A hybrid heuristic algorithm for the no-wait flowshop problem with sequence-dependent setup times

Zhanbin Wu
School of Computer Science and Engineering, Southeast University, Nanjing, P.R. China

Xia Zhu
School of Computer Science and Engineering, Southeast University, Nanjing, P.R. China

Jatinder N. D. Gupta
College of Business Administration, University of Alabama in Huntsville, Huntsville, Al 35899, USA

Xiaoping Li
School of Computer Science and Engineering, Southeast University, Nanjing, P.R. China (xpli@seu.edu.cn)

Abstract
We consider the NP-hard no-wait flowshop scheduling problem with sequence-dependent setup times to minimize makespan. We propose a hybrid heuristic which combines the iterated greedy process with local search method based on a block swap operator. Experimental results show that the proposed heuristic is comparatively more effective than the existing metaheuristics.

Keywords: No-wait flowshop, Setup time, Makespan

Introduction
The no-wait flowshop scheduling problems (NWFSP) have been investigated for many years. In a NWFSP, all jobs visit each machine in the same order. Each machine can only process one job at a time. The operations of the same job are processed contiguously from start to end without interruptions either on or between machines. Such situations are widespread in many practical applications. In steel manufacturing, for example, different operations (such as molding into ingots, unmolding, reheating, and rolling) must be done in no-wait fashion. Similar situations can also be found in chemical and pharmaceutical industries. Hall and Sriskandarajah (1996) and MacCarthy and Liu (1993) conducted surveys on NWFSP.

In recent years, a great deal of attention is being paid on no-wait flowshop scheduling problems involving setup times. It is reasonable to separate setup times from

processing times because they are always not ignorable in real industrial environments. Generally, there are two types of setup times: sequence independent setup times (SIST), and sequence dependent setup times (SDST). Further, these setup times could be completed in anticipation of the arrival of a job on a machine or be attached to the job. In the former case, idle time of a machine can be used to perform part or all of the needed setup to processing the next job. In this paper, we deal with situations involving anticipatory setup times.

Recently, SDST-FSP with makespan minimization has attracted great attention. Ruiz et al. (2005) proposed two genetic algorithms, named GA, and HGA, that outperformed several existing algorithms including the HYBRID algorithm developed by Ríos-Mercado and Bard (1999). Gupta et al. (2005) developed a penalty-based (PB) heuristic algorithm for the same problem and compared their heuristic with a saving index (SI) algorithm. Overall the proposed algorithm outperformed SI algorithm. Later, Ruiz and Stützle (2008) proposed two iterated greedy (IG) algorithms. The first IG algorithm was a straightforward adaption of the IG principle, while the second incorporated a simple descent local search. Experimental results showed that their algorithms performed better than the genetic algorithms developed by Ruiz et al. (2005). An ant colony optimization (ACO) algorithm was proposed by Mirabi (2011) to solve the same problem. He also presented a new approach for computing the initial pheromone values. The proposed algorithm was tested on randomly generated problem instances and results indicated it was very competitive when compared with GA and HGA (Ruiz et al. 2005).

However, the existing research to solve the flowshop problem with sequence dependent setup times and no-wait (SDST-NWFSP) constraints is quite scarce. Gupta (1986) showed that the SDST-NWFSP problem is NP-hard in the strong sense. Therefore, no polynomial time algorithm exists to find an optimal solution. Realizing this, Gupta (1986) showed that the SDST-NWFSP problem is equivalent to a asymmetric TSP and can be solved using any solution techniques to solve the TSP problem. Later on, Bianco et al. (1999) addressed SDST-NWFSP with release dates to minimize makespan and showed that it was equivalent to the asymmetric TSP with additional visiting time constraints. They proposed a mathematical formulation, two lower bounds, and a heurisitic, called BIH. Allahverdi and Aldowaisan (2001) designed heuristic SL1~SL5 based on iterated insertion operation for the 2-machine flowshop with setup and removable times to minimize total flowtime and presented five heuristics. Basing on a structural property, Araújo and Nagano (2010) proposed GAPH1~GAPH4 for solving the SDST-NWFSP. Experimental results showed the proposed algorithm outperformed BIH (Bianco et al. 1999) and TRIPS (Brown et al. 2004).

In this paper, the *m*-machines SDST-NWFSP to minimize makespan is considered. A hybrid heuristic (IGBS) which combines the iterated greedy process with local search method based on block swap operator (BS) is proposed to solve the considered problem. In the proposed algorithm, the well-known NEH algorithm is used to generate an initial solution. Then, an IG method is deployed to avoid becoming trapped in a local optimum, followed by a BS method to find a good solution in a short time.

The rest of the paper is organized as follows: After introducing and defining the SDST-NWFS problem, we discuss the details of the proposed algorithm. The effectiveness and efficiency of the proposed algorithm are then empirically compared

with existing algorithms and the comparison results are reported. Finally, the paper concludes with some fruitful directions for future research.

Problem Description

The SDST-NWFSP can be described as follows: A set of *n* jobs is to be processed on a set of *m* machines. Let $\pi = (J_0, J_1 ..., J_n)$ denote the scheduling sequence or permutation of jobs to be processed, where J_0 is a virtual job whose setup times and processing times are zero, and t_{ji} is the processing time of job J_i on machine *j,* S_{ijk} is the sequence dependent setup time between two adjacent jobs (J_i , J_j) on machine *k*, D_{ij} is the distance between the finish times of the two adjacent jobs (J_i , J_j) on machine *m*. According to Li (2005), the make span of π and D_{ij} can be calculated by the following equations.

$$C_{max}(\pi) = \sum_{i=0}^{n-1} D^{\pi}_{[i],[i+1]} \quad (1)$$

$$D_{ij} = \max_{k=1,2,...,m} \{\sum_{p=k}^{m} (t_{p,j} - t_{p,i}) + t_{k,i} + S_{ijk}\} \quad (2)$$

Then, the SDST-NWFSP is one of finding a permutation schedule π such that the makespan $C_{max}(\pi)$ is minimum.

The proposed algorithm

Our proposed IGBS algorithm is a hybrid heuristic approach which considers the intensification and diversification of algorithms for solving flowshop problems. Firstly, it uses the well known NEH method to generate an initial solution, as a good initial solution may speed up the search effectiveness. If the termination criterion is not satisfied, an iterated greedy (IG) method is then used to improve the solution and explore a large solution space. Then, BS is used as a local search method to find better solutions. It has to be noted that BS may take a lot of time. That is to say, if the stopping criterion is elapsed CPU time, there is little time to explore a large solution space. The frequency of IG and BS should, therefore, be controlled in one loop. The pseudo-code of the proposed IGBS algorithm can be depicted as follows:

IGBS Algorithm:

1. Generate the initial sequence π_{best} .
   $\pi_1 = \pi_{best}$
2. While not (stopping criterion) do
   π_t = IG (π_1).
   If (rand < β), BS (π_t).
   If (f (π_t) < f (π_1)), $\pi_1 = \pi_t$
   If (f (π_t) < f (π_{best})) $\pi_{best} = \pi_t$
3. Return π_{best} .

Here, the stopping criterion is the maximum CPU time or maximum iterate number

N depending on the compared algorithms, and *rand* is a random number uniformly distributed in the range [0, 1]. β is a threshold value which is determined later.

Initial solution generation

Nawaz et al. (1983) proposed the famous NEH heuristic for solving the traditional FSP with the makespan criterion. Here, it is used to generate an initial solution. Firstly, the jobs are sorted in a decreasing order of their total processing times on all machines. Then, the first two jobs are taken to generate a partial sequence with minimum makespan. Then the rest of the jobs are inserted into the partial sequence step by step with the smallest makespan increment. The pseudo-code of NEH is showed as follows:

NEH procedure:

1: Arrange the jobs by decreasing sums of their total processing times. Let π denote the resulting sequence.

2: Take the first two jobs from π, and generate a partial sequence π_1 with minimum makespan. Set k = 2.

3: Update k=k+1, select the kth job from π and insert it in k possible positions of π_1. Select the one with the minimum partial makespan and declare it as the current π_1.

4: If k = n, return π_1, stop; else return to Step 3.

An iterated greedy method (IG)

The basic IG algorithm starts from some initial solution and then iterates through a main loop in which a partial candidate solution s_d is obtained by removing a number of solution components from a complete candidate solution *s*. Then, a complete solution s_1 is re-constructed starting with s_d by inserting the removed jobs. The number of removed solution components is kept constant.

The proposed IG method contains three main procedures. Firstly, it chooses *k* jobs among all the jobs randomly, and removes them from the job sequence, where $k = \lfloor n \times \alpha \rfloor$, and α is a fraction ranging from [0, 1]. These removed jobs form a new sequence, called π_R. Thus, the schedule sequence is decomposed into two subsequences: one is the sequence without the removed jobs, namely π_D; the other is π_R. Secondly, a new sequence π_1 is generated from π_D using the NEH method without the initial sorting procedure. Finally, the reinsertion phase performs k steps in which the jobs in π_R are reinserted into π_1. The pseudo-code of IG is showed as follows:

IG procedure:

1. For i =1 to $\lfloor n \times \alpha \rfloor$

Remove a job randomly from π, and insert it into π_R.

2. Use NEH method to generate a new partial sequence π_1 from π_D.

3. Generate a random number *r* uniformly distributed in the range [0, 1].

4. For i =1 to $\lfloor n \times \alpha \rfloor$

len = the length of π_1.

Insert the job $\pi_R(i)$ into all possible positions *j* of π_1. And select the position with minimum makespan as the final insertion position. If (r <0.5), $1 \le j \le$ len. Else, len/2 $\le j \le$ len.

5. Return π_2.

A Local search method

To further improve the performance of the IG method, a local search method based on block swap (BS) is proposed. In fact, it is a common hybrid constructive method with local search. The basic idea is as follows. Suppose the current scheduling sequence is π. Firstly, generating an integer set I: $I = \{i \mid 1 \le i \le n-2,\ \mathrm{i} \in \mathrm{Z}\}$. Next, an integer i from set I is picked and to generate an integer set J: $J = \{j \mid i+1 \le j \le n-1,\ \mathrm{j} \in \mathrm{Z}\}$. Also, we randomly select an integer j from set J and generate set K: $K = \{k \mid j+1 \le k \le n,\ \mathrm{k} \in \mathrm{Z}\}$. An integer k is randomly picked from K, and switch the jobs $[\pi_{[i+1]}, \pi_{[i+2]}, ..., \pi_{[j]}]$ and $[\pi_{[j+1]}, \pi_{[j+2]}, ..., \pi_{[k]}]$ to generate a new sequence $\pi_1 = ([\pi_{[0]}, \pi_{[i]}, \pi_{[j+1]} \cdots, \pi_{[k]}, \pi_{[i+1],\ldots,}\pi_{[j]}, \pi_{[k+1],\ldots,}\pi_{[n]}])$. If $f(\pi_1) \le f(\pi)$, the sequence π_1 will be returned. Otherwise, the next loop will not end until the stopping criterion is satisfied. The detailed steps of the BS procedure are as follows.

BS procedure:

1. Suppose the input sequence is π. $I = \{i \mid 1 \le i \le n-2,\ \mathrm{i} \in \mathrm{Z}\}$.
2. Select i randomly from I, $I \leftarrow I - \{i\}$, $J = \{j \mid i+1 \le j \le n-1,\ \mathrm{j} \in \mathrm{Z}\}$.
3. Select j randomly from J, $J \leftarrow J - \{j\}$, $K = \{k \mid j+1 \le k \le n,\ \mathrm{k} \in \mathrm{Z}\}$.
4. Select k randomly from K, $K \leftarrow K - \{k\}$, and generate new sequence π_1.
5. If $f(\pi_1) \le f(\pi)$, return π_1. Stop.
6. If (termination criterion), go to 4.
7. If (J≠∅), go to 3.
8. If (I≠∅), go to 2.
9. Return π_1.

The *termination criterion* depends on the compared algorithms. When comparing with heuristics BIH (Bianco et al.1999) and GAPH1~GAPH3 (Araújo and Nagano 2010), the termination criterion is $K \neq \emptyset$. The used CPU time will be recorded. Otherwise, IGBS chooses *the remained CPU time* && $K \neq \emptyset$ as termination criterion when compared with metaheuristics HGA (Ruiz et al. 2005), IG_LS (Ruiz and Stützle 2008), and ACO (Mirabi 2011) with the same CPU time.

Experimental results

The performance of IGBS is compared with HGA (Ruiz et al. 2005), IG_LS (Ruiz and Stützle 2008), ACO (Mirabi 2011), BIH (Bianco et al.1999), and GAPH1~GAPH3 (Araújo and Nagano 2010). Since, HGA, IG_LS, and ACO algorithms are proposed for SDST-FSP originally, they are adjusted to SDST-NWFSP by considering the no-wait constraint. Besides, HGA, IG_LS, and ACO are metaheuristics, while BIH, and GAPH1~GAPH3 are heuristics. The comparisons among them are divided into two parts. Part I contains ACO, IG_LS, and HGA, and 2) Part II contains BIH, and GAPH1~GAPH3. For part I experiments, the termination criterion of all algorithms is based on the elapsed CPU time. The maximum CPU time is set to *nxmx15* milliseconds. In order to provide a fair comparison with existing algorithms in Part II, IGBS termination criterion is adjusted to ensure that the IGBS does not perform a large number of iterations. Therefore, for experiments in Part II, the IGBS algorithm terminates when

the maximum number of iterations, N=10.

These algorithms have been coded in Java and executed on a Pentium(R) 4, 3.00GHZ and 816MB of RAM. Test data from Ruiz et al. (2005) was used (which can be downloaded from http://www.upv.es/gio/rruiz). In this data set, there are four instances sets: SSD10, SSD50, SSD100, SSD125, and each contain 120 instances, which are organized in 12 groups with 10 instances each. The groups contain several combinations of the jobs and machines, that is, {20, 50, 100} × {5, 10, 20}, 200 × {10, 20}, and 500 × 20. The processing times are distributed in the range [1, 100], while the ratio of the setup time to processing time are at most 10%, 50%, 100%, 125% for four instances sets respectively.

The average relative percentage deviation (ARPD) is adopted as the performance criterion which is defined as folllows:

$$\text{ARPD} = \sum_{i=1}^{R} \left(\frac{F(\pi) - f^*}{f^*} \times 100 \right) / \text{R} \qquad (3)$$

where $F(\pi)$ denotes the makespan of the instance obtained by a given algorithm, and f^* is the best solution that have been calculated for this instance by all algorithms.

Parameters α, β are tested for determination. Parameter α influences the efficiency of construction phase. A small α may lead insufficient information provided. On the other side, a large one will destroy some useful sequence relation among jobs. Parameter β determines the relation between IG method and VNS method. A low value may not use the VNS efficiently, while a large value may waste lot time in the VNS. Through experiments, we set α =0.15, and β = 0.50.

These algorithms are run 10 times on each instance, and the results are finally averaged. Table 1 shows the comparison results among algorithms in Part I for the SSD10 and SSD50, SSD100 and SSD125 set. The results are grouped by instance type and size at each cell. Table 2 shows the comparison results among algorithms in Part II for SSD10 SSD50, SSD100 and sets, while Table 3 gives out the results among algorithms for SSD125 sets. Time is short for CPU time.

The experimental results in Table 1 show that the proposed IGBS algorithm outperforms all other algorithms in Part I by a wide margin, and the ARPD of IGBS is zero. The second best algorithm is IG_LS. An interesting thing is that, IGBS is good at solving large problems. For example, IG_LS and IGBS can get a good solution for instances with combinations of 20 jobs. But, for instances with more than 20 jobs, IGBS can get better solutions. For the SSD10 instance sets, the ARPD of IGBS is 0, the second best algorithm is IG_LS with a 2.098% ARPD. Among all the compared algorithms, ACO is the one that yields the worst results with a 4.632% ARPD. For combination of 20 jobs from SSD10 instance sets, the ARPD of IG_LS is 0.147%, 0.071, and 0.068% respectively for combinations of 20 × 5, 20 × 10, and 20 × 20. For instances with more than 100 jobs from the SSD10 instance sets, IG_LS can only get an ARPD of 4.099%, 3.566%, and 4.063% respectively for combinations of 200 × 10, 200 × 20, and 500 × 20.

In Table 2 and Table 3, we can see that IGBS has the least ARPD, almost zero. The average ARPD value of IGBS for instance sets SSD10, SSD50, SSD100, and SSD125 is 0.021%, 0.033%, 0.113%, and 0.157% respectively, while the best average among other compared algorithms for these four instances sets is 3.643%, 3.080%, 3.306%, and 3.448% of GAPH1. For the SSD10 instances, the ARPD of IGBS ranges from 0 to 0.08%,

and the average of all instances is 0.021%. The second best algorithm is GAPH1, and the average is 3.643%, ranging from 2.016% to 5.816%. Regarding the CPU times, IGBS takes much more time than BIH, but smaller than GAPH2. This happens in instances SSD50, SSD100, and SSD125.

Table 1. Comparison of results for the SDST-NWFSP on instances set SSD10, SSD50, SSD50 and SSD125 with the termination criteria set as $n \times m \times 15$ milliseconds maximum CPU time.

Instances	ARPD				ARPD			
	IG_LS	HGA	ACO	IGBS	IG_LS	HGA	ACO	IGBS
Instance SSD10					Instance SSD50			
20* 5	0.147	0.576	0.450	0	0.058	0.667	0.699	0
20*10	0.071	0.510	0.514	0	0.028	0.26	0.356	0
20* 20	0.068	0.208	0.314	0	0.003	0.325	0.320	0
50* 5	1.728	2.476	2.998	0	1.631	3.116	3.885	0
50*10	1.134	1.670	3.921	0	1.109	2.202	3.521	0
50* 20	0.991	1.745	4.011	0	0.941	1.968	3.546	0
100* 5	3.411	4.577	4.447	0	3.888	4.900	6.164	0
100*10	2.983	3.263	5.819	0	2.707	3.290	5.403	0
100* 20	2.914	2.590	6.480	0	2.676	2.721	5.815	0
200*10	4.099	4.449	7.360	0	4.171	4.163	7.186	0
200* 20	3.566	3.408	8.488	0	3.429	3.433	7.603	0
500* 20	4.063	4.296	10.780	0	3.815	4.205	9.482	0
Avg	2.098	2.481	4.632	0	2.038	2.604	4.498	0
Instance SSD100					Instance SSD125			
20* 5	0.128	1.172	1.243	0	0.115	1.063	1.007	0
20*10	0.016	0.666	0.534	0	0.024	0.566	0.519	0
20* 20	0.032	0.194	0.334	0	0.034	0.423	0.265	0
50* 5	2.464	4.298	5.055	0	2.758	5.073	5.586	0
50*10	1.545	2.878	3.806	0	1.782	2.991	3.992	0
50* 20	1.221	1.665	3.247	0	1.194	1.896	3.193	0
100* 5	5.149	5.917	8.308	0	5.645	6.522	8.989	0
100*10	3.596	4.152	6.13	0	3.652	4.338	6.239	0
100* 20	2.467	2.968	5.178	0	2.880	3.374	5.196	0
200*10	5.037	5.171	8.011	0	5.672	5.728	8.525	0
200* 20	3.535	3.609	6.871	0	3.651	3.837	6.719	0
500* 20	4.351	4.533	8.867	0	4.768	4.893	8.720	0
Avg	2.462	3.102	4.799	0	2.681	3.392	4.913	0

Table 2. Comparison of ARPD and CPU time (milliseconds) for the SDST-NWFSP on instances set SSD10, SSD50, and SSD100

Instance	BIH		GAPH1		GAPH2		GAPH3		IGBS	
	ARPD	Time	ARPD	Time	ARPD	Time	ARPD	Time	ARPD	Time
SSD10 instances										
20* 5	2.087	0	2.087	0	6.241	5	2.909	0	0.019	5
20*10	3.373	0	3.373	0	5.205	0	2.954	0	0.000	6
20*20	2.016	0	1.631	0	5.957	2	2.693	0	0.057	0
50* 5	4.783	0	4.438	20	4.923	28	5.561	8	0.000	56
50*10	2.861	0	3.045	16	4.397	47	3.605	11	0.000	36
50*20	3.168	0	3.297	17	4.367	108	3.506	6	0.000	55
100 *5	5.816	5	5.902	190	5.495	473	7.552	123	0.000	479
100*10	3.618	9	3.933	180	4.663	968	4.77	116	0.078	662
100*20	3.613	2	3.398	194	5.323	1968	3.746	130	0.000	635
200*10	4.913	30	4.989	3462	5.403	17633	5.649	2229	0.080	5711
200*20	4.051	34	3.983	3794	4.155	41030	4.289	2432	0.000	5160
Avg	3.664	7	3.643	716	5.103	5660	4.294	460	0.021	1164
SSD50 instances										
20* 5	2.552	0	2.552	0	3.881	2	3.502	2	0.000	0
20*10	2.050	0	1.725	0	5.640	0	2.789	0	0.194	0
20*20	1.817	0	1.813	0	6.603	2	2.055	0	0.045	5
50* 5	4.168	0	4.017	6	4.069	14	4.775	3	0.000	51
50*10	3.057	2	3.065	10	4.044	25	3.534	6	0.000	44
50*20	2.38	2	2.206	2	5.204	60	2.978	5	0.000	44
100 *5	4.479	3	4.512	89	4.208	234	5.852	56	0.125	542
100*10	3.300	2	3.423	93	4.151	479	4.082	63	0.000	393
100*20	2.985	2	3.098	95	4.194	999	3.455	70	0.000	454
200*10	4.214	19	4.173	1703	4.896	8989	5.268	1109	0.000	5639
200*20	3.274	16	3.300	1974	3.707	20612	3.911	1231	0.000	5219
Avg	3.116	4	3.080	361	4.600	2856	3.836	231	0.033	1126
SSD100 instances										
20* 5	3.585	0	3.828	0	3.853	2	3.119	0	0.160	2
20*10	2.528	0	2.528	0	4.327	2	3.156	0	0.000	0
20*20	1.852	0	1.674	0	4.83	2	2.194	0	0.089	3
50* 5	3.945	0	3.774	2	3.906	16	5.842	3	0.316	82
50*10	2.994	0	2.987	6	3.243	28	3.201	3	0.185	71
50*20	2.158	0	2.23	6	3.16	56	3.24	3	0.014	81
100 *5	5.477	5	5.418	94	4.143	254	6.753	59	0.303	543
100*10	3.746	5	3.583	96	3.587	488	5.022	66	0.000	626
100*20	2.977	5	2.903	96	3.844	989	3.716	63	0.000	702
200*10	4.737	16	4.785	1693	4.435	8814	5.569	1111	0.000	6349
200*20	2.716	12	2.654	1826	3.005	19601	3.239	1200	0.18	4611
Avg	3.338	4	3.306	347	3.848	2750	4.096	228	0.113	1188

Table 3. Comparison of ARPD and CPU time (milliseconds) for the SDST-NWFSP on instances set SSD125

Instance	BIH		GAPH1		GAPH2		GAPH3		IGBS	
	ARPD	Time	ARPD	Time	ARPD	Time	ARPD	Time	ARPD	Time
SSD125 instances										
20* 5	3.333	0	3.431	2	4.51	2	3.139	0	0.174	3
20*10	2.012	0	1.839	0	3.488	0	2.945	0	0.095	5
20*20	1.743	0	1.705	0	4.84	3	2.083	0	0.098	0
50* 5	5.57	0	5.304	11	4.904	9	6.038	8	0000	61
50*10	2.777	0	3.015	8	3.719	28	3.692	5	0.117	44
50*20	2.255	0	2.191	3	3.343	59	2.792	3	0.000	31
100 *5	5.481	2	5.256	86	4.006	244	6.738	59	0.782	582
100*10	4.061	2	4.051	94	3.672	487	5.093	61	0.000	449
100*20	2.908	2	2.883	96	3.398	949	3.463	62	0.157	500
200*10	4.893	16	4.849	1697	3.856	8846	5.838	1097	0.379	6025
200*20	3.497	14	3.404	1853	3.495	19546	4.041	1228	0.000	6245
Avg	3.503	3	3.448	350	3.930	2743	4.169	229	0.164	1268

Conclusions

In this paper, a hybrid heuristic algorithm which combines an iterated greedy algorithm with local search method based on swap operator (IGBS) is proposed for solving SDST-NWFSP to minimize makespan. IGBS comprises three components: a heuristic method to generate an initial solution, an iterated greedy (IG) method to avoid becoming trapped in a local optimum, followed by *BS* to improve the solution. The proposed algorithm is compared with existing heuristic methods with different termination criterion. The results show that the proposed algorithm remains efficient in terms of quality of solutions in the same time, and get better solution with more time when compared with existing heuristic methods.

The proposed IGBS algorithm in this paper is quite general and can solve flowshop problems under varying conditions, with or without setup time. Since the total completion of time of a set of jobs is also very important in practical production shops, one fruitful direction for future research is to develop solution procedures for the SDST-NWFSP with total completion time minimization. At the same time, we want to point out that decision making becomes very complex when many objectives should be taken into account at the same time. Therefore, multi-objective flowshop scheduling problems have gained wide attention both in practical and academic fields. Therefore, extending the application of the proposed IGBS algorithm and to develop new solution procedures for the multi-objective SDST-NWFSP is another interesting future research direction.

References

Allahverdi, A. and T. Aldowaisan. 2001. "Minimizing total completion time in a no-wait flowshop with sequence-dependent additive changeover times." *JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY* 52:449-462.

Araújo, D. C. and M. S. Nagano. 2010. "A new effective heuristic method for the no-wait flowshop with sequence-dependent setup times problem." *International Journal of Industrial Engineering Computations* 2:155-166.

Bianco, L., P. Dell'Olmo and S. Giordani. 1999. "Flow shop no-wait scheduling with sequence dependeent setup times and release dates." *INFOR* 37:3-19.

Brown, S. I., R. G. McGarvey and J. A. Ventura. 2004. "Total flowtime and makespan for a no-wait m-machine flowshop with set-up times separated." *Journal of the Operational Research Society* 55:614-621.

Gupta, J. N. D. 1986. "Flowshop schedules with sequence dependent setup times," *Journal of the Operations Research Society of Japan* 29:206-219.

Hall, N. G. and C. Sriskandarajah. 1996. "A survey of machine scheduling problems with blocking and no-wait in process." *Operations research* 44:510-525.

MacCarthy, B. L. and J. Liu. 1993. "Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling." *The International Journal of Production Research* 31:59-79.

Mirabi, M. 2011. "Ant colony optimization technique for the sequence-dependent flowshop scheduling problem." *The International Journal of Advanced Manufacturing Technology* 55:317-326.

Li, X. 2005. "Heuristics for Several large-scale Flow Shop Scheduling Problems.", Tsinghua University,(in Chinese).

Nawaz, M., E. E. Enscore and I. Ham. 1983. "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem." *Omega* 11:91-95.

Ríos-Mercado, R. Z. and J. F. Bard. 1999. "An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times." *Journal of Heuristics* 5:53-70.

Ruiz, R. and T. Stützle. 2007. "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem." *European Journal of Operational Research* 177:2033-2049.

Ruiz, R. and T. Stützle. 2008. "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives." *European Journal of Operational Research* 187:1143-1159.

Ruiz, R., C. Maroto and J. Alcaraz. 2005. "Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics." *European Journal of Operational Research* 165:34-54.

Stafford, E. F. and F. T. Tseng. 2002. "Two models for a family of flowshop sequencing problems." *European Journal of Operational Research* 142:282-293.