Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2015, Article ID 548363, 6 pages
http://dx.doi.org/10.1155/2015/548363

Research Article

A Hybrid Algorithm for the Permutation Flowshop Scheduling Problem without Intermediate Buffers

Xiaobo Liu,[1] Kun Li,[2] and Huizhi Ren[3]

[1]*College of Resources and Civil Engineering, Northeastern University, Shenyang 110819, China*
[2]*School of Management, Tianjin Polytechnic University, Tianjin 300387, China*
[3]*School of Mechanical Engineering, Shenyang University of Technology, Shenyang 110870, China*

Correspondence should be addressed to Xiaobo Liu; lxb_58@163.com

Received 29 September 2014; Revised 4 February 2015; Accepted 4 February 2015

Academic Editor: Bixiang Wang

This paper deals with the permutation flowshop scheduling problem without intermediate buffers and presents a hybrid algorithm based on the scatter search and the variable neighborhood search. In the hybrid algorithm, the solutions with good quality and diversity are maintained by a reference set of scatter search, and the search at each generation starts from a solution generated from the reference set so as to improve the search diversity while guaranteeing the quality of the initial solution. In addition, a variable neighbourhood based on the notion of job-block is developed, and the neighbourhood size can adaptively change according to the construction of the job-block. Such a dynamic strategy can help to obtain a balance between search depth and diversity. Extensive experiments on benchmark problems are carried out and the results show that the proposed hybrid algorithm is powerful and competitive with the other powerful algorithms in the literature.

1. Introduction

The traditional permutation flow shop scheduling problem (PFSP) has been the focus of research in the field of production scheduling due to its strong industrial background [1]. This problem usually is assumed to have unlimited capacity to store jobs in an intermediate buffer between adjacent machines. However, in many practical industries, due to the need for the continuous production, a production line usually has no intermediate buffers between adjacent machines [2]. For example, in the hot rolling process of heavy plates in iron and steel industry, a heavy plate will be continuously processed by a series of consecutive stages, that is, reheating, hot rolling, cooling, cutting, and quenching, and there are no intermediate buffers between any two stages. In this case, after the completion of processing on machine j, a job i has to wait on this machine until the next machine $j+1$ becomes available (i.e., the next machine $j+1$ is idle). Such a phenomenon of job i and machine j is called the blocking, and such a kind of PFSP is then called the PFSP with blocking. When the objective is the minimization of *makespan* ($C_{\max}$), this problem can be represented as F_m/blocking/$C_{\max}$, which has drawn a lot of attention from many researchers recently.

Hall and Sriskandarajah [3] presented a survey of F_m/blocking/$C_{\max}$ and introduced some instances of this problem in practical industries. Since this problem has been proved to be NP-complete with $m \geq 3$, the exact algorithms based on branch-and-bound can only solve very small-size problems. For large-size problems, heuristics or metaheuristics becomes the only candidate algorithm.

McCormick et al. [4] proposed a heuristic algorithm called PF, which in each iteration will insert the job into a position of a partial sequence obtained currently so that the resulted new sequence has a minimum increase of machine idle time and blocking time. Based on the NEH heuristic algorithm proposed by Nawaz et al. [5] and the PF heuristic, Ronconi and Armentano [6] designed a hybrid algorithm hybridizing the PF and NEH and had achieved good results. Caraffa et al. [7] proposed an improved genetic algorithm (GA) to solve the problem with the minimization of $C_{\max}$, and the experimental results showed that the algorithm is effective and its performance was superior to heuristics proposed by

Abadi et al. [8]. Grabowski and Pempera [2] proposed a Tabu search (TS+M) for this problem, and the computational results based on benchmark problems showed that the TS+M algorithm outperformed the previous algorithms in the literature. Jarboui et al. [9] designed a hybrid estimation of distribution algorithms (H-EDA) for the problem, and the test results showed that the algorithm was superior to the algorithms proposed in [2, 7]. Liang et al. [10] proposed a multigroup particle swarm optimization, and the results showed that the algorithm problem was better than the TS+M algorithm proposed in [2]. Wang and Tang [11] proposed a discrete particle swarm algorithm in which a variable neighborhood search algorithm is used as local search and the computational results showed that the algorithm is superior to TS+M algorithm.

Variable neighborhood search (VNS) is a simple but effective intelligent optimization algorithm [12]. Different from the traditional local search algorithm that searches only a neighborhood, the idea of VNS is to use two or more neighborhoods in the search process, and in the search process the algorithm will perform an alternate search according to a given rule. Because the neighborhood structures are different from each other, a local optimal solution in one neighborhood is not necessarily the local optimal solution in the other neighborhoods. So the alternate search along different neighborhoods can help to improve the search diversity of VNS. VNS algorithm has been widely used to solve various kinds of combinatorial optimization problems, and many promising results have been achieved by VNS [13]. Scatter search (SS) is a population-based approach [14], and its main idea is to select solutions with good quality and diversity from the population to construct the reference set and then use the reference set to produce the next generation of the population. The algorithm has been successfully applied in many combinatorial optimization problems, such as the vehicle scheduling problem [15] and the production scheduling problem [16].

Therefore, in this paper we hybridize the SS with VNS and propose a hybrid algorithm for the F_m/blocking/$C_{\max}$ problem so as to combine the global search ability of SS and local search ability of VNS. Through their combination, a balance of global search and local search can be achieved. In the neighborhood structure, we develop a job-block based neighborhood so as to improve the search efficiency of VNS. The main contributions of this paper are as follows.

(1) Different from previous research that incorporates VNS into SS to act as the local search, this paper incorporates the SS into VNS to store solutions with good quality and diversity and develops a new hybrid algorithm.

(2) In the hybrid memetic algorithm, an adaptive selection strategy is presented to select appropriate solutions for the VNS based on both the previous selection times and search result starting from it.

(3) A job-block neighborhood structure is designed that can dynamically adjust its size so as to achieve a balance between search depth and diversity.

The rest of this paper is organized as follows. The F_m/ blocking/$C_{\max}$ problem is described in Section 2. The description of the proposed memetic algorithm is presented in Section 3, and in Section 4 we carry out extensive experiments to test the performance of our algorithm and compare it to the other powerful algorithms in the literature. At last, the paper is concluded in Section 5.

2. The F_m/Blocking/$C_{\max}$ Problem

The F_m/blocking/$C_{\max}$ problem considered in this paper can be described as follows. There is a set of jobs $J = \{1, \ldots, n\}$ that have arrived at time zero and a set of processing machines $M = \{1, \ldots, m\}$. Each job $i \in J$ must be processed on these m machines in the same order from the first one to the last one, that is, $1, 2, \ldots, m$. The processing time of job $i \in J$ on machine $j \in M$ is p_{ij}, which is a fixed and nonnegative value. It is assumed that the processing of each job cannot be interrupted once the processing starts. At any time, each job can be processed on at most one machine, and each machine can process at most one job. There are no intermediate buffers between any two consecutive machines, and, consequently, a job i, which has completed processing on machine j, should remain on machine j until the next machine $j + 1$ becomes idle and available. That is, the job i and machine j are *blocked* if the next machine $j + 1$ is not idle. The scheduling task is to determine a job sequence for these n jobs on the m machines so as to minimize the maximum completion time (i.e., *makespan*).

Let $s = (s(1), s(2), \ldots, s(n))$ be a permutation of n jobs, in which $s(k)$ denotes the index of the job allocated at the kth position of s. The departure time of job $s(k)$ on machine j (denoted as $D_{s(k),j}$) can be then calculated as follows:

$$
\begin{aligned}
&D_{s(1),0} = 0;\\
&D_{s(1),j} = \sum_{l=1}^{j} p_{s(1),l}, \quad j = 1, \ldots, m-1;\\
&D_{s(k),0} = D_{s(k-1),1}, \quad k = 2, \ldots, n;\\
&D_{s(k),j} = \max\left\{D_{s(k),j-1} + p_{s(k),j}, D_{s(k-1),j+1}\right\},\\
&\qquad\qquad k = 2, \ldots, n, \quad j = 1, \ldots, m-1;\\
&D_{s(k),m} = D_{s(k),m-1} + p_{s(k),m}, \quad k = 1, \ldots, n;
\end{aligned}
\tag{1}
$$

then the makespan of solution s can be calculated as $C_{\max}(s) = D_{s(n),m}$.

3. Hybrid Algorithm

3.1. Main Procedure of the Algorithm. In the hybrid algorithm, the reference set R is introduced into VNS to help the algorithm to get more power to get out of local optimum. In this algorithm, R is used to store solutions with good quality and diversity. The main procedure of the proposed algorithm is given in Algorithm 1, in which a solution is represented by a processing sequence of all jobs, and the details of each procedure are presented in the following sections.

```
Generate an initial solution s, and set the neighborhoods based on job-block N_k (k = 1, ..., K)
s := LocalSearch(s, N_1)                    //perform local search
Add the improved solution s into R
while the termination criterion is not reached do
    k := 1
    while k ≤ K do
        S' := Shaking(R)                    //generate a new solution from R
        s'' := LocalSearch(s', N_k)         //perform local search to s'
        if C_max(s'') < C_max(s)            //update solution and neighborhood
            s := s''
            k := 1
        else
            k := k + 1
        end if
        R := UpdateRefSet(s'')              //update R
    end while
end while
```

Algorithm 1: Main procedure of the proposed hybrid algorithm.

In the hybrid algorithm, we replace the *Shaking* procedure in traditional VNS with the generation of a new solution s' based on solutions selected from R so that the new solution can have good initial quality and diversity, which in turn can help the algorithm to avoid being trapped in local optimum. In addition, traditional VNS proposed for PFSP generally adopts the *insertion* move (remove a job from the job sequence and then insert it into another position) and the *swap* move (swap two jobs in the job sequence). Bozejko et al. [17] pointed out that compound moves (i.e., multiple insertions or swaps) generally have better performance than simple insertion or swap for the scheduling problems in which a solution is represented as a job sequence. The reason is that the compound move can have a larger solution space. Therefore, in this paper we develop a job-block neighborhood whose size can be dynamically adjusted according to the size of job blocks. In the hybrid algorithm, the neighborhood size changes in an ascending order so as to guarantee a balance between search depth and diversity.

3.2. Generation of Initial Solution. Due to the fact that the NEH method can obtain good solutions for PFSP, we also adopt this method to generate an initial solution and then apply local search to improve it. The procedure can be described as follows.

Step 1. Sequence all the jobs according to the nonascending order of total processing times of each job on all stages and obtain a job sequence as $x = (x(1), \ldots, x(n))$.

Step 2. Determine the optimal sequence of the first two jobs in the sequence as the partial solution and then delete the two jobs $x(1)$ and $x(2)$ from the sequence x.

Step 3. From the first job in the remaining sequence, insert it to the best position of the partial solution (i.e., the increased objective function value is minimum) and remove it from the sequence. Repeat this insertion procedure until every job has been inserted and subsequently an initial solution is obtained.

Step 4. Perform a local search with the insertion neighborhood to improve the initial solution and take the improved solution as the true initial solution.

3.3. Update of Reference Set R. In order to ensure the solutions of reference set R ($|R| = b$) can have good quality and diversity, the following updating strategy is used. If $|R| < b$, then the improved solution s'' can be added into R directly; otherwise, the improved solution s'' can be added into R if it can satisfy $(C_{\max}(s'') - C_{\max}(s_{\text{worst}}))/C_{\max}(s_{\text{worst}}) \leq a$ where s_{worst} denotes the worst solution in R and a is a threshold value. After the insertion of s'', the solution with the worst diversity in R will be removed. The diversity of a solution in R is denoted as follows: the distance between two solutions $s_1 = (s_1(1), s_1(2), \ldots, s_1(n))$ and $s_2 = (s_2(1), s_2(2), \ldots, s_2(n))$ is defined as $d(s_1, s_2) = \sum_{j=1}^{n} \text{sgn}(j)$, in which $\text{sgn}(j) = 1$ if $s_1(j) \neq s_2(j)$ and otherwise $\text{sgn}(j) = 0$. If two solutions have the same worst diversity, then the solution with worse quality will be deleted.

3.4. Generation of New Solution from R. Initially, each solution in R has the same selection probability. If a solution s_i has been selected for local search and the resulting improved solution fails to enter the reference set R, then we set $\text{sel}_i = \text{sel}_i + 1$ (sel_i is the selection time of solution i); otherwise, sel_i remains unchanged. According to sel_i, the selection probability of a solution can be defined as

$$p_i = \frac{r_i}{\sum_{j \in R} r_j}, \quad \text{where } r_i = \sum_{j \in R} \frac{\text{sel}_i}{\text{sel}_j}. \tag{2}$$

3.5. Local Search Based on Job-Block Neighborhood. As mentioned earlier, the job-block based neighborhood is a kind of compound-move based neighborhood. For a given solution

$s = (s(1), \ldots, s(n))$, let the size of the job-block be k and the corresponding neighborhood is denoted as N_k; then the local search procedure can be described as follows:

Step 1. Set the iteration index $l = 1$ and the best solution to be $s_b = s$.

Step 2. Randomly remove k adjacent jobs from s and denote this job-block as $s(d_1), s(d_2), \ldots, s(d_k)$ and the left partial solution as s'.

Step 3. Set $j = 1$.

Step 4. Insert $s(d_j)$ to the best position in s'.

Step 5. Set $j = j + 1$. If $j \leq k$, go to Step 4; otherwise, obtain a new solution s'.

Step 6. If $C_{\max}(s') < C_{\max}(s_b)$, then set $s = s_b = s'$.

Step 7. Set $l = l + 1$. If $l \leq l_{\max}$ (maximum iteration), go to Step 2; otherwise, terminate and output s_b.

In the above local search, it is clear that the local search is not performed on the entire neighborhood but only a number of $l_{\max}$ random searches. Such a strategy can help to save computational time while maintaining the solution quality. In addition, the search depth of neighborhoods is in an ascending order so as to obtain a balance between search depth and diversity. Since the values of $l_{\max}$ and k are much less with comparison to n (the number of jobs), the complexity of this kind of local search will not exceed that of the local search based on the *insertion* neighborhood or the *swap* neighborhood, whose complexity is $O(n^2)$. So the search efficiency can be guaranteed in the local search of our algorithm.

4. Computational Experiments and Results

In the experiment, we tested our proposed algorithm on a set of benchmark problems for PFSP proposed in Taillard [18] and compared it with other powerful algorithm in the literature. There are a total of 12 groups of problems in the benchmark suit and there are 10 instances for each group. The number of jobs n changes from 20 to 500 and the number of machines m changes from 5 to 20.

The hybrid algorithm was implemented in C++ language and tested on a personal computer with Intel Core i3 (2.8 GHz) CPU and 4 GB memory. The stopping criterion is set to the maximum computational time of $T_{\max} = m \times n \times 0.02$ seconds. Parameters of the hybrid algorithm are set as follows: the size of R is equal to 10, the maximum size of job-block is set to 5, and the maximum iteration of local search $l_{\max}$ is set to 10 for instances with $n \leq 50$ and 20 for the other instances.

To compare the statistical performance, 10 independent runs were performed for each instance and the minimum percentage relative difference (minPRD) and average percentage relative difference (APRD) are collected. The relative difference PRD is defined as $\text{PRD} = 100 \times (C_{\max}(s) - C_{\max}(\text{RON}))/C_{\max}(\text{RON})$, where s and RON, respectively, represent the solution obtained by an algorithm and the solution given in [19].

Table 1: Comparison results between VNS and SSVNS.

Problem ($n \times m$)	VNS_1		SSVNS	
	APRD	minPRD	APRD	minPRD
20×5	**−0.43**	**−0.43**	**−0.43**	**−0.46**
20×10	−2.29	−2.38	**−2.38**	**−2.40**
20×20	−2.96	−2.94	**−3.22**	**−3.30**
50×5	−2.19	−2.72	**−4.06**	**−4.42**
50×10	−4.02	−4.55	**−4.85**	**−5.48**
50×20	−4.92	−5.39	**−5.18**	**−5.53**
100×5	−2.37	−2.52	**−2.32**	**−2.63**
100×10	−4.36	−4.61	**−5.11**	**−5.76**
100×20	−3.72	−4.14	**−4.02**	**−4.60**
200×10	−2.71	−3.39	**−3.10**	**−4.08**
200×20	−2.77	−2.91	**−3.28**	**−3.90**
500×20	−1.36	−1.72	**−1.82**	**−1.99**
Average	−2.84	−3.14	**−3.31**	**−3.71**

4.1. Efficiency Analysis of Incorporating the Reference Set. To test and analyze the performance of incorporating the reference set of SS into VNS, in this section we compared our hybrid algorithm (denoted as SSVNS) and another version in which the reference set is not included (denoted as VNS). Please note that the VNS also shares the same stopping criterion as used in our SSVNS. The comparison results are shown in Table 1, in which a better result is shown in a bold style.

From the results shown in Table 1, it can be found that the algorithm's performance can be significantly improved by incorporating the reference set. The main reason behind this phenomenon is that the incorporation of reference set can help to provide an initial solution with good quality and diversity to the VNS, which in turn can help to improve the solution's quality and at the same time make the algorithm have a good ability of getting out of local optimum regions.

4.2. Comparison with the Other Powerful Algorithms. In the above section, we have shown the efficiency of incorporating the reference set into VNS and analyzed its impact on the algorithm. To further illustrate the efficiency of the proposed hybrid algorithm, in this section we compare it to the other powerful algorithms in the literature, for example, the TS+M algorithm proposed in [2], the H-EDA algorithm recently proposed in [9], and the DMS-PSO algorithm proposed in [10].

The comparison results are given in Table 2, in which the results of TS+M, H-EDA, and DMS-PSO are those reported in the corresponding references. From the comparison results, it appears that for the APRD metric the H-EDA obtains the best performance. However, our hybrid algorithm can obtain the best results for more instance groups than the H-EDA. Furthermore, it is clear that our SSVNS algorithm achieves the best result for 10 out of the 12 instance groups for the minPRD metric and these results are better than those obtained by the H-EDA algorithm. In addition,

Table 2: Comparison results between SSVNS and the other powerful algorithms.

Problem	TS+M		H-EDA		DMS-PSO		SSVNS	
	APRD	minPRD	APRD	minPRD	APRD	minPRD	APRD	minPRD
20 × 5	0.34	0.34	**−0.43**	−0.43	−0.30	−0.36	**−0.43**	**−0.46**
20 × 10	−1.77	−1.77	**−2.39**	−2.39	−2.32	−2.39	−2.38	**−2.40**
20 × 20	−2.94	−2.94	**−3.27**	−3.27	−3.26	**−3.30**	−3.22	**−3.30**
50 × 5	−0.55	−0.55	−4.00	−4.01	−2.78	−3.12	**−4.06**	**−4.42**
50 × 10	−3.52	−3.52	−4.80	−4.80	−3.45	−4.09	**−4.85**	**−5.48**
50 × 20	−4.26	−4.26	−4.94	−4.95	−4.91	−5.09	**−5.18**	**−5.53**
100 × 5	2.62	2.62	**−2.49**	−2.49	−1.01	−1.52	−2.32	**−2.63**
100 × 10	−2.66	−2.66	−5.02	−5.03	−4.43	−4.87	**−5.11**	**−5.76**
100 × 20	−3.02	−3.02	−3.85	−3.86	**−4.72**	**−5.14**	−4.02	−4.60
200 × 10	−0.58	−0.58	**−3.70**	−3.70	−1.93	−2.36	−3.10	**−4.08**
200 × 20	−2.31	−2.31	**−3.37**	−3.37	−2.94	−3.31	−3.28	**−3.90**
500 × 20	−1.47	−1.47	−1.89	−1.89	**−2.00**	**−2.28**	**−1.82**	**−1.99**
Average	−1.68	−1.68	**−3.35**	−3.35	−2.89	−3.10	−3.31	**−3.71**

the SSVNS is better than the DMS-PSO algorithm in both APRD and minPRD metrics for all the instance groups except the instance groups 100 × 20 and 500 × 20. Therefore, it can be then concluded that the proposed hybrid algorithm is competitive with or superior to some powerful algorithms in the literature.

5. Conclusions

In this paper, the permutation flowshop scheduling problem with no intermediate buffer is investigated. In this problem, a job and a machine may be blocked due to the fact that there is no intermediate buffer between consecutive machines, and such a condition often occurs in practical industries. Although there have been many papers dealing with this problem, this paper presents a new kind of hybrid algorithm by incorporating SS into VNS and developing a job-block based neighborhood search. The incorporation of SS into VNS can provide an initial solution with good quality and diversity, which in turn can help to guarantee the good quality of solutions improved by the VNS and at the same time maintain a good search diversity (i.e., the hybrid algorithm can obtain a good ability of getting out of local optimum regions in the search space). The adaptive selection strategy of solutions from the reference set is able to further improve the search diversity. The local search used in the VNS is based on the job-block which is a compound-move neighborhood. This kind of neighborhood can provide a much larger search space and consequently can increase the probability of finding more promising solutions. In addition, the variable search depth of this kind of local search can also help to achieve a balance between the search intensity and the diversity. Extensive experiments on benchmark problems are carried out to test the hybrid algorithm and the results show that the proposed algorithm is very efficient and it is even competitive with or superior to some other powerful algorithms in the literature.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research is partly supported by National Natural Science Foundation of China (Grant no. 51104033), National Natural Science Foundation of China (Grant no. 61403277), and National Natural Science Foundation of China (Grant no. 71201105).

References

[1] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics," *European Journal of Operational Research*, vol. 165, no. 2, pp. 479–494, 2005.

[2] J. Grabowski and J. Pempera, "Sequencing of jobs in some production system," *European Journal of Operational Research*, vol. 125, no. 3, pp. 535–550, 2000.

[3] N. G. Hall and C. Sriskandarajah, "A survey of machine scheduling problems with blocking and no-wait in process," *Operations Research*, vol. 44, no. 3, pp. 510–525, 1996.

[4] S. T. McCormick, M. L. Pinedo, S. Shenker, and B. Wolf, "Sequencing in an assembly line with blocking to minimize cycle time," *Operations Research*, vol. 37, no. 6, pp. 925–935, 1989.

[5] M. Nawaz, E. E. Enscore Jr., and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[6] D. P. Ronconi and V. A. Armentano, "Lower bounding schemes for flowshops with blocking in-process," *Journal of the Operational Research Society*, vol. 52, no. 11, pp. 1289–1297, 2001.

[7] V. Caraffa, S. Ianes, T. P. Bagchi, and C. Sriskandarajah, "Minimizing makespan in a blocking flowshop using genetic algorithms," *International Journal of Production Economics*, vol. 70, no. 2, pp. 101–115, 2001.

[8] I. N. K. Abadi, N. G. Hall, and C. Sriskandarajah, "Minimizing cycle time in a blocking flowshop," *Operations Research*, vol. 48, no. 1, pp. 177–180, 2000.

[9] B. Jarboui, M. Eddaly, P. Siarry, and A. Rebaï, "An estimation of distribution algorithm for minimizing the makespan in blocking flowshop scheduling problems," *Studies in Computational Intelligence*, vol. 230, pp. 151–167, 2009.

[10] J. J. Liang, Q.-K. Pan, T. J. Chen, and L. Wang, "Solving the blocking flow shop scheduling problem by a dynamic multi-swarm particle swarm optimizer," *The International Journal of Advanced Manufacturing Technology*, vol. 55, no. 5–8, pp. 755–762, 2011.

[11] X. P. Wang and L. X. Tang, "A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking," *Applied Soft Computing Journal*, vol. 12, no. 2, pp. 652–662, 2012.

[12] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[13] P. Hansen, N. Mladenović, and J. A. Moreno Pérez, "Variable neighborhood search," *European Journal of Operational Research*, vol. 191, no. 3, pp. 593–595, 2008.

[14] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision Sciences*, vol. 8, no. 1, pp. 156–166, 1977.

[15] R. A. Russell and W.-C. Chiang, "Scatter search for the vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 169, no. 2, pp. 606–622, 2006.

[16] E. Nowicki and C. Smutnicki, "Some aspects of scatter search in the flow-shop problem," *European Journal of Operational Research*, vol. 169, no. 2, pp. 654–666, 2006.

[17] W. Bozejko, J. Grabowski, and M. Wodecki, "Block approach-tabu search algorithm for single machine total weighted tardiness problem," *Computers and Industrial Engineering*, vol. 50, no. 1-2, pp. 1–14, 2006.

[18] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[19] D. P. Ronconi, "A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking," *Annals of Operations Research*, vol. 138, no. 1, pp. 53–65, 2005.