An Efficient Iterated Greedy Algorithm for the Makespan Blocking Flow Shop Scheduling Problem

Nouha Nouri and Talel Ladhari

***Abstract*—We propose in this paper a Blocking Iterated Greedy algorithm (BIG) which makes an adjustment between two relevant destruction and construction stages to solve the blocking flow shop scheduling problem and minimize the maximum completion time (makespan). The greedy algorithm starts from an initial solution generated based on some well-known heuristic. Then, solutions are enhanced till some stopping condition and through the above mentioned stages. The effectiveness and efficiency of the proposed technique are deduced from all the experimental results obtained on both small randomly generated instances and on Taillard's benchmark in comparison with state-of-the-art methods.**

***Index Terms*—Blocking, flow shop, makespan, iterated greedy method.**

I. Problem Description

In the Blocking Flow Shop Scheduling Problem (BFSP), there is a finite set of N jobs that must be processed on M machines in the same order. Indeed, since there is no buffer storage between each consecutive pair of machines, intermediate queues of jobs waiting for their next process are not allowed. So, a job cannot leave its current machine till the next downstream machine is clear. This blocking state avoids progressing of other jobs on the blocked shop.

Furthermore, each job i $(i = 1, 2, ..., N)$ ready at time zero and requiring non-negative time p_{ij} as a processing delay has to be processed first on machine $M1$, then on machine $M2$ and so on till on machine Mm $(j = 1, 2, ..., M)$. That is the sequence in which the jobs are to be processed is identical for each machine. Besides, the processing of a given job at a machine cannot be interrupted once started. Each job can be processed only on one machine at a time and each machine can process at most one job at a time. Based on the above definitions, the final objective is to find out a sequence for processing all jobs on all machines so that its maximum completion time (makespan) is minimized. Formally, the BFSP aborted in this research is the $Fm|block|C_{max}$ in conformance with the classifications mentioned by Graham et al. [1]. The most popular eccentric work done on this problem is [2] who showed that the $F2|blocking|C_{max}$ instance may be reduced to a special case of the traveling salesman problem which may be solved in polynomial time using Gilmore and Gomory algorithm [3]. When the number of machines exceeds two $(m > 2)$, then the problem becomes strongly NP-hard [4]. The BFSP may be sketched in many real-life situations. We may cite the robotic cell [5], the iron and steel production [6], the manufacturing of concrete blocks and other.

As well, let $\Pi := (\pi_1, \pi_2, ..., \pi_N)$ be a possible solution for the BFSP, where π_i denotes the i^{th} job in the specific sequence; $d_{\pi_i,j}$ $(i = 1, 2, ..., N; j = 0, 1, 2, ..., M)$ defines the departure time of job π_i on machine j, where $d_{\pi_i,0}$ represents the time job π_i begins its processing on the first machine. The corresponding values of makespan of Π may then be calculated as $C_{max}(\Pi) = C_{\pi_N,M}(\Pi)$ in $O(nm)$, where $C_{\pi_i,M} = d_{\pi_i,M}$ is the completion time of job π_i on machine M that can be calculated generally using expressions presented in [7]. We choose in this work to refer to the method based on tails calculation to express the makespan of a given permutation as $C_{max}(\Pi) = f_{1,1}$ where $f_{i,j}$ defines the length of time between the latest loading time of operation o_{ij} and the end of the operations for $j : M, M-1, ..., 1$; and $f_{i,M+1}$ is the duration between the latest completion time of operation o_{iM} and the end of the operations [8]. Consequently, we obtain the following recursive equations:

$$f_{N,M+1} = 0$$

$$f_{N,j} = f_{N,j+1} + p_{Nj} \qquad j = M, M-1, ..., 2 \qquad (1)$$

$$f_{i,M+1} = f_{i+1,M} \qquad i = N-1, N-2..., 1 \qquad (2)$$

$$f_{i,j} = max\{f_{i,j+1} + p_{ij}, f_{i+1,j-1}\} \qquad i = N-1, ..., 1; \quad j = M, ..., 2 \qquad (3)$$

$$f_{i,1} = f_{i,2} + p_{i1} \qquad i = N, N-1, ..., 1 \qquad (4)$$

In the beyond recursion, the tails of the last job on every machine are calculated first, then the second last job, and so on up to the first job.

Due to the NP-hardiness of the BFSP, small number of methods have been proposed to solve it. They are

Manuscript received on February 23, 2016, accepted for publication on June 19, 2016, published on June 25, 2016.

Nouha Nouri is with the Ecole Supérieure des Sciences Economiques et Commerciales de Tunis, University of Tunis, Tunisia (e-mail: nouri.nouha@yahoo.fr).

Talel Ladhari is with the Ecole Supérieure des Sciences Economiques et Commerciales de Tunis, University of Tunis, Tunisia and College of Business, Umm Al-Qura University, Umm Al-Qura, Saudi Arabia (e-mail: talel_ladhari2004@yahoo.fr).

ranged from exact methods to meta-heuristic ones. Some of the most important techniques are briefly presented in the following. Solving the BFSP using exact methods has attracted few attention in comparison with other original flow shop problems [9], [10], [11]. In [12], a lower bounding schemes were exposed, next an exact method based on the Branch-and-Bound (B&B) technique which uses a new compounded lower bounds was developed in [13]. Besides, a double B&B algorithm using the reversibility property of the problem is proposed in [14]. As well, two MBIP models and one B&B algorithm were lately presented to solve to optimality the BFSP under the total completion time measure [15]. Certainly, exact methods are unable to solve the problem within a reasonable computational time. Therefore, this incapacity explains the necessity to employ heuristics and meta-heuristics. As constructive heuristics, we may cite the Profile Fitting (PF) [16] and the Nawaz-Enscore-Ham (NEH) technique [17]. Some priority rules and tie breaking strategies were proposed in [18]. In [19] and [20] the NEH-WPT heuristic and a constructive and a GRASP-based heuristics for the BFSP were introduced respectively. Basically, the NEH-WPT sorts all jobs in a non-decreasing order of the sum of their processing times on all machines.

Afterward, meta-heuristics algorithms appear as a complement to their counterparts heuristics. The (Ron) algorithm was presented in [13] regarding the blocking constraints, and a Tabu Search (TS) and an enhanced TS techniques were used by Grabowski et al. In [21], we locate an Iterated Greedy (IG) method based on the insertion stage of the NEH. Under the total flow time criterion, we cite the hybrid modified global-best Harmony Search (hmgHS) algorithm and the Discrete Artificial Bee Colony algorithm (DABC_D) technique exposed in [19] and [22], respectively. Under C_{max} criterion, an effective Revised Artificial Immune Systems (RAIS) algorithm is proposed in [23], a three-phase algorithm presented in [24], and a Discrete Particle Swarm Optimization algorithm (DPSO) with self-adaptive diversity control was treated in [25]. Subsequently, we refer to the Memetic Algorithm (MA) in [26], the Iterated Local Search algorithm (ILS) coupled with a Variable Neighborhood Search (VNS) in [27], and the Blocking Genetic Algorithm (BGA) and Blocking Artificial Bee Colony (BABC) algorithms in [28]. Experimental results demonstrated that both of the two later proposed algorithms are more efficient in finding better solutions than all other leading techniques.

Now, among meta-heuristics, we focused on the IG algorithm which is being applied to many scheduling problems and subsequently to flow shop variants [29]. It is simple and effective: the approach applies constructive methods iteratively to a selected solution and then uses an acceptance criterion to decide whether the obtained solution substitutes the old one. Indeed, a sequence of solution is obtained using some destruction and construction stages. The destruction phase removes some elements from one selected solution. Next, in the construction phase, a new solution is created by reconstructing a complete solution using a greedy constructive heuristic, which reinserts the removed elements in some order to form a new complete sequence. Facultatively, a local search algorithm may be added to boost the constructed solution.

The basic steps of the IG algorithm are given as shown in Table I. After presenting the problem background, the rest of this paper is organized as follows: In Section 2, the Blocking IG algorithm (BIG) is stated. In Section 3, the computational results and comparisons are provided, and Conclusions are made together with future research direction in Section 4.

TABLE I
PSEUDO-CODE OF IG ALGORITHM

Begin
Generate initial solution Π_0;
Apply local search to Π_0, and *Put* the modified solution into Π_s; % Optional
Repeat
Π_d= Destruction(Π_s);
Π_c= Construction(Π_d);
Π_l= Local search(Π_c); % Optional
Π_f= Acceptance criterion(Π_s,Π_l);
Until termination condition met
End

II. SOLVING THE BLOCKING FLOW SHOP PROBLEM BASED ON THE ITERATED GREEDY ALGORITHM

In this section, the details of the BIG proposed for the problem under discuss is introduced. We recall that the key components of all existing heuristics for the BFSP have been developed based on the NEH heuristic which is made up of two stages: the first stage is the creation of the preliminary sequence of the jobs, and next comes the iterative process of insertion of the resulting jobs depending on the initial sequence obtained in phase 1.

A. Initial solution

To generate the initial solution we have used the PF-NEH(x) heuristic as in [28]. However, instead of generating x solutions at the end of the heuristic we choose only the permutation with the minimum objective value. As well, we employ the insertion-based local search to produce a neighboring solution. In this local search, a job is removed from its original position and reinserted in all other possible places. The local technique is applied with a probability P_{ls}. Next, if the objective value is enhanced then the solution is replaced. The final permutation Π^s thus obtained is the seed sequence.

B. Destruction and construction stages

On the basis of an initial solution, the destruction phase is applied. This stage begins with a complete solution Π^s and then extracts $[q * \Pi^s]$ randomly chosen jobs from Π^s in

an iterative way. The degree of destruction q is in the range [0,1]. This creates two subsequences: the first one contains the removed jobs Π^r, and the second subsequence is the rest of the initial sequence obtained after removing some jobs Π^s.

Now, based on these resulting subsequences, in the construction phase a final solution Π^c is then reconstructed using a greedy constructive algorithm by reinserting the previously removed jobs in the order in which they were extracted.

The pseudo-codes of the destruction and construction steps are as in Table II and Table III.

TABLE II
PSEUDO-CODE OF DESTRUCTION STAGE (Π^s,q)

Begin
Stage 1: Set Π^r empty
Stage 2: Let $\Pi^q \leftarrow \Pi^s$
Stage 3: For $i = 1$ ***to*** $(q * |\Pi^q|)$ ***Do***
1) $\Pi^q \leftarrow$ remove a randomly selected job from Π^q
2) $\Pi^r \leftarrow$ include the removed job in Π^r
End

TABLE III
PSEUDO-CODE OF CONSTRUCTION STAGE (Π^q,Π^r)

Begin
Stage 1: Let $\Pi^c \leftarrow \Pi^q$
Stage 2: For $j = 1$ ***to*** $|\Pi^r|$ ***Do***
1) $\Pi^c \leftarrow$ best permutation obtained after inserting job π^r_j in all possible positions of Π^c
End

C. Acceptance criterion

Once a newly reconstructed solution has been obtained, an acceptance criterion is applied to decide wether it will replace the current incumbent solution or not. We consider the Simulated Annealing (SA) acceptance criteria that may be achieved by accepting worse solutions with a certain probability as used in [29], [30]. This acceptance criterion is used with a constant temperature value, which depends on the number of jobs, the number of machines, and on other adjustable parameter λ:

$$Tempt = \lambda * \frac{\sum_{i=1}^{N}\sum_{j=1}^{M} p_{ij}}{10 * M * N} \quad (5)$$

Let $Mksp(\Pi^s)$ and $Mksp(\Pi^c)$ be respectively the makespan values of the current incumbent solution and the new reconstructed solution. Also, let rand() be a function returning a random number sampled from a uniform distribution between 0 and 1.

If $Mksp(\Pi^c) \geq Mksp(\Pi^s)$ Then Π^c is accepted as the new incumbent solution if:

$$rand() \leq exp\{Mksp(\Pi^c) - Mksp(\Pi^s)/Tempt\} \quad (6)$$

D. Final BIG algorithm

Considering all previous subsections, the proposed BIG algorithm for the BFSP goes as in Table IV.

TABLE IV
PSEUDO-CODE OF BIG ALGORITHM

Begin
Stage 1: Set the parameters: P_{ls}, q, λ and MCN.
Stage 2: Obtain the initial solution using the PF-NEH(x) heuristic. Depending on the local probability rate P_{ls}, improve the initial solution using the insertion-based local search technique. Let the final permutation Π^s be the seed sequence.
Stage 3: Let $\Pi^* = \Pi^s$
Stage 4:
While termination condition is not met ***Do***
1) Π^q= Destruction-phase(Π^s, q)
2) Π^c= Construction-phase(Π^q)
3) $\Pi^{c'}$= Local-phase(Π^c, P_{ls})
4) If $Mksp(\Pi^{c'}) < Mksp(\Pi^s)$ Then
   a) $\Pi^s := \Pi^{c'}$
   b) If $Mksp(\Pi^s) < Mksp(\Pi^*)$ Then
      i) $\Pi^* := \Pi^s$
5) Else If $(rand() \leq exp\{Mksp(\Pi^s) - Mksp(\Pi^{c'})/Tempt\})$ Then
   a) $\Pi^s := \Pi^{c'}$
Stage 5: Return the best solution found Π^*
End

III. COMPUTATIONAL RESULTS

In the following, to confirm the effectiveness and competitiveness of BIG, its performances are compared against some leading methods in the literature. As usually done, we have used the Taillard instances [31] to test our technique. This benchmark include 120 problems of multiple sizes arranged into 12 subsets. Each subset entails ten instances with equal size (20*5, 20*10, 20*20, 50*5, 50*10, 50*20, 100*5, 100*10, 100*20, 200*10, 200*20, and 500*20) where the first number define the job size and the second one represent the machine size.

Each instance is independently run 10 times and in each run we compute the percentage relative difference (PRD) using the following expression:

$$RPD(A) = \frac{(Mksp^A - Mksp^{Min}) \times 100}{Mksp^{Min}} \quad (7)$$

where, $Mksp^A$ defines the value of the makespan reached by the BIG algorithm; and $Mksp^{Min}$ defines the minimum mekespan value obtained among all the compared algorithms.

The BIG algorithm is coded in $C++8.0$ and the experiments are executed on an Intel Pentium IV 2.4 GHz PC with 512 MB of memory.

The final experimental setup is given in Table V where the main purpose of the experiment was to compare the optimization performances of the algorithm under various system conditions.

TABLE V
THE EXPERIMENTAL SETUP

Factors	P_{ls}	q	λ	MCN
Values	0.2	0.3	2	100

A. Results on randomly generated instances

Before testing BIG algorithm on benchmark sets, the computational experiments have been at first carried out on a set of randomly generated instances obtained following the procedure explained in Taillard.

In our tests, the problem sizes are determined by varying the number of jobs and machines from 10 jobs and 3 machines to 100 jobs and 10 machines as was the case in [28].

This choice is fixed such to make comparison between BIG and BGA algorithm under this type of instances. Next, the C_{max} values of the best-found solutions for these generated instances were memorized for each of the compared heuristics.

A statistic for the solution quality for each set is given (Average RPD (ARPD)) as in Table VI. According to

TABLE VI
ARPD ON RANDOMLY GENERATED INSTANCES

Inst	BIG	BGA
10×3	0,000%	0,000%
10×5	0,000%	0,000%
10×7	0,000%	0,000%
20×3	0,000%	0,000%
20×5	0,000%	0,000%
20×7	0,000%	0,000%
50×3	0,000%	0,000%
50×5	0,007%	0,026%
50×7	0,000%	0,013%
70×3	0,010%	0,005%
70×5	0,005%	0,063%
70×7	0,000%	0,103%
100×3	0,007%	0,026%
100×5	0,006%	0,043%
100×7	0,000%	0,077%
Avrg	0,002%	0,024%

the above table, the proposed algorithm is more likely to get better solutions than BGA which is outperformed. For small instances, the two algorithms behave in the same way. Difference is observed by increasing the number of jobs.

B. Comparing BIG with leading heuristics

In this subsection, we enlarge the domain of comparison and consider the BIG versus IG [21], MA [26], RAIS [23], and BGA [28] algorithms.

From Table VII, we can observe that the proposed BIG gives the best performance in terms of the overall solution quality, since it yields the minimum overall mean ARPD value equal to $0,041\%$, which is much better than those by the IG (0.744%), MA (0.174%), RAIS (0.426%), and BGA (0.055%).

More specifically, the BIG gives much better APRD than all compared heuristics and improves 86 out of 120 best-known solutions of Taillard's instances for the BFSP with the makespan criterion. The worst results are given by the IG [21].

Indeed, BIG algorithm behaves much more effective than the BIG algorithm as the size of instances increases. So, regardless its simplicity, we may assert that the BIG algorithm is an efficient heuristic in solving the BFSP and so may be used as a basis of comparison for future research.

TABLE VII
ARPD ON TAILLARD INSTANCES

Inst	BIG	MA	IG	RAIS	BGA
20×5	0,000%	0,000%	0,000%	0,000%	0,000%
20×10	0,000%	0,000%	0,000%	0,000%	0,000%
20×20	0,000%	0,000%	0,000%	0,000%	0,000%
50×5	0,022%	0,238%	0,322%	0,129%	0,032%
50×10	0,003%	0,199%	0,402%	0,203%	0,025%
50×20	0,005%	0,046%	0,267%	0,263%	0,030%
100×5	0,032%	0,572%	0,936%	0,109%	0,050%
100×10	0,027%	0,325%	1,032%	0,141%	0,058%
100×20	0,004%	0,245%	0,962%	0,242%	0,032%
200×10	0,000%	0,062%	0,631%	0,299%	0,015%
200×20	0,394%	0,052%	1,576%	0,936%	0,411%
500×20	0,001%	0,349%	2,805%	2,789%	0,011%
Avrg	0,041%	0,174%	0,744%	0,426%	0,055%

IV. CONCLUSION AND FUTURE WORK

In our study, BIG algorithm is proposed to solve the BFSP under makespan measure. This greedy method is very simple, and hybridized with a form of local search, enhanced much more the solutions quality.

The algorithm is developed to solve both randomly generated instances and a number of test problems (Taillard instances). The experiment results attest that BIG is better than other leading algorithms on all group instances specifically on high dimensional problems.

In the future, we will hybridize our technique using some hybrid evolutionary heuristics such as SA to improve its performance and design some better NEH heuristic variant to improve its efficiency.

REFERENCES

[1] R. Graham, E. Lawler, J. Lenstra, and K. Rinnooy, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–362, 1979.

[2] S. Reddi and C. Ramamoorthy, "On the flow-shop sequencing problem with no wait in process," *Operational Research Quarterly*, vol. 3, pp. 323–31, 1972.

[3] P. Gilmore and R. Gomory, "Sequencing a one state variable machine: A solvable case of the traveling salesman problem," *Operations Research*, vol. 5, pp. 655–79, 1964.

[4] N. Hall and C. Sriskandarajah, "A survey of machine scheduling problems with blocking and no-wait in process," *Operations Research*, vol. 44, pp. 510–25, 1996.

[5] S. Sethi, C. Sriskandarajah, G. Sorger, J. Blazewicz, and W. Kubiak, "Sequencing of parts and robot moves in a robotic cell," *International Journal of Flexible Manufacturing Systems*, vol. 4, pp. 331–358, 1992.

[6] H. Gong, L. Tang, and C. Duin, "A two-stage flowshop scheduling problem on batching machine and a discrete machine with blocking and shared setup times," *Computers and Operations Research*, vol. 37, pp. 960–4, 2010.

[7] M. Pinedo, *Scheduling: theory, algorithms, and systems*. USA: Pretice Hall, 2008.

[8] L. Wang, Q. Pan, P. Suganthan, W. Wang, and Y. Wang, "A novel hybrid discrete differential evolution algorithm for blocking flowshop scheduling problems," *Computers and Operational Research*, vol. 3, pp. 509–20, 2010.

[9] E. Levner, "Optimal planning of parts machining on a number of machines," *Automation and Remote Control*, vol. 12, pp. 1972–8, 1969.

[10] I. Suhami and R. Mah, "An implicit enumeration scheme for the flowshop problem with no intermediate storage," *Computers and Chemical Engineering*, vol. 2, pp. 83–91, 1981.

[11] S. Karabati and P. Kouvelis, "Cycle scheduling in flow lines: modeling observations, effective heuristics and a cycle time minimization procedure," *Naval Research Logistics*, vol. 2, pp. 211–31, 1996.

[12] D. Ronconi and V. Armentano, "Lower bounding schemes for flowshops with blocking in-process," *Journal of the Operational Research Society*, vol. 11, pp. 1289–97, 2001.

[13] D. Ronconi, "A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking," *Annals of Operations Research*, vol. 1, pp. 53–65, 2005.

[14] R. Companys and M. Mateo, "Different behaviour of a double branch-and-bound algorithm on $fm|prmu|cmax$ and $fm|block|cmax$ problems," *Computers and Operations Research*, vol. 34, pp. 938–953, 2007.

[15] G. Moslehi and D. Khorasanian, "Optimizing blocking flowshop scheduling problem with total completion time criterion," *Computers and Operations Research*, vol. 40, pp. 1874–1883, 2013.

[16] S. McCormick, M. Pinedo, S. Shenker, and B. Wolf, "Sequencing in an assembly line with blocking to minimize cycle time," *Operations Research*, vol. 37, pp. 925–935, 1989.

[17] M. Nawaz, J. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, pp. 91–95, 1983.

[18] R. Companys, I. Ribas, and M. Mateo, "Note on the behaviour of an improvement heuristic on permutation and blocking flow-shop scheduling," *International Journal of Manufacturing Technology and Management*, vol. 20, pp. 331–57, 2010.

[19] L. Wang, Q. Pan, and M. Tasgetiren, "Minimizing the total flow time in a flowshop with blocking by using hybrid harmony search algorithms," *Expert Syst. Appl*, vol. 12, pp. 7929–7936, 2010.

[20] D. Ronconi and L. Henriques, "Some heuristic algorithms for total tardiness minimization in a flowshop with blocking," *Omega*, vol. 2, pp. 272–81, 2009.

[21] I. Ribas, R. Companys, and X. Tort-Martorell, "An iterated greedy algorithm for the flowhsop scheduling problem with blocking," *Omega*, vol. 3, pp. 293–301, 2011.

[22] G. Deng, Z. Xu, and X. Gu, "A discrete artificial bee colony algorithm for minimizing the total flow time in the blocking flow shop scheduling," *Chinese Journal of Chemical Engineering*, vol. 20, pp. 1067–1073, 2012.

[23] S. Lin and K. Ying, "Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm," *Omega*, vol. 41, pp. 383–389, 2013.

[24] Q. Pan and L. Wang, "Effective heuristics for the blocking flowshop scheduling problem with makespan minimization," *Omega*, vol. 2, pp. 218–29, 2012.

[25] X. Wang and L. Tang, "A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking," *Applied Soft Computing*, vol. 12, pp. 652–662, 2012.

[26] Q. Pan, L. Wang, H. Sang, J. Li, and M. Liu, "A high performing memetic algorithm for the flowshop scheduling problem with blocking," *IEEE Transactions on Automation Science and Engineering*, vol. 10, pp. 741–756, 2013.

[27] I. Ribas, R. Companys, and X. Tort-Martorell, "An efficient iterated local search algorithm for the total tardiness blocking flow shop problem," *International Journal of Production Research*, vol. 51, pp. 5238–5252, 2013.

[28] N. Nouri and T. Ladhari, "Minimizing regular objectives for blocking permutation flow shop scheduling: Heuristic approaches," in *GECCO 2015*, 2015, pp. 441–448.

[29] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, pp. 2033–49, 2007.

[30] ——, "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives," *European Journal of Operational Research*, vol. 187, pp. 1143–1159, 2008.

[31] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, pp. 278–85, 1993.