies. The imperialist with its colonies, is called an empire. Competition among imperialists continues and the most powerful imperialist has a higher chance to take the weakest colony of the weakest imperialist. This process continues until just one imperialist remains. Finally all the imperialists and colonies become the same.

HICASA is a novel meta-heuristic algorithm that integrates ICA with the SA algorithm. SA has functions as a neighborhood search process and improves the convergence of the solutions. During this process, some colonies are chosen randomly from each empire and the SA algorithm is used as neighborhood search to alter chosen colonies. The altered colonies replace the previous ones in each empire and the algorithm goes on. The structure of HICASA is as follows (all equations of ICA are captured from Atashpaz and Lucas [9]):

Establishing Initial Empires. In ICA the initial population, which is generated randomly, as colonies, is located in the form of array. Each individual in the population is equivalent to the chromosomes in GA. Each array in the FJSP consists of two strings [24]. The first ordered string represents the number of machines and the second string represents the order of the job operations. Figure 1 shows an example of an array for our experiments in Sect. 4. Note that the fifth element in the first string is 2 and the counterpart element in the second one is 3. It means the second operation of job 3 should be performed on machine 2 (O_{ij}: operation j from job i). The number of initial population is considered as NC (number of countries) in this paper.

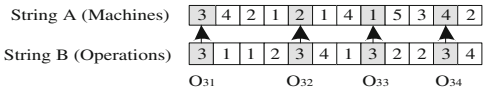

Fig. 1. Array structure of a country (colony).

Calculating Objective Function and Generating Colonies. In order to evaluate the colonies' power, a cost function should be calculated. In the FJSP, the cost function is equivalent to the value of makespan (C_{max}). For each colony the value of C_{max} is equal to:

$$C_{\max} = \max\{C_{K(j)ji}\}; \quad K \in K(j); j = 1, 2, \ldots, n; \quad i = 1, 2, \ldots, m \tag{5}$$

Then N_{imp} (number of empires) of the best members are chosen as imperialists. N_{col} is the number of remaining countries which should be distributed between the imperialists. The number of colonies for each imperialist depends on the imperialists' power. In order to calculate their power, first the normalized cost should be computed according to Eq. (6) (i.e., normalized makespan):

$$C_O = c_O - \max_{i=1,2,\ldots,N_{imp}} \{c_i\}; \forall O = 1, 2, \ldots, N_{imp}, \tag{6}$$

where c_O is the makespan of the Oth imperialist and C_O is its normalized value. Now the relative power of each imperialist (P_O) is calculated through the following:

$$p_O = \left| \frac{C_O}{\sum_{i=1}^{N_{imp}} C_i} \right|; \forall O = 1, 2, \ldots, N_{imp} \tag{7}$$

Obviously, each imperialist's power is the portion of colonies that should be possessed by that imperialist. Hence the number of colonies of each imperialist is computed by Eq. (8):

$$N.C._O = round\{p_O \times N_{col}\}; \forall O = 1, 2, \ldots, N_{imp}, \tag{8}$$

where $N.C._O$ is the initial number of *Oth* imperialist's colonies. *round* is a function which rounds a decimal number into the nearest integers. So each imperialist in an empire, has $N.C._O$ colonies, which are chosen from the remained initial countries randomly.

Assimilating. The empires attempt to increase their power by improving their colonies. In other words, they propel their colonies to become similar to their imperialist through making constructive changes in their structures. This changing process is similar to the crossover process in GA. The assimilating process in the FJSP is shown as a designed algorithm in Fig. 2.

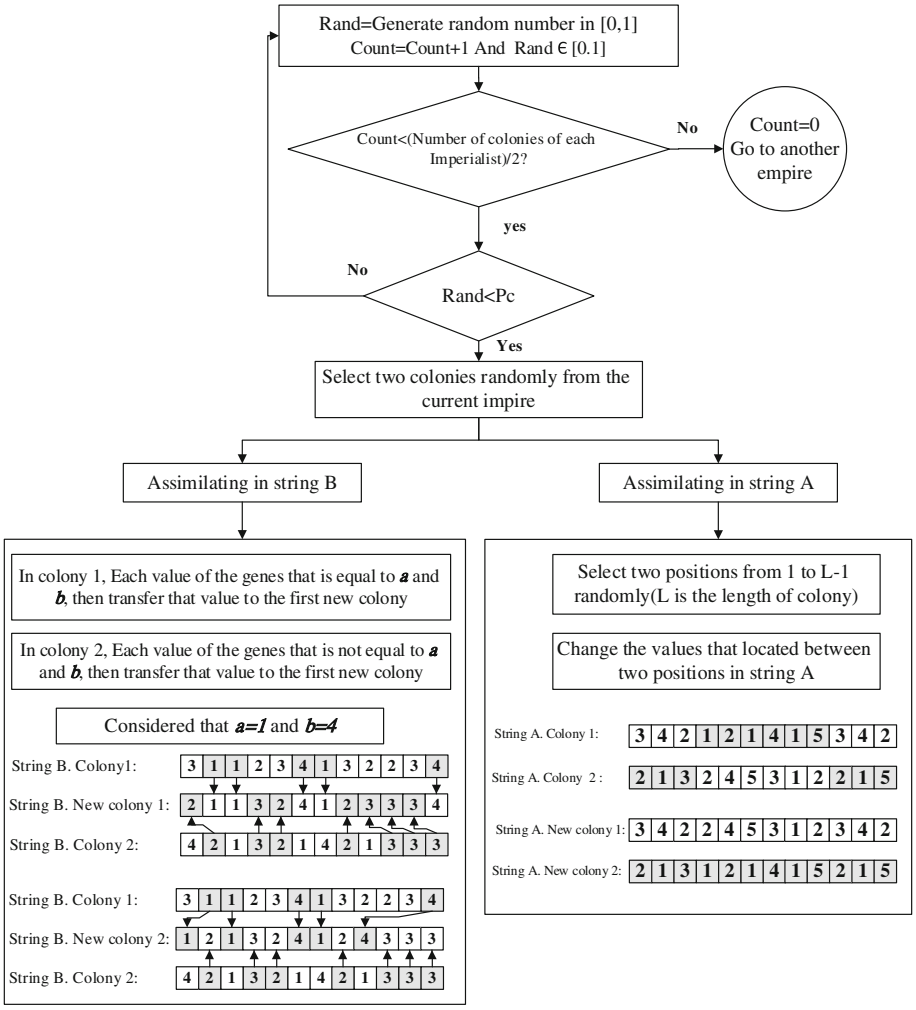

Fig. 2. Assimilating process.

Now the cost function (makespan) for the colonies resulting from assimilating process is calculated by Eq. (1).

Neighborhood Search Through SA Algorithm. As mentioned before, neighborhood search is suitable for further fine-tuning solutions produced by global optimization methods. In this paper, SA is integrated with ICA to deal with the FJSP problem. The algorithm works as follows: *NSA* colonies are chosen randomly from each empire and SA is carried out for each chosen colony. At first the neighborhood search algorithm works with the second string of each array, if it does not get improved after *NI* iterations, it goes to the first string (machines string) of that array. The ultimate array replaces the current solution. The function of SA algorithm in this neighborhood search is shown in Fig. 3.

```
for i=1 to N_imp
        NS0=0
        while NS0<NSA do
                XB=Select one colony from current empire randomly
                F_XB = C_max(XB) ;(Eq. 5)
                for R0= 1 to R do
                        for S0=1 to S do
                                X_N=Modify selected colony
                                F_X_N = C_max(X_N) ;(Eq. 5)
                                if F_XB ≥ F_X_N then
                                        XB=X_N
                                        F_XB = F_X_N
                                        NC0=0
                                else
                                        Y=Generate random number in [0,1]
                                        if Y<exp(-(F_XN-F_XB))/T then
                                                XB=X_N
                                                F_XB = F_X_N
                                                NC0=NC0+1
                                        else
                                                NC0=NC0+1
                                        end if
                                end if
                                S0=S0+1
                        end for
                        T=T*α
                        R0=R0+1
                end for
                NS0=NS0+1
        end while
end for
```

Fig. 3. Pseudo code of neighborhood search using SA.

The Replacement of Colony and Imperialist. During the movement of colonies towards the imperialist and the neighborhood search, some colonies have better objective values than their imperialist. In this case, in each empire, the imperialist is replaced by the colony that has the smallest cost.

The total power of an imperialist is the summation of the power related to the imperialist and a percentage of its all colonies' power and calculated by the following:

$$T.C._n = C_n(imperialist_n) + \xi \cdot mean\{C_n(\ colonies \quad of \quad impire_n\ )\}, \tag{9}$$

where $T.C._n$ is the total cost of the *nth* empire and ζ is a positive value, which shows the level of influence of colonies' power in calculating the total power of the empire.

During an algorithm run, each empire that cannot increase its power loses its competition power gradually. There is always competition among empires and the stronger empires have the higher chance to seize the colony, and for this purpose the chance of possession is defined by possession probability. Total normalized cost of each empire ($N.T.C._o$) is calculated by Eq. (10) and the possession probability of each empire is calculated by Eq. (11):

$$N.T.C_O = T.C._O - \max_{i=1,2,\ldots,N_{imp}} \{T.C._i\}; \forall O = 1, 2, \ldots, N_{imp} \tag{10}$$

$$p_{p_O} = \left| \frac{N.T.C._O}{\sum_{i=1}^{N_{imp}} N.T.C._i} \right|; \forall O = 1, 2, \ldots, N_{imp} \tag{11}$$

The colonies are divided among empires randomly and with regards to the probability of acquiring each empire. To do this, first P vector is formed as follows: $P = [p_{p_1}, p_{p_2}, p_{p_3}, \ldots, p_{pN_{imp}}]$. Then vector R should be generated randomly in the closed interval [0,1] with the same size as P ($R = [r_1, r_2, r_3, \ldots, r_{N_{imp}}]$). Finally we calculate vector D ($D = P - R = [D_1, D_2, D_3, \ldots, D_{N_{imp}}] = [p_{p_1} - r_1, p_{p_2} - r_2, p_{p_3} - r_3, \ldots, p_{pN_{imp}} - r_{N_{imp}}]$) and a colony belongs to the empire which has the maximum index in D vector.

In each iteration, the algorithm eliminates the empire which has no colonies. In the algorithm's each iteration, all the empires collapse gradually except for the strongest one. The algorithm stops when just one empire is remained. Notations of parameters for HICASA are as follows: NC is the number of countries, N_{imp} is the number of empires, Pc is assimilation rate, ζ is constant value, S and R are the number of internal and external loop in SA respectively, NSA is the number of local search performed by SA algorithm and α is decreasing rate for temperature.

3.2 Genetic Algorithm

GA is one of the most widely-used population-based stochastic search algorithms proposed by Holland [25]. GA begins with an initial population and improves the solutions based on the evolutionary process. In this regard, GA utilizes two important operators to modify solutions and produce offspring. A selection procedure is used to generate offspring in the next generation. In the selection procedure better solutions have higher probability to be chosen. This process continues until the termination condition is satisfied. The crossover and mutation are captured from [23, 24].

4 Design of Experiments

In this paper, to evaluate the proposed algorithm, the FJSP has been considered. The problem includes 4 jobs and 5 machines. Data containing processing times have been extracted from [26] and shown here in Table 1. The objective function of the above-mentioned FJSP problem has been treated as a single-objective through minimizing the makespan. The results of HICASA, ICA and GA have been compared.

Table 1. Processing time of 4 × 5 problem.

Job	Operation	Processing time for machine MI				
		M1	*M2*	*M3*	*M4*	*M5*
J1	1	2	5	4	1	2
	2	5	4	5	7	5
	3	4	5	5	4	5
J2	1	2	5	4	7	8
	2	5	6	9	8	5
	3	4	5	4	54	5
J3	1	9	8	6	7	9
	2	6	1	2	5	4
	3	2	5	4	2	4
J4	4	4	5	2	1	5
	1	1	5	2	4	12
	2	5	1	2	1	2

4.1 Taguchi Parameter Design

Since the three algorithms are population-based and their parameters' values affect the final solution qualities significantly. There are different methods to calibrate the parameters of algorithms [14]. In this paper Taguchi method is used. Taguchi method has been utilized for optimization [27, 28] including evolutionary algorithms [29, 30]. Taguchi method has three phases: system design, parameter design and tolerance design. In this paper, Taguchi method is used as a robust parameter design. In this approach parameters' design is used to define factors which provide the best performance of processes/products.

In Taguchi method, instead of doing full factorial trails, an orthogonal array is used to carry out fewer experiments which examine the effect of noise. The orthogonal array suggests a definite number of combinations of factor levels which have the same results as full factorial trails. A robust parameter design tries to minimize the effect of noise factor through achieving a higher ratio of signal-to-noise (S/N). In other words, a higher value of S/N causes less effect of uncontrollable and noise factors in the performance of the algorithm. The value of the S/N is calculated as [28]:

$$S/N = -10 \times \log_{10}(objectivefunction)^2 \quad (12)$$

In this study, we select crucial factors of the algorithms (GA, ICA) according to the previous researches. Three factors of HICASA are the same as ICA but we add NSA parameter to show the effect of neighborhood search in the proposed algorithm. Interested readers can refer to [14, 30, 31]. By using the Taguchi method the best combination of the factors and their levels can be obtained for each algorithm. This process is used to compare the performance of the algorithms. The factors and their levels for the algorithms are shown in Table 2. Notations for GA algorithm are as follows: *GN* is the number of generation, *Pop_size* is the number of individuals, *Pc* and *Pm* are the probabilities of crossover and mutation respectively.

Table 2. Factors and their level in GA, HICASA and ICA.

Factors in GA				
	A(GN)	***B(Pop_size)***	***C(Pc)***	***D(Pm)***
Levels	A1:100	B1:50	C1:0.9	D1:0.1
	A2:150	B2:100	C2:0.95	D2:0.15
	A3:200	B3:200	C3:.98	D3:0.2
Factors in *HICASA*				
	A(NSA)	***B(NC)***	***C(Pc)***	***D(ζ)***
Levels	A1:4	B1:40	C1:0.9	D1:1.4
	A2:5	B2:50	C2:0.94	D2:1.5
	A3:8	B3:35	C3:0.91	D3:1.6
Factors ICA				
	A(NC)	***B(Pc)***	***C(ζ)***	
Levels	A1:40	B1:0.9	C1:1.4	
	A2:50	B2:0.94	C2:1.5	
	A3:35	B3:0.91	C3:1.6	

As shown in Table 2 for GA there are four 3-level factors, for ICA three 3-level factors and for HICASA four 3-level factors. In order to facilitate and decrease the number of the experiments, the orthogonal array is used. Appropriate orthogonal arrays assigned for GA and HICASA is L9 and for ICA is L9 [14]. Table 3 shows the orthogonal arrays.

Table 3. Orthogonal array L9 for GA and HICASA.

	L9 for GA and HICASA				L9 for ICA		
Trail	A	B	C	D	A	B	C
1	1	1	1	1	1	1	1
2	1	2	2	2	1	2	2
3	1	3	3	3	1	3	3

(*Continued*)

Table 3. (*Continued*)

	L9 for GA and HICASA				L9 for ICA		
Trail	A	B	C	D	A	B	C
4	2	1	2	3	2	1	2
5	2	2	3	1	2	2	3
6	2	3	1	2	2	3	1
7	3	1	3	2	3	1	3
8	3	2	1	3	3	2	1
9	3	3	2	1	3	3	2

The mean value of S/N is calculated and shown in Figs. 4, 5 and 6 for all algorithms.

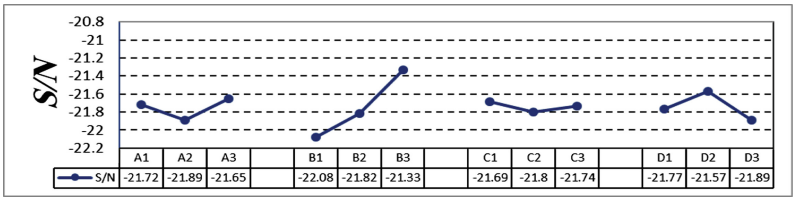

Fig. 4. Mean S/N ratio for each level of factors in GA.

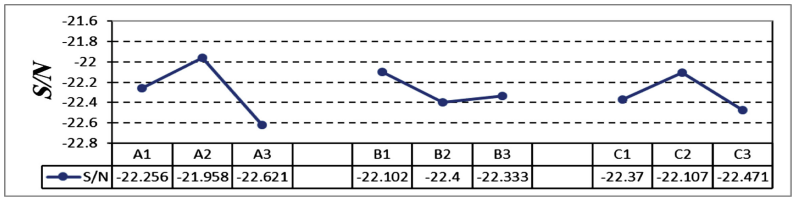

Fig. 5. Mean S/N ratio for each level of factors in ICA.

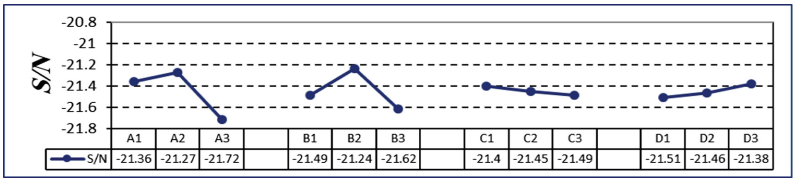

Fig. 6. Mean S/N ratio for each level of factors in HICASA.

According to the Figs. 4, 5 and 6 the optimal levels of the factors are the set {A3, B3, C1 and D2}, the set {A2, B1 and C2} and the set {A2, B2, C1 and D3} for GA, ICA and HICASA respectively.

4.2 Experimental Results

The algorithms are implemented in Visual Basic Application (VBA) and run on PC 2 GHz with 512 MB RAM. According to the parameters set in the previous section, the problem presented in Table 1 is solved by the proposed HICASA, ICA and GA. Each algorithm has been run 50 times and the averaged value was recorded as the final results. As shown in Fig. 7, the objective function values (makespan) for HICASA in all 50 runs converged to the optimal value of 11 but for GA and ICA the objective function values (makespan) did not converge to the optimal makespan in most of the runs. Clearly, the proposed algorithm (HICASA) obtains better solution in solving the same benchmark. Figure 8 illustrates the solution obtained by HICASA for the problem presented in Table 1. The numbers in the Gantt chart (Fig. 8) illustrate jobs' number.

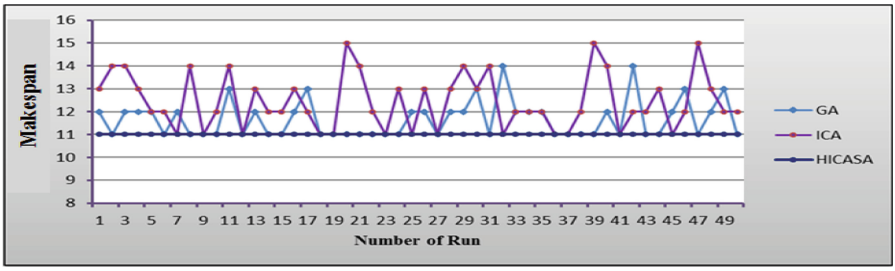

Fig. 7. Results of GA, ICA and HICASA.

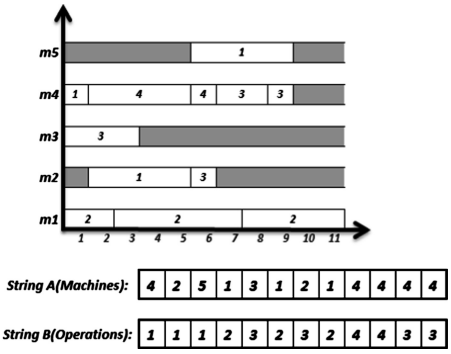

Fig. 8. Gantt chart of the obtained solution by HICASA for 4 × 5 Problem.

5 Conclusions

In this paper a novel hybrid meta-heuristic method (HICASA) has been developed to solve the FJSP. The proposed algorithm is a hybridization of ICA and SA algorithm which attempts to minimize makespan as the objective function. HICASA algorithm is compared with ICA and GA. Results from calculating and comparing between these three algorithms demonstrate that HICASA algorithm performs better than GA and ICA. By using the neighborhood search in the procedure of HICASA algorithm, it is able to solve the FJSP optimization problems effectively. HICASA can be used for solving different scheduling problems. It is also possible to integrating this algorithm with other meta-heuristic algorithms.

References

1. Gary, M.R., Johnson, D.S., Sethi, R.: The complexity of flow shop and job shop scheduling. Math. Oper. Res. **1**(2), 117–129 (1976)
2. Rossi, A., Dini, G.: Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimization method. Robot. Comput. Integr. Manuf. **23**(5), 503–516 (2007)
3. Brandimarte, P.: Routing and scheduling in a flexible job shop by tabu search. Ann. Oper. Res. **41**(3), 157–183 (1993)
4. Brucker, P., Schlie, R.: Job-shop scheduling with multi-purpose machines. Computing **45**(4), 369–375 (1990)
5. Thamilselvan, R., Balasubramanie, P.: Integrating genetic algorithm, tabu search approach for job shop scheduling. Int. J. Comput. Sci. Inf. Secur. **2**(1), 1–6 (2009)
6. Najid, N.M., Dauzere-Peres, S., Zaidat, A.: A modified simulated annealing method for flexible job shop scheduling problem. In: 2002 IEEE International Conference on Systems, Man and Cybernetics, vol. 5 (2002)
7. Colorni, A., Dorigo, M., Maniezzo, V., Trubian, M.: Ant system for job-shop scheduling. Belg. J. Oper. Res. Statist. Comput. Sci. **34**, 39–54 (1994)
8. Chen, H., Ihlow, J., Lehmann, C.: A genetic algorithm for flexible job-shop scheduling. In: Proceeding of IEEE International Conference on Robotics, pp. 1120–1125 (1999)
9. Atashpaz-Garagari, E., Lucas, C.: Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: IEEE Congress on Evolutionary Computation, pp. 4661–4667 (2007)
10. Khabbazi, A., Atashpaz-Gargari, E., Lucas, C.: Imperialist competitive algorithm for minimum bit error rate beam forming. Int. J. Bio-Inspired Comput. **1**(1–2), 125–133 (2009)
11. Nazari-Shirkouhi, S., Eivazy, H., Ghods, R., Rezaie, K., Atashpaz-Gargari, E.: Solving the integrated product mix-outsourcing problem using the imperialist competitive algorithm. Expert Syst. Appl. **37**(12), 7615–7626 (2010)
12. Lucas, C., Nasiri-Gheidari, Z., Tootoonchian, F.: Application of an imperialist competitive algorithm to the design of a linear induction motor. Energy Convers. Manag. **51**(7), 1407–1411 (2010)
13. Kaveh, A., Talatahari, S.: Optimum design of skeletal structures using imperialist competitive algorithm. Comput. Struct. **88**(21–22), 1220–1229 (2010)

14. Shokrollahpour, E., Zandieh, M., Dorri, B.: A novel imperialist competitive algorithm for bi-criteria scheduling of the assembly flow shop problem. Int. J. Prod. Res. **49**(11), 3087–3103 (2011)
15. Attar, S.F., Mohammadi, M., Tavakkoli-moghaddam, R.: A novel imperialist competitive algorithm to solve flexible flow shop scheduling problem in order to minimize maximum completion time. Int. J. Comput. Appl. **28**(10), 27–32 (2011)
16. Tavakkoli-Moghaddam, R., Khalili, M., Naderi, B.: A hybridization of simulated annealing and electromagnetic-like mechanism for job shop problems with machine availability and sequence-dependent setup times to minimize total weighted tardiness. Soft. Comput. **13**(10), 995–1006 (2009)
17. Naderi, B., Tavakkoli-Moghaddam, R., Khalili, M.: Electromagnetism-like mechanism and simulated annealing algorithms for flow shop scheduling problems minimizing the total weighted tardiness and makespan. Knowl. Based Syst. **23**(2), 77–85 (2010)
18. Soke, A., Bingul, Z.: Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems. Eng. Appl. Artif. Intel. **19**(5), 557–567 (2006)
19. Li, W.D., Ong, S.K., Nee, A.Y.C.: Hybrid genetic algorithm and simulated annealing approach for the optimization of process plans for prismatic parts. Int. J. Prod. Res. **40**(8), 1899–1922 (2002)
20. Osman, I.H., Christofides, N.: Capacitated clustering problems by hybrid simulated annealing and tabu search. Int. Trans. Oper. Res. **1**(3), 317–336 (1994)
21. Swarnkar, R., Tiwari, M.K.: Modeling machine loading problem of FMSs and its solution methodology using a hybrid tabu search and simulated annealing-based heuristic approach. Robot. Comput. Integr. Manuf. **20**(3), 199–209 (2004)
22. Behnamian, J., Zandieh, M., Fatemi Ghomi, S.M.T.: Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm. Expert. Syst. Appl. **36**(6), 9637–9644 (2009)
23. Shahsavari-pour, N., Ghasemishabankareh, B.: A novel hybrid meta-heuristic algorithm for solving multi objective flexible job shop scheduling. J. Manuf. syst. **32**(4), 771–780 (2013)
24. Zhang, G.H., Shao, X.Y., Li, P.G., Gao, L.: An effective hybrid particle swarm optimization algorithm for multi-objective flexible job shop scheduling problem. Comput. Ind. Eng. **56**(4), 1309–1318 (2009)
25. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
26. Kacem, I., Hammadi, S., Borne, P.: Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Math. Comput. Simul. **60**(3–5), 245–276 (2002)
27. Taguchi, G.: Introduction to Quality Engineering. Asian Productivity Organization/ UNIPUB, White Plains (1986)
28. Phadke, M.S.: Quality Engineering Using Robust Design. Prentice-Hall, New Jersey (1986)
29. Molla-Alizadeh-Zavardehi, S., Hajiaghaei-Keshteli, M., Tavakoli-Moghaddam, R.: Solving a capacitated fixed-charge transportation problem by artificial immune and genetic algorithms with a Prüfer number representation. Expert Syst. Appl. **38**(8), 10462–10474 (2011)
30. Hajiaghaei-Keshteli, M., Molla-Alizadeh-Zavardehi, S., Tavakoli-Mogaddam, R.: Addressing a nonlinear fixed-charge transportation problem using a spanning tree-based genetic algorithm. Comput. Ind. Eng. **59**(2), 259–271 (2010)
31. Behnamian, J., Zandieh, M.: A discrete colonial competitive algorithm for hybrid flowshop scheduling to minimize earliness and quadratic tardiness penalties. Expert Syst. Appl. **38**(13), 14490–14498 (2011)