DOI: 10.21917/ijsc.2012.0066

A BAT ALGORITHM FOR REALISTIC HYBRID FLOWSHOP SCHEDULING PROBLEMS TO MINIMIZE MAKESPAN AND MEAN FLOW TIME

M. K. Marichelvam[1] and T. Prabaharan[2]
[1]*Department of Mechanical Engineering, Kamaraj College of Engineering and Technology, India*
E-mail: mkmarichelvamme@gmail.com
[2]*Department of Mechanical Engineering, Mepco Schlenk Engineering College, India*
E-mail: tpraba@mepcoeng.ac.in

Abstract

This paper addresses the multistage hybrid flow shop (HFS) scheduling problems. The HFS is the special case of flowshop problem. Multiple parallel machines are considered in each stage in the HFS. The HFS scheduling problem is known to be strongly NP-hard. Hence, many researchers proposed metaheuristic algorithms for solving the HFS scheduling problems. This paper develops a bat algorithm (BA) to the HFS scheduling problem to minimize makespan and mean flow time. To verify the developed algorithm, computational experiments are conducted and the results are compared with other metaheuristic algorithms from the literature. The computational results show that the proposed BA is an efficient approach in solving the HFS scheduling problems.

Keywords:

Scheduling, Hybrid Flowshop, NP-Hard, Bat Algorithm, Makespan, Mean Flow Time

1. INTRODUCTION AND LITERATURE REVIEW

Scheduling is a decision making process that plays vital role in both manufacturing and service industries. Different types of scheduling problems were addressed in the literature. This paper considers a hybrid flowshop scheduling problem. The HFS may also be called as a flexible flow shop. A HFS is a special case of flowshop with a series of production stages. Each stage consists of parallel machines that can be identical, uniform or unrelated. Some stages may have only one machine. But, at least one stage must have multiple machines. The layout of a HFS environment is shown in Fig.1.

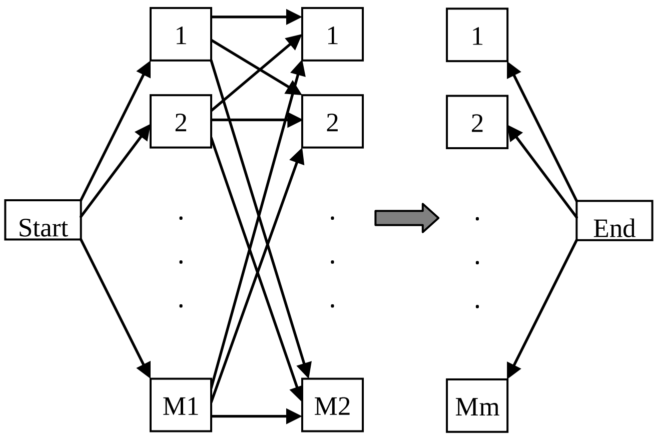

Fig.1. Layout of a HFS environment

The HFS scheduling model was first developed by Arthanary and Ramamurthy [1]. The HFS scheduling problem has been proved to be NP-hard [2]. Due to the complexity of the problem, it is difficult to develop exact methods to solve this problem. Hence, researchers proposed different heuristics and metaheuristics to solve the HFS scheduling problems. Gupta [3] presented heuristic techniques for solving a two-stage flexible flow shop scheduling problems to minimize makespan. They considered only one machine in the second stage. Sriskandarajah and Sethi [4] developed simple heuristic algorithms for the two-stage flexible flow shop problems. Soewnadi and Elmaghraby [5] presented several heuristics for three-stage HFS scheduling problems to minimize makespan. Many other researchers have also proposed heuristics to solve the HFS scheduling problems [6–8].

Recently, researchers adapted different metaheuristics to solve the HFS scheduling problems. Genetic algorithm (GA) [9], simulated annealing algorithm (SA) [10–11], ant colony optimization (ACO) algorithm [12], particle swarm optimization (PSO) algorithm [13] and artificial immune system (AIS) algorithm [14–15]. Recently, an exhaustive review on research carried out in the field of HFS problem has been presented by Ruiz and Vázquez-Rodríguez [16].

The rest of the paper is organized as follows. The problem is defined in section 2. The proposed algorithm is presented in section 3. Finally, the conclusions and future research opportunities are discussed in section 4.

2. PROBLEM DEFINITION

The problem is defined as follows. Let us consider a set of *n* jobs to be processed on *M* consecutive stages to minimize both makespan and mean flow time. Makespan is the completion time of the last job in the production system. Makespan is important for measuring the system utilization. Mean flow time is the average time spent by the jobs in the production system.

2.1 MATHEMATICAL MODEL

The problem is mathematically formulated as follows. The following notations are used in this paper.

C_{js}	Completion time of job *j* at stage *s*
C_{max}	Makespan
$\bar{f}$	Mean flow time
K	A constant ($K\to\infty$)
R_j	Ready time of job *j*
S_{js}	Starting time of job *j* at stage *s*

y	objective function
w_1	weightage for makespan
w_2	weightage for mean flow time

$$\min\ y = w_1 C_{\max} + w_2 \bar{f} \tag{1}$$

Subject to,

$$C_{\max} \geq C_{js}, \text{ for all } s = 1, 2, \ldots, M, j = 1, 2, \ldots, n \tag{2}$$

$$C_{js} = S_{js} + P_{sj} \tag{3}$$

$$\sum_{i=1}^{m_s} Y_{jis} = 1, \text{ for all } s = 1,2,\ldots,M, j = 1,2,\ldots.n \tag{4}$$

$$C_{js} \leq S_{j(s+1)}, \text{ for } s = 1, 2, \ldots, M-1 \tag{5}$$

$$S_{hs} \geq C_{js} - KW_{hjs}, \text{ for all job pairs } (h, j) \tag{6}$$

$$S_{js} \geq C_{hs} + K - 1, \quad \text{for all job pairs } (h, j) \tag{7}$$

$$S_{j1} \geq R_j, \text{ for all } j = 1, 2, \ldots, n \tag{8}$$

$$Y_{jis} \in \{0,1\}, \quad W_{jhj} \in \{0,1\}, \text{ for all } j = 1,\ldots n, i = 1,2,\ldots m_s \text{ and } s = 1,2,\ldots M \tag{9}$$

$$C_{js} \geq 0, \text{ for all } s = 1, 2, \ldots, M, j = 1, 2, \ldots, n \tag{10}$$

$$F = \sum C_{jM} - R_j \text{ for all } j = 1, 2, \ldots, n \tag{11}$$

$$F \geq 0 \tag{12}$$

$$w_1 \geq 0 \tag{13}$$

$$w_2 \geq 0 \tag{14}$$

$$w_1 + w_2 = 1 \tag{15}$$

The objective function (1) considers the minimization of the makespan and mean flow time. The constraint (2) ensures that the makespan is at least equal to the completion times of the last job. Because the objective is to minimize the makespan, constraints in this set will be tight at optimality whenever $C_{\max}$ is positive. The constraint (3) corresponds to the computation of the completion time of job P_{sj} being the processing time of job j at stage s. The constraint (4) ensures that each job is assigned to exactly one machine at each stage. The constraint (5) forces to start the processing of each job only when it has been completed at the precedent stage. The set of constraints (5) and (6) ensure that only one job is on a machine of a stage at any one time. When $W_{hjs} = 1$, and job h is before job j, the constraint in Eq.(6) is trivially satisfied. Eq.(7) requires that the starting time of job j at stage s must be after the completion time for job h. When $W_{hjs} = 0$, indicating that job j is before job h, the constraint in Eq.(7) is trivially satisfied and the starting time of job h at stage s must be the completion time for job j at stage s to satisfy Eq.(6). The constraint (8) bounds the job starting times to be after job release times in the system. The constraint (9) forces both variable Y_{jis} and W_{hjs} to assume binary values 0 or 1 [43]. The Eq.(11) is the calculation of flow time. The constraints (10) and (12-14) represent the non-negative constraints. The constraint (15) indicates that the sum of the weights to be one.

2.2 ASSUMPTIONS

1) The number of stages and the number of machines at each stage are known in advance.
2) The numbers of jobs, their processing times are known in advance and are fixed.
3) All the jobs and the machines are available at time zero.
4) No preemption is allowed.
5) The setup and transportation times of the jobs are independent of the sequence and are included in the processing times.
6) Each machine can process only one job at a time.
7) All the machines are available for the entire period of scheduling. (No machine breakdown).

3. BAT ALGORITHM

Bat algorithm (BA) is a recent, population-based metaheuristic optimization algorithm developed by Yang [17] in 2010 for solving constrained optimization problems. BA is based on the echolocation behaviour of microbats with varying pulse rates of emission and loudness. Yang [18] adapted a multi-objective BA to solve design optimization problems such as welded beam design problems. Radha and Valarmathi [19] presented a modified BA for detecting and optimizing Phishing websites. They compared the BA with other metaheuristics like ACO and PSO algorithms. Bora, Coelho and Lebensztajn [20] developed the BA to solve the Brushless DC Wheel Motor Problem. Tsai et al. [21] addressed a BA inspired algorithm for solving the numerical optimization problems. There is no addressed application of BA to solve the scheduling problems. Hence, in this paper, we present an improved bat algorithm to solve the multi-objective HFS scheduling problems.

3.1 BAT BEHAVIOUR

Bats are fascinating animals. They are the only mammals with wings and they also have advanced capability of echolocation. There are about 996 different species. Their size ranges from the tiny bumblebee bat (of about 1.5 to 2 g) to the giant bats with wingspan of about 2 m and weight up to about 1 kg. Microbats typically have forearm length of about 2.2 to 11cm. Most bats uses echolocation to a certain degree; among all the species, microbats are a famous example as microbats use echolocation extensively while megabats do not.

Microbats use a type of sonar, called, echolocation, to detect prey, avoid obstacles, and locate their roosting crevices in the dark. These bats emit a very loud sound pulse and listen for the echo that bounces back from the surrounding objects. Their pulses vary in properties and can be correlated with their hunting strategies, depending on the species. Most bats use short, frequency-modulated signals to sweep through about an octave, while others more often use constant-frequency signals for echolocation. Their signal bandwidth varies depends on the species, and often increased by using more harmonics.

3.2 ECHOLOCATION OF MICROBATS

Each pulse only lasts a few thousandths of a second (up to about 8 to 10 ms). However, it has a constant frequency which is usually in the region of 25 kHz to 150 kHz. The typical range of frequencies for most bat species are in the region between 25 kHz and 100 kHz, though some species can emit higher frequencies up to 150 kHz. Each ultrasonic burst may last typically 5 to 20 ms, and microbats emit about 10 to 20 such sound bursts every second. When hunting for prey, the rate of pulse emission can be sped up to about 200 pulses per second when they fly near their prey. Such short sound bursts imply the fantastic ability of the signal processing power of bats. In fact, studies shows the integration time of the bat ear is typically about 300 to 400 μs. As the speed of sound in air is 340 m/s, the wavelength of the ultrasonic sound bursts with a constant frequency. The wavelength is calculated as,

$$\lambda = \frac{v}{f} \tag{16}$$

where, v is the speed of sound in air and f is the frequency. The wavelength is in the range of 2 mm to 14 mm for the typical frequency range from 25 kHz to 150 kHz. Such wavelengths are in the same order of their prey sizes.

Studies show that microbats use the time delay from the emission and detection of the echo, the time difference between their two ears, and the loudness variations of the echoes to build up three dimensional scenario of the surrounding. However, here we are only interested in the echolocation and the associated behaviour. Such echolocation behaviour of microbats can be formulated in such a way that it can be associated with the objective function to be optimized, and this makes it possible to formulate new optimization algorithms.

3.3 BAT ALGORITHM

If we idealize some of the echolocation characteristics of microbats, we can develop various bat-inspired algorithms or bat algorithms. In the basic bat algorithm developed by Yang (2010), the following approximate or idealized rules were used.

1) All bats use echolocation to sense distance, and they also ‘know’ the difference between food/prey and background barriers in some magical way;
2) Bats fly randomly with velocity v_i at position x_i with a frequency f_{min}, varying wavelength λ and loudness A_0 to search for prey. They can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission $r \in [0, 1]$, depending on the proximity of their target;
3) Although the loudness can vary in many ways, we assume that the loudness varies from a large (positive) A_0 to a minimum constant value $A_{\min}$.

Another obvious simplification is that no ray tracing is used in estimating the time delay and three dimensional topography. Though this might be a good feature for the application in computational geometry, however, we will not use this feature, as it is more computationally extensive in multidimensional cases.

In addition to these simplified assumptions, we also use the following approximations, for simplicity. In general the frequency f in a range [$f_{\min}$, $f_{\max}$] corresponds to a range of wavelengths [$\lambda_{\min}$, $\lambda_{\max}$]. For example a frequency range of [20 kHz, 500 kHz] corresponds to a range of wavelengths from 0.7 mm to 17 mm in reality. Obviously, we can choose the ranges freely to suit different applications.

The basic steps of the BA can be summarized as the pseudo code shown below,

```
Objective function $f(x)$, $x = (x_1, ..., x_d)^T$
Initialize the bat population $x_i$ (i = 1, 2, ..., $n$) and $v_i$
Define pulse frequency $f_i$ at $x_i$
Initialize pulse rates $r_i$ and the loudness $A_i$
while ($t$ < Max number of iterations)
Generate new solutions by adjusting frequency and updating velocities and locations/solutions
    if (rand > $r_i$)
        Select a solution among the best solutions
        Generate a local solution around the selected best solution
    end if
        Generate a new solution by flying randomly
    if (rand < $A_i$ & $f(x_i) < f(x_*)$)
        Accept the new solutions
        Increase $r_i$ and reduce $A_i$
    end if
Rank the bats and find the current best $x_*$
end while
Postprocess results and visualization
```

Pseudo code of the bat algorithm

3.4 BAT MOTION

For the bats in simulations, we have to define the rules how their positions x_i and velocities v_i in a d-dimensional search space are updated. The new solutions x_i^t and velocities v_i^t at time step t are given by,

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta \tag{17}$$

$$v_i^{t+1} = v_i^t + (x_i^t - x_*) f_i \tag{18}$$

$$x_i^{t+1} = x_i^t + v_i^t \tag{19}$$

where, $\beta \in$ [0,1] is a random vector drawn from a uniform distribution. Here x_* is the current global best location (solution) which is located after comparing all the solutions among all the n bats at each iteration t. As the product $\lambda_i f_i$ is the velocity increment, we can use λ_i (or f_i) to adjust the velocity change while fixing the other factor f_i (or λ_i), depending on the type of the problem of interest. In our implementation, we will use $f_{\min}$ = 0 and $f_{\max}$ = $O(1)$, depending on the domain size of the problem of interest. Initially, each bat is randomly assigned a frequency which is drawn uniformly from [$f_{\min}$, $f_{\max}$].

For the local search part, once a solution is selected among the current best solutions, a new solution for each bat is generated locally using random walk,

$$x_{new} = x_{old} + \varepsilon A^t \tag{20}$$

where, ε is a random number vector drawn from [−1, 1], while $A^t = < A_i^t >$ is the average loudness of all the bats at this time step.

The update of the velocities and positions of bats have some similarity to the procedure in the standard PSO, as f_i essentially controls the pace and range of the movement of the swarming particles. To a degree, BA can be considered as a balanced combination of the standard PSO and the intensive local search controlled by the loudness and pulse rate.

3.5 LOUDNESS AND PULSE EMISSION

Furthermore, the loudness A_i and the rate r_i of pulse emission have to be updated accordingly as the iterations proceed. As the loudness usually decreases once a bat has found its prey, while the rate of pulse emission increases, the loudness can be chosen as any value of convenience. For example, we can use A_o = 100 and A_{min} = 1. For simplicity, we can also use A_o = 1 and A_{min} = 0, assuming A_{min} = 0 means that a bat has just found the prey and temporarily stop emitting any sound. Now we have,

$$A_i^{t+1} = \alpha A_i^t, \quad r_i^t = r_i^o\left[1-\exp(-\gamma t)\right] \tag{21}$$

where, α and γ are constants. In fact, α is similar to the cooling factor of a cooling schedule in the simulated annealing [18]. For any $0 < \alpha < 1$ and $\gamma > 0$, we have,

$$A_i^t \to 0, \quad r_i^t \to r_i^o, \quad \text{as } t \to \infty \tag{22}$$

In the simplest case, we can use $\alpha = \gamma$, and we have used $\alpha = \gamma = 0.9$ in our simulations.

3.6 BAT ALGORITHM FOR HFS SCHEDULING

Preliminary studies by Yang [18] suggested that bat algorithm is very promising for solving nonlinear global optimization problems. However, the major drawback of applying BA to combinatorial problems is due to its continuous nature. In order to enable the continuous BA to be applied to the discrete scheduling problems, we apply the smallest position value (SPV) proposed by Bean [22]. The SPV rule is presented to convert the continuous position values to a discrete job permutation.

In the BA for the HFS scheduling problems, parameters were initialized and a population was generated randomly. Each bat is initialized with some random position. The SPV rule applies to each bat to find its corresponding permutation. Thus, each bat will be evaluated by using the permutation to compute the objective function for the HFS scheduling problems with makespan criterion.

The solution representation is explained in the following section.

3.6.1 Solution representation:

The vector $X_i^t = \left(X_{i1}^t, X_{i2}^t, \ldots X_{in}^t\right)$ represents the continuous position values of the bats in the search space. The SPV rule is used to convert the continuous position values of the bats to the discrete job permutation. The solution representation for a 4 job problem is described in Table.1.

Table.1. Solution representation in bat algorithm

	Dimension *j*			
	1	**2**	**3**	**4**
x_{ij}	0.41	0.36	0.78	0.52
Jobs	2	1	4	3

The smallest position value is $x_{i2}^t = 0.14$ and the dimension j = 2 is assigned to be the first job in the permutation according to the SPV rule. The second smallest position value is $x_{i1}^t = 0.41$ and the dimension j = 1 is assigned to be the second job in the permutation. Similarly, all the jobs are assigned in the permutation.

3.7 COMPUTATIONAL RESULTS

To test the performance of the proposed algorithm, we consider a furniture manufacturing company in Chennai, India. The company produces different types of furniture components. Among them we consider the three drawer furniture components. Each component consists of 20 parts. Each part is considered as a job. The jobs are processed in different stages. The production system consists of five different stages namely punching stage, bending stage, welding stage, power pressing stage and drilling stage. Each job has a fixed processing time at each stage. Hence the problem is to obtain an optimal schedule to minimize the makespan and mean flow time. In the company, the production schedule is performed manually. Hence, it is not possible to obtain optimal schedules. We try to propose the BA to obtain the optimal schedule.

Moreover, we conduct the design of experiments to test the performance of the proposed algorithm for large scale problems. The factor levels for the experiments are shown in Table.2 to define the production systems: the number of stages, number of machines, the number of jobs, processing time of jobs, Population size, product types and the number of iterations.

Table.2. Factor levels for the design of experiments

Factors	Levels
Number of stages	5, 10 and 20
Number of machines in each stage	2, 4, 5 and 10
Number of jobs	10, 20, 50 and 100
Processing time distribution	Uniform (1 – 50)
Population size	100 and 200
Product types	5
Number of iterations	100, 200 and 500

Hence, we generate 3*3*4*1*2*1*3 = 216 problem instances randomly. The BA is coded in C++ and ran on a PC Pentium 4 processor with 3 GHz and 1 GB memory.

The proposed BA is compared with the other metaheuristics in the literature namely GA [9], SA [10], ACO [12], PSO [13] and AIS [14]. To measure the performance of different metaheuristics, we use percentage improvement (*PI*) as a performance measure. The *PI* is calculated as follows,

$$PI = \frac{y_{algo} - y_{best}}{y_{algo}} \times 100 \tag{23}$$

where,

y_{algo} – objective function obtained by different metaheuristic algorithms

y_{best} – best objective function

The result comparison of different algorithms is shown in Fig.2. From the figure it is easily concluded that the BA is more efficient than the GA, SA, ACO, PSO and AIS algorithms addressed in the literature. The *PI* obtained by the BA is more than 20%.

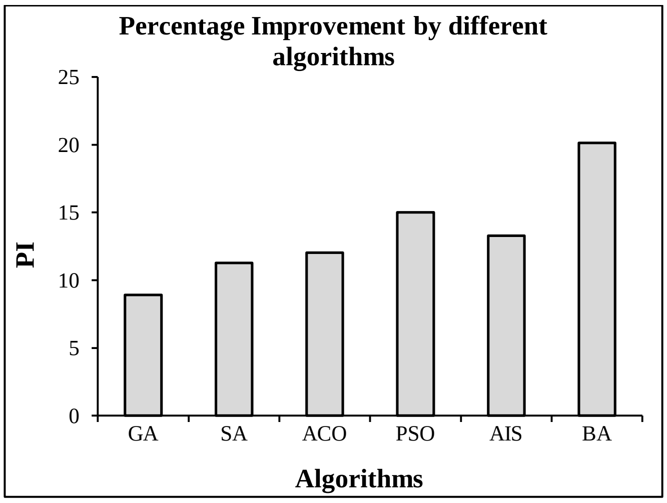

Fig.2. Result comparison

4. CONCLUSION

In this paper, we present a recently developed Bat algorithm to solve a real-world HFS scheduling problems. To the best of our knowledge, this is the first attempt to solve HFS scheduling problems using the Bat algorithm. Computational results show that the proposed Bat algorithm provides better results than many other metaheuristics. The proposed bat algorithm may be used in hybrid with other heuristics and metaheuristics. It would be interesting to apply the proposed algorithm to solve HFS scheduling problems with other objective functions. Developing multi-population bat algorithm to solve the multi-objective HFS scheduling problems would be another possible scope of this research work.

REFERENCES

[1] T.S. Arthanary and K.G. Ramaswamy, “An extension of two machines sequencing problem”, *OPSEARCH, Journal of the Operational Research Society of India*, Vol. 8, No. 4, pp. 10–22, 1971.

[2] Chung-Yee Lee and George L. Vairaktarakis, “Minimizing makespan in hybrid flowshops”, *Operations Research Letters*, Vol. 16, No. 3, pp. 149–158, 1994.

[3] J.N.D. Gupta, A.M.A. Hariri and C.N. Potts, “Scheduling a two-stage hybrid flow shop with parallel machines at the first stage”, *Annals of Operations Research*, Vol. 69, pp. 171–191, 1997.

[4] C. Sriskandarajah and S.P. Sethi, “Scheduling algorithms for flexible flowshops: worst and average case performance”, *European Journal of Operational Research*, Vol. 43, No. 2, pp. 143–160, 1989.

[5] H. Soewnadi and S.E. Elmaghraby, “Sequencing three-stage flexible flowshops with identical machines to minimize makespan”, *IIE Transactions*, Vol. 33, No. 11, pp. 985–994, 2001.

[6] S.A. Brah and L.L. Loo, “Heuristics for scheduling in a flow shop with multiple processors”, *European Journal of Operational Research*, Vol. 113, No. 1, pp. 113–122, 1999.

[7] F.Y. Ding and D. Kittichartphayak, “Heuristics for scheduling flexible flow lines”, *Computers and Industrial Engineering*, Vol. 26, No. 1, pp. 27–34, 1994.

[8] E. Figielska, “A new heuristic for scheduling the two-stage flowshop with additional resources”, *Computers and Industrial Engineering*, Vol. 54, No. 4, pp. 750–763, 2008.

[9] E. Rashidi, M. Jahandar M and M. Zandieh, “An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines”, *The International Journal of Advanced Manufacturing Technology*, Vol. 49, No. 9-12, pp. 1129–1139, 2010.

[10] C. Low, “Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines”, *Computers and Operations Research*, Vol. 32, No. 8, pp. 2013–2025, 2005.

[11] H.M. Wang, F.D. Chou and F.C. Wu, “A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan”, *The International Journal of Advanced Manufacturing Technology*, Vol. 53, No. 5-8, pp. 761–776, 2011.

[12] K. Alaykyran, O. Engin and A. Doyen, “Using ant colony optimization to solve hybrid flow shop scheduling problems”, *The International Journal of Advanced Manufacturing Technology*, Vol. 35, No. 5-6, pp. 541–550, 2007.

[13] M.R. Singh and S.S. Mahapatra, “A swarm optimization approach for flexible flow shop scheduling with multiprocessor tasks”, *The International Journal of Advanced Manufacturing Technology*, Vol. 62, No. 1-4, pp. 267-277, 2012.

[14] Q. Niu, T. Zhou and S. Ma, “A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion”, *Journal of Universal Computer Science*, Vol. 15, No. 4, pp.765–785, 2009.

[15] O. Engin and A. Doyen, “A new approach to solve hybrid flow shop scheduling problems by artificial immune system”, *Future Generation Computer Systems*, Vol. 20, No. 6, pp. 1083–1095, 2004.

[16] R. Ruiz and J. A. Vázquez-Rodríguez, “The hybrid flowshop scheduling problem”, *European Journal of Operational Research*, Vol. 205, No. 1, pp. 1–18, 2010.

[17] X.S. Yang, “A new Metaheuristic Bat-inspired algorithm”, *Nature Inspired Cooperative Strategies for Optimization*, Vol. 284, pp. 65-74, 2010.

[18] X.S. Yang, “Bat Algorithm for Multi-objective Optimization”, *International Journal of Bio-Inspired Computation*, Vol. 3, No. 5, pp. 267-274, 2011.

[19] D. Radha and M.L. Valarmathi, “Phishing website detection and optimization using Modified bat algorithm”, *International Journal of Engineering Research and Applications*, Vol. 2, No. 1, pp. 870–876, 2012.

[20] T.C. Bora, L.S. Coelho and L. Lebensztajn, “Bat-Inspired Optimization Approach for the Brushless DC Wheel Motor

Problem", *IEEE Transactions on Magnetics*, Vol. 48, No. 2, pp. 947–950, 2012.

[21] P.W. Tsai, J.S. Pan, B.Y. Liao, M.J. Tsai and V. Istanda, "Bat Algorithm Inspired Algorithm for Solving Numerical Optimization Problems", *Applied Mechanics and Materials*, Vol. 148-149, pp. 134–137, 2012.

[22] J.C. Bean, "Genetic algorithms and random keys for sequencing and optimization", *INFORMS Journal of Computing*, Vol. 6, No. 2, pp.154–160, 1994.