International Journal of Industrial Engineering & Production Research

December 2012, Volume 23, Number 4
pp. 301-308

http://IJIEPR.iust.ac.ir/

ISSN: 2008-4889

An Iterated Greedy Algorithm for Solving the Blocking Flow Shop Scheduling Problem with Total Flow Time Criteria

D. Khorasanian & G. Moslehi*

Danial Khorasanian *is an M.S. Student of Department of Industrial Engineering, Isfahan University of Technology, Isfahan, Iran*
Ghasem Moslehi *is a Professor of Department of Industrial Engineering, Isfahan University of Technology, Isfahan, Iran*

KEYWORDS

Constructive heuristic,
Iterated greedy algorithm,
Blocking flow shop,
Total flow time

ABSTRACT

In this paper, we propose an iterated greedy algorithm for solving the blocking flow shop scheduling problem with total flow time minimization objective. The steps of this algorithm are designed very efficient. For generating an initial solution, we develop an efficient constructive heuristic by modifying the best known NEH algorithm. Effectiveness of the proposed iterated greedy algorithm is tested on the Taillard's instances. Computational results show the high efficiency of this algorithm with comparison state-of-the-art algorithms. We report new best solutions for 88 instances of 120 Taillard's instances.

1. Introduction

The flow shop scheduling problem is one of the most popular machine scheduling problems with extensive engineering relevance, representing nearly a quarter of manufacturing systems, assembly lines and information service facilities in use nowadays [1]. In the general flow shop model, it is assumed that the buffers have unlimited capacity. However, in many real flow shop environments, the buffers may have limited capacity due to technological requirements or process characteristics [2].

A special case of these environments is the flow shop with zero capacity buffers that the related problem is known as the blocking flow shop scheduling problem (BFSP). Since there are no buffers between machines, no intermediate queues of jobs are allowed in the production system for their next operations [3]. In BFSP, the completed job on a machine may block it until the next downstream machine is free. Grabowski and Pempera [4] present a real life example in the production of concrete blocks that does not allow storage in some stages. A detailed review of studies and applications of the BFSP can be found in [2]. It has been proved that the BFSP with makespan criteria and more than two machines is strongly NP-hard [2]. Also, Rock [5] showed that the BFSP with total flow time criteria and two machines is strongly NP-hard.

In recent years, the BFSP has attracted much attention among researchers. Since the problem is strongly NP-hard for large numbers of jobs and machines, as is usual in real systems, it is more practical to use heuristics and metaheuristics to solve it. The simplest heuristics are constructive procedures, which use rules to assign a priority index to each job, in each step, to build a sequence.

Most studies of the BFSP have dealt with makespan criteria. Among the proposed constructive heuristics for the BFSP with makespan criteria, one can refer to Profile Fitting (PF) [6], MinMax (MM) [7], MME [7], PFE [7], NEH2 [8].

The most important proposed constructive heuristic for the BFSP with total flow time criteria is NEH-WPT [9]. Most of the proposed heuristics for the BFSP have

* **Corresponding author: Ghasem Moslehi**
Email: *. moslehi@cc.iut.ac.ir*
Paper first received July. 05, 2012, and in revised form Oct. 9, 2012.

been developed by modifying the best known NEH [10] heuristic. As we know, the NEH algorithm is as follows [10]:

Step 1: Sort the jobs according to the non-increasing sums of their processing times and let the obtained sequence $\pi = (\pi(1),\ldots,\pi(n))$.

Step 2: The first two jobs of the π are taken and two possible partial sequences of these two jobs are evaluated. Then, the better partial sequence is chosen as the current sequence and let $L = 3$.

Step 3: Find the best partial sequence by placing the job $\pi(L)$ in all possible positions of the current sequence. The best partial sequence becomes the next L-job current sequence.

Step 4: If $L = n$, end and otherwise, let $L = L + 1$ and back to step 3.

Recently, different types of metaheuristics have been developed for the BFSP. Genetic algorithm (GA) [11], tabu search (TS) [3], hybrid discrete differential evolution (HDDE) algorithm [1], iterated greedy (IG) algorithm [12] and hybrid modified global-best harmony search (hmgHS) algorithm [13] are the most important metaheuristics so far presented for the BFSP with makespan criteria.

Among these algorithms, Wang et al [13] have claimed that their hmgHS algorithm outperforms the others. Studies about BFSP with total flow time criteria are scarce. To the best of our knowledge, only one research paper has so far been reported on the BFSP with total flow time criteria using metaheuristics. Wang et al [9] have presented three modified harmony search (hHS, hgHS and hmgHS) algorithms where hmgHS outperforms the others. In this study, NEH-WPT heuristic has been used for an initial solution generation.

In this paper, we want to solve the BFSP with total flow time criteria using the IG algorithm. The IG algorithm is a metaheuristic that, despite its simple structure, can find very good solutions efficiently. This algorithm has been used successfully for both the permutation flow shop scheduling problem (PFSP) [14] and the BFSP [12] with makespan criteria. However, it has not so far been used for the BFSP with total flow time criteria. General frame of the IG algorithm is shown in Fig. 1.

This algorithm includes two phases. In phase 1, an efficient initial solution (π_0) is generated. This solution is considered as both the current solution (π) and the best solution (π_{best}). Then, until the stopping criteria are met, phase 2 which includes three main steps repeats. In step 1, efforts are made to improve the current solution by a local search procedure. In step 2, acceptance criteria are checked for the current solution and if it is rejected, the best solution is considered as the current solution. In step 3, a deconstruction and reconstruction is done for the current solution to escape from deep local optima.

Procedure iterated greedy
% Phase 1
Generate initial solution (π_0)
$\pi = \pi_0$
$\pi_{best} = \pi_0$
%Phase 2
repeat
% step 1
π' = Local Search (π)
if total flow time (π') < total flow time (π_{best})
then $\pi_{best} = \pi'$ **endif**
% step 2
π = Acceptance criteria (π_{best}, π')
%step 3
π' = deconstruction (π)
π = reconstruction (π')
if total flow time (π) < total flow time (π_{best})
then $\pi_{best} = \pi$ **endif**
until stopping criteria is met
end

Fig. 1. General frame of the IG algorithm

After this introduction, we will formulate the BFSP in Section 2. Section 3 details the proposed IG algorithm. Then, in Section 4, we will show the parameter settings and computational results. Finally, we will present some conclusions and suggestions for future studies in Section 5.

2. Problem Formulation

In the BFSP with total flow time criteria, there are n jobs and m machines and every job must be processed on machine 1 to m. Because intermediate buffers have zero capacity, there are not intermediate queues of jobs between machines. The completed job on a machine may block it until the next downstream machine is free. Suppose that the release time of all jobs is zero and that the set-up time on each machine is included in the processing time. At any time, each machine can process at most one job and each job can be processed on at most one machine. The objective of this problem is to find a permutation that minimizes the total flow time.

Because the jobs may be blocked after their completion times, departure times may be different from completion times. Consider the permutation $\pi = (\pi(1),\ldots,\pi(n))$.

The departure time of job $\pi(j)$ from machine i is shown by $D_{i,\pi(j)}$. Departure times of jobs in the permutation π from each machine can be calculated by the following equations:

$$D_{0,\pi(1)} = 0 \qquad (1)$$

$$D_{i,\pi(1)} = D_{i-1,\pi(1)} + p_{i,\pi(1)};\ i = 1,\ldots,m-1 \quad (2)$$

$$D_{0,\pi(k)} = D_{1,\pi(k-1)};\qquad k = 2,\ldots,n \quad (3)$$

$$D_{i,\pi(k)} = \max\left(D_{i-1,\pi(k)} + p_{i,\pi(k)}, D_{i+1,\pi(k-1)}\right);\quad i = 1,\ldots,m-1; k = 2,\ldots,n \quad (4)$$

$$D_{m,\pi(k)} = D_{m-1,\pi(k)} + p_{m,\pi(k)}; k = 1,\ldots,n \quad (5)$$

In these equations, $D_{0,\pi(j)}$ represents the starting time of job $\pi(j)$ on machine 1 and $D_{m,\pi(j)}$ designates the departure time of job $\pi(j)$ from machine m. Because the jobs are not blocked on machine m, for every job, departure time from machine m is equal to completion time on machine m. So, we can calculate total flow time of the permutation π, as follows:

$$TFT = \sum_{j=1}^{n} D_{m,\pi(j)} \quad (6)$$

TFT calculation of the π has $O(mn)$ complexity [1].

3. The Proposed IG Algorithm

As previously mentioned, the IG algorithm includes two phases. In phase 1, an initial solution is generated and in phase 2, three main steps are repeated until the stopping criteria are met. The proposed procedures for each part of the IG algorithm are as follows:

3-1. Initial Solution

As we know, the NEH [10] heuristic was originally developed for the PFSP with makespan criteria. Nevertheless, this heuristic and its modifications have been efficiently used for the PFSP and the BFSP with different criteria, such as [12] [15] [16] and [17]. An alternative method of modifying the NEH heuristic to improve its efficiency for a specific problem is changing the step 1 of its algorithm.

Wang et al [9] present NEH-WPT by sorting the jobs according to the non-decreasing sum of their processing times on each machine in the step 1. They showed that NEH-WPT outperforms the NEH heuristic. In this paper, we present a new sorting procedure for the jobs in the step 1 which is called step1-MK. In this procedure, first a job with minimum sum of processing times on each machine is selected as the first job. Then an unsorted job with minimum value of the following expression is selected as the next job. This process is repeated until the entire jobs are sorted.

$$(1-\alpha)n\sum_{i=1}^{m} p_{ij} + \alpha(n-L)\sum_{i=1}^{m-1}(m-l)\left|p_{ij} - p_{i+1,q}\right| \quad (7)$$

In the above expression, j and q represent the candidate job for assignment and the last assigned job, respectively. This expression includes the $\sum_{i=1}^{m} p_{ij}$ and $\sum_{i=1}^{m-1}(m-l)\left|p_{ij} - p_{i+1,q}\right|$ terms that their weights are $(1-\alpha)n$ and $\alpha(n-L)$, respectively. If the weight of the second term be equal to zero; i.e. $\alpha = 0$, obtained sequence by the step1-MK procedure will be same as the sequence which is obtained by sorting the jobs according to non-decreasing sum of processing times on each machine.

The second term represents the sum of weighted approximate of block and idle times on each machine generated by placing the candidate job after the last assigned job.

In this term, the weights are considered to decrease with machine stage number. This is because the larger becomes the machine stage number, the smaller will be the effect of its block and idle times on starting times of the following jobs. The weight of this term is considered to be decreasing with respect to the assigned jobs number (L). This is because the effect of the block and idle times of the candidate job on starting times of the following jobs become smaller as the assigned jobs number increases and so that of the unassigned jobs decreases. Algorithm of the step1_MK procedure is as follows:

Step 1: Let a job with minimum sum of processing times on each machine as first job. If more than one job have this characteristic, select a job with minimum processing time on machine 1 among them. If more than one job have this characteristic yet, assign one of them randomly. Let $L = 2$.

Step 2: Among the unsorted jobs, assign a job with minimum value of expression (7) as Lth job. If more than one job have this characteristic, select one of them, randomly. Let $L = L + 1$.

Step 3: If $L = n$, end; otherwise, back to step 2.

Simply, we can find that the complexity of step1-MK procedure is $O(mn^2)$. Another alternative for improving the NEH [10] algorithm is modifying the step 3 of its algorithm. In step 3, after finding the best place for the new job, we can improve the obtained current sequence by a procedure like a local search. In following, a procedure is presented instead of step 3 of NEH [10] heuristic that is called step3-MK(k). We can adjust improving level (k) with considering the time limit. Suppose that j_{new} is the new job to assign in step 3 and the current sequence has $L - 1$ jobs. For this case, algorithm of the step3-MK(k) procedure is as follows:

Step 1: Find the best partial sequence by placing j_{new} in all possible positions of current sequence. The best partial sequence becomes the next L-job current sequence.

Step 2: Let $d = \lfloor k \times (L-1)/n \rfloor$. If d = 0 end; otherwise, go to step 3.

Step 3: Select d non-repetitive jobs except j_{new} among the jobs in the current sequence randomly and save them in $S_{1\times d}$ matrix. Let i = 1.

Step 4: Remove the job $S[i]$ from its position in the current sequence and place it in its L – 1 other possible positions in this sequence. Among these obtained L sequences, select the best one as the next L-job current sequence.

Step 5: If $i = d$, end; otherwise, let $i = i + 1$ and back to step 4.

In step 2 of the above algorithm, the operator $\lfloor \rfloor$ represents the integer part of a number. In the step3-MK(k) procedure, a specific portion of the jobs of the current sequence is reinserted. This portion is related to parameter k and n. The larger the values of k, the more the number of jobs reinserted for a given problem. Because, for a specific value of the mentioned portion, execution time of the above step extremely increases with increasing the value of n, this value is considered decreasing respect to n for regulating the execution time of this step. Obviously, when k = 0, this procedure is equivalent to step 3 of the NEH [10] algorithm.

In the following, we present a constructive heuristic that is called NEH-MK(k). This heuristic is a modification of the NEH [10] heuristic in which step1-MK and step3-MK(k) procedures are used instead of steps 1 and 3 of its algorithm, respectively. This heuristic has two parameters (α, k) and its algorithm is as follows:

Step 1: Obtain the sequence $\pi = (\pi(1),\ldots,\pi(n))$ by using the step1-MK procedure.

Step 2: The first two jobs of π are taken and two possible partial sequences of these two jobs are evaluated. Then, choose the better partial sequence as the current sequence and let L = 3.

Step 3: By considering $\pi(L)$ as the new job and with the current sequence with L – 1 jobs, obtain the next L-job current sequence using the step3-MK(k) procedure.

Step 4: If $L = n$, end; otherwise, let $L = L + 1$ and go back to step 3.

A glance at the algorithm reveals that step 3 of the above algorithm determines its complexity. This step requires $L + (\lfloor k \times (L-1)/n \rfloor \times (L-1))$ times objective calculation of an L-job sequence. Because the objective calculation of an L-job sequence has $O(mL)$ complexity [1], we can say that the complexity of the NEH-MK(k) algorithm is about $O(mn^3k)$. In the proposed IG algorithm, we use the NEH-MK(k) heuristic for the initial solution generation.

3-2. Local Search

By using the local search, the solution obtained from the deconstruction and reconstruction step is accepted with more probability. Two consecutive pair-wise and insert-based local searches are used for this purpose. The pair-wise based local search considered is a non-exhaustive decent algorithm that tries to improve the current sequence by swapping any two of its positions. If, during this process, a new sequence improves the value of the objective function, it becomes the new current sequence and the process continues until all of the positions of the current sequence have been permuted and no more improvement takes place [12]. After this pair-wise local search, the insert-based local search is executed on the current sequence thus obtained.

In this local search, job 1 is placed in the L – 1 other possible positions of the current sequence. Among the obtained L – 1 sequences and the current sequence, the best one is considered as the next current sequence. This process is repeated up to job n.

3-3. Acceptance Criteria

Acceptance criteria are those that determine the acceptance or rejection of the solution obtained in the local search step (π') as the next current solution (π). If this solution is equal to or better than the best solution (π_{best}), it is accepted. Else if the π' is worse than the π_{best}, it is accepted with a specific value of probability. If π' is rejected, then π_{best} is considered as the next π. In this study, for the sake of the parameters adjustment of the proposed IG algorithm becomes manageable, the probability of acceptance of the worse sequence is considered equal to 0.5, like the reference [12].

3-4. Deconstruction and Reconstruction

In the deconstruction part of this step, N_r jobs are randomly removed from the current sequence and saved in a $R_{1\times N_r}$ matrix. The result is a current sequence with $(n - N_r)$ jobs. Then in the reconstruction part, $R[1]$ is considered as the new job for this current sequence and the next current sequence with $(n - N_r + 1)$ jobs is obtained using the step3-MK(k) procedure.

This process is repeated up to job $R[N_r]$. Before the execution of the proposed IG algorithm, the parameter k of the step3-MK(k) procedure used in this step, should be adjusted.

4. Experimental Evaluation

In this section, we intend to evaluate the effectiveness of the proposed IG algorithm. For this purpose, we use the Taillard instances [18] combining 20, 50, 100, 200 and 500 jobs with 5, 10 and 20 machines. These benchmark instances are composed of

12 groups each of which has 10 instances of the same size.

In these evaluations, a time limit of $20.m.n$ milliseconds is considered for solving each instance. Before these evaluations, the parameters of the proposed algorithm should be adjusted. To avoid over learning in parameter adjustment, we have generated a new random set of instances by using the Taillard procedure [10] and random seeds. In this random set, there are 20 groups, 10 instances per each group for every combination of m and n where $m \in \{5,10,15,20\}$ and $n \in \{20,60,100,150,200\}$. In the parameter adjustment, a time limit of $5.m.n$ milliseconds is considered for solving each instance.

The algorithms have been coded in C# and tested on a Core 2 Due T9600, 2.8 GHz and 4 GB of RAM memory.

To analyze the effectiveness of solution A for instance s, we used the relative percentage deviation (RPD) calculated by equation (8):

$$RPD_{As} = \frac{(TFT_{As} - bestTFT_s)}{bestTFT_s} \times 100 \qquad (8)$$

Where, "TFT_{AS}" is the total flow time of solution A for instance s and "$bestTFT_s$" is the best known total flow time for the same instance.

4-1. Experimental Parameter Setting

The NEH-MK(k) heuristic which generates the initial solution has two parameters, α and k. With a little attention to this heuristic, we can find that different α levels have not any effect on the execution time of it. However this issue is not true for the parameter k. We have evaluated the effect of k levels on average execution time (in seconds) of NEH-MK(k) in Fig. 2. As we see in this figure, execution time of the NEH-MK(k) increases with k. Because of the time limit for the proposed IG algorithm, both the quality of the initial solution and the time taken to obtain are important.

So, we can tune the parameter α separately. But, we have to adjust the k with respect to the other parameters of the proposed IG algorithm. We considered the following levels for each parameter of the NEH-MK(k):

- α, 11 levels: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0;
- k, 4 levels: 0, 2, 5, 10.

Considering the time limit chosen for the proposed IG algorithm, we cannot recommend the larger levels for k; because the execution time of the initial solution generation for the instances with 500 jobs and 20 machines will be more than a half of the time limit. We performed a 11×4 full factorial experiment for finding the best level of the parameter α. Due to the randomness of the NEH-MK(k), for each 44 different combinations of α and k parameters, we performed 5 runs per instance. So, instance s was solved 220 times by the NEH-MK(k) heuristic and the best TFT among them is selected as the $bestTFT_s$ for RPD calculation. The above results were analyzed by a multi-way analysis of variance (ANOVA).

First, the normality, homoscedasticity, and independence assumptions were checked and no considerable departure was found. This analysis showed that both parameters are significant, but their interaction is not like this (P-value < 0.05). By analysis of different levels of the parameter α with Tukey HSD 95% confidence intervals, $\alpha = 0.2$ is selected as the best level for the NEH-MK(k) heuristic.

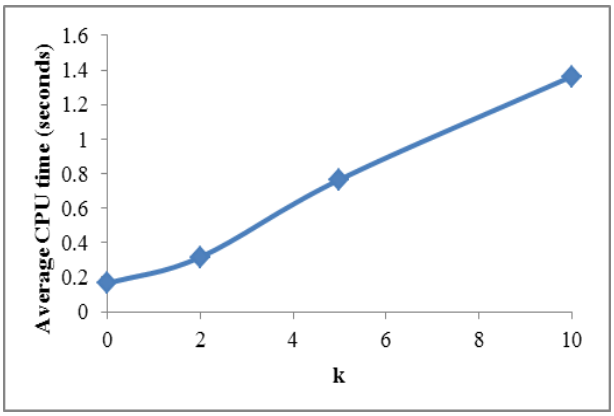

Fig. 2. Effect of *k* on execution time of the NEH-MK(k)

We are now left with the adjustment of the three parameters in the proposed IG algorithm, namely k in the NEH-MK(k) heuristic in the initial solution generation phase which we designated as k_1; N_r in the deconstruction part and finally k_2 in the step3-MK(k_2) procedure in the reconstruction part. We considered the following levels for each parameter:

- k_1, 4 levels: 0, 2, 5, 10;
- N_r, 3 levels: 3, 5, 10;
- k_2, 4 levels: 0, 2, 5, 10.

For finding the best level of each parameter, we performed a $4 \times 3 \times 4$ full factorial experiment. Due to the randomness of the IG algorithm, we performed 5 runs per instance for each 48 different combination of k_1, N_r and k_2. So, each instance was solved 240 times by the IG algorithm.

As already mentioned above, the time limit in these evaluations is $5.m.n$ milliseconds. For instance s, the *best* TFT_s is considered equal to the objective of the best solution among the ones obtained for this instance

in this experiment. These results were analyzed by a multi-way ANOVA.

The normality, homoscedasticity, and independence assumptions were checked and no considerable departures were found.

This analysis showed that all the parameters and their interactions are significant, except the $k_1 \times k_2$, $k_1 \times N_r$ and $k_1 \times k_2 \times N_r$ interactions. So first we found the best level of k_1. By the Tukey HSD 95% confidence interval, we found that the best level for k_1 is 10. Then we compared the different combination of the N_r and k_2 levels by the Tukey HSD 95% confidence interval and with assumption of $k_1 = 10$. We found that (5, 2), (5, 5) and (5, 10) combinations of the (N_r, k_2) are in the best homogenous subset that we select (5, 2) which has the minimum average RPD. So the value of the parameters α, k_1, N_r and k_2 are finally considered equal to 0.2, 10, 5 and 2, respectively.

4-2. Experimental Results

As already mentioned, we used the Taillard benchmark instances for evaluating the effectiveness of the proposed IG algorithm. The total flow time values of the best known solutions for these instances are shown in Table 1, which we used for calculating the RPD in this section. In this Table, the Dataset column indicates the size (n|m) and number of the instances, Best column indicates the total flow time of the best known solution, and S represents the algorithm that obtained the best known solution. In the S column, "1" designates the hmgHS [9] and "2" represents the proposed IG algorithm.

First we compared the effectiveness of the NEH-MK(k) heuristics, $k \in \{0,2,5,10\}$, with the NEH-WPT [9] heuristic. For this purpose, each instance was solved 5 times with each heuristic. Average RPD values of the solutions and their average execution times for each group of Taillard instances obtained by each heuristic are shown in Tables 2 and 3, respectively.

According to the data in Table 2, the average RPD values of the Taillard instances for NEH-WPT, NEH-MK(0), NEH-MK(2), NEH-MK(5), and NEH-MK(10) are equal to 4.10, 3.68, 3.12, 2.42 and 2.07, respectively.

All the NEH-MK(k) heuristics, $k \in \{0,2,5,10\}$, exhibited better efficiency than the NEH-WPT heuristic. Larger values of k resulted in better efficiencies for the NEH-MK(k) heuristic. These claims have been confirmed by the Tukey HSD 95% confidence intervals. The only difference between the NEH-WPT [9] and NEH-MK(0) algorithms lies in their step 1. So, the above results clearly show the more effectiveness of the proposed step1-MK procedure with comparison step 1 of the NEH-WPT [9].

Tab. 1. Best known solutions for Taillard benchmark instances

Dataset	Best	S	Dataset	Best	S	Dataset	Best	S
20|5			**20|10**			**20|20**		
1	14953	1,2	11	22358	1,2	21	34683	1,2
2	16343	1,2	12	23881	1,2	22	32855	1,2
3	14297	1,2	13	20873	1,2	23	34825	1,2
4	16483	1,2	14	19916	1,2	24	33006	1,2
5	14212	1,2	15	20196	1,2	25	35328	1,2
6	14624	1,2	16	20126	1,2	26	33720	1,2
7	14936	1,2	17	19471	1,2	27	33992	1,2
8	15193	1,2	18	21330	1,2	28	33388	1,2
9	15544	1,2	19	21585	1,2	29	34798	1,2
10	14392	1,2	20	22582	1,2	30	33174	1,2
50|5			**50|10**			**50|20**		
31	73101	2	41	100193	2	51	137222	2
32	78411	2	42	96135	2	52	130386	2
33	73499	2	43	92234	2	53	128102	1
34	77621	2	44	98820	2	54	132378	2
35	78824	2	45	98502	2	55	131058	2
36	75543	2	46	97721	2	56	131936	2
37	74291	2	47	100138	2	57	135148	2
38	74056	2	48	98565	2	58	133379	2
39	71161	2	49	97372	2	59	133012	2
40	79306	2	50	98368	2	60	136249	1
100|5			**100|10**			**100|20**		
61	292863	2	71	357766	2	81	430626	2
62	283784	2	72	339274	2	82	440768	2
63	280031	2	73	347673	2	83	435725	2
64	265390	2	74	364470	2	84	438568	2
65	278352	2	75	342828	2	85	433014	2
66	273840	2	76	333139	2	86	436769	2
67	279652	2	77	341278	2	87	443088	2
68	275113	2	78	347754	2	88	446417	2
69	289241	2	79	361197	2	89	437340	2
70	285839	2	80	353220	2	90	444264	2
200|10			**200|20**			**500|20**		
91	1309697	2	101	1531757	2	111	8983587	2
92	1304588	2	102	1563365	2	112	9120898	2
93	1308056	2	103	1580211	2	113	9023587	2
94	1299421	2	104	1564420	2	114	9111421	2
95	1303077	2	105	1545823	2	115	9078369	2
96	1281034	2	106	1557750	2	116	9158850	2
97	1332859	2	107	1559874	2	117	9025125	2
98	1325551	2	108	1568342	2	118	9096221	2
99	1303549	2	109	1545995	2	119	9073555	2
100	1299273	2	110	1562535	2	120	9115986	2

Tab. 2. Mean of RPD comparisons of the NEH-MK(k) with NEH-WPT

n|m	NEH-WPT	NEH-MK(0)	NEH-MK(2)	NEH-MK(5)	NEH-MK(10)
20|5	3.31	3.09	2.39	1.80	1.42
20|10	3.09	3.08	2.40	1.47	1.35
20|20	3.58	2.27	1.90	1.23	0.94
50|5	5.50	4.65	4.35	3.87	3.47
50|10	5.12	4.66	4.01	3.14	2.68
50|20	4.33	4.21	3.41	2.60	2.37
100|5	6.17	5.48	4.70	4.04	3.61
100|10	4.75	4.39	3.84	2.92	2.66
100|20	3.87	3.47	2.92	2.31	1.89
200|10	4.37	4.06	3.51	2.77	2.25
200|20	2.92	2.72	2.28	1.67	1.38
500|20	2.21	2.08	1.69	1.20	0.85
Average	4.10	3.68	3.12	2.42	2.07

Tab. 3. Average CPU time (second) comparisons of the NEH-MK(k) with NEH-WPT

n|m	NEH-WPT	NEH-MK(0)	NEH-MK(2)	NEH-MK(5)	NEH-MK(10)
20|5	0.00	0.00	0.00	0.00	0.00
20|10	0.00	0.00	0.00	0.00	0.01
20|20	0.00	0.00	0.00	0.01	0.01
50|5	0.01	0.01	0.01	0.02	0.04
50|10	0.01	0.01	0.02	0.03	0.06
50|20	0.01	0.02	0.03	0.07	0.11
100|5	0.03	0.04	0.07	0.16	0.27
100|10	0.06	0.06	0.11	0.26	0.45
100|20	0.10	0.11	0.20	0.46	0.85
200|10	0.44	0.45	0.83	2.08	3.62
200|20	0.79	0.80	1.50	3.75	6.40
500|20	12.36	12.50	23.61	55.93	99.09
Average	1.15	1.17	2.20	5.23	9.24

The proposed IG algorithm was evaluated by comparing its efficiency with that of the hmgHS algorithm [9] which had in previous studies shown to have the best efficiency. The procedure presented for the local search step of the proposed IG algorithm was also evaluated by using a modified form of the IG algorithm (mIG) which differed only in its local search step. The local search step of the mIG lacked the insert-based local search part of that of the proposed IG algorithm. All these three algorithms were coded in the same structure.

As previously mentioned, we considered a time limit of $20.m.n$ millisecond in this evaluation. Due to the randomness of these algorithms, each Taillard instance was solved 10 times using each algorithm. Average RPD values of each group of the Taillard instances for each algorithm are shown in Table 4. It is seen that the proposed IG algorithm enjoys a higher efficiency than the others. This result has been confirmed using the Tukey HSD 95% confidence intervals. These results also indicate that the application of two consecutive pair-wise and insert-based local searches lead to the higher efficiency of the IG algorithm than when one pair-wise based local search is used.

Tab. 4. Effectiveness evaluation of the proposed IG algorithm

n|m	hmgHS	IG	mIG	Time (seconds)
20|5	0.06	0.02	0.02	2.00
20|10	0.01	0.01	0.03	4.00
20|20	0.01	0.00	0.01	8.00
50|5	1.78	0.70	0.71	5.00
50|10	1.13	0.57	0.73	10.00
50|20	0.66	0.42	0.57	20.00
100|5	3.70	0.63	0.80	10.00
100|10	2.59	0.61	0.84	20.00
100|20	1.68	0.43	0.62	40.00
200|10	3.25	0.61	0.79	40.00
200|20	2.03	0.48	0.59	80.00
500|20	2.03	0.39	0.42	200.00
Average	1.58	0.41	0.51	36.58

5. Conclusion

In this paper, we presented an IG algorithm for solving the BFSP with total flow time criteria. For generating an efficient initial solution for this algorithm, we developed the NEH-MK(k) that is obtained by modifying the steps 1 and 3 of the NEH [10] algorithm.

Computational results showed that for each $k \in \{0,2,5,10\}$, the efficiency and the execution time of the NEH-MK(k) increases with k. It was also found that all NEH-MK(k) heuristics, $k \in \{0,2,5,10\}$, have higher efficiencies than the NEH-WPT [9] heuristic. The results revealed that the proposed IG algorithm is more efficient than the hmgHS [9] algorithm. In the proposed IG algorithm, we used two consecutive pair-wise and insert-based local searches and we showed that it is more efficient than a pair-wise based local search.

The proposed concepts maybe used for developing efficient algorithms for solving similar problems such as the BFSP with other criteria. Also, with respect to the fact that the most computational burden of the proposed algorithms lies with the objective computation of the parent sequences' offspring, it may be interesting to develop rules that reject some of the offspring before their objective functions are computed.

References

[1] Wang, L., Pan, Q.K., Suganthan, P.N., Wang, W.H., & Wang, Y.M., "*A Novel Hybrid Discrete Differential Evolution Algorithm for Blocking Flow Shop Scheduling Problems*", Computers and Operations Research, Vol. 37, 2010, pp. 509-520.

[2] Hall, N.G., Sriskandarajah, C., "*A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process*", Operations Research, Vol. 44, 1996, pp. 510-525.

[3] Grabowski, J., Pempera, J., "*The Permutation Flow Shop Problem with Blocking. A Tabu Search Approach*", Omega, Vol. 35, 2007, pp. 302-311.

[4] Grabowski, J., Pempera, J., "*Sequencing of Jobs in Some Production System*", European Journal of Operational Research, Vol. 125, 2000, pp. 535-550.

[5] Rock, H., "*Some New Results in Flow Shop Scheduling*", Zeitschrift fur Operations Research, Vol. 28, 1984, pp. 1-16.

[6] McCormick, S.T., Pinedo, M.L., Shenker, S., Wolf, B., "*Sequencing in an Assembly Line with Blocking to Minimize Cycle Time*", Operations Research, Vol. 37, 1989, pp. 925-935.

[7] Ronconi, D.P., "*A Note on Constructive Heuristics for the Flow Shop Problem with Blocking*", International Journal of Production Economics, Vol. 87, 2004, pp. 39-48.

[8] Companys, R., Mateo, M., "*Different Behaviour of a Double Branch-and-Bound Algorithm on Fm|Prmu|Cmax and Fm|Block|Cmax Problems*", Computers and Operations Research, Vol. 34, 2007, pp. 938-953.

[9] Wang, L., Pan, Q.K., Tasgetiren, M.F., "*Minimizing the Total Flow Time in a Flow Shop with Blocking by using Hybrid Harmony Search Algorithms*", Expert Systems with Applications, Vol. 37, 2010, pp. 7929-7936.

[10] Nawaz, M., Enscore Jr, E.E., Ham, I., "*A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem*", Omega, Vol. 11, 1983, pp. 91-95.

[11] Caraffa, V., Ianes, S., Bagchi, T.P., Sriskandarajah, C., "*Minimizing Makespan in a Blocking Flow Shop using Genetic Algorithms*", International Journal of Production Economics, Vol. 70, 2001, pp. 101-115.

[12] Ribas, I., Companys, R., Tort-Martorell, X., "*An Iterated Greedy Algorithm for the Flow Shop Scheduling Problem with Blocking*", Omega, Vol. 39, 2011, pp. 293-301.

[13] Wang, L., Pan, Q.K., Tasgetiren, M.F., "*A Hybrid Harmony Search Algorithm for the Blocking Permutation Flow Shop Scheduling Problem*", Computers & Industrial Engineering, Vol. 61, 2011, pp. 76-83.

[14] Ruiz, R., & Stutzle, T., "*A Simple and Efective Iterated Greedy Algorithm for the Permutation Flow Shop Scheduling Problem*", European Journal of Operational Research, Vol. 177, 2007, pp. 2033-2049.

[15] Ronconi, D.P., Henriques, L.R.S., "*Some Heuristic Algorithms for Total Tardiness Minimization in a Flowshop with Blocking*", Omega, Vol. 37, 2009, pp. 272-281.

[16] Framinan, J.M., Leisten, R., "*An Eficient Constructive Heuristic for Flowtime Minimisation in Permutation Flow Shops*", Omega, Vol. 31, 2003, pp. 311-317.

[17] Kalczynski, P.J., Kamburowski, J., "*An Improved NEH Heuristic to Minimize Makespan in Permutation Flow Shops*", Computers and Operations Research, Vol. 35, 2008, pp. 3001-3008.

[18] Taillard, E., "*Benchmarks for Basic Scheduling Problems*", European Journal of Operational Research, Vol. 64, 1993, pp. 278-285.