Scientific Research and Essays Vol. 6(14), pp. 3094-3100, 18 July, 2011
Available online at http://www.academicjournals.org/SRE
DOI: 10.5897/SRE10.1014
ISSN 1992-2248 ©2011 Academic Journals

Full Length Research Paper

An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems

Jian Gao and Rong Chen*

College of Information Science and Technology, Dalian Maritime University, Dalian, Liaoling, Province, 116026, China.

Accepted 10 June, 2011

Distributed Permutation Flowshop Scheduling Problem (DPFSP) is a newly proposed scheduling problem with a strong engineering background. Whereas there is a body of work on scheduling problems in the past decades, the literature on DPFSPs is scant and in its infancy. Motivated by the good performances of some heuristics in a very recent work, we propose a constructive heuristic algorithm enhanced through a novel dispatching rule to deal with the DPFSP. Given multiple factories in a DPFSP, our dispatching rule will insert a group of jobs to the factories at one time instead of inserting one job at one time like the original rules. The time complexity of the proposed heuristic algorithm is the same as that of the NEH with the original rule. To validate the proposed heuristic, intensive benchmark experiments are carried out on the large problem instances, and the results show that the proposed algorithm outperforms the existing heuristics in terms of tradeoff between solution quality and running time.

Key words: Distributed scheduling problem, flowshop, heuristic, branch and bound.

INTRODUCTION

In the past decades, the Permutation Flowshop Scheduling Problem (PFSP) has been a central and well studied scheduling problem that has a strong engineering background in manufacturing and chemical industries. Most of the literature deals with the PFSP whereby each job is processed by the same set of machines in the same order, and all jobs are assumed to be assigned in the unique factory (Yin and Li., 2011; Fondrevelle et al., 2006; Rajendran and Ziegler, 2004; Li et al., 2010; Chakraborty and Laha, 2007). However, such an assumption is not suitable for modeling some real-world scheduling problems nowadays in that more and more companies handle large volume of manufacturing in distributed environments in order to achieve better product quality, lower production cost and lower management risks (Jia et al., 2003).

With the great shift in manufacturing from the traditional single-factory mode to the nowadays multi-factory mode, more factories are built to set up such distributed manufacturing environments (Chan et al., 2005), hence Naderi and Ruiz (2010) have recently phrased the distributed permutation flowshop scheduling problem (DPFSP) which generalizes the classical PFSP with a set of factories whereby each job is allowed to be processed in one factory with an aim to minimizing the maximum completion time among all the factories.

Because of the computational complexity of the PFSP (Garey et al., 1976), it has been extensively studied by many researchers. In particular, a large amount of the literature deals with the PFSP with makespan criterion. Algorithms for solving PFSP can be categorized into exact algorithms and heuristics approaches. Some exact algorithms are mathematical programming (Yin et al., 2010) methods, branch and bound (B&B) algorithms (Land and Doig, 1960) and backtracking approaches. Some early constructive heuristics include: the index heuristic proposed by Palmer (1965), the CDS method by Campbell et al. (1970) and the NEH algorithm by Nawaz et al. (1983). Moreover, metaheuristic algorithms for the

*Corresponding author. E-mail: rchen@dl.cn. Tel: +86 +411 84723669; Fax: +86 +411 84723669.

Abbreviations: DPFSP, Distributed permutation flowshop scheduling problem; **PFSP, permutation** flowshop scheduling problem.

PFSP have been investigated to obtain better solution recently, such as differential evolution algorithm (Onwubolu and Davendra, 2006), genetic algorithm (Reeves, 1995) and particle swarm optimization (Erdogmus, 2010), but most of those algorithms start from the solution produced by the constructive heuristic algorithms (e.g. NEH).

Whereas there is a body of work on the PFSP in the past decades, the literature on DPFSPs is scant and in its infancy. Such algorithms for solving DPFSP are mainly discussed in the work by Naderi and Ruiz (2010). They solve the DPFSP with mixed integer linear programming and heuristics; some mixed integer linear programming models have been investigated and implemented on highly optimized CPLEX11.1 package. In evaluating their experiment performances, only small instances (16 jobs and 4 factories) are solved by the mixed integer linear programming methods. To solve large instances, they use heuristics approaches derived from existing heuristics for the well known PFSP. Those heuristics are extended with two alternative rules for job assignments: one locates the job to the factory with the lowest partial makespan; the other one tries all possible positions of all the factories for a job and places the job in the position that has the lowest partial makespan after including the job. Moreover, they also present a local search approach for the DPFSP, called variable neighborhood descent. The approach starts from the solution of NEH heuristic method, and moves jobs in each factory or between factories with the aim at minimizing the maximal makespan of the factories. Experiments on large instances indicate that the local search methods can get better solutions than the constructive heuristic algorithms, while the CPU times consumed by the local search methods are quite longer than those by heuristic algorithms.

Motivated by the good performances of some heuristics in a very recent work, we propose a new constructive heuristic enhanced through a new dispatching rule for job assignments to compute the optimal solutions to DPFSPs. Given multiple factories in a DPFSP, our job assignment will insert a group of jobs to the factories at one time instead of inserting one job at one time like the original rules. Also we use the sort method proposed by Dong et al. (2008), namely jobs are ordered by the sum of average processing times and standard deviations of the processing times of the jobs. Regarding the time complexity, the proposed heuristic is same as NEH with the original rule. By carefully analyzing experimental results on the benchmark instances, we find that the proposed heuristic obtains better solutions than the existing NEH-based heuristic algorithms, though it's total CPU run-times are slightly longer than those of the existing algorithms. Statistical analyses also show that our heuristic is significantly better than the original NEH for DPFSP.

The remainder of this paper is organized as follows. DPFSP is formally defined, existing heuristics for DPFSP, which is NEH method are discussed with two original rules, and job sort methods in the NEH method. We also present a novel rule for job assignments, and then propose an NEH-based heuristic enhanced by this rule. Finally experimental results are analyzed and concluded.

DISTRIBUTED PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Formally the permutation flowshop scheduling problem is described as follows (Grabowski and Wodecki, 2004), each of n jobs from the set $J=\{1,2,\cdots,n\}$ has to be processed on m machines in the order of $1,2,\cdots,m$. Job j, $j\in J$ consists of a sequence of m operations $O_{j1},O_{j2},\cdots,O_{jm}$; operation O_{jk} corresponds to the processing of job j on machine k and is associated with a processing time p_{jk}. All jobs are uninterrupted. The objective is to find a sequence of the jobs that meet a given criterion. The criterion we think of is the maximum completion time or makespan.

Let π be a sequence of all jobs and $C(j,k)$ denotes the completion time of O_{jk}, then $C(j,k)$ can be calculated by the following formulas (Reeves, 1995).

$$\pi=\{j_1,j_2,\cdots,j_n\}$$

$$C(j_1,1)=p_{j_1 1}$$

$$C(j_i,1)=C(j_{i-1},1)+p_{j_i 1} \text{ for } i=2,\cdots,n$$

$$C(j_i,k)=C(j_i,k-1)+p_{j_i k} \text{ for } k=2,\cdots,m$$

$$C(j_i,k)=\max\{C(j_{i-1},k),C(j_i,k-1)\}+p_{j_i k} \text{ for } i=2,\cdots,n;\ k=2,\cdots,m$$

$$C_{max}(\pi)=C(n,m)$$

where $C_{max}(\pi)$ is the makespan. The task of solving a permutation flowshop scheduling problem is to find a π such that $C_{max}(\pi)$ is minimized.

When it comes to a DPFSP, we follow the definition (Naderi and Ruiz, 2010): n jobs from the set $J=\{1,2,\cdots,n\}$ have to be processed on F factories, where each factory $f\in G=\{1,\cdots,F\}$ contains the same set of m machines, which is same as the PFSP. All factories are able to process all jobs. When a job j is assigned to a factory f, it can not be transferred to another factory and all operations of it can only be processed at factory f. Each operation O_{jk} is associated with a processing time p_{jk}. It is noted that this processing time of the operation is available for all factories. Namely, the processing times of O_{jk} in all factories are same. A schedule of jobs is a set of job sequences, denoted by $\prod$, $\prod$ contains F job sequences. The intersection of any two job sequences is empty and the union of all job sequences is the set J. The makespan of a schedule $\prod$ is defined as the maximum makespan among all factories, which can be formulated

as follows.

$$C_{max}(\prod) = \max\{C_{max}(\pi_f)\} \text{ for } f \in G$$

where π_f denotes the job sequence of the *f*-th factory. The goal of a DPFSP is to find the minimal makespan of the DPFSP.

NEH HEURISTIC

Heuristic algorithms play an important role in scheduling problems. Whereas exact algorithms like mixed integer programming are used to obtain the optimized solution to the small-sized problem, heuristics are proposed to solve large problem instances effectively. Many researchers have worked on developing heuristics to find a near optimal solution in a reasonable time, that is, build a feasible solution in polynomial time. Besides early constructive heuristics like the index heuristic, the CDS method and the NEH algorithm (Nawaz et al., 1983), the newly proposed constructive methods in (Li and Li, 2007; Li et al., 2004), as well as more complex heuristic algorithms in (Agarwal et al., 2006). Also the NEH has recently been improved (Dong et al., 2008; Chakraborty and Laha, 2007), some heuristics for optimizing maximum tardiness and makespan have been presented (Allahverdi, 2004; Braglia and Grassi, 2009). In comparison the NEH algorithm is still one of the most efficient heuristics (Dong et al., 2008). Next we recall the NEH algorithm presented (Nawaz et al., 1983).

The NEH algorithm has two steps, sorts all the jobs by decreasing sums of processing times for the jobs on all machines, and for the *k*th job, *k*=1, . . . , *n*, finds the best position among *k* possible ones that minimizes the partial makespan, then inserts it into the position.

The sorting step was improved by Li et al. (2004), they propose to sort jobs using average processing times and deviations of the processing times of the jobs. Dong et al. (2008) order jobs by the sums of the average processing times AVG_j and the standard deviations of processing times STD_j, where AVG_j and STD_j are described as follows:

$$AVG_j = \frac{1}{m}\sum_{i=1}^{m} p_{ij}$$

$$STD_j = \left[\frac{1}{m-1}\sum_{i=1}^{m}(p_{ij} - AVG_j)^2\right]^{1/2}$$

Since a solution to a DPFSP is a set of *F* job sequences (each sequence is associated with a factory), the NEH heuristic should adapt to the DPFSP to construct multi-factory sequences at its second step. Through the empirical study (Naderi and Ruiz, 2010), select the following rules to construct DPFSP solutions in NEH methods.

Rule 1: assign job *j* to the factory with the lowest current C_{max}, not including job *j*. Once assigned, the job is inserted in all possible positions of the job sequence and is placed at the position with the lowest makespan.
Rule 2: assign job *j* to the factory which completes it at the earliest time, that is, the factory with the lowest C_{max} when including job *j*. All possible positions of all factories will be tried and job *j* will be placed at the position with the lowest makespan when including *j*.

For convenience, we denote the NEH heuristic with Rule 1 for solving DPFSP instances as NEH1, while the NEH heuristic with Rule 2 as NEH2. An empirical comparison of NEH1 and NEH2 reveals that NEH2 performs better (Naderi and Ruiz, 2010).

THE PROPOSED CONSTRUCTIVE HEURISTIC

The aforementioned NEH rules select only one job at a time. This strategy works well because the single-factory problem contains only one job sequence. Care should be taken to exploit NEH rules for multiple factories in DPFSPs. Next we present such a constructive heuristic algorithm with a novel job insertion rule.

Unlike the previous job insertion rule, we insert *F* jobs at a time, and each job is assigned to a factory. To do so, *F* jobs are first selected, and the best position on each factory as well as the makespan is computed for each of them. Next jobs are assigned to the factories by using a bijective mapping from jobs to factories, which has the lowest partial C_{max} in comparison to alternative job assignments. To select the best job assignments, a simple B&B method (Land and Doig, 1960) is adopted to determine the optimized association between jobs and factories. Formally, Algorithm 1 depicts the procedure of our job insertion, which is denoted by Rule-f.

For each job, Algorithm 1 starts from computing a factory and its position if it has the smallest partial makespan. Then it assigns jobs by using function B&B (0, *max* (*L*), *L, P*), where *max* (*L*) is the largest number of L_{ij} ($1 \le i, j \le F$) to initialize the parameter *b*, the upper bound of C_{max}. B&B is a recursive function that runs a depth-first search, backtracks to the parent level to escape the local optimum, and thus to find and return the best job placement with the smallest partial C_{max}. After B&B finds the optimized job assignments so that C_{max} will be the smallest after those *F* jobs are inserted into the corresponding factories, Algorithm 1 ends with applying this job assignment by inserting jobs into the associated factories.

Note that the solution constructed only by Rule-f has the same number of jobs for each factory (or difference of a job), but a good solution may have job sequences with different length. So a job will be placed into a factory if

the insertion does not increase the total makespan before each execution of Rule-f. This is done in the heuristic algorithm NEH-df for DPFSPs.

Step 1: sorts all the jobs by decreasing the sums of the average processing times AVG_j and the standard deviations of processing times STD_j, where AVG_j and STD_j are defined earlier in the NEH heuristic; the sorted job sequence is denoted by J_s.
Step 2: repeats the following until $|J_s|$ is less than F; insert the first job j in J_s by Rule 2 and remove j from J_s if C_{max} after including j does not change, otherwise, perform Rule-f to insert F jobs at a time.
Step 3: assigns the remaining jobs by Rule 2.

In this algorithm, NEH-df first employs $AVG_j + STD_j$ for job ordering which has been studied by Dong et al. (2008) and shows the good performance. Then Rule 2 and Rule-f will be performed alternately until the number of jobs in the job list J_s are less than F. Finally the remaining jobs will be inserted by Rule 2 as NEH2 does.

Note that the time complexity of NEH-df mainly depends on the computation of lowest makespan for job insertion. We use Taillard (1990) accelerations, when the C_{maxl} is calculated during each job insertion, which can decrease the time complexity greatly. The times of a job insertion for our heuristic does not increased compared to NEH2 where Rule 2 is used. It requires to sequence all m tasks of a job at all factories, that is, $O(mF)$, which is same as NEH2. Though a B&B algorithm is used during the solution construction, the worst case of its time complexity is only $O(F!)$. Since F is a constant, it can be ignorant when evaluating the time cost level of the entire algorithm.

EXPERIMENTS

To evaluate the performance of the proposed heuristic NEH-df, intensive computational experiments are carried out on the DPFSP benchmark available at http://soa.iti.es. In this paper, only large-scale instances are concerned. Whereas the number of jobs is up to 16 and the number of machines is only 5 at most in small-scale instances, the set of large-scale instances is extended from the benchmark of Taillard by adding the number of factories F from {2,3,4,5,6,7}. The Taillard instances are composed of 12 combinations of $n \times m$, and for each combination there are 10 different instances. Each instance is combined with 6 values of F to yield 6 instances of DPFSP benchmark, so the number of total instances reaches 720. The best solutions to the benchmark instances are obtained by implementing the heuristic approaches proposed (Naderi and Ruiz, 2010). Many heuristic approaches have been tested (Naderi and Ruiz, 2010). In our experiments, we would rather consider NEH1 and NEH2 than other heuristics because all heuristics are extensions of the well-known existing heuristics for solving classical PFSP, and they have much better performance than other heuristic algorithms. For simplicity, we denote NEH2 with $AVG_j + STD_j$ sorting by NEH-d and NEH-df with original job sorting method by NEH-f. The competing algorithms are incorporated in a C++ program and implemented within VC++6.0. We run all those algorithms on an Intel Core Duo 2.4GHz machine with 2GB RAM under Windows XP.

To measure performance, we chose to use the following relative percentage deviation (RPD):

$$RPD = \frac{alg - opt}{opt} \times 100$$

Where *opt* is the best solution published and *alg* stands for the solution obtained by the heuristic algorithms. We analyze *RPD* of the aforementioned experiment. From Table 1, it can be seen that NEH-df algorithm outperforms all the other algorithms as it has the best average *RPD*. We can also see that both NEH-d and NEH-f have better performance than their competitor NEH2.

We also give the *RPD* results grouped by the combination of m and n, as shown in Table 2. Similar conclusion can be drawn from Table 2, that is, NEH-df performs best for 7 out of 12 combinations of m and n.

To draw a better picture of the results, we also check whether the differences in Table 1 and Table 2 made by those algorithms are statistically significant. In this case, hypotheses of normality, homocedasticity and independence of the residuals are checked and satisfied, and the ANOVA test is preformed. Figure 1 shows the results, from which we can clearly observe that the NEH-df is significantly better than the original NEH-based heuristics, that is, NEH1 and NEH2. But there is no statistical significance between NEH2 and NEH-d or NEH2 and NEH-f.

The run-times of those heuristic algorithms are also compared. Table 3 indicates the results. From the table, we can observe that the run-times of NEH-df is slightly larger than those of others. NEH1 is the fastest, but the solutions produced by NEH1 are rather worse than those of others. Furthermore, NEH-df can solve the instance that has 500 jobs and 20 machines only in 47 ms on average, so NEH-df is still very efficient.

CONCLUSIONS

The distributed permutation flowshop scheduling problem is a newly proposed scheduling problem, which is in the set of NP-hard. Several heuristic and local search algorithms have been presented, as well as maxed integer programming methods that can only solve small-

Table 1. Average relative percentage deviation (*RPD*) of algorithms grouped by *F*.

F	Algorithms				
	NEH1	NEH2	NEH-d	NEH-f	NEH-df
2	2.90	1.19	1.18	0.95	1.03
3	3.57	1.08	1.05	0.94	0.88
4	4.26	1.19	1.01	0.89	0.69
5	4.30	0.89	0.89	0.86	0.83
6	4.61	1.00	0.85	1.05	0.95
7	4.70	0.75	0.60	0.90	0.68
Average	4.06	1.02	0.93	0.93	0.84

The 5 heuristic algorithms were carried out once for all 720 instances. For each heuristic algorithm, average *RPD* results grouped by *F* are listed. At the bottom of the table, the total average values of all *f* are reported as well.

Table 2. Average relative percentage deviation (RPD) of algorithms grouped by *m* and *n*.

n×*m*	Algorithms				
	NEH1	NEH2	NEH-d	NEH-f	NEH-df
20×5	3.86	1.35	0.49	1.14	0.84
20×10	3.48	0.82	0.69	0.65	0.60
20×20	2.61	0.83	0.56	0.86	0.60
50×5	6.26	1.11	1.27	0.90	1.31
50×10	4.83	1.16	1.67	1.33	1.33
50×20	3.66	1.16	0.98	1.00	0.77
100×5	5.54	0.78	0.63	0.73	0.75
100×10	4.77	0.86	1.07	1.04	0.86
100×20	3.38	1.02	0.91	0.87	0.83
200×10	4.51	0.97	0.90	0.89	0.55
200×20	3.17	1.21	1.12	0.99	0.93
500×20	2.64	0.93	0.87	0.79	0.72

Similar to the results in table 1, average *RPD* results of each heuristic algorithm grouped by *n*×*m* are listed.

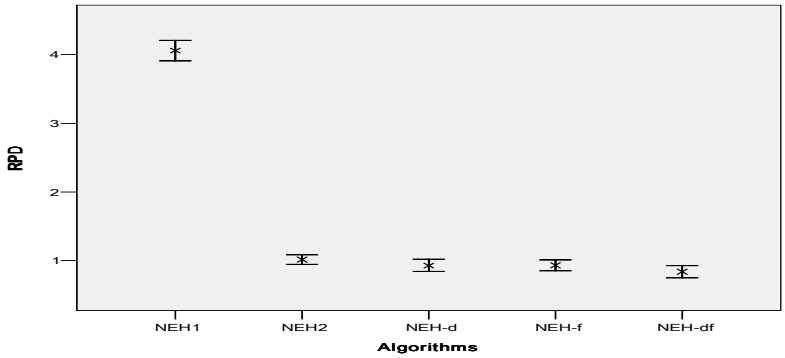

Figure 1. Mean plot with intervals at 95% confidence level for the algorithm factor. For each algorithm, RPD values of all the 720 instances are employed to calculate its confidence intervals.

Table 3. CPU run-times of algorithms grouped by *F*.

F	Algorithms				
	NEH1	**NEH2**	**NEH-d**	**NEH-f**	**NEH-df**
2	392	626	685	654	673
3	266	517	594	627	632
4	218	544	535	562	607
5	155	469	491	554	576
6	184	497	478	594	609
7	125	434	489	556	581
Total	1340	3087	3272	3547	3678

The 5 heuristic algorithms are performed by 100 runs for each instance. Average CPU run-time of each instance in milliseconds is recorded. The sums of the average run-times are listed grouped by *F*. In addition, the sum of the average run-times are also calculated at the bottom of the table.

scale instances though they are exact methods. Among those heuristics, algorithms based on NEH usually have good performance. Two job insertion rules are employed, where jobs are assigned to factories one by one. This paper presents a new constructive heuristic by introducing a novel job insertion rule for constructing solutions to DPFSPs. It assigns a group of jobs to factories at a time, since a DPFSP instance has many job sequences to construct. Furthermore, we use the strategy discussed by Dong et al. (2008) for job sorting in the first step of NEH algorithms, where jobs are ordered by the sum of average processing times and standard deviations of the processing times of the jobs. Experimental results indicate that our NEH-based heuristic outperforms all the other heuristics on average *RPD*, and also show it is significantly better than the original heuristics. In addition, we also show that the proposed method does not increase the time complexity.

ACKNOWLEDGEMENTS

This work is partially supported by the National Natural Science Foundation of China under Grant No.60775028, the Major Projects of Technology Bureau of Dalian No.2007A14GXD42, and IT Industry Development of Jilin Province.

REFERENCES

Agarwal A, Colak S, Eryarsoy E (2006). Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. Eur. J. Oper. Res., 169: 801-815.

Allahverdi A (2004). A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. Computers & OR, 31: 157-180.

Braglia M, Grassi A (2009). A new heuristic for the flowshop scheduling problem to minimize makespan and maximum tardiness. Int. J. Prod. Res., 47: 273-288.

Campbell HG, Dudek RA, Smith ML (1970). Heuristic algorithm for N-job, M-machine sequencing problem. Management Science Series B-Application. 16: 630-637.

Chan FTS, Chung SH, Chan PLY (2005). An adaptive genetic algorithm with dominated genes for distributed scheduling problems. Expert Syst. Appl., 29: 364-371.

Chakraborty UK, Laha D (2007). An improved heuristic for permutation flowshop scheduling. Int. J. Inf. Commun. Technol., 1: 89-97.

Dong X, Huang H, Chen P (2008). An improved NEH-based heuristic for the permutation flowshop problem. Computers & OR., 35: 3962-3968.

Erdogmus P (2010). Particle swarm optimization performance on special linear programming problems. Sci. Res. Essays, 5: 1506-1518.

Fondrevelle J, Oulamara A, Portmann MC (2006). Permutation flowshop scheduling problems with maximal and minimal time lags. Computers & OR, 33: 1540-1556.

Garey MR, Johnson DS, Sethi R (1976). The complexity of flowshop and jobshop scheduling. Math. Operat. Res., 1: 117-129.

Grabowski J, Wodecki M (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers & OR, 31: 1891-1909

Jia HZ, Nee AYC, Fuh JYH, Zhang YF (2003). A modified genetic algorithm for distributed scheduling problems. J. Intel. Man., 14: 351-362.

Land AH, Doig AG (1960). An automatic method of solving discrete programming problems. Econometrica, 28:497-520.

Li T., Li Y (2007). Constructive Backtracking Heuristic for Hybrid Flowshop Scheduling with Limited Waiting Times. In: International Conference on Wireless Communications, Networking and Mobile Computing. pp. 6671-6674.

Li XP, Wang YX, Wu C (2004). Heuristic algorithms for large flowshop scheduling problems. In: Proceedings of the 5th world congress on intelligent control and automation. pp. 2999-3003.

Li X, Wang J, Zhou J, Yin M (2010). An effective GSA based memetic algorithm for permutation flow shop scheduling. In: IEEE Congress on Evolutionary Computation, pp. 1-6.

Naderi B, Ruiz R (2010). The distributed permutation flowshop scheduling problem. Computers & OR, 37: 754-768.

Nawaz M, Enscore Jr. EE, Ham I (1983). A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem. Omega-Int. J. Manage. Sci., 11: 91-95.

Onwubolu GC, Davendra D (2006). Scheduling flowshops using differential evolution algorithm. Eur. J. Oper. Res., 171: 674-692.

Palmer DS (1965). Sequencing jobs through a multi-stage process in the minimum total time: a quick method of obtaining a near optimum. Oper. Res. Q., 16: 101-107.

Rajendran C, Ziegler H (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/ total flowtime of jobs. Eur. J. Oper. Res., 155:426-438.
Reeves CR (1995). A genetic algorithm for flowshop sequencing. Computers & OR. 22: 5-13.
Taillard E (1990). Some efficient heuristic methods for the flow-shop sequencing problem. Eur. J. Operat. Res., 47: 65–74.
Yin M, Zou T, Gu W (2010). Reverse Bridge Theorem under Constraint Partition, Mathematics Problems in Engineering. doi:10.1155/2010/617398.
Yin M, Li X (2011). A hybrid bio-geography based optimization for permutation flow shop scheduling. Sci. Res. Essays, 6(10): 2078-2100.