2021 11th International Conference on Information Science and Technology (ICIST) | 978-1-6654-1266-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICIST52614.2021.9440648

# A quick and effective iterated greedy algorithm for energy-efficient hybrid flow shop scheduling problem with blocking constraint

1st Hao-xiang Qin
*School of Computer Science*
*Liaocheng University*
Liaocheng, China
987352978@qq.com

2nd* Yuyan Han
*School of Computer Science*
*Liaocheng University*
Liaocheng, China
hanyuyan@lcu-cs.com

3rd Junqing Li
*School of Computer Science,*
*Shandong Normal University*
Jinan, China
lijunqing@lcu-cs.com

4th Hongyan Sang
*School of Computer Science*
*Liaocheng University*
Liaocheng, China
sanghongyan@lcu-cs.com

5th Qingda Chen
*School of Computer Science*
*Liaocheng University*
Liaocheng, China
chenqingda@lcu-cs.com

6th Leilei Meng
*School of Computer Science*
*Liaocheng University*
Liaocheng, China
mengleilei@lcu-cs.com

7th Biao Zhang
*School of Computer Science*
*Liaocheng University*
Liaocheng, China
zhangbiao@lcu-cs.com

***Abstract*—With the continuous development of national economy, the problems of energy consumption are becoming more and more prominent. Similarly, the energy efficient scheduling problem in manufacturing has attracted much attention of the researchers. Some flow-shop scheduling problems have made progress on energy saving, but the research on hybrid flow shop with blocking constraint(BHFSP) is very few. Due to its NP-hard characteristic, in this paper, we suggest a mathematical model of BHFSP and a meta heuristic method named quick iterative greedy algorithm (IGQ) to solve this problem. Compared with other existing IG algorithms, for BHFSP, three main contributions of the IGQ are described as follows: First, we use the MME to initialize the solution. Second, we develop a quick local perturbation strategy to ensure the convergence of the algorithm. Third, a quick global perturbation strategy is proposed to guarantee the diversity of the algorithm. Computational results and comparisons verify the effectiveness of proposed algorithm IGQ for solving BHFSP.**

***Keywords—BHFSP, energy-efficient, quick, perturbation strategy***

## I. INTRODUCTION

The hybrid flow shop scheduling problem (HFSP) has been widely studied by many researchers [1]. For HFSP, there are different stages of processing jobs, in which has a different number of machines, a job sequence to be scheduled must passes through all stages. In recent years, due to the practicability of HFSP, many scholars have made a lot of achievements on HFSP.

For the HFSP with single objective, Li et al. [2] proposed a hybrid variable neighborhood search(HVNS) and an improved artificial bee colony (DABC)algorithm [3] to solve this problem. Liu et al. [4] proposed a two stage flow shop with deteriorating effect and different sizes of jobs. In multi-objective of HFSP, an iterated local search(ILS) algorithm is proposed to minimize the total energy costs, peak load and makespan [5]. Zhang et al. employed a three-stage multi-objective algorithm to solve HFSP with energy-efficient [6]. In this paper, we consider the constraint of adding blocking constraint to HFSP problem.

IG algorithm is a simple and effective algorithm. It firstly used to optimize the permutation flow shop scheduling problem (PFSP) [7], unlike other swarm intelligence algorithms, it generates only one solution at each iteration. Firstly, it generates an initial solution, and then continuously generates better solution based on the current one by using the iterative methods. In this paper, an improved iterated greedy algorithm (IG) algorithm is proposed to reduce the energy consumption of BHFSP.

The main contributions of this paper are as follows:

(1) It has a simple code structure and can reduce the time complexity of the local search from $O(n^3)$ to $O(n^2)$.
(2) In addition to these jobs that need to be removed in the destruction phase, there is no need for extra adjustment parameters.
(3) Compared with the inserted local search, the proposed algorithm is more simple and can better prevent the solution from falling into the local optimum.

## II. ENERGY-EFFICIENT HYBRID FLOW SHOP SCHEDULING PROBLEM WITH BLOCKING CONSTRAINT

Comparing with the HFSP, there are no intermediate buffers existing for any adjacent machines for BHFSP. Once a job is completed at the current stage, it must be transferred to the next stage to process. Thus, it is necessary to consider the blocking and idle time of each machine. In the following two main steps, one is the machine assignment for each job, another is allocation the collections of jobs on selected machines. Except of these constraints, the BHFSP is subject to the following constrains.

1) At a specific stage, a job is operated by exactly one

machine at any time.

2) Each machine can only process at most one job and each job can only be processed on at most one machine.
3) All jobs should be continuously processed not be preempted and interrupted.
4) If there is no available machine at next stage, the job will be blocked on current machine until the machine of next stage is available.
5) Both transportation time and setup time are included in its processing time.

The mathematical notations are summarized as follows:

$J$ : The total number of jobs, indexed by $j = 1,2,\ldots,J$.

$S$ : The total number of stages, indexed by $s = 1,2,...,S$.

$M_s$ : The number of parallel machines at stage $s$ and $M_s = \{1,...,m,...,M_s\}$.

$M_{s,m}^{Idle}$ : The available time of machine $m$ at stage $s$.

$p_{s,j}$ : The processing time of job $j$ at stage $s$.

$B_{s,j}$ : The beginning time of job $j$ at stage $s$.

$E_{s,j}$ : The ending time of job $j$ at stage $s$.

$Block_{s,j}$ : The blocking time of job $j$ at stage $s$.

$m^{before}$ : Record the number of machine with the job $j$ processing at the preceding stage.

$m^{current}$ : The number of machine with the job $j$ processing at the current stage $s$.

$P_{s,m}^{Process}$ : Power of machine $m$ at stage $s$ processing a job per unit time.

$P_{s,m}^{Idel}$ : Power of machine $m$ at stage $s$ staying at the idle state per unit time.

$P_{s,m}^{Block}$ : Power of machine $m$ at stage $s$ staying at the block state per unit time.

To further illustrate the difference between HFSP and BHFSP, a simple example is given in Fig.1.

As shown in Fig.1, if these is no intermediate buffer between any two consecutive machines, the completion time will be 20. Whereas, when there are sufficient buffers exist for any adjacent machines, the completion time is reduced to 16. Thus, in this example, the blocking constraint can increase the completion time. The reason is that some jobs are blocked on current machine that the machine of next stage is not available for processing, for instance, due to the machine 3 and 4 are not available at 5 moments, job 3 is blocked on machine 1 (noted by shadow rectangular).

The blocking constraint increases the total completion time. Thus, we are encouraged to study BHFSP to reduce the blocking time. We construct the mathematical model of BHFSP to the further explain this problem.

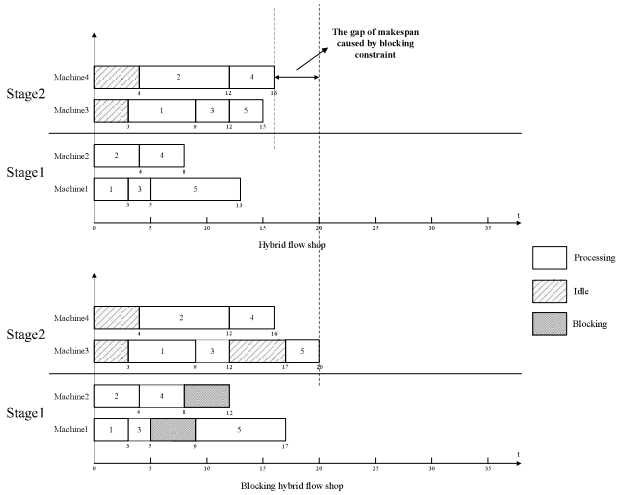


Fig. 1. The Gantt comparison of HFSP with and without blocking constraint

$$M_{s,m^{\text{current}}}^{Idle} = \min\{M_{s,1}^{Idle},...,M_{s,m}^{Idle},\text{K},M_{s,M_s}^{Idle}\},$$
$$s = 1, m, m^{current} \in M_1 \quad (1)$$

$$B_{s,j} = M_{s,m^{\text{current}}}^{Idle}, s = 1, j = 1,2...,J \quad (2)$$

$$E_{s,j} = B_{s,j} + p_{s,j}, s = 1, j = 1,2...,J \quad (3)$$

$$m^{before} = m^{\text{current}}, m^{before} \in M_1, m^{current} \in M_1 \quad (4)$$

$$M_{s,m^{\text{current}}}^{Idle} = \min\{M_{s,1}^{Idle},...,M_{s,m}^{Idle},\text{K},M_{s,M_s}^{Idle}\}, \quad (5)$$
$$s = 2,...S, m, m^{current} \in M_s$$

$$B_{s,j} = \max\{M_{s,m^{current}}^{Idle}, E_{s-1,j}\}, s = 2,...S, \quad (6)$$
$$j = 1,2...,J, m^{current} \in M_s$$

$$E_{s-1,j} = B_{s,j}, s = 2,...S, j = 1,2...,J \quad (7)$$

$$M_{s-1,m^{before}}^{Idle} = E_{s-1,j}, s = 2,...S, \quad (8)$$
$$j = 1,2...,J, m^{before} \in M_{s-1}$$

$$E_{s,j} = B_{s,j} + p_{s,j}, s = 2,...S, j = 1,2...,J \quad (9)$$

$$M_{s,m^{\text{current}}}^{Idle} = E_{s,j}, s = 2,...S, \quad (10)$$
$$j = 1,2...,J, m^{current} \in M_s$$

$$m^{before} = m^{\text{current}}, m^{before} \in M_s, m^{current} \in M_s \quad (11)$$

Except for the objective of makespan, the energy-efficient plays a key role in the practical factory production. We know that the energy consumption exists in any stages, i.e., the processing stage, the blocking stage, the idle stage. Furthermore, different scheduling job sequences may emerge different length of idle time and blocking time that lead to increase energy consumption. Thus, we not only consider the energy consumption of processing time, but also consider the energy consumption of the blocking and idle time. The objective of energy consumption is given as follows.

***Objective*:**

**Min** $TEC = EC_P + EC_B + EC_I$

(12)

$$EC_P = \sum_{s\in\{1,...,S\}} \sum_{m\in M_s} \sum_{j\in\{1,2,...,J\}} P_{s,m}^{Process} p_{s,j} \quad (13)$$

$$EC_B = \sum_{s\in\{2,...,S\}} \sum_{m^{before}\in M_{s-1}} \sum_{m\in M_s} \sum_{j\in\{1,2,...,J\}} (M_{s,m}^{Idle} - E_{s-1,j}) \times P_{s-1,m^{before}}^{Block} \quad (14)$$

$$EC_I = \sum_{s\in\{2,...,S\}} \sum_{m\in M_s} \sum_{j\in\{1,2,...,J\}} (E_{s-1,j} - M_{s,m}^{Idle}) \times P_{s,m}^{Idle} \quad (15)$$

where TEC is the total energy consumption, $EC_P$ shows the energy consumption when machines process jobs, $EC_B$ represents the energy consumption that the jobs are blocked on machines, $EC_I$ reflects the energy consumption when machines stay at the idle state.

## III. PROPOSED QUICK ITERATED GREEDY ALGORITHM

### *A. Basic iterated greedy algorithm*

The existing IG mainly includes five parts: initialization, destruction, construction phase, iterative improvement, and acceptance criterion. Firstly, it uses a heuristic NEH to produce an initial solution, then an iterative improvement is used to improve the performance of obtained solution by continuous insertion. In destruction phase of this procedure, the job sequence $\pi=(\pi_1,\pi_2,...,\pi_J)$ is partially destroyed, the number of these removed jobs are controlled by parameter d. When the phase is finished, we obtain two partial job sequence $\pi^{Remove}$ and $\pi^{origin}$, where $\pi^{Remove}$ includes the removed d jobs and $\pi^{origin}$ consists of the rest jobs. In construction phase, every job in $\pi^{Remove}$ is inserted into every possible slots of $\pi^{origin}$ to test, the best slot that satisfies the optimization objective value will be admitted until all jobs in $\pi^{Remove}$ are tested. After that, the same local search algorithm as before is used again to improve the convergence of obtained solution. Finally, with the acceptance criterion performing, besides checking if it is a new best permutation, a simulated annealing algorithm is applied to determine whether select the improved job sequence to replace the current sequence or not. The procedure of basic IG is given as follows.

**Algorithm 1 Basic iterated greedy algorithm**

**Begin**

Set parameters $d$, termination criterion, job sequence, $\pi=(\pi_1,\pi_2,...,\pi_J)$.

$\pi=NEH(\pi)$

$\pi= IterativeImprove\ (\pi)$

$\pi^{best}=\pi$

**While**(termination criterion is not satisfied) **do**

$\pi^{origin}=\pi$

$\pi^{Remove}=Destruction(\pi^{origin},d)$, $\pi^{origin}=\pi^{origin} \setminus \pi^{Remove}$

$\pi^{origin} = Construction(\pi^{origin},\pi^{Remove})$

$\pi^{temp} = IterativeImprovement\ (\pi^{origin})$

$\pi = AcceptanceCriterion\ (\pi^{temp},\ \pi^{best},\ \pi)$

**Endwhile**

**Return** $\pi^{best}$

**End**

### *B. The proposed quick iterated greedy algorithm*

In this paper, we developed a quick iterated greedy algorithm (IGQ) to solve the BHFSP. Based on this problem, a heuristic method MME (MinMax combining NEH (Nawaz, Enscore and Ham)) is employed to generate an initial solution by reducing the critical path length. And then, in order to maintain a balance between strengthening convergence and achieving more rapid evolution, we proposed a quick local perturbation strategy to increase the solution performance with continuous updating iteration. In traditional IG, the simulated annealing acceptance criterion is utilized to ensure the diversity of the solutions. In this paper, we developed a global perturbation strategy which is put into the simulated annealing strategy. Fig. 2 and Algorithm 2 show the basic framework of proposed IGQ.

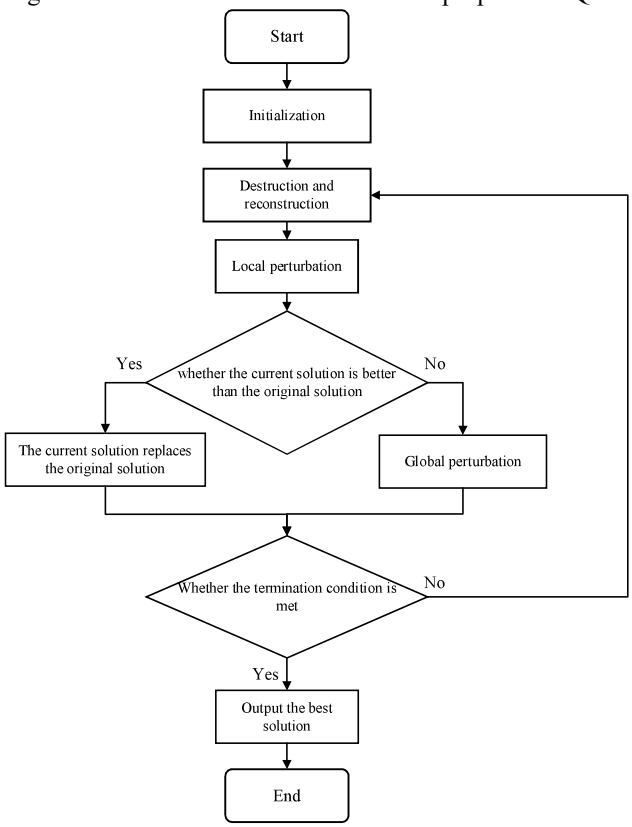


Fig. 2. The framework of proposed IGQ

**Algorithm 2 Proposed quick iterated greedy algorithm**

**Input**: $d$, termination criterion, Sequence $\pi=(\pi_1,\pi_2,...,\pi_J)$

**Output**: $\pi^{best}$

**Begin**

$\pi = MME(\pi)$ /*heuristic algorithm which we used*/

$\pi^{best} = \pi$

**While**(termination criterion is not satisfied) **do**

$\pi^{origin} = \pi$

$\pi^{origin}, \pi^{Remove} = Destruction(\pi^{origin}, d)$

$\pi^{origin} = Construction(\pi^{origin}, \pi^{Remove})$

$\pi^{temp} = \pi^{origin}$

**If** ( $\pi^{temp}$ better than $\pi^{best}$ )

$\pi^{best} = \pi^{temp}$

**EndIf**

***The following parts are the main innovation points of this paper***

$\pi^{temp} = PerturbationStrategy(\pi^{temp})$ /*based on swap*/

**If** ( $\pi^{temp}$ better than $\pi$ )

$\pi = \pi^{temp}$

**If** ( $\pi$ better than $\pi^{best}$ )

$\pi^{best} = \pi$

**EndIf**

**EndIf**

**Else**

$\pi^{temp} = PerturbationStrategy(\pi^{temp})$ /*based on Half Swap*/

$\pi = \pi^{temp}$

**If** ( $\pi$ better than $\pi^{best}$ )

$\pi^{best} = \pi$

**EndIf**

**EndIf**

**Endwhile**

**End**

*C. Solution encoding and decoding*

For the HFSP, the permutation-based representation is widely used for solution encoding. Thus, in BHFSP we also apply an n-dimensional integer sequence $\pi=(\pi_1, \pi_2, ...\pi_j, ..., \pi_J)$ to represent a solution, where $\pi_j$ denotes a job and $J$ represents the number of a job sequence.

*D. Initial solution*

In IG, a good initial solution can make the algorithm generate better performance, so as to achieve the goal, the application of heuristic algorithm becomes very important. In single-objective optimization of flow shop problems, MME [8], with a shortest critical path to shorten the blocking time of jobs has been successfully applied. To the best knowledge of us, in BHFSP, few people used the MME to complete the initialization operation. Thus, in this paper, MME is utilized to solve the BHFSP. Algorithm 3 depicts the step of MME.

**Algorithm 3 MME heuristic method**

**Input:** sequence $\pi=(\pi_1, \pi_2, ..., \pi_J)$ , parameter $\eta$ , unscheduled job Set $U$

**Output:** $\pi=(\pi_1, \pi_2, ..., \pi_J)$

**Begin**

Calculate $PTime_j = \sum_{s=1}^{S} p_{s,j}$ , $j \in J$

Select $\pi_j$ with the shortest $PTime_j$ as $\pi_1$ and assign it at the first position of job sequence

Select $\pi_J$ with the second shortest $PTime_j$ as $\pi_J$ and place it at the last position in the job sequence

$U=U-\{\pi_1, \pi_J\}$

Compute $Block_j = \sum_{s=1}^{S} Block_{s,j}$ , $j \in J$

**For** $i = 2$ **to** $J-1$

**For** $j = 1$ **to** $J$

**If** ( $\pi_j \notin U$ )

Compute $f_j = \eta * Block_j + (1-\eta) * PTime_j$

**EndIf**

**EndFor**

Find the smallest $f_j$ of $\pi_j$ and take it as $\pi_i$ , then insert it into the position $i$ ,

$U=U-\{\pi_i\}$

**EndFor**

Take out the first two jobs $\pi_1$ and $\pi_2$ , arrange them to form two partial sequence $\{\pi_1, \pi_2\}$ and $\{\pi_2, \pi_1\}$ , then compute the objective function value (Energy consumption)

The sorted sequence with lower objective function value is selected as $\pi^{best}$, $k = 2$

Extract the $(k+1)th$ job from $\pi$ and test it in all possible slots of $\pi^{best}$ , Select the partial sort with the smallest target value as the new $\pi^{best}$ ,until $k = J$ :all the other jobs are tested

$\pi = \pi^{best}$

**End**

*E. Destruction and construction phase*

The task of these two phases is to construct a completed and feasible scheduling sequence. The destruction of job sequence, in which $\pi^{Remove}$ is generated by randomly extracting $d$ jobs from job sequence and put them into $\pi^{Remove}$ one by one. $\pi^{origin}$ consisted of the rest jobs. Then these jobs in $\pi^{Remove}$ are inserted into all positions of $\pi^{origin}$ .The best one sequence is selected to replace the current best sequence. The destruction phase and construction phase are shown in Algorithm 4.

**Algorithm 4 Destruction and construction phase**

**Input:** $\pi=(\pi_1, \pi_2, ..., \pi_J)$ , parameter $d$

**Output:** $\pi^{origin}$

**Begin**

$count = 1$, $\pi^{Remove} = \Phi$, $\pi^{origin} = \pi$

**While** ( $count <= d$ ) **do**

$num = rand()\%J + 1$

**If**( $\pi_{num}^{origin}$ is not selected)

Extract $\pi_{num}^{origin}$ from $\pi^{origin}$ and put it into $\pi^{Remove}$

$\pi^{origin} = \pi^{origin} \setminus \{\pi_{num}^{origin}\}$, $count++$

**EndIf**

**Endwhile**

**For** $j = 1$ **to** $d$

$\pi_j = \pi_j^{Remove}$

**For** $p = 1$ **to** $J - d$ // $p$ is the insertion position

insert $\pi_j$ in the $p^{th}$ position of $\pi^{origin}$

calculate the energy consumption value, $Min_{EC}$

**EndFor**

select the position $p$ with minimal $Min_{EC}$, and insert $\pi_j$ into the p*th* of $\pi^{origin}$

**EndFor**

**End**

*F. The quick local perturbation strategy*

In the existing IG algorithms, the local search strategy is based on the insertion operation, such as the local improved insertion operation. In a given series of jobs, the number of n jobs should be selected first and the time complexity is O(n), Then, after inserting into all the positions, a total of n-1 insertion operations are required, so the time complexity becomes O(n(n-1)). Finally, after finding the position that minimizes the corresponding objective value, the selected position and all the jobs after the selected position are moved, assuming that the selected position is p, then the number of moving jobs becomes n-p+1, so the time complexity of the whole local search strategy is O(n(n-1)(n-p+1)). Moreover, if the value of objective is improved after all the above operations, there is a need to re execute this step until there is no way to improve, which will greatly improve the execution time. Thus, in this paper, we abandon the traditional IG local search strategy which will be improved forever if it is improved, and use the quick local perturbation strategy, which time complexity is only O(n), n is the number of iterations, each iteration contains a swap operation.

The quick local perturbation strategy, denoted as IGQ, is as follows:(1) Randomly select two different jobs. (2) exchange these two jobs and obtain a new sequence $\pi^{temp2}$. If $\pi^{temp2}$ is better than $\pi^{temp}$, $\pi^{temp2}$ replaces $\pi^{temp}$ .Otherwise, $\pi^{temp}$ remains unchanged. (3) Continue the above steps (1), (2) until the termination condition is met. The local perturbation strategy based on swap operator is given in Algorithm 5.

**Algorithm 5 The local perturbation strategy**

**Input:** $\pi^{temp}$

**Output:** $\pi^{temp}$

**Begin**

**For** $j = 1$ **to** $J*J$ // the number of iterations

$\pi^{2} = \pi$

Randomly select two different jobs in $\pi^{temp2}$ to swap with each other

**If**( $\pi^{temp2}$ better than $\pi^{temp}$ )

$\pi^{temp} = \pi^{temp2}$

**EndIf**

**EndFor**

**End**

*G. The quick global perturbation strategy*

In traditional IG algorithm, simulated annealing acceptance criterion is used to improve the diversity of the solution. In order to expand the search neighborhood of the current solution, we suggested a quick global perturbation strategy based on half-swap, denoted as $IG_{half\text{-}swap}$, in which time complexity is also $O(n^2)$, and integrate it into the simulated annealing stage. In this way, we can find a better solution which is close to the global optimal solution more quickly. After the disturbance operation, the acceptance criterial is performed. If the new solution is better than the best one, it will replace the best solution. Algorithm 6 illustrates the quick global perturbation strategy based on half-swap.

**Algorithm 6 The quick global perturbation strategy**

**Input:** $\pi^{temp}$

**Output:** $\pi^{temp}$

**Begin**

The sequence $\pi^{temp}$ is divided equally into two subsequences $\pi^{Front}$, $\pi^{Back}$

**If**(The $\pi^{Back}$ is better than $\pi^{Front}$ )

$\pi^{Front\_temp} = \pi^{Front}$

**For** $p = 1$ **to** $Sizeof(\pi^{Front\_temp})$ /*select these jobs in order */

**For** $j = 1$ **to** $Sizeof(\pi^{Front\_temp})$ /* swap with other jobs*/

**If**( $\pi_p^{Front\_temp}$ is not equal to $\pi_j^{Front\_temp}$ )

Swap $\pi_p^{Front\_temp}$ with $\pi_j^{Front\_temp}$

**EndIf**

**If**( $\pi^{Front\_temp}$ is better than $\pi^{Front}$ )

$\pi^{Front} = \pi^{Front\_temp}$

Merge $\pi^{Front}$ and $\pi^{Back}$ , noted the new sequence as $\pi^{temp\_new}$

**If**( $\pi^{temp\_new}$ is better than $\pi^{temp}$ )

$\pi^{temp} = \pi^{temp\_new}$

**EndIf**

**EndIf**

**EndFor**

**EndFor**

**ElseIf** (The energy consumption of $\pi$ is smaller than $\pi^{Back}$ )

Similar to the steps performed above.

**EndIf**

**End**

## IV. EXPERIMENTAL RESULTS AND ANALYSIS

### A. Parameter settings

The different number of stages and jobs can be combined to form different instances. We set the number of jobs as $J$ and the number of stages as $S$, where $J \in \{20, 40, 60, 80, 100, 200, 300\}$ and $S \in \{5, 10\}$. For each $J \times S$ combination, ten instances are produced, so the test instances number is $7 \times 2 \times 10 = 140$. The processing times are generated uniformly distributed in distribution range [1, 30], the number of machines in each stage is produced in distribution range [1, 5], randomly, where the energy consumption per unit time of idling, processing and blocking are generated from the uniform distribution range [1, 3], [3, 5] and [5, 7], randomly.

For proving the performance of the proposed IGQ algorithm and fair tests, the termination condition of all the algorithms is the cpu running time, denoted as $TimeLimit$. $TimeLimit = J \times S \times CPU$, $CPU = 5$, All test algorithms are compiled and coded by C++, the Visual Studio 2019, running on Microsoft Windows 10, 2.60 GHZ Intel Core i7 Pentium processor and a 16GB RAM memory. Each instance is repeated for 5 times and the best one is selected.

### B. Evaluation indicator

Due to the best true solution of BHFSP is unknown, we utilize the relative percentage increment (RPI) to evaluate the performance of all the algorithms. The calculation formula is as given as follows.

$$RPI = (c_i - c_{best}) / c_{best} \times 100\% \quad (16)$$

where $c_{best}$ is the minimum energy consumption value obtained by all the algorithms, $c_i$ is the energy consumption value generated by algorithm $i$. Obviously, the algorithm which has the smaller RPI is better than other algorithms.

From the objective analysis of BHFSP, we know that the value of energy consumption is so large to make the difference between the numerator and the denominator small. The value of RPI obtained by all algorithms is also very small. Therefore, to comprehensively and intuitively evaluating the proposed algorithm, we compare not only the RPI value, but also the average and minimum energy consumption(AVG and MIN for short), respectively.

### C. Comparisons of overall performance

We compare IGQ with 9 different algorithms. We first compare our proposed IGQ with some classical swarm intelligence algorithms, i.e., the GA [9], DABC [10], EMBO [11], DPSO [12]. Then, we compare IGQ with a series of the existing IG algorithms, i.e., the original IG [7], the improved IGRS , IGT , IGTALL , and VBIH [8]. Because GA, DABC, EMBO and DPSO are representative and highly recognized in swarm intelligence algorithm, they all show their performance advantages after long-term test, and the selected improved IG algorithm also shows good performance in test, and the published year of these comparative algorithms is relatively recent. Therefore, we choose these algorithms as comparative algorithms. About the termination limit time, $TimeLimit = J \times S \times CPU$, we set 5 as the parameter CPU, all compared algorithms have the same termination time. In TABLE I and II, the best result of the comparative methods is highlighted.

For 14 different instance sets, TABLE I and II list the average and minimum values of the energy consumption obtained by 10 compared algorithms. In addition, the RPI values of all the algorithms are listed in the brackets in TABLE I and II. we can see that IGQ has the best RPI in all test sets. When comparing with other classical swarm intelligence optimization algorithms, as seen in Table 1, IGQ algorithm has the smallest average values for all tests. About the minimum values, the IGQ can get 12 smallest values for 14 tests. In Table 2, IGQ can get 12/14 and 9/14 best average and minimum values when contrast with other existing IG algorithms.

## V. CONCLUSIONS

In this paper, we firstly use a mathematical model of energy efficient BHFSP, then we develop a quick and effective IG algorithm, namely IGQ to solve this problem. In IGQ, MME is utilized to produce an initial solution, after the initialization, a quick local perturbation strategy and global perturbation strategy are suggested in this algorithm. The experiment shows the superiority of our proposed IGQ.

The future research would develop more methods based on the IG algorithm. We may study some new operators based on swap. Similarly, multi-objective problems will be researched. Besides, we may consider a distributed factories problem or batching parts problem and further take the uncertainty conditions into account, such as machine breakdowns, the due date.


## ACKNOWLEDGMENT

This work was jointly supported by National Natural Science Foundation of China with grant No. 61803192, 61973203, 61966012, 61773192, 61603169, 61773246, and 71533001. Thanks for the support of Shandong province colleges and universities youth innovation talent introduction and education program.


## REFERENCES


[1] Rubén Ruiz, José Antonio Vázquez-Rodríguez. The hybrid flow shop scheduling problem. European Journal of Operational Research, 2010, 205( 1):1-18.

[2] Li J Q , Pan Q K , Wang F T . A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem. Applied Soft Computing Journal, 2014, 24:63-77.

[3] Li J Q , Pan Q K , Duan P Y . An Improved Artificial Bee Colony Algorithm for Solving Hybrid Flexible Flowshop With Dynamic Operation Skipping. IEEE Transactions on Cybernetics, 2016, 46(6):1311-1324.

[4] Liu S , Pei J , Cheng H , et al. Two-stage hybrid flow shop scheduling on parallel batching machines considering a job-dependent deteriorating effect and non-identical job sizes. Applied Soft Computing, 2019, 84:105701.

[5] Schulz S , Neufeld J S , Buscher U . A multi-objective iterated local search algorithm for comprehensive energy-aware hybrid flow shop scheduling. Journal of Cleaner Production, 2019, 224(JUL.1):421-434.

[6] Zhang B , Pan Q K , Gao L , et al. A Three-Stage Multiobjective Approach Based on Decomposition for an Energy-Efficient Hybrid Flow Shop Scheduling Problem. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2019, PP(99):1-16.

[7] Rubén Ruiz, Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 2007, 177( 3):2033-2049.

TABLE I. EXPERIMENTAL RESULTS OF ENERGY CONSUMPTION COMPARED WITH CLASSIC ALGORITHMS WHEN CPU = 5

| Instance | IGQ | | GA | | DABC | | EMBO | | DPSO | |
|---|---|---|---|---|---|---|---|---|---|---|
| J×S | AVG(RPI) | MIN | AVG(RPI) | MIN | AVG(RPI) | MIN | AVG(RPI) | MIN | AVG(RPI) | MIN |
| 20×5 | **10583.7(0.00)** | 7895 | 10600.1(0.01) | 7918 | 10624.4(0.02) | 8007 | 10614.3(0.01) | **7870** | 11425.1(0.09) | 8581 |
| 20×10 | **27898.1(0.00)** | **22953** | 28018.7(0.01) | 22990 | 28003(0.02) | 22982 | 27948.3(0.01) | 22961 | 29332.9(0.05) | 24114 |
| 40×5 | **18591(0.01)** | **13030** | 18945.6(0.04) | 13431 | 19761.5(0.09) | 14118 | 18839.5(0.03) | 13168 | 20328(0.11) | 15092 |
| 40×10 | **56959(0.01)** | **47190** | 57663(0.02) | 48044 | 58940.1(0.05) | 49178 | 57354(0.02) | 47935 | 59446.5(0.05) | 51338 |
| 60×5 | **43269(0.00)** | **33478** | 43819.3(0.02) | 33946 | 44258.9(0.03) | 34625 | 43535(0.01) | 33646 | 44354.9(0.03) | 34338 |
| 60×10 | **84587.2(0.01)** | **60377** | 86211.6(0.03) | 61576 | 89217.2(0.07) | 64227 | 85713.6(0.02) | 60997 | 88574.8(0.05) | 62015 |
| 80×5 | **55250.1(0.00)** | **27474** | 56067.8(0.02) | 28002 | 56742.4(0.04) | 28727 | 55549.3(0.01) | 27782 | 56592(0.03) | 28410 |
| 80×10 | **119137(0.01)** | **81925** | 123869.5(0.05) | 85828 | 126837.2(0.09) | 88624 | 120785.6(0.02) | 83899 | 123582.3(0.04) | 83237 |
| 100×5 | **64937.9(0.00)** | **31413** | 66094.7(0.03) | 32512 | 67369.1(0.05) | 34609 | 65460.4(0.02) | 32193 | 66323.1(0.03) | 33702 |
| 100×10 | **149353.5(0.01)** | 111465 | 157099.9(0.07) | 117072 | 162530.7(0.11) | 119878 | 152704.2(0.04) | **113305** | 154705.7(0.04) | 113537 |
| 200×5 | **143562.7(0.01)** | **60608** | 150717.5(0.07) | 65860 | 152821.6(0.09) | 68022 | 146281.2(0.04) | 64081 | 146853.6(0.14) | 63627 |
| 200×10 | **302144.5(0.01)** | **222839** | 327796.3(0.1) | 240067 | 336635.3(0.13) | 240855 | 315935.2(0.05) | 234505 | 312026.4(0.03) | 224800 |
| 300×5 | **250643.4(0.00)** | **151898** | 257671.3(0.03) | 152914 | 257787.8(0.03) | 152737 | 252998.2(0.01) | 151904 | 252950.1(0.05) | 151957 |
| 300×10 | **490051.5(0.00)** | **332151** | 525004.7(0.08) | 341201 | 526498.5(0.08) | 341669 | 507247.2(0.04) | 336586 | 499511.8(0.02) | 334636 |
| **AVG** | **129783.5(0.006)** | | 136398.6(0.041) | | 138430.6(0.065) | | 132926.1(0.023) | | 133286.2(0.054) | |

TABLE II. EXPERIMENTAL RESULTS OF ENERGY CONSUMPTION COMPARED WITH OTHER EXISTING IG ALGORITHMS WHEN CPU = 5

| Instance | IGQ | | IGA | | IGRS | | IGT | | IGTALL | | VBIH | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| J×S | AVG(RPI) | MIN | AVG(RPI) | MIN | AVG(RPI) | MIN | AVG(RPI) | MIN | AVG(RPI) | MIN | AVG(RPI) | MIN |
| 20×5 | **10583.7(0.00)** | **7895** | 10614.5(0.01) | **7895** | 10591.8(0.01) | 7950 | 10605(0.01) | 7941 | 10599.8(0.01) | 7906 | 10595.5(0.02) | **7895** |
| 20×10 | **27898.1(0.00)** | **22953** | 27945.3(0.01) | 22953 | 27940.6(0.01) | 22982 | 27936.1(0.01) | 22953 | 27939.2(0.01) | 22970 | 27943.6**(0.00)** | 22953 |
| 40×5 | **18591(0.01)** | **13030** | 18853.8(0.03) | 13563 | 18933.1(0.03) | 13665 | 18859.3(0.03) | 13274 | 18870.7(0.03) | 13621 | 18771.7(0.03) | 13404 |
| 40×10 | **56959(0.01)** | **47190** | 57344.7**(0.01)** | 48093 | 57445.6(0.02) | 48092 | 57281.7**(0.01)** | 47975 | 57196**(0.01)** | 47731 | 57103.8**(0.01)** | 47725 |
| 60×5 | **43269(0.00)** | **33478** | 43436.6(0.01) | 33613 | 43565.9(0.01) | 33837 | 43371.1(0.01) | 33639 | 43426.5(0.01) | 33770 | 43354.7(0.01) | 33589 |
| 60×10 | **84587.2(0.01)** | **60377** | 85547.5(0.02) | 60964 | 85846.9(0.02) | 60847 | 85292.4**(0.01)** | 60953 | 85239.7**(0.01)** | 60730 | 85011.1**(0.01)** | 60730 |
| 80×5 | **55250.1(0.00)** | 27474 | 55497.5(0.01) | 27795 | 55570.1(0.01) | 27825 | 55356.2**(0.00)** | **27468** | 55402.3(0.01) | 27638 | 55318.1**(0.00)** | 27554 |
| 80×10 | **119137(0.01)** | **81925** | 120091(0.02) | 82199 | 120432.1(0.02) | 82550 | 119790.9**(0.01)** | 82705 | 119778.9**(0.01)** | 82495 | 119231.9**(0.01)** | 82007 |
| 100×5 | **64937.9(0.00)** | **31413** | 65329.1(0.01) | 32036 | 65413.1(0.01) | 31671 | 65042.4(0.01) | 31732 | 65068.3(0.01) | 31418 | 64984.4(0.01) | 31515 |
| 100×10 | **149353.5(0.01)** | 111465 | 150967.8(0.02) | 112098 | 151237.6(0.02) | 112477 | 149890.5**(0.01)** | 111257 | 149532.9**(0.01)** | 111615 | 149420.7**(0.01)** | **111171** |
| 200×5 | 143562.7**(0.01)** | 60608 | 144762.2(0.02) | 62218 | 144986.8(0.02) | 62501 | **143422.4(0.01)** | **60591** | 143540.7**(0.01)** | 61125 | 143564**(0.01)** | 61356 |
| 200×10 | **302144.5(0.01)** | 222839 | 306334.3(0.02) | 222944 | 306545.3(0.02) | 222943 | 302848.2**(0.01)** | 222525 | 302946.4**(0.01)** | **221619** | 302687.7**(0.01)** | 221771 |
| 300×5 | 250643.4**(0.00)** | **151898** | 251379.2(0.01) | 151912 | 251213.1**(0.00)** | 151904 | **250544.5**(0.01) | **151898** | 250563.7**(0.00)** | 151957 | 250601.8**(0.00)** | **151898** |
| 300×10 | **490051.5(0.00)** | 332151 | 494387.8(0.01) | 333824 | 493297.2(0.01) | 333098 | 490602.8**(0.00)** | 332326 | 490726.3**(0.00)** | **332126** | 490340.8**(0.00)** | **332126** |
| **AVG** | **129783.5(0.006)** | | 130892.2(0.015) | | 130929.9(0.015) | | 130060.3(0.011) | | 130059.4(0.01) | | 129923.6(0.008) | |


[8] Ronconi D P. A note on constructive heuristics for the flowshop problem with blocking. International Journal of Production Economics, 2004, 87, 39-48.

[9] Nejati M , Mahdavi I , Hassanzadeh R , et al. Multi-job lot streaming to minimize the weighted completion time in a hybrid flow shop scheduling problem with work shift constraint. International Journal of Advanced Manufacturing Technology, 2014, 70(1-4):501-514.

[10] Pan Q K , Wang L , Li J Q , et al. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. Omega, 2014, 45(jun.):42-56.

[11] Zhang B , Pan Q K , Gao L , et al. An effective modified migrating birds optimization for hybrid flowshop scheduling problem with lot streaming. Applied Soft Computing, 2017, 52:14-27.

[12] Marichelvam M K , Geetha M , mür Tosun. An improved particle swarm optimization algorithm to solve hybrid flowshop scheduling problems with the effect of human factors – a case study. Computers & Operations Research, 2019, 114:104812.

[13] Ztop H , Tasgetiren M F , Deniz Türsel Eliiyi, et al. Metaheuristic algorithms for the hybrid flowshop scheduling problem. Computers & Operations Research, 2019, 111:177-196.