Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems

Mohammad Kazem Sayadi[a], Reza Ramezanian[a*] and Nader Ghaffari-Nasab[a]

[a]*Department of Industrial Engineering, Iran University of Science and Technology, Tehran, Iran*

A R T I C L E I N F O

Article history:
Received 23 January 2010
Received in revised form
23 April 2010
Accepted 26 April 2010
Available online 26 April 2010

Keywords:
Meta-heuristic
Firefly meta-heuristic
Ant colony
Permutation flow shop
Scheduling
Combinatorial optimization
Mixed integer programming

A B S T R A C T

During the past two decades, there have been increasing interests on permutation flow shop with different types of objective functions such as minimizing the makespan, the weighted mean flow-time etc. The permutation flow shop is formulated as a mixed integer programming and it is classified as NP-Hard problem. Therefore, a direct solution is not available and meta-heuristic approaches need to be used to find the near-optimal solutions. In this paper, we present a new discrete firefly meta-heuristic to minimize the makespan for the permutation flow shop scheduling problem. The results of implementation of the proposed method are compared with other existing ant colony optimization technique. The preliminary results indicate that the new proposed method performs better than the ant colony for some well known benchmark problems.

1. Introduction

The flow shop scheduling problem (FSSP) is normally classified as a complex combinatorial optimization problem, in which there is a set of n jobs ($1, \ldots, n$) to be processed in a set of m machines ($1, \ldots, m$) in the same order. We normally look for a special sequence of processing the jobs in the machines to minimize one or more criteria such as minimization of makespan, mean flow, etc. There are different most commonly used criteria such as the minimization of the total completion time or makespan of the schedule ($C_{\max}$) which is sometimes referred to as maximum flow time or $F_{\max}$. The processing times needed for the jobs on the machines are assumed to be non-negative and deterministic denoted as p_{ij}, with $i = 1 \ldots, n$ and $j = 1, \ldots, m$. Although the optimal solution of the flow shop scheduling problem can be determined in polynomial time when $m=2$ (Johnson, 1954), the general form of this kind of problem is known to be NP-Complete in the strong sense when $m \geq 3$ (see Garey et al., 1976) and generally $(n!)^m$ schedules need to be considered. That is why the problem is somewhat restricted in by not allowing job passing. In this case, "only" $n!$ schedules must be considered and the problem is then known as permutation flow shop which is classified as $n/m/P/F_{\max}$ or as $F/prmu/C_{\max}$ (see Pinedo, 2002) and the primary focus of the work of this paper is the last type of flow shop environment. Johnson (1954) is believed to be the first who introduced flow shop scheduling. Since then, flow shop scheduling has become one of the most interesting topics among researchers and practitioners. There are

* Corresponding author. Tel./fax: +98-21-77240482.
E-mail addresses: Reza_Ramezanian@ind.iust.ac.ir (R. Ramezanian),

doi: 10.5267/j.ijiec.2010.01.001

different forms of flow shop optimization such as minimization of the makespan which is one of the most popular one. The solution procedure for the flow shop problem is often either heuristic or meta-heuristics. Turner and Booth (1987) compared two famous heuristics with a set of 350 random problems. Ponnambalam et al. (2001) compared five different heuristics against only 21 typical test problems. Ruiz and Maroto (2005) presented a review and comparative evaluation of heuristics and meta-heuristics for the permutation flow shop problem with the makespan criterion. They compared 25 methods, ranging from the classical Johnson's algorithm to the most recent meta-heuristics. Lian et al. (2006) applied an efficient similar particle swarm optimization algorithm (SPSOA) to the PFSS problem with the objective of minimizing the makespan. Tasgetiren et al. (2007) solved the permutation flow shop sequencing problem (PFSP) with a particle swarm optimization algorithm (PSO). They considered the objectives of minimizing makespan and the total flow time of jobs. Ruiz and Stutzle (2007) presented a new iterated greedy algorithm that applies two phases iteratively, named destruction, where some jobs are eliminated from the incumbent solution, and construction, where the eliminated jobs are reinserted into the sequence using the well known NEH construction heuristic. Naderi and Ruiz (2010) studied a new generalization of the regular permutation flow shop scheduling problem (PFSP) referred to as the distributed permutation flow shop scheduling problem or DPFSP. Under this generalization, they assumed that there are a total of F identical factories or shops, each one with *m* machines disposed in series. A set of *n* available jobs have to be distributed among the *F* factories and then a processing sequence has to be derived for the jobs assigned to each factory. Their optimization criterion was the minimization of the maximum completion time or makespan among the factories. Dong et al. (2009) presented an integrated local search algorithm to solve the permutation flow shop sequencing problem with total flow time criterion. They showed the effectiveness and superiority of their method over three constructive heuristics, three ant-colony algorithms and a particle swarm optimization algorithm. Vallada and Ruiz (2009) worked on a cooperative meta-heuristic method for the permutation flow shop scheduling problem considering two objectives separately: total tardiness and makespan. They adopted the island model where each island runs an instance of the method and communications begin when the islands are reached to a certain level of evolution. Farahmand Rad et al. (2009) showed five new methods that outperform the well-known NEH heuristic as supported by careful statistical analyses using the well-known instances of Taillard. The proposed methods attempt to counter the excessive greediness of NEH by carrying out re-insertions of already inserted jobs at some points in the construction of the solution. Vallada and Ruiz (2010) presented three genetic algorithms for the permutation flow shop scheduling problem with total tardiness minimization criterion. The algorithms include advanced techniques like path re-linking, local search and a procedure to control the diversity of the population.

In this paper, we consider a permutation flow shop scheduling problem with the objective of minimizing the makespan. The method is then solved using a discrete firefly algorithm (DFA) and it is employed to solve the problem. Firefly algorithm, developed by Xin-She Yang (2008), is a novel population based technique for solving continues optimization problem, especially for continues NP-hard problems and has been motivated by the simulation of the social behavior of fireflies. The flashing light of fireflies is a fantastic sight in the sky and fireflies normally attract mating partners and potential prey by using such flashes. Both genders join together by the rhythmic flash, the rate of flashing and the amount of time of flashing. Females respond to a male's unique and peerless pattern of flashing. It is possible to formulate optimization algorithms because the flashing light can be formulated in such a way that it is associated with the objective function to be optimized. Based on the Xin-She Yang's paper, firefly algorithm is very efficient in finding the global optima with high success rates. Xin-She Yang's simulation results for finding the global optima of various test functions suggest that particle swarm optimization often outperforms customary algorithms such as genetic algorithms, while the firefly algorithm is superior to both PSO and GA in terms of both efficiency and success rate (2009). In 2009, Lukasik and Zak (2009) deliberate the firefly algorithm for continuous constrained optimization task. Their experimental evaluation demonstrates efficiency of the firefly algorithm.

The remainder of the paper is organized as follows: In section 2, a mathematical model for permutation flow shop scheduling problems is presented. Section 3 describes our proposed discrete firefly algorithm for solving the model. Section 4 presents the computational results acquired and, finally, section 5 provides conclusions and suggestions for further research.

2. Mathematical formulation

In this section, a mathematical model for permutation flow shop scheduling problem is presented. The assumptions of the model are as follows:

- All *n* jobs to schedule are independent and available for processing at time zero.
- Machine cannot process two jobs at the same time.
- Each job is processed on at most one machine at a time.
- Setup times for the operations are sequence-independent and are included in processing times.
- There is only one of each type of machine.
- No more than one operation of the same job can be executed at a time.

Parameters:

n: Number of jobs

m: Number of machines

i: Machine index, (*i*=1,..,*m*)

j: Job index, (*j*=1,…,*n*)

k: Order index, (*k*=1,…,*n*)

t_{ij} : Processing time of job *j* on machine *i*

Decision variables:

q_{ijk} : Completion time of job *j* on machine *i* in *k*th order

x_{jk} : Binary variable taking value 1 if job *j* is processed in *k*th order and 0, otherwise.

The mathematical model for minimizing the makespan is as follow:

$$\min \quad \sum_{j=1}^{n} q_{mjn}.x_{jn} \tag{1}$$

subject to

$$\sum_{k=1}^{n}(q_{(i+1)jk} - t_{(i+1)j})x_{jk} \geq \sum_{k=1}^{n} q_{ijk}x_{jk} \quad i=1,...,m-1; j=1,..,n \tag{2}$$

$$\sum_{j=1}^{n}(q_{ij(k+1)} - t_{ij})x_{j(k+1)} - \sum_{j=1}^{n} q_{ijk}x_{jk} \geq 0, \quad i=1,...,m; k=1,..,(n-1) \tag{3}$$

$$\sum_{j=1}^{n} x_{jk} = 1, \quad \forall i,k \tag{4}$$

$$\sum_{k=1}^{n} x_{jk} = 1, \quad \forall i,j \tag{5}$$

$$q_{ijk} \le M x_{jk}, \ \forall i, j, k \tag{6}$$

$$q_{ijk} \ge 0, \forall i, j, k \tag{7}$$

$$x_{ijk} = \{0,1\}, \forall i, j, k \tag{8}$$

The objective function (1) represents the minimization of the makespan. Constraints (2) and (3) certify that a job does not start on a machine until it finishes processing on the previous machine and its predecessor has completed processing on that machine. Constraint (4) insures that each sequence position is filled with only one job and constraint (5) insures that each job is assigned to only one position in the job sequence. The constraint set (6) is a relationship between the binary variables and the completion time variables. Note that when each binary variable becomes zero, then its completion time variable will be also equal to zero. Finally, (7) and (8) are logical constraints.

3. Discrete firefly algorithm

3.1. **Methodology**

Nature-inspired methodologies are among the most powerful algorithms for optimization problems. Firefly algorithm is a novel nature-inspired algorithm inspired by social behavior of fireflies. Fireflies are one of the most special, captivating and fascinating creature in the nature. There are about two thousand firefly species, and most fireflies produce short and rhythmic flashes. The rate and the rhythmic flash, and the amount of time form part of the signal system which brings both sexes together. Therefore, the main part of a firefly's flash is to act as a signal system to attract other fireflies. By idealizing some of the flashing characteristics of fireflies, firefly-inspired algorithm was presented by Xin-She Yang (2008). Firefly-inspired algorithms use the following three idealized rules: 1) All fireflies are unisex which means that they are attracted to other fireflies regardless of their sex; 2) The degree of the attractiveness of a firefly is proportion to its brightness, thus for any two flashing fireflies, the less brighter one will move towards the brighter one and the more brightness means the less distance between two fireflies. If there is no brighter one than a particular firefly, it will move randomly; 3) The brightness of a firefly is determined by the value of the objective function. For a maximization problem, the brightness can be proportional to the value of the objective function. Other forms of brightness can be defined in a similar way to the fitness function in genetic algorithms (2009). Based on the effectiveness of the firefly algorithm in optimizing continues problems, it is predictable that this algorithm would be impressive to solve discrete optimization problems which creates the motivation for proposing a discrete firefly algorithm. In the discrete firefly algorithm, there are four important issues:

Attractiveness: In the firefly algorithm, the main form of attractiveness function β(r) can be any monotonically decreasing functions such as the following generalized form:

$$\beta(r) = \beta_0 e^{-\gamma r^m}, \qquad (m \ge 1) \tag{9}$$

where r is the distance between two fireflies, β_0 is the attractiveness at $r = 0$ and γ is a fixed light absorption coefficient.

Distance: The distance between any two fireflies i and j at X_i and X_j is the Cartesian distance as follows,

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2}, \tag{10}$$

where $x_{i,k}$ is the k-th component of the i-th firefly(X_i).

Movement: The movement of a firefly, *i* is attracted to another more attractive (brighter) firefly *j*, is determined by

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha\,(\text{rand} - \frac{1}{2}), \tag{11}$$

where the second term is due to the attraction while the third term is randomization with α being the randomization parameter and "rand" is a random number generator uniformly distributed in [0, 1].

Discretization: When the firefly *i* moves toward firefly *j*, the position of firefly *i* is changed from a binary number to a real number. Therefore, we must replace this real number by a binary number. The following sigmoid function restricts $S(x_{ik})$ to be in the interval of zero to one,

$$S(x_{ik}) = \frac{1}{1+\exp(-x_{ik})}, \tag{12}$$

where $S(x_{ik})$ denotes the probability of bit x_{ik} taking 1.

3.2. **Discrete firefly algorithm for permutation flow shop scheduling problems**

The position of the *i*-th firefly in the *t*-th generation is denoted as $Y_i^t = (y_{i11}^t, y_{i12}^t, \dots, y_{imc}^t)$, $y_{ijk}^t = 1$ if job *j* of firefly *i* is placed in the *k*-th priority at *t*-th generation and 0, otherwise. For example, suppose that we have $y_{i13}^t = y_{i21}^t = y_{i32}^t = y_{i44}^t = 1$ and all other $y_{ijk}^t = 0$. This firefly is represented in Table 1.

Table 1
The representation of i-th firefly

		Priority (k)			
		1	2	3	4
Job (j)	1	0	0	1	0
	2	1	0	0	0
	3	0	1	0	0
	4	0	0	0	1

Consider a flow shop scheduling problem with bypass consideration with four jobs and three machines. Processing time of each job on each machine is given in Table 2.

Table 2
Processing times

		Job			
	Machine	1	2	3	4
Machine	1	5	7	3	1
	2	2	4	5	8
	3	6	2	4	3

The sequence of jobs on different machines for *i*-th firefly represented in Table 1 is shown in **Fig** 1.

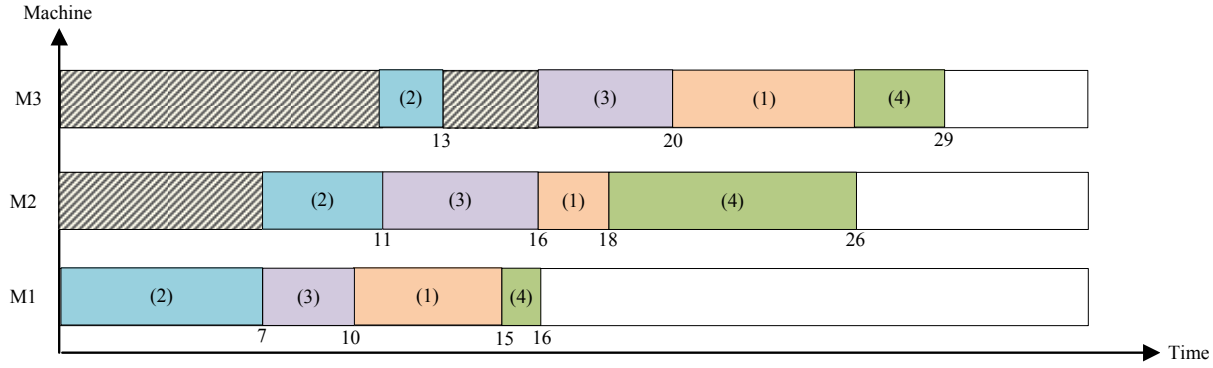

Fig. 1. Gant chart of job sequence vector on machines

When a particular firefly *i* moves toward firefly *j*, the position of firefly *i* is changed from a binary number to a real number. So, when firefly *i* moves toward the firefly *j*, the position of firefly *i* needs to be converted from real numbers to the changes of the probabilities by the following sigmoid function:

$$S(y_{ijk}) = \frac{1}{1+\exp(-y_{ijk})}, \tag{13}$$

where $S(y_{ijk})$ represents the probability of y_{ijk} taking the value of zero to one. For example, in Table 3, $S(y_{i12})$ indicates that there is a 33% chance that the first job of firefly *i* is placed in the second priority.

Table 3
The probability of y_{ijk} taking the value 1

		Priority (k)			
		1	2	3	4
Job (j)	1	0.29	0.33	0.10	0.62
	2	0.21	0.43	0.87	0.89
	3	0.66	0.41	0.70	0.22
	4	0.99	0.12	0.55	0.11

Each firefly assigns jobs to priorities based on its changes of probabilities. For job *j*, the priority *k* with the highest probability is selected and job *j* is assigned to this priority if there is a vacant position in this priority. Otherwise, without considering priority *k*, a position with the highest probability is selected and job *j* is assigned to this position if there is a vacant position in this priority. This action continues until job *j* is assigned to a position.

Local search

In each iteration of the discrete firefly algorithm, we improve the quality of the best solution (firefly) using some local search mechanism. We start from the first job placed in the first position and consider the exchange of the jobs placed in *i*-th and (*i*+1)-th positions. If the exchange improves the objective function, it will be executed; otherwise no action is needed to take place in the existing step and go to the next step of the algorithm.

4. Computational results

The performance of the presented approach is compared with other existing approaches through experimentation on a number of benchmark problems obtained from the literature.

4.1 **Test problems**

In order to study the performance of the proposed method of this paper, we selected 40 benchmark problems given by Demirkol et al. (1998). The test instances involve two machine number values (m=15, 20) and four job number values (n=20, 30, 40, 50), resulting in eight combinations of *m* and *n*, and a total number of operations ranging from 300 to 1000. Five instances were generated for each of the eight combinations of *m* and *n*. A total number of 40 test instances were obtained for the F||Cmax problem. Table 4a and Table 4b demonstrate the details of the performance of the proposed method. In these tables, the instances are denoted by the term flcmax_n_m_InstanceNumber.

4.2 **Results and discussion**

The presented approach was coded in MATLAB R2007(b), and it was run on a personal computer with an Intel Pentium(R) (3GHz) CPU 1GB RAM. The approach was employed to solve each instance in ten trials, and the best trial was taken as the objective function value obtained. In addition to comparisons with the upper bound (UB) determined by the eight algorithms proposed by Demirkol et al. (1998), the computational results of the presented GA were also compared with an ant colony optimization meta-heuristic, namely MHD-ACS, proposed by Ying and Lin (2007). The final results are shown in Table 4(a) and Table 4(b). To calculate the average improvement rates of the approaches, the formula (14) is employed to calculate the improvement rate of each problem, for the solution methods.

$$\frac{C_{\max(b)} - C_{\max(m)}}{C_{\max(b)}} \times 100, \tag{14}$$

where Cmax(A) denotes the objective function value of MHD-ACS and discrete firefly approaches, and Cmax(B) denotes the benchmark values for each individual problem. The average improvement rates of the MHD-ACS and the proposed method for each of the eight problem sets are shown in Table 5. As we can observe, the proposed approach demonstrates a better average improvement rate than the MHD-ACS approach.

Table 4 (a)
Results obtained by the MHD-ACS and proposed discrete-firefly approaches

Problems	Benchmark problems			MHD-ACS		Discrete firefly	
	LB	UB	Time	Cmax	Time	Cmax	Time
flcmax_20_15_3	3354	4437	69.93	4420	46	4164	44.07
flcmax_20_15_6	3168	4144	0.09	4044	46	4010	44.91
flcmax_20_15_4	2997	3779	57.86	3786	46	3734	51.42
flcmax_20_15_10	3420	4302	66.68	4265	45	4192	44.36
flcmax_20_15_5	3494	4373	57.54	4310	45	4215	47.93
flcmax_20_20_1	3776	4821	159.12	4819	59	4740	51.63
flcmax_20_20_3	3758	4779	148.21	4723	60	4515	57.23
flcmax_20_20_9	3902	4944	175.74	4922	60	4810	45.87
flcmax_20_20_2	3881	4886	202.58	4878	60	4736	38.47
flcmax_20_20_10	3823	4717	164.07	4715	60	4619	30.98

Table 4(b)
Results obtained by the MHD-ACS and proposed discrete-firefly approaches

Problems	Benchmark problems			MHD-ACS		Discrete firefly	
	LB	UB	Time	Cmax	Time	Cmax	Time
flcmax_30_15_3	4020	5226	148.99	5210	93	5083	60.46
flcmax_30_15_4	4080	5304	163.22	5284	94	5218	84.97
flcmax_30_15_9	4022	5079	108.63	5075	95	5045	72.56
flcmax_30_15_8	4490	5605	0.17	5593	94	5308	87.67
flcmax_30_15_6	4184	5147	147.33	5149	93	5056	79.59
flcmax_30_20_3	4806	6183	0.24	5987	121	5658	87.29
flcmax_30_20_1	4772	6037	470.73	5989	124	6120	106.12
flcmax_30_20_6	5004	6241	394.33	6195	124	6012	114.45
flcmax_30_20_10	4899	6095	320.22	5923	121	5851	189.80
flcmax_30_20_2	4757	5822	388.40	5840	123	5859	114.27
flcmax_40_15_5	5560	6986	155.97	6972	154	6343	126.09
flcmax_40_15_9	5119	6351	223.64	6310	154	6385	146.01
flcmax_40_15_2	5290	6506	289.02	6532	154	6459	221.17
flcmax_40_15_10	5596	6845	186.49	6712	156	6612	151.78
flcmax_40_15_8	5576	6783	0.24	6771	156	6713	147.19
flcmax_40_20_3	5693	7154	615.49	7132	210	7330	178.26
flcmax_40_20_9	5998	7528	645.20	7496	208	7459	127.64
flcmax_40_20_6	5990	7469	673.92	7476	209	7646	115.97
flcmax_40_20_7	6170	7608	681.51	7588	297	7445	156.07
flcmax_40_20_5	6011	7219	605.76	7217	210	7072	199.49
flcmax_50_15_6	6290	7673	313.15	7631	238	7635	231.64
flcmax_50_15_5	6355	7679	298.54	7496	240	7556	216.53
flcmax_50_15_1	6198	7416	283.88	7402	240	7430	235.91
flcmax_50_15_8	6312	7548	307.38	7558	237	7667	321.28
flcmax_50_15_2	6531	7750	353.77	7712	236	7447	216.06

The results show that the performance of the presented discrete firefly is suitable and can reach to good-quality solutions within a reasonable computational time. Thus, we can use the discrete firefly to solve large-sized permutation flow shop scheduling problems.

Table 5
Average improvement rates (%) of the MHD-ACS and discrete firefly approaches

Problem sets									Average Improvement rate
n	20	20	30	30	40	40	50	50	
m	15	20	15	20	15	20	15	20	
MHD-ACS	0.98	0.37	0.19	1.44	0.51	0.19	0.70	0.13	0.56
Proposed method	3.35	3.01	2.42	2.83	2.77	0.05	0.85	0.93	2.03

5. Conclusion

In this study, we have proposed a discrete firefly algorithm to solve the permutation flow shop scheduling problem where the objective function is the minimization of makespan. The proposed method of this paper has been implemented for some existing benchmark problems in small and medium sizes. The results of the implementation of the proposed method for these benchmark problems have been compared with an alternative ant colony method. The preliminary results indicate that the proposed method performs better than the existing ant colony one. The proposed solution scheme is easy to apply to other algorithms and problems.

Acknowledgment

The author would like to thank the anonymous referee for his/her constructive comments on the earlier version of this work.

References

Demirkol, E., Mehta, S., & Uzsoy, R. (1998). Benchmarks for shop scheduling problems, *European Journal of Operational Research*, 109, 137–141.

Dong, X., Huang, H., & Chen, P. (2009). An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion, *Computers & Operations Research*, 36, 1664 -1669.

Farahmand Rad, S., Ruiz, R., & Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan in permutation flowshops, *Omega*, 37, 331 – 345

Garey, M., Johnson, D., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling, *Mathematics of Operations Research*, 1 (2), 117–129.

Johnson, S., 1954. Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly*, 1, 61.

Lian, Z., Gu, X., & Jiao, B. (2006). A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan, *Applied Mathematics and Computation*, 175, 773–785.

Lukasik, S., & Zak, S. (2009). Firefly algorithm for continuous constrained optimization task, ICCCI 2009, Lecture Notes in Artificial Intelligence (Eds. N. T. Ngugen, R. Kowalczyk, S. M. Chen), 5796, 97-100.

Naderi, B. & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem, *Computers & Operations Research*, 37, 754 – 768.

Pinedo, M. (2002). Scheduling: Theory, Algorithms and Systems, 2nd ed. Prentice-Hall, Englewood Cliffs, NJ.

Ponnambalam, S.G., Aravindan, P., & Chandrasekaran, S. (2001). Constructive and improvement flow shop scheduling heuristics: An extensive evaluation, *Production Planning and Control*, 12 (4), 335–344.

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics, *European Journal of Operational Research*, 165, 479–494.

Ruiz, R., & Stutzle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *European Journal of Operational Research*, 177, 2033–2049.

Tasgetiren, M. F., Liang, Y-C., Sevkli, M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem, *European Journal of Operational Research*, 177, 1930–1947.

Turner, S., & Booth, D. (1987). Comparison of heuristics for flow shop sequencing, *Omega*, 15 (1), 75–78.

Vallada, E. & Ruiz, R. (2009). Cooperative metaheuristics for the permutation flowshop scheduling problem, *European Journal of Operational Research*, 193- 365–376.

Vallada, E., & Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem, *Omega* , 38 , 57 -67.

Yang, X-S. (2008). Nature-Inspired Metaheuristic Algorithm. Luniver Press.

Yang, X-S. (2009). Firefly algorithms for multimodal optimization, in: Stochastic Algorithms: Foundations and Applications, SAGA, *Lecture Notes in Computer Sciences*, 5792, 169-178.

Ying, K.-C., & Lin, S.-W. (2007). Multi-heuristic desirability ant colony system heuristic for non-permutation flowshop scheduling problems, *International Journal of Advanced Manufacturing Technology*, 33, 793–802.