Improving the Multi-Restart Local Search Algorithm by Permutation Matrices and Sorted Completion Times for the Flow Shop Scheduling Problem

J. C. Seck-Tuoh-Mora[1], **L. Garcia-Lechuga**[2], **J. Medina-Marin** [1], **N. Hernandez-Romero** [1], **and E. S. Hernandez-Gress**[1]

[1]Área Academica de Ingeniería, Universidad Autónoma del Estado de Hidalgo, Pachuca, Hidalgo, Mexico

[2]Área de Electromecánica Industrial, Universidad Tecnológica de Tulancingo, Tulancingo, Hidalgo, Mexico

Abstract— *Iterated local search (ILS) is a metaheuristic used successfully to solve the flow shop scheduling problem. In particular, the multi-restart ILS (MRSILS) is an easily implementable algorithm which obtains good results. In this paper, we modify the MRSILS algorithm in two ways. First, small changes in the initial solution are generated by permutation matrices in order to improve it before using the MRSILS. Second, a minor variation is made in the strategy of the MRSILS. Sorted completion times are taken to select the job to be inserted in new positions to obtain a better scheduling. The original MRSILS and both modifications are evaluated with well-known benchmark instances. The experiments show that the new modifications produce slightly better results than the original one, especially for a large number of jobs.*

Keywords: Flow shop scheduling problem, iterated local search, permutations, work time costs

1. Introduction

The flow shop scheduling is a classic combinatorial problem established by Johnson in [1]. In this problem, n jobs must be processed on m machines, where each machine is required to process the set of all jobs in the same order. The most common objective in this problem is to minimize the completion time of the last job, or makespan.

Many evolutionary methods have been proposed for makespan minimization. Some of them are based on the modification of continuous evolutionary algorithms. For instance, recent modifications of particle swarm optimization methods have been proposed in [2], [3], [4], and a bee colony algorithm has been presented in [5].

These are only a few examples of the great number of papers devoted to adapt evolutionary and metaheuristic algorithms to minimize the makespan in the flow shop scheduling problem. Very often these papers present complex algorithms in order to construct factible and better solutions. The basic operations in these algorithms, however, are a mixture and/or extensions of job insertions and swaps.

Past research shows that simpler algorithms are able to obtain effective results without need of complex strategies. In particular, the iterated local search method (ILS) has been applied in the minimization of the total flow time for the flow shop scheduling problem [6]. This method has been recently improved using multi-restart points (MRSILS), showing better results than many more intricate and sophisticated metaheuristics [7]. This method is only based in insertions of jobs; without considering swaps.

This paper takes the MRSILS algorithm proposed by Dong et al. in [7] in order to minimize the makespan of the flow shop scheduling problem. Then, two modifications are presented. The first one consists of applying a matrix which defines job permutations to improve a given initial solution. This improved solution is passed to the MRSILS algorithm to obtain a final result. This modified algorithm is called MRSILS_PM.

The second modification is a slightly change in the strategy of the MRSILS, where the job to be inserted is selected sorting the differences of completion job times. For this reason, this modification is called MRSILS_SD.

The three algorithms have been evaluated taking the benchmark instances proposed in [8] and [9], which can be downloaded as the data file *flowshop1* from the web site OR Library.[1] The results show that MRSILS_PM and MRSILS_SD have obtained slightly better results than the original MRSILS in minimizing the execution time and/or the makespan.

2. Flow shop scheduling problem

The flow shop scheduling problem consists of the assignment of a set of n jobs $W = \{J_1, \ldots, J_n\}$, each job J_i is composed of a set of operations $J_i = \{O_{i1}, \ldots, O_{im}\}$ processed by a set of machines $M = \{M_1 \ldots M_m\}$. Every operation O_{ij} has associated a value T_{ij} that represents the processing time of operation O_{ij}; for $1 \leq i \leq n$ and $1 \leq j \leq m$. Each machine can process only one job at a time. It is assumed that the machine sequence is identical for all jobs, and the job sequence is the same for all machines.

[1]http://people.brunel.ac.uk/~mastjjb/jeb/orlib/flowshop info.html. Consulted in January 27th, 2014.

Therefore, a job schedule is represented as a permutation $\pi = \{\pi_1, \ldots, \pi_n\}$ of W. The completion time of the operation O_{ij} is indicated by C_{ij}. The objective of the flow shop scheduling problem is the minimization of the makespan C_{max}, or the completion of the last operation in π.

The model of the flow shop scheduling problem can be defined in the following way:

$$\begin{aligned} C_{\pi_1 1} &= T_{\pi_1 1} \\ C_{\pi_1 j} &= C_{\pi_1 j-1} + T_{\pi_1 j} \\ C_{\pi_i 1} &= C_{\pi_{i-1} 1} + T_{\pi_i 1} \\ C_{\pi_i j} &= \max\{C_{\pi_{i-1} j}, C_{\pi_i j-1}\} + T_{\pi_i j} \end{aligned} \quad (1)$$

where $i \in \{2, \ldots, n\}$ and $j \in \{2, \ldots, m\}$. Thus $C_{max} = C_{nm}$.

3. MRSILS algortihm

The framework of the MRSILS algorithm is very simple [7]. First, a well-known heuristic is applied in order to generate a *good* initial solution. This paper applies the NEH method described in [10], which constructs a solution π in negligible time.

The MRSILS algorithm improves the solution by inserting a selected job into different positions of π to generate a new solution π'. If a better solution is obtained, then π is replaced by π'.

Let π^* be the best solution found by the algorithm. In order to avoid that π^* be trapped in a local optimum, the search space is extended by generating restart solutions from a set Π (or pool) of solutions. This pool keeps the best q solutions calculated so far. If the MRSILS algorithm does not improve the solution after n iterations, the local optimum π is added to Π if it is not already there. If Π has more than q solutions, the worst solution of the pool is deleted.

When Π is not full, the restart solution is generated from π^*; otherwise, a solution π' is randomly chosen from Π. The selected solution is perturbed by inserting one randomly chosen job into another randomly chosen position. In this way, a new restart solution π is defined.

Figure 1 shows the pseudocode of the MRSILS algorithm taking the parameters specified in [7].

```
1: Set q = 20, lim = 1000
2: Set cnt = 0, flag = 0, Π = ∅
3: Generate π* using NEH and set π = π*
4: for i = 1 to lim do
5:     for j = 1 to n do
6:         Find k such that π_k = π*_j
7:         Insert π_k into other n − 1 positions in π,
           let π' be the best solution from the
           n − 1 generated permutations
8:         if π' is better than π then
               Set π = π', cnt = 0
9:         else
               Set cnt = cnt + 1
10:        if π is better than π* then
               Set π* = π, flag = 1
11:        if cnt = n then
12:            if flag = 1 then
                   Set Π = ∅, flag = 0
13:            if π ∉ Π then
                   Add π into Π
14:            if |Π| > q then
                   Delete the worst π' from Π
15:            if |Π| < q then
                   Perturb π* to obtain a new π
16:            else
                   Select π' at random from Π
                   and perturb it to obtain a new π
17:            if π is better than π* then
                   Set π* = π
18:            Set cnt = 0
19: return π*
```

Fig. 1: Pseudocode of MRSILS.

4. Permutation matrix

The MRSILS algorithm is only based on insertions. It can be considered as an algorithm which performs small changes in every solution in order to achieve an improvement. So the MRSILS algorithm is suitable to refine solutions.

The application of swaps and permutations of jobs, however, are useful to improve an initial solution at the beginning of the optimization process. These operations may produce the exploration of a bigger space search in order to obtain a better schedule which can be refined after with the MRSILS.

These swaps and permutations can be specified in a matrix M, where it has as many rows as n (number of jobs) and r columns. Here, r is the number of different solutions to be obtained for the swaps and permutations defined by M.

For a job schedule $\pi = \{\pi_1 \ldots \pi_n\}$, every column j in M defines a new permutation π'. The entry m_{ij} indicates the new place of the job π_i. If $m_{ij} = i$, then $\pi'_i = \pi_i$; that is, the job i in π conserves the same position in π'. Otherwise, for $m_{ij} = z \neq i$, $\pi'_i = \pi_z$. The job at position z in π changes into position i in π'. Therefore, every column j in M is a permutation of $Z_n = \{1, \ldots, n\}$. Since M consists only of permutations of Z_n, it is easily defined and can be applied quickly to π to produce a new schedule.

In order to define the matrix M, we are taking into account four parameters:

$$\begin{aligned} r &\rightarrow \text{number of columns} \\ \alpha &\rightarrow \text{proportion of columns which define large changes} \\ \gamma &\rightarrow \text{proportion of permuted jobs for columns defining large changes} \\ \delta &\rightarrow \text{proportion of permuted jobs for columns defining small changes} \end{aligned} \quad (2)$$

In Eq. 2, r can be defined as a multiple of q, the size of the pool Π in the MRSILS algorithm. In our experiments,

the best performance has been obtained with $r = 10 * q$, $\alpha = 0.55$, $\gamma = 0.6$ and $\delta = 0.1$.

```
1: Set r = q * 10, α = 0.55, γ = 0.6 δ = 0.1
2: Set lim = 1000
3: Generate π* using NEH
4: for i = 1 to lim do
5:     Set π = π* and P = ∅
6:     for j = 1 to r do
7:         Set a column vector c = Z_n^T
8:         Generate a random number s ∈ [0, 1]
9:         if s < α then
               Permute round(γ * n) elements in c
10:        else
               Permute round(δ * n) elements in c
11:        Add the column c to M
12:    for j = 1 to r do
13:        Obtain π' by permuting π according to
           column j in M
14:        Add π' into P
15:    Select the best solution π' from P.
16:    if π' is better than π* then
17:        Set π* = π'
18: return π*
```

Fig. 2: Pseudocode of PM.

```
1: Set q = 20, lim_a = 500, lim_b = 500
2: Set r = q * 10, α = 0.55, γ = 0.6 δ = 0.1
3: Set cnt = 0, flag = 0, Π = ∅
4: Generate π* using algorithm PM with lim_a
   iterations.
5: Improve π* using algorithm MRSILS with
   lim_b iterations.
6: return π*
```

Fig. 3: Pseudocode of MRSILS_PM.

The algorithm that defines and applies the permutation matrix (PM) is presented in Fig. 2. This algorithm takes first an improved solution from the NEH heuristic, where Z_n^T is the transpose of Z_n and round(a) is the closest integer of a real number a. Figure 3 describes the integration of the MRSILS and PM algorithms (MRSILS_PM). This algorithm takes first an improved solution from PM using small and large random permutations. Finally, the best solution is taken from PM and it is refined (or exploited) by the MRSILS algorithm. Notice that half of the iterations are done by PM and the final half are realized by MRSILS.

5. Strategy using sorted completion times

The second modification proposed in this paper is a slight change in the selection of the job to be inserted in the line 6 of the MRSILS algorithm (Fig. 1). In the original algorithm, the job is selected finding the position k in π containing the job j in the current best solution π^*. In the proposed modification, first the completion times of every job in π^* are taken. These completion times define a vector D as follows:

$$D = \{C_{im} : i = 1, \ldots, n\} \tag{3}$$

Equation 3 can be obtained when the makespan is calculated with Eq. 1, therefore, this calculus does not imply an extra numerical work. Then, the differences E between completion times is calculated as follows:

$$E = \{D_{j+1} - D_j : j = 1, \ldots, n-1\} \tag{4}$$

Notice that $|E| = n-1$ and all the values of E are positive because $D_j < D_{j+1}$ for every j in Eq. 4.

Once calculated Eq. 4, E is sorted in a descending way, where I is the index vector associated with E. That is, the first element of I is the index of the highest element in E; the second element of I is the index of the second highest element in E, and so on.

Take the first index μ in I. This index represents the pair of jobs π_μ and $\pi_{\mu+1}$ with the greatest completion time difference. A big difference would indicate that these jobs are not well-ordered in π^*, and a best solution could be obtained if one of the jobs is inserted in a different position.

Consequently, each π_μ and $\pi_{\mu+1}$ are inserted in the other positions, similar to the process done in the MRSILS algorithm. If some insertion improves the current solution, D and E are also updated. This process is iteratively repeated taking the i-th element of I in every step. This modification is called MRSILS_SD. In short, MRSILS_SD takes as criterion the descending sort of completion time differences to select the jobs to be inserted, in order to improve the current schedule. Figure 4 presents the pseudocode of the MRSILS_SD algorithm.

6. Experimental results

The original MRSILS algorithm and the two modifications (MRSILS_PM and MRSILS_SD) have been evaluated taking 29 flow shop scheduling instances commonly used in the literature (car1,. . ., car8; and rec01, rec03,. . ., rec43) [8] [9].

The parameters selected to compare the three algorithms are presented in Table 1. For each case, 50 independent runs have been calculated for every flow shop scheduling instance. Table 2 presents the data of each problem: name (N), size (n, m) indicating n jobs and m machines, and their optimum value (O). In order to compare the results, Table 2

1: Set $q = 20$, $lim = 1000$
2: Set $cnt = 0$, $flag = 0$, $\Pi = \emptyset$
3: Set $D = \emptyset$, $E = \emptyset$, $I = \emptyset$
4: Generate π^* using NEH and set $\pi = \pi^*$
5: **for** $i = 1$ to lim **do**
6: **for** $j = 1$ to $n - 1$ **do**
7: Set $D = \{C_{im} : i = 1, \ldots, n\}$
8: Set $E = \{D_{j+1} - D_j : j = 1, \ldots, n - 1\}$
9: Set I as the index vector associated with E sorted in a descending way
10: Take index $\mu = I_j$ and insert π_μ and $\pi_{\mu+1}$ into the other positions in π, let π' be the best solution from the $2n - 2$ generated permutations
11: Follow algorithm MRSILS from line 8.
12: **return** π^*

Fig. 4: Pseudocode of MRSILS_SD.

Table 1: Parameters to compare the three algorithms.

Parameter	Value
Pool size (q)	20
Number of iterations (lim)	1000
Number of iterations for the PM part (lim_a) in MRSILS_PM	500
Number of iterations for the MRSILS part (lim_b) in MRSILS_PM	500
Number of permutations (r) in PM	200
Proportion of columns (α) in PM defining large changes	0.55
Proportion of permuted jobs (γ) in PM for columns defining large changes	0.6
Proportion of permuted jobs (δ) in PM for columns defining small changes	0.1

also shows the results obtained by each algorithm: minimum (M), mean value (MV) and mean execution time (MET, in seconds). The best results obtained by any of the algorithms are presented in bold. Notice that more than one algorithm may reach the best value in every instance.

Table 4 shows a summary of the results obtained by the algorithms. This table presents the number of times in which each algorithm has obtained the best results.

From Tables 2 and 4, we can see that the three algorithms optimize easily the problems car1 to car8. In these ones, the original MRSILS has the minimum MET. For the problems rec01 to rec41, the three algorithms had similar results to find the optima, and MRSILS and MRSILS_SD have obtained almost the same results in finding the best MV. We can see, however, that MRSILS_SD had a better performance than the other algorithms to find the best M. In particular, MRSILS_SD has outperformed the other algorithms to find a minimum makespan for the greater instances of the rec problems (from rec19 to rec 41). Nevertheless, MRSILS_SD is a more time-consuming algorithm than the other ones. This is notorious as well in Table 2 for the greater instances of the rec problems. The MET parameter of MRSILS_SD is almost 75% greater than the same parameter of MRSILS in the last three problems.

Finally, it is clear that MRSILS_PM is the fastest algorithm for the rec problems. Actually, MRSILS_PM is executed in almost 50% less time that the original MRSILS for the last three problems in Table 2.

For each of the 12 hardest problems (rec19 to rec41), another set of 50 independent runs have been executed for every algorithm, but taking now $lim = 4000$. This implies that $lim_a = lim_b = 2000$ in the MRSILS_PM algorithm. Table 3 displays the results of this experiment and Table 5 presents the result summary. In the second experiment, we can see again that MRSILS_PM is faster than the other two algorithms. All the algorithms improved their results, especially the algorithm MRSILS_PM which improves the performance of MRSILS in the *Optimum* and *Best M* parameters. We can notice that MRSILS_SD outperforms the other algorithms in the second experiment as well, mainly in the last three problems.

7. Concluding remarks

The MRSILS is a very simple algorithm capable to obtain very good results in the optimization of the flow shop scheduling problem. On the contrary to more complex methods of optimization, the MRSILS algorithm is only based in insertions, starting from a single initial solution. In this manuscript, two modifications of the MRSILS have been proposed in order to improve its performance.

The first one (MRSILS_PM) consists of executing random permutations in the initial solution. One part of these permutations take into account a minimal part of the jobs, and the other part permutes a greater number of jobs. The experimental results show that this process is faster in a rough 50% than the original MRSILS algorithm for the bigger problems presented in this paper. Besides, MRSILS_PM obtains similar results to MRSILS for the *Best M* parameter.

The second modification (MRSILS_SD) is a small change in the strategy of the original algorithm. It selects the jobs to be inserted according to the difference of completion times between contiguous works. We suppose that bigger differences may indicate bad placed jobs that need to be reallocated in order to improve the solution. The experimental results show that MRSILS_SD outperforms slightly the other algorithms. This performance is not for free, the sorting operation involved in the step 9 of the algorithm (Fig. 4) increases the execution time in a rough 75% compared to the original MRSILS, for the bigger cases.

Future work could consider the implementation of an algorithm where the first part applies the matrix permutation and the second one implements the strategy of differences between completion times.

Table 2: Results of the three algorithms for $l = 1000$.

N	S (n, m)	O	MRSILS			MRSILS_PM			MRSILS_SD		
			M	MV	MET	M	MV	MET	M	MV	MET
car1	$(11, 5)$	7038	**7038**	**7038**	**2.29**	**7038**	**7038**	3.52	**7038**	**7038**	3.1
car2	$(13, 4)$	7166	**7166**	**7166**	**3.07**	**7166**	**7166**	3.89	**7166**	**7166**	4.29
car3	$(12, 5)$	7312	**7312**	**7312**	**2.65**	**7312**	7315.24	3.71	**7312**	**7312**	3.71
car4	$(14, 4)$	8003	**8003**	**8003**	**3.52**	**8003**	**8003**	4.14	**8003**	**8003**	4.97
car5	$(10, 6)$	7720	**7720**	**7720**	**1.91**	**7720**	**7720**	3.32	**7720**	**7720**	2.55
car6	$(8, 9)$	8505	**8505**	**8505**	**1.31**	**8505**	**8505**	3.06	**8505**	**8505**	1.66
car7	$(7, 7)$	6590	**6590**	**6590**	**1.01**	**6590**	**6590**	2.81	**6590**	**6590**	1.23
car8	$(8, 8)$	8366	**8366**	**8366**	**1.29**	**8366**	**8366**	3.01	**8366**	**8366**	1.64
rec01	$(20, 5)$	1247	**1247**	**1248.32**	7.5	**1247**	1248.64	**6.4**	**1247**	1248.68	11.3
rec03	$(20, 5)$	1109	**1109**	**1109**	7.52	**1109**	**1109**	**6.4**	**1109**	**1109**	11.3
rec05	$(20, 5)$	1242	1245	1245	7.50	**1242**	**1244.88**	**6.45**	**1242**	1244.94	11.43
rec07	$(20, 10)$	1566	**1566**	**1566.04**	8.91	**1566**	1567.44	**7.4**	**1566**	1566.36	13.65
rec09	$(20, 10)$	1537	**1537**	**1537**	8.79	**1537**	**1537**	**7.33**	**1537**	**1537**	13.66
rec11	$(20, 10)$	1431	**1431**	**1431**	8.89	**1431**	**1431**	**7.39**	**1431**	**1431**	13.58
rec13	$(20, 15)$	1930	**1930**	1933.48	10.28	**1930**	1934.7	**8.43**	**1930**	**1932.82**	16.04
rec15	$(20, 15)$	1950	**1950**	**1950.24**	10.24	**1950**	1952.64	**8.36**	**1950**	1950.36	15.98
rec17	$(20, 15)$	1902	**1902**	**1902**	10.07	**1902**	1903.18	**8.35**	**1902**	1902.28	16.07
rec19	$(30, 10)$	2093	**2099**	2104.44	22.65	**2099**	2102.92	**14.75**	**2099**	**2102.84**	37.03
rec21	$(30, 10)$	2017	**2020**	2044.04	27.81	2021	2045.08	**17.29**	**2020**	**2040.78**	38.35
rec23	$(30, 10)$	2011	2014	2019.74	28.1	2014	2020.18	**17.43**	**2013**	**2019.24**	39.03
rec25	$(30, 15)$	2513	**2513**	**2523.3**	32.49	**2513**	2532.24	**20.06**	**2513**	2529.26	47.44
rec27	$(30, 15)$	2373	2378	2389.34	32.62	2378	2392.34	**20.14**	**2377**	**2389.08**	47.84
rec29	$(30, 15)$	2287	**2287**	**2299.96**	32.81	2290	2304.56	**20.21**	**2287**	2301.02	47.35
rec31	$(50, 10)$	3045	**3053**	3056.94	95.38	**3053**	3061	**51.73**	**3053**	**3056.86**	143.27
rec33	$(50, 10)$	3114	**3114**	**3119.06**	93.16	**3114**	3122.04	**50.46**	**3114**	3126.46	138.31
rec35	$(50, 10)$	3277	**3277**	**3277**	94.47	**3277**	**3277**	**51.16**	**3277**	**3277**	140.64
rec37	$(75, 20)$	4951	5007	**5039.18**	395.04	5022	5051.24	**205.15**	**5000**	5042.78	681.12
rec39	$(75, 20)$	5087	5120	**5134.84**	402.62	5124	5142.86	**208.38**	**5114**	5135.9	702.96
rec41	$(75, 20)$	4960	5018	5049.84	400.62	5026	5058.7	**206.99**	**5017**	**5048.12**	696.53

Table 3: Results of the three algorithms for $l = 4000$.

N	S (n, m)	O	MRSILS			MRSILS_PM			MRSILS_SD		
			M	MV	MET	M	MV	MET	M	MV	MET
rec19	$(30, 10)$	2093	2096	2100.76	110	**2093**	**2100.08**	**68.15**	2099	2100.2	153.02
rec21	$(30, 10)$	2017	**2020**	2038.8	111	**2020**	2038.62	**68.44**	**2020**	**2036.18**	153.3
rec23	$(30, 10)$	2011	2013	**2017.08**	111.2	**2011**	2017.86	**68.85**	2013	2018.12	154.95
rec25	$(30, 15)$	2513	**2513**	**2518.46**	127.61	**2513**	2520.48	**78.93**	**2513**	2520.72	187.62
rec27	$(30, 15)$	2373	2376	2382.04	128.96	2376	2384.68	**79.56**	**2373**	**2381.7**	189.75
rec29	$(30, 15)$	2287	**2287**	**2292.42**	129.14	**2287**	2293.88	**79.72**	**2287**	2292.88	190.1
rec31	$(50, 10)$	3045	3053	3053.5	377.22	**3048**	3054.22	**204.99**	**3048**	**3053.22**	560.71
rec33	$(50, 10)$	3114	**3114**	3114.68	371.17	**3114**	3115.96	**201.07**	**3114**	**3114.16**	545.77
rec35	$(50, 10)$	3277	**3277**	**3277**	376.84	**3277**	**3277**	**204.05**	**3277**	**3277**	558.32
rec37	$(75, 20)$	4951	4979	**5014.06**	1558.13	4993	5022.88	**809.45**	**4964**	5016.14	2642.87
rec39	$(75, 20)$	5087	5120	5128.74	1595.96	5120	5130.14	**827.02**	**5114**	**5126.92**	2712.61
rec41	$(75, 20)$	4960	4993	5026.7	1575.54	5000	5033.02	**822.53**	**4991**	**5023.64**	2651.83

Table 4: Best results obtained by the algorithms for $l = 1000$.

Algorithm	Optimum	Best M	Best MV	Best MET
MRSILS	20	23	21	8
MRSILS_PM	20	22	12	21
MRSILS_SD	21	29	19	0

Table 5: Best results obtained by the algorithms for $l = 400$.

Algorithm	Optimum	Best M	Best MV	Best MET
MRSILS	4	5	5	0
MRSILS_PM	6	8	2	12
MRSILS_SD	5	10	7	0

Other possibility is to investigate other criteria in order to select the jobs to be inserted in the different places to improve the current schedule. For instances, the idle times between contiguous jobs or machines.

References

[1] S. R. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Nav. Res. Log.*, vol. 1, no. 1, pp. 61–68, March 1954.

[2] C. Zhang and J. Sun, "An alternate two phases particle swarm optimization algorithm for flow shop scheduling problem," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 5162–5167, April 2009.

[3] H. Liu, L. Gao, and Q. Pan, "A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem," *Expert Syst. Appl.*, vol. 38, no. 4, pp. 4348–4360, April 2011.

[4] Z. Lian, X. Gu, and B. Jiao, "A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan," *Chaos Soliton. Fact.*, vol. 35, no. 5, pp. 851–861, March 2008.

[5] M. F. Tasgetiren, Q. Pan, P. Suganthan, and A. H.-L. Chen, "A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops," *Inform. Sciences*, vol. 181, no. 16, pp. 3459–

3475, August 2011.
[6] X. Dong, H. Huang, and P. Chen, "An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion," *Comput. Oper. Res.*, vol. 36, no. 5, pp. 1664–1669, May 2009.
[7] X. Dong, P. Chen, H. Huang, and M. Nowak, "A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time," *Comput. Oper. Res.*, vol. 40, no. 2, pp. 627–632, February 2013.
[8] J. Carlier, "Ordonnancements a contraintes disjonctives," *R. A. I. R. O. Oper. Res.*, vol. 12, no. 4, pp. 333–350, February 1978.
[9] C. R. Reeves, "A genetic algorithm for flowshop sequencing," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 5–13, January 1995.
[10] M. Nawaz, E. E. E. Jr., and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.