THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 13, Number 1/2012, pp. 71–79

ON SOLVING FLOWSHOP SCHEDULING PROBLEMS

Mircea ANCĂU

Technical University of Cluj-Napoca
E-mail: mircea.ancau@tcm.utcluj.ro

This paper proposes two variants of heuristic algorithms to solve the classic permutation flowshop scheduling problem. Both algorithms are simple and very efficient. The first algorithm is a constructive heuristic which builds the optimal schedule of jobs on the basis of a selective-greedy process. To escape the trap of local optimum points, for the second heuristic algorithm a stochastic feature is added. The numerical results show the good position of the proposed algorithms within the top known as best heuristic algorithms in the field.

Key words: heuristic, greedy, flowshop scheduling problem, optimization.

1. INTRODUCTION

By the concept of Group Technology (GT) the industrial parts are categorized according to different criteria, on part families (i.e. spindle family, gear family etc.). The most important feature of parts which belong to the same family is the technological itinerary, which is the same, no matter their size or specific geometry feature. The technological itinerary dictates the order in which the parts go by one machine to another, from rough material to final product. Optimal manufacturing means best quality and minimum price, while a minimum price involves always a minimum manufacturing time, essentially nowadays.

To generalize the above problem, we have to consider a set of n parts from the same family, which must be manufactured on m different machines. The main objective is to find the optimal order of parts manufacturing so that the total manufacturing time, denoted C_{max} (total completion time or makespan) is minimum. The values of the manufacturing time on each machine, denoted t_{ij} (i = 1,2, …, n, j = 1,2, …, m), are previously known, constant and positive. They include also all the necessary auxiliary times involved in the technological process. This problem must be completed with a series of assumptions: all jobs are independent and available at the time t = 0, all machines are permanently available, each job can be manufactured at a specific moment on a single machine, each machine can do a single operation at a specific time, the machine cannot be interrupted once it started an operation, the set-up and auxiliary times are included in the manufacturing times, if a machine is not available (being engaged in another operation) the following jobs are assigned to a waiting queue etc. A comprehensive list of these assumptions, grouped on categories, can be found in [1]. In the literature this problem is called Flowshop Scheduling Problem (FSP) and it has an essential role in the field of combinatorial optimization problems. If the operations sequence, or the technological itinerary is the same for each job on the m given machines, then the problem is called Permutation Flowshop Scheduling Problem (PFSP).

Generally, the methods to solve PFSP are grouped on two categories: constructive heuristics and improvement heuristics. The first category, called constructive heuristics, builds the ordered sequence of jobs, based on some specific rules or decisions. The second category of improvement heuristics, by specific rules, makes better solutions, starting from an existing feasible solution usually found by one of the previous techniques.

The algorithm of Johnson [2] is a classic method which solves to optimum the problem of ordering n jobs on two machines, in a polynomial time. If there are n jobs on three machines, then the problems become NP-complete (i.e. cannot be solved optimally in polynomial time) and the Johnson's algorithm can be applied only for some few particular cases that obey some primary conditions. Besides the classic algorithm

of Johnson which has much limitation, there were designed several heuristic algorithms which attempt to solve the general problem of finding the optimal order of manufacturing n jobs on m machines.

One of them is CDS algorithm of Campbell *et al.*, which divides all m machines into two groups which are considered as two virtual machines [3]. Therefore, the problem is solved by applying Johnson's algorithm. Subsequently, the best solution is chosen among (m-1) processing sequences. The HFC heuristic of Koulamas [4] uses in a first phase the algorithm of Johnson, later on improving the previously feasible solution. Many other heuristic methods such as those of Palmer [5], Gupta [6], Hundal and Rajgopal [7] make use of a slope index assigned to each job. These heuristic algorithms sort the list of jobs using that weight as a sort key to create a feasible schedule.

The algorithm RA (Rapid Access) of Dannenbring [8] is a combination between previous methods based on Johnson and Palmer algorithms. Two virtual machines are defined, as in CDS heuristic method. Then there are created two assignment systems of weights, one for each virtual machine. Finally, the flowshop scheduling problem is solved by applying Johnson's algorithm.

The NEH heuristic algorithm made by Nawaz *et al.* [9] is recognized by Taillard [10] as one of the efficient heuristic method in this field. NEH is based neither on Johnson's algorithm, nor on assigning weights techniques. First of all, the algorithm calculates the total completion time of each job taken by alone, on all machines. Then the jobs are sorted in descending order of these individual values. The total completion time for the first two jobs is calculated. There is taken the best one from two possible variant. It follows the third job from the ordered list. For this job there are three possible variants to place it in the manufacturing sequence. As in the precedent case, the variant with minimum completion time is selected, and so on, until all jobs are placed in the manufacturing sequence. In this way, the manufacturing sequence is generated, by placing each job J_k ($2 < k \leq n$) in the most favorable position in the sequence J_1, J_2, …, $J_{k\text{-}1}$ already formed. As a consequence, there are necessary $n \cdot (n-1)/2 - 1$ evaluation sequences to get the final result. Many other heuristic methods as Framinan *et al.* [11] are based on NEH and proposes different starting sequences.

The SPIRIT algorithm of Widmer and Hertz [12] is also a constructive heuristic. This algorithm is based on analogy with traveling salesman problem. Based on a formula proposed by the authors, productive time of related jobs becomes the distance between them. Thus, job sequencing problem turns into a path down of one job to another, depending on the distance between them.

Various constructive heuristic algorithms such as Chandrasekharan [13], Yuvraj and Chandrasekharan [14], use insertion techniques such as ant colony principle, to design the optimal sequence of jobs.

Unlike constructive heuristic algorithms, the improvement heuristic algorithms begins from a manufacturing ordered sequence already made and try to improve it by applying different procedures. The techniques are usually based on the simple procedure of permutation of adjacent jobs as in Dannenbring [8], with variants *Rapid Access with Close Order Search* (RACS) and *Rapid Access with Extensive Search* (RAES), both starting from an initial schedule made by RA. Other techniques uses as optimization criterion some intermediary processing times as Ho and Chang [15]. In Agarwal *et al.* [16] three very known constructive heuristics are used to build a good starting solution. Then all three heuristics are modified using a weight parameter which perturb the starting solution in order to get a better one. The weight parameter is permanently adjusted based on an adaptive-learning strategy which involve reinforcement and backtracking, as in neural networks. In Chakraborty and Laha [17], the same NEH heuristic has been modified in order to get better performances.

It is worth to mention many other heuristics whose techniques are used to solve other combinatorial optimization problems and are applied in this case of flowshop scheduling. Within the most known there are the techniques based on Simulated Annealing which can be found in Osman and Potts [18], Widmer and Hertz [12], Taillard [10], Ogbu and Smith [19], Ishibuchi *et al.* [20], the Tabu Search techniques in Moccellin [21], Nowicki and Smutnicki [22] or techniques based on genetic algorithms in Murata *et al.* [23], Reeves and Yamada [24], Caraffa *et al.* [25], Ponnambalam *et al.* [26] etc. A comprehensive study on the performances of these methods may be found in Ruiz and Morato [27].

This paper is structured on four paragraphs. In the first paragraph, an introduction of the flowshop scheduling problem in the context of combinatorial optimization problems is made. It continues with a review of most relevant results from the literature. The second paragraph presents the general principle of the proposed algorithms and shows their outline, to ease the algorithms encoding on a computer. The third

paragraph shows a detailed presentation of the numerical results, according to other top results. In the fourth paragraph the main numerical results are commented and the main conclusion are summarized. The paper ends with a comprehensive reference list.

2. THE HEURISTIC ALGORITHMS

Although intensely studied, PFSP continue to attract interest from researchers. The main reason is that methods for solving PFSP is a starting platform to develop other methods to solve new problems that derive from PFSP. This paper proposes two variants of heuristic algorithms to solve the classic permutation flowshop scheduling problem. The first proposed algorithm is a constructive heuristic, in which each job is placed in the optimal schedule based on a greedy-type selections. The second algorithm, change the construction of optimal schedule in a stochastic manner.

2.1. The constructive greedy heuristic (CG)

In the constructive heuristic algorithm, both the job and its position in the optimal schedule are designed in a greedy way. The algorithm uses two lists, namely 'job list' and 'optimal schedule'. At first, 'job list' has n elements (J_1, J_2, ..., J_n), while the 'optimal schedule' is empty (Fig. 1). From the n elements in 'job list' will be selected an ordered pair of jobs, which satisfies the criterion 'minimum completion time'. This choice will require $n(n–1)$ checks. The two jobs that satisfy the criterion of optimization 'minimum completion time' will be removed from the 'job list' and will pass to the list of 'optimal schedule'. In the next step, from the $(n–2)$ remaining elements in 'job list'\', will get that job and its position with respect to the elements in 'optimal schedule' so that the completion time is minimum. For this new selection will be required $3(n–2)$ checks. After that, keeping the greedy principle of choice and positioning of jobs in the 'optimal schedule', from the $(n–3)$ the remaining jobs in 'job list', will choose that job and its position in the 'optimal schedule' so that manufacturing time is minimized. For this new selection will be required $4(n–3)$ checks and so on. To insert the last job from the 'job list' in the 'optimal schedule', n verifications will be made.

```
set 'job list' = {J1, J2, …, Jn};
        'optimal schedule' is empty;
find the ordered pair of jobs (Jπi,Jπi+1) from 'job list', which
    corresponds at a minimum makespan;
set k = 2;
while ('job list' is not empty)
{
        from 'job list' select the best candidate and its relative
    position in the 'optimal schedule';
  set k = k + 1;
}
print the 'optimal schedule' and the makespan;
```

Fig. 1 – The pseudocode of constructive greedy heuristic algorithm.

Apart from manufacturing time calculation, the selection of a job J_k from 'job list' and determining its position in the 'optimal schedule' in a greedy way, requires $k(k–n+1)$ checks and has a complexity of de $\Theta(n^3)$, while NEH heuristic has a complexity of $\Theta(n^2)$ [17]. The NEH heuristic algorithm arranges the list of jobs in descending order of total manufacturing time of each job. Thus, the introduction in the proper place in 'optimal schedule' of each job is done in the order given by the ordered list in advance. At the algorithm proposed in this research, both job and its appropriate position in the 'optimal schedule' are determined based on the greedy principle, leading to a degree of complexity of the calculations a bit higher than to NEH. It is worth noting that the algorithm leads to determine a local optimal point. This is supported by the fact that a local search procedure (i.e. k-opt), does not lead to significant improvements.

2.2. The stochastic greedy heuristic (SG)

To escape the trap of local optimal point, we generate a random permutation of *n* jobs $\{J\pi_1, J\pi_2, \ldots, J\pi_n\}$ constituting the *job list*. From this list choose the first pair of jobs $(J\pi_1, J\pi_2)$ as start jobs and find their optimal order so that makespan criterion is minimized. These two jobs are deleted from the 'job list' and then are inserted in the list of 'optimal schedule'. The remaining jobs in 'job list' are introduced one by one in 'optimal schedule' list, in the proper position done by greedy rule, so that the increase of the makespan criterion is minimum. From the random permutation, the next pair of jobs $(J\pi_2, J\pi_3)$ is chosen as start pair of jobs, and the process of building the 'optimal schedule' is repeated as described above. The selection of the start pair of jobs continue until the last pair of jobs from the random permutation. After that, we can generate a new random permutation and so on. In this case, as a stopping criterion, the maximum number of steps s_{max}, can be selected. By the random way of introducing the jobs in 'optimal schedule' as well as its iterative aspect, the new algorithm pass from constructive into a stochastic greedy heuristic. Taking into account that the insertion order of jobs from the 'job list' into the 'optimal schedule' is done by the random permutation (see the statement "*select the best candidate and its relative position*" in the constructive greedy heuristic with respect to "*select the next candidate and its relative position*" in the stochastic greedy heuristic), the step corresponding to construction of 'optimal schedule" list has a complexity of $\Theta(n^2)$. The main principle of the algorithm, in pseudocode, can be seen in Fig. 2.

```
Set s = 0;
while (s ≤ s_max)
{
    generate a random permutation of jobs {Jπ_1, Jπ_2, …, Jπ_n};
    for (i=1; i≤n; i++)
    {
        select (Jπ_i, Jπ_i+1) as start jobs; find their optimal ordered sequence;
        for (k=3; k≤n; k++)
    {
        from the remaining (n-k) jobs of the random permutation,
        select the next candidate and its relative position to be appended
        in the ordered partial sequence. The job's relative position
        which corresponds to the minimum increasing of the makespan
        for (k+1) jobs will be selected.

        if ('job list' is empty)
                print (the n-job's 'optimal schedule' and makespan);
    }
        }
        s = s + 1;
}
```

Fig. 2 – The pseudocode of stochastic greedy heuristic algorithm.

3. NUMERICAL RESULTS

Both variants of the algorithm were tested on four groups of benchmark problems from Prof. E. Taillard's page (http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html) and OR Library (http://people.brunel.ac.uk/~mastjjb/jeb/info.html). Tables 1–6 show the computational results of these test instances. The results corresponding to NEH heuristic were taken from Agarwal *et al.* [16] only for performances evaluation. Each instance was solved by both selective greedy (CG) and selective greedy with stochastic start (SG) algorithms. The *Gap*, in percents, which refer to as the difference between the *Makespan* and *Upper Bound*, was calculated by:

$$\text{Gap} = \frac{\text{Makespan} - \text{UB}}{\text{UB}} \cdot 100\% . \quad (1)$$

Present researches demonstrated the efficiency of NEH heuristic, according to most recognized heuristics seen in Ruiz and Morato [27], Agarwal *et al.* [16], Gupta and Stafford [1], Chakraborty and Laha [17]. This is another reason why, in this paper, the performances of CG and SG algorithms are related to those of NEH algorithm. If we analyze the results which correspond to the first set of Taillard's 10 instances of size 20×5 (20 jobs on 5 machines) from Table 1, we can see that the results of CG are very near from NEH's. There are instances to which CG and NEH gave the same result (ta001), others to which NEH offer a slight better solution than CG (ta002, ta004, ta008) or others to which CG offer a slight better solution than NEH (ta003, ta005, ta006, ta007, ta009, ta010). For CG, the average CPU time was 0.062 seconds on an Intel Pentium processor at 1.6 GHz. The results given by SG variant are far better than those of CG or NEH. At two instances SG found the upper bound and the average gap is 0.851% while for CG and NEH the average gap is 3.365% and 3.246% respectively (see Table 3). In the case of SG at each instance the average CPU time was 19.5 s.

Table 1

Makespans and gaps of Taillard's 20×5 benchmark problems

Problem instance	Upper bound	Makespan			Gap [%]		
		NEH	CG	SG	NEH	CG	SG
ta001-20x5	1278	1286	1286	1278	0.62	0.62	**0.00**
ta002-20x5	1359	1365	1367	1366	0.44	0.58	0.51
ta003-20x5	1081	1159	1141	1097	7.21	5.55	1.48
ta004-20x5	1293	1325	1358	1306	2.47	5.02	1.00
ta005-20x5	1236	1305	1301	1244	5.58	5.25	0.64
ta006-20x5	1195	1228	1224	1210	2.76	2.42	1.25
ta007-20x5	1239	1278	1264	1251	3.14	2.01	0.96
ta008-20x5	1206	1223	1268	1206	1.41	5.14	**0.00**
ta009-20x5	1230	1291	1277	1253	4.95	3.82	1.86
ta010-20x5	1108	1151	1144	1117	3.88	3.24	0.81

Table 2

Makespans and gaps of Taillard's 50×5 benchmark problems

Problem instance	Upper bound	Makespan			Gap [%]		
		NEH	CG	SG	NEH	CG	SG
ta031-50x5	2724	2733	2761	2724	0.33	5.11	**0.00**
ta032-50x5	2834	2843	2889	2848	0.31	1.94	0.49
ta033-50x5	2621	2640	2674	2622	0.72	2.02	0.04
ta034-50x5	2751	2782	2782	2782	1.12	1.12	1.12
ta035-50x5	2863	2868	2908	2863	0.17	1.57	**0.00**
ta036-50x5	2829	2850	2863	2840	0.74	1.20	0.38
ta037-50x5	2725	2758	2781	2732	1.21	2.05	0.25
ta038-50x5	2683	2721	2780	2701	1.41	3.61	0.67
ta039-50x5	2552	2576	2595	2562	0.94	1.68	0.39
ta040-50x5	2782	2790	2787	2784	0.28	0.17	0.07

At instances of size 50×5 the NEH results are better than those of CG with an average of 1.3%, at size 50×10 with 0.92% and at size 50×20 with 0.28%. The results of SG are still better than those of NEH, with an average gap of 0.341% (see Table 2). At two instances of this size (ta031, ta035) SG reached the upper bound. For the instances size of 50×5 the average CPU time was 3.12 seconds for CG and 120.2 seconds for SG respectively. For big size instances (100×5, 200×10) the relative situation remain the same as before. The solutions found by CG are quite similar to those of NEH, slight in favor of NEH, as we can see in Tables 3

and 4. The average gap from the upper bound is 0.476% at NEH compared with 0.945% at CG for the instance size of 100×5 and 1.282% at NEH compared with 1.55% at CG for the instance size of 200x10. SG offer to these instances higher quality solutions, with an average gap of 0.295% for the size instance of 100×5 and 0.862% for 200×10.

Table 3

Makespans and gaps of Taillard's 100×5 benchmark problems

Problem instance	Upper bound	Makespan			Gap [%]		
		NEH	CG	SG	NEH	CG	SG
ta061-100x5	5493	5519	5527	5495	0.47	0.61	0.04
ta062-100x5	5268	5348	5304	5275	1.51	0.68	0.13
ta063-100x5	5175	5219	5250	5206	0.85	1.44	0.59
ta064-100x5	5014	5023	5052	5021	0.17	0.75	0.13
ta065-100x5	5250	5266	5345	5262	0.30	1.81	0.22
ta066-100x5	5135	5139	5200	5156	0.07	1.26	0.41
ta067-100x5	5246	5259	5287	5273	0.24	0.78	0.51
ta068-100x5	5106	5120	5144	5108	0.27	0.74	0.04
ta069-100x5	5454	5489	5485	5467	0.64	0.56	0.23
ta070-100x5	5328	5341	5372	5363	0.24	0.82	0.65

Table 4

Makespans and gaps of Taillard's 200×10 benchmark problems

Problem instance	Upper bound	Makespan			Gap [%]		
		NEH	CG	SG	NEH	CG	SG
ta091-200x10	10868	10942	11040	10940	0.68	1.58	0.66
ta092-200x10	10494	10716	10625	10600	2.11	1.24	1.01
ta093-200x10	10922	11025	11107	11043	0.94	1.69	1.11
ta094-200x10	10889	11057	11069	10974	1.54	1.65	0.78
ta095-200x10	10524	10645	10692	10603	1.14	1.59	0.75

Concerning Carlier benchmark problems from Table 5, we can see that the results situation is almost the same as for Taillard's benchmark problems. So, at three of Carlier's instances (Car6-8×9, Car7-7×7, Car8-8×8), NEH offer better solutions than CG, while for all the other of them CG is better than NEH. At three of Carlier instances CG reached the upper bound. Yet again SG results are far better than NEH's, at all seven of them reaching the upper bound.

Table 5

Makespans and gaps for some of Carlier's benchmark problems

Problem instance	Upper bound	Makespan			Gap [%]		
		NEH	CG	SG	NEH	CG	SG
Carlier							
Car1-11x5	7038	7038	7038	7038	0.00	**0.00**	**0.00**
Car2-13x4	7166	7376	7166	7166	2.93	**0.00**	**0.00**
Car3-12x5	7312	7399	7366	7312	1.18	0.73	**0.00**
Car4-14x4	8003	8003	8003	8003	0.00	**0.00**	**0.00**
Car6-8x9	8505	8773	8776	8505	3.15	3.18	**0.00**
Car7-7x7	6590	6590	6760	6590	0.00	2.57	**0.00**
Car8-8x8	8366	8564	8732	8366	2.36	4.37	**0.00**

For the two test instances of Heller (see Table 6) it was considered as upper bound the results found by Agarval *et al.* [16] with NEH_ALA algorithm (a modified version of NEH). For the instance of size 20×10, CG has a gap of 5.88%, NEH's gap is 3.67% and the gap of SG is 0.73%. But for 100×10 Heller instance, SG found a new upper bound, better with 0.19% than that found by NEH-ALA.

Table 6

Makespans and gaps for some of Heller's and Reeves' benchmark problems.

Problem instance	Upper bound	Makespan			Gap [%]		
		NEH	CG	SG	NEH	CG	SG
Heller							
Heller-20x10	136	141	144	137	3.67	5.88	0.73
Heller-100x10	516	518	525	515	0.38	1.74	**- 0.19**
Reeves							
ReC13-20x15	1930	2002	2019	1966	3.73	4.61	1.86
ReC15-20x15	1950	2013	2011	1965	3.23	3.13	0.77
ReC17-20x15	1902	2019	2021	1955	6.15	6.25	2.78
ReC07-20x10	1566	1626	1619	1584	3.83	3.38	1.15
ReC10-20x10	1537	1594	1571	1540	3.70	2.21	0.19
ReC12-20x10	1431	1550	1505	1458	8.31	5.17	1.88
ReC25-30x15	2513	2644	2657	2568	5.21	5.73	2.18
ReC29-30x15	2287	2391	2419	2357	4.54	5.77	3.06
ReC31-50x10	3045	3173	3198	3129	4.20	5.02	2.75
ReC33-50x10	3114	3241	3221	3143	4.07	3.43	0.93

Table 7

Average gaps for Taillard and Reeves benchmark problems.

Instance size	Number of instances	Average Gap [%]		
		NEH	CG	SG
Taillard				
20 x 5	10	3.246	3.365	0.851
20 x 10	10	4.587	5.643	2.240
20 x 20	10	3.721	5.460	1.963
50 x 5	10	0.723	2.047	0.341
50 x 10	10	4.567	5.485	2.355
100 x 5	10	0.476	0.945	0.295
200 x 10	5	1.282	1.550	0.862
Reeves				
20 x 5	6	3.23	4.285	0.318
20 x 10	6	5.162	5.878	1.165
20 x 15	3	4.370	4.663	1.803
30 x 10	3	5.733	8.093	2.410
30 x 15	3	5.003	6.740	2.490
50 x 10	3	3.120	3.780	1.336
75 x 20	3	5.980	7.720	4.876

The same relative situation between NEG, CG and SG remains in the case of Reeves test instances. So, the results of CG are very near to those of NEH. In five test problems from ten, the results of CG algorithm are beter that those of NEH. In the case of ReC10-20×10 and ReC12-20×10 the difference between CG and NEH results is quite big in favour of CG. If we take a close look at the results obtained by SG algorithm, all the results are far better than those of NEH, and this fact is owing to the stochastic search. Looking at the average gaps in Table 7, from Taillard's instances of size 20×5, 20×10 and 20×20, one may conclude that for CG and SG algorithms, the average gap increase as the number of machines increase.

But this conclusion does not stand while the average gap for the size 20×20 is lower than that of size 20×10. Nevertheless, more important thing to remark is the constancy of the results quality at the same size instance.

4. CONCLUSION

This paper presents two original heuristic algorithms for solving permutation flowshop scheduling problem. First algorithm is a constructive heuristic based on a greedy selection, while the second algorithm is a modified version of the previous, based on iterative stochastic start. Both variants are elegant, very simple

to implement and offers very good quality solutions. The numerical results prove a high quality solutions for CG variant of the heuristic algorithm, as good as those of NEH. It must be remarked the fine results offered by SG variant of the proposed heuristic algorithm, far better than those of CG or NEH. Although, due to its relatively high computational degree, the proposed heuristic is slower than NEH. For very large size problems (more than 200 jobs on more than 20 machines) good results require from tens of minutes to even several hours CPU time, which might be costly. This is a main reason to recommend this heuristic mainly for medium and large size problems.

ACKNOWLEDGEMENTS

This work was supported by CNCSIS –UEFISCSU, project number 579/2008, PNII – IDEI, code ID_188.

REFERENCES

1. Gupta JND, Stafford Jr. EF., *Flowshop scheduling research after five decades*, European Journal of Operational Research, **169**, pp. 699–711, 2006.
2. Johnson SN., *Optimal two-and three-stage production schedules with setup times included*, Naval Research Logistics Quarterly, **1**, pp. 61–68, 1954.
3. Campbell HG, Dudek RA, Smith ML., *A heuristic algorithm for the n job, m machine sequencing problem*, Management Science, **16**, *10*, pp. B630–B637, 1970.
4. Koulamas C., *A new constructive heuristic for the flowshop scheduling problem*, European Journal of Operational Research Society, **105**, pp. 66–71, 1988.
5. Palmer D., *Sequencing jobs through a multi-stage process in the minimum total time - a quick method of obtaining a near optimum*, Operational Research Quarterly, **16**, *1*, pp. 101–107, 1965.
6. Gupta JND., *A functional heuristic algorithms for the flowshop scheduling problem*, Operational Research Quarterly, **22**, 1, pp. 39–47, 1971.
7. Hundal TS, Rajgopal J., *An extension of Palmer's heuristic for the flow shop scheduling problem*, International Journal of Production Research, **26**, *6*, pp. 1119–1124, 1988.
8. Dannenbring DG., *An evaluation of flow shop sequencing heuristics*, Management Science, **23**, *11*, pp. 1174–1182, 1977.
9. Nawaz M, Enscore Jr. EE, Ham I., *A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem*, OMEGA, The International Journal of Management Science, **11**, *1*, pp. 91–95, 1983.
10. Taillard E., *Some efficient heuristic methods for the flow-shop sequencing problem*, European Journal of Operational Research, **47**, pp. 67–74, 1990.
11. Framinan JM, Leisten R, Rajendran C., *Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem*, International Journal of Production Research, **41**, *1*, pp. 121–148, 2003.
12. Widmer M, Hertz A., *A new heuristic method for the flow shop sequencing problem*, European Journal of Operational Research, **41**, pp. 186–193, 1989.
13. Chandrasekharan R., *Heuristic algorithm for scheduling in a flowshop to minimize total flowtime*, International Journal of Production Economics, **29**, *1*, pp. 65–73, 1993.
14. Yuvraj G, Chandrasekharan R., *An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops*, International Journal of Production Economics, **101**, *2*, pp. 259–272, 2006.
15. Ho JC, Chang YL., *A new heuristic for the n-job, m-machine flow-shop problem*, European Journal of Operational Research, **52**, pp. 194–202, 1991.
16. Agarwal A, Colak S, Eryarsoy E., *Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach*, European Journal of Operational Research, **169**, pp. 801–815, 2006.
17. Chakraborty UK, Laha D., *An improved heuristic for permutation flowshop scheduling*. International Journal of Information and Communication Technology, **1**, *1*, pp. 89–97, 2007.
18. Osman I, Potts C., *Simulated annealing for the permutation flow-shop scheduling*, OMEGA, The International Journal of Management Science, **17**, *6*, pp. 551–557, 1989.
19. Ogbu F, Smith D., *The application of the simulated annealing algorithm to the solution of the n/m/C_{max} flowshop problem*, Computers and Operations Research, **17**, *3*, pp. 243–253, 1990.
20. Ishibuchi H, Misaki S, Tanaka H., *Modified simulated annealing algorithms for the flow shop sequencing problem*. European Journal of Operational Research, **81**, pp. 388–398, 1995.
21. Moccellin JV., *A new heuristic method for the permutation flow shop scheduling problem*. Journal of Operational Research Society, **46**, pp. 883–886, 1995.

22. Nowicki E, Smutnicki C., *A fast tabu search algorithm for the permutation flow-shop problem*, European Journal of Operational Research, **91**, pp. 160–175, 1996.
23. Murata T, Ishibuchi H, Tanaka H., *Genetic algorithms for flowshop scheduling problem*, Computers and Industrial Engineering, **30**, *4*, pp. 1061–1071, 1996.
24. Reeves C, Yamada T., *Genetic algorithms, path relinking and the flowshop sequencing problem*, Evolutionary Computation, **6**, *1*, pp. 45–60, 1998.
25. Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C., *Minimizing makespan in a blocking flowshop using genetic algorithms*. International Journal of Production Economics, **70**, *2*, pp. 101–115, 2001.
26. Ponnambalam SG, Aravindan P, Chandrasekaran S., *Constructive and improvement flow shop scheduling heuristics: An extensive evaluation*, Production Planning and Control, **12**, *4*, pp. 335–344, 2001.
27. Ruiz R, Morato C., *A comprehensive review and evaluation of permutation flowshop heuristics*, European Journal of Operational Research, **165**, pp. 479–494, 2005.

Received October 22, 2011