Int J Adv Manuf Technol (2008) 38:778–786
DOI 10.1007/s00170-007-1120-y

ORIGINAL ARTICLE

An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion

Quan-Ke Pan · Ling Wang · Bao-Hua Zhao

Received: 17 January 2007 / Accepted: 4 June 2007 / Published online: 10 July 2007

Abstract An improved iterated greedy algorithm (IIGA) is proposed in this paper to solve the no-wait flow shop scheduling problem with the objective to minimize the makespan. In the proposed IIGA, firstly, a speed-up method for the insert neighborhood is developed to evaluate the whole insert neighborhood of a single solution with $(n-1)^2$ neighbors in time $O(n^2)$, where n is the number of jobs; secondly, an improved Nawaz-Enscore-Ham (NEH) heuristic is presented for constructing solutions in the initial stage and searching process; thirdly, a simple local search algorithm based on the speed-up method is incorporated into the iterated greedy algorithm to perform exploitation. The computational results based on some well-known benchmarks show that the proposed IIGA can obtain results better than those from some existing approaches in the literature.

Keywords No-wait flow shop · Makespan · Iterated greedy algorithm · Speed-up method · Insert neighborhood · Local search

Q.-K. Pan (✉) · B.-H. Zhao
Department of Computer Science,
University of Science and Technology of China,
Hefei 230026, People's Republic of China
e-mail: qkpan@lcu.edu.cn

Q.-K. Pan
College of Computer Science, Liaocheng University,
Liaocheng 252059, People's Republic of China

L. Wang
Department of Automation, Tsinghua University,
Beijing 100084, People's Republic of China

1 Introduction

Production scheduling plays a key role in the manufacturing systems of enterprises for maintaining a competitive position in fast-changing markets, so it is very important to develop effective, efficient, and advanced manufacturing and scheduling technologies and approaches [1–8]. In this paper, the no-wait flow shop scheduling problem with makespan criterion is considered, which has important applications in different industries, including chemical processing [9], food processing [10], concrete ware production [11], and pharmaceutical processing [12]. In the no-wait flow shop, the processing of each job has to be continuous. That is, once a job is started on the first machine, it has to be processed through all machines without any interruption. Therefore, when needed, the start of a job on the first machine must be delayed in order to meet the no-wait requirement. The no-wait condition ensures that any no-wait schedule must be a permutation schedule, and it is well known that the no-wait flow shop scheduling problem with more than two machines is strongly NP-hard [13]. Therefore, only small-sized instances of the no-wait flow shop scheduling problem can be solved optimally within a reasonable computational time using exact algorithms. As the problem size increases, the computational time of the exact methods grows exponentially. On the other hand, heuristic algorithms have generally acceptable time and memory requirements to obtain a near-optimal or optimal solution. In the past few decades, most research focused on developing heuristic algorithms. These solution techniques can be broadly classified into two groups, referred to as constructive methods [9, 14–16] and metaheuristics, including simulated annealing (SA) [17], genetic algorithm (GA) [17], hybrid GA and SA (GASA) [18], variable neighborhood search (VNS) [18], descending search (DS) [19], tabu search (TS)

[19], hybrid particle swarm optimization (HPSO) [5], and discrete particle swarm optimization (DPSO) [2]. A comprehensive survey on the no-wait flow shop scheduling problem can be found in [10].

As a very simple and powerful metaheuristic, the iterated greedy algorithm (IGA) has been proved to be highly effective when compared to the state-of-the-art methods for the permutation flow shop scheduling problem [20]. In IGA, a greedy constructive heuristic is repeatedly applied to an incumbent solution, and an acceptance criterion is employed to decide whether the newly constructed solution will replace the incumbent solution after the construction phase. Due to the simple concept and easy implementation, IGA was successfully applied to the set covering [21, 22] and flow shop scheduling problems [20]. To the best of our knowledge, however, there is no work published on dealing with the no-wait flow shop scheduling problem using IGA. In this paper, an improved IGA (IIGA) is proposed for the no-wait flow scheduling problem with makespan criterion. In the proposed IIGA, a speed-up method for the insert neighborhood is developed to evaluate the whole insert neighborhood of a single solution with $(n-1)^2$ neighbors in time $O(n^2)$, where n is the number of jobs. The well known Nawaz-Enscore-Ham (NEH) heuristic [23] and its modified variant are employed to construct new solutions, and a local search based on the speed-up method is applied to perform exploitation and a simple SA-like acceptance criterion is employed to accept new solutions.

The rest of the paper is organized as follows. In Sect. 2, the no-wait flow shop scheduling problem is introduced. In Sect. 3, the speed-up method for the insert neighborhood is proposed. In Sect. 4, the proposed IIGA is presented in detail. The computational results and comparisons are provided in Sect. 5. Finally, we end the paper with some conclusions in Sect. 6.

2 The no-wait flow shop scheduling problem

The no-wait flow shop scheduling problem can be described as follows. Each of n jobs from the set $J=\{1, 2,..., n\}$ will be sequenced through m machines ($k=1, 2,..., m$). Job $j\in J$ has a sequence of m operations ($o_{j1}, o_{j2},..., o_{jm}$). Operation o_{jk} corresponds to the processing of job j on machine k during an uninterrupted processing time $p(j, k)$. At any time, each machine can process at most one job and each job can be processed on at most one machine. To follow the no-wait restriction, the completion time of operation o_{jk} must be equal to the earliest start time of operation $o_{j,\,k+1}$ for $k=1, 2,..., m-1$. In other words, there must be no waiting time between the processing of any consecutive operations of each of the n jobs. The problem is, then, to find a schedule such that the processing order of the jobs is the same on each machine and the maximum completion time, so called makespan, is minimized.

Suppose that the job permutation $\pi=\{\pi_1, \pi_2,..., \pi_n\}$ represents the schedule of jobs to be processed. Let $d(\pi_{j-1}, \pi_j)$ be the minimum delay restricted on the first machine between the start of jobs π_{j-1} and π_j by the no-wait constraint when job π_j is directly processed after job π_{j-1}, and let $C_{j-1,\,j}(\pi_j, m)$ denote the makespan of jobs π_{j-1} and π_j in the $2/m/P/C_{\max}$ problem. Since the makespan for $2/m/$ *no-wait* P/C and $2/m/P/C_{\max}$ is the same, and in the no-wait flow shop scheduling problem the difference between the completion time of a job's last operation and the starting time of its first operation is equal to the sum of its operation times on all machines [19], $d(\pi_{j-1}, \pi_j)$ can be computed as follows:

$$d(\pi_{j-1},\ \pi_j) = C_{j-1,\,j}(\pi_j,\ m) - \sum_{k=1}^{m} p(\pi_j,\ k) \quad \text{for } j = 2, \ldots,\ n \tag{1}$$

Then, the makespan of the job permutation $\pi=\{\pi_1, \pi_2,..., \pi_n\}$ can be given by:

$$C_{\max}(\pi) = \sum_{j=2}^{n} d(\pi_{j-1},\ \pi_j) + \sum_{k=1}^{m} p(\pi_n,\ k) \tag{2}$$

Therefore, the no-wait flow shop scheduling problem with makespan criterion is to find a permutation π^* in the set of all permutations Π such that:

$$C_{\max}(\pi^*) \le C_{\max}(\pi), \qquad \forall \pi \in \Pi \tag{3}$$

The complexity of calculation with Eq. 1 is $O(m)$, and that with Eq. 2 is $O(mn)$. For a no-wait flow shop with n jobs, since all of the possible pairs of $d(\pi_{j-1}, \pi_j)$ is no more than $n(n-1)$, and the number of $\sum_{k=1}^{m} p(\pi_j,\ k)$ is no more than n, they can be computed in advance to be used in the evaulation of a permutation. Thus, the complexity of Eq. 2 is reduced to $O(n)$.

3 Speed-up method

Two different kinds of neighborhoods for a permutation π of jobs, i.e., insertion and swap, are widely used in the literature. However, from the literature [19] and our previous experience, it is found that the swap neighborhood is not better than the insert neighborhood for efficiently obtaining permutations with better quality. For this reason, only the insert neighborhood is considered in this paper. The insert neighborhood of a permutation π is defined by considering all possible insert moves $v(j, k)$, $j, k\in\{1, 2,..., n\}$ by removing a job of π from its original position j and inserting it into position k ($k\notin\{j, j-1\}$).

Thus, the move $v(j, k)$ generates a permutation π' from π in the following way:

$$\pi' = \{\pi_1, ..., \pi_{j-1}, \pi_{j+1}, ..., \pi_k, \pi_j, \pi_{k+1}, ..., \pi_n\} \quad \text{if } j < k \tag{4}$$

$$\pi' = \{\pi_1, ..., \pi_{k-1}, \pi_j, \pi_{k+1}, ..., \pi_j, \pi_{j+1}, ..., \pi_n\} \quad \text{if } j > k \tag{5}$$

Based on the similarity of π' and π, a shortcut to calculate the makespan of π' can be given as follows:

Step 1: Let $\pi''=\{\pi_1'', \pi_2'',..., \pi_{n-1}''\}$ be a partial permutation generated by removing job π_j from permutation π, and $T(\pi_j)$ be the total processing time of job π_j. As seen in Fig. 1, the makespan of the partial permutation π'' can be obtained as:

$$C_{\max}(\pi'') = C_{\max}(\pi) - d(\pi_j, \pi_2) \quad \text{if } j = 1 \tag{6}$$

$$C_{\max}(\pi'') = C_{\max}(\pi) - d(\pi_{j-1}, \pi_j) - T(\pi_j) + T(\pi_{j-1}) \quad \text{if } j = n \tag{7}$$

$$C_{\max}(\pi'') = C_{\max}(\pi) - d(\pi_{j-1}, \pi_j) - d(\pi_j, \pi_{j+1}) + d(\pi_{j-1}, \pi_{j+1}) \quad \text{if } 1 < j < n \tag{8}$$

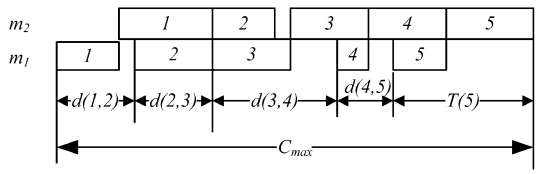
Step 2: As seen in Fig. 2, the makespan of permutation π' generated by inserting π_j into position k ($k \notin \{j, j-1\}$) of π'' can be calculated as:

$$C_{\max}(\pi') = C_{\max}(\pi'') + d\left(\pi_j, \pi_k''\right) \quad \text{if } k = 1 \tag{9}$$

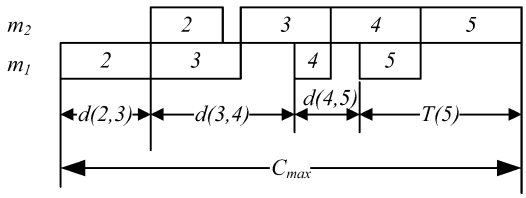
$$C_{\max}(\pi') = C_{\max}\left(\pi''\right) + d\left(\pi_{k-1}'', \pi_j\right) - T\left(\pi_{k-1}''\right) + T(\pi_j) \quad \text{if } k = n \tag{10}$$

$$C_{\max}(\pi') = C_{\max}(\pi'') + d\left(\pi_{k-1}'', \pi_j\right) + d\left(\pi_j, \pi_k''\right) - d\left(\pi_{k-1}'', \pi_k''\right) \quad \text{if } 1 < k < n \tag{11}$$

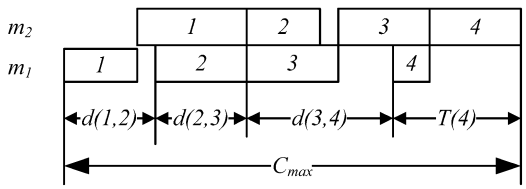
All of these steps can be executed in time $O(1)$ and the number of neighbors in the insert neighborhood is $(n-1)^2$.



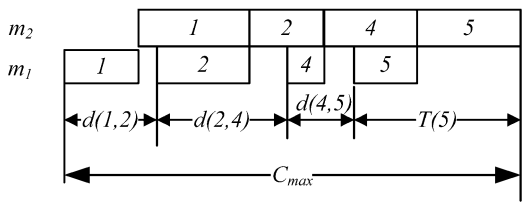
a makespan of π



b makespan of π'' generated by removing job 1 from π



c makespan of π'' generated by removing job 5 from π



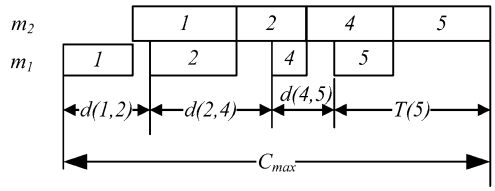
d makespan of π'' generated by removing job 3 from π

Fig. 1 Calculating the makespan after removing a job

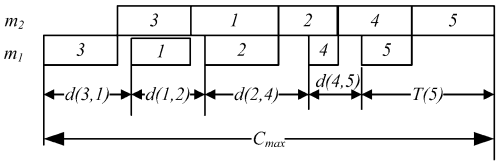
So, the complexity is $O(n^2)$ to evaluate the whole insert neighborhood of a single solution.

4 The improved iterated greedy algorithm

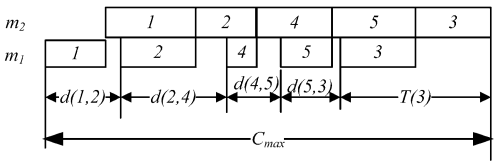
The iterated greedy algorithm (IGA) is a simple and effective metaheuristic that iteratively applies constructive heuristics to the current search solution [20]. Once a



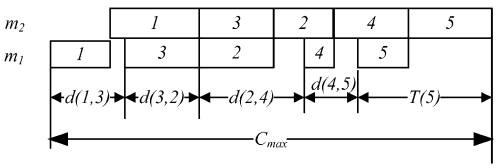
a makespan of π''



b makespan of π' generated by inserting job 3 into position 1



c makespan of π' generated by inserting job 3 into position 5



d makespan of π' generated by inserting job 3 into position 2

Fig. 2 Calculating the makespan after inserting job 3

constructed solution has been generated, an acceptance criterion is used to decide which solution will be reserved in the next iteration. Here, we will propose an improved IGA (IIGA) after introducing some main elements.

4.1 Initial solution and construction phase

It is concluded that the NEH heuristic [23] is the best constructive method for the permutation flow shop scheduling problem for a wide variety of instances [24, 25]. The procedure of the NEH heuristic can be described as follows:

Step 1: Sort the jobs according to the descending sums of their processing times. Let the resulting sequence be $\pi=\{\pi_1, \pi_2,..., \pi_n\}$.

Step 2: The first two jobs of π are taken and the two possible sub-sequences of these two jobs are evaluated. Then, the better sub-sequence is selected as the current sequence.

Step 3: Repeat the previous steps until all jobs are sequenced. Take job π_j, j=3, 4,..., n and find the best sub-schedule by placing it in all possible positions of the sub-sequence of jobs that have been already scheduled. The best sub-sequence is selected for the next generation.

For the no-wait flow shop scheduling problem, Eq. 2 can be easily adopted in the NEH heuristic to calculate the makespan of a sequence. Because the NEH heuristic evaluates a total of $[n(n+1)/2]-1$ schedules, the complexity of the NEH heuristic is $O(n^3)$. By using the speed-up method proposed in Sect. 3, the complexity of the NEH heuristic will be reduced to $O(n^2)$.

In addition, we present a modified NEH heuristic (named the M_NEH heuristic) by slightly modifying Step 2 and Step 3 of the NEH heuristic. The procedure of the M_NEH heuristic is described as follows:

Step 1: Sort the jobs according to the descending sums of their processing times. Let the resulting sequence be $\pi=\{\pi_1, \pi_2,..., \pi_n\}$.

Step 2: The last two jobs from π are taken and the two possible sub-schedules of these two jobs are evaluated. Then, the better sub-sequence is selected as the current sequence.

Step 3: Repeat the previous steps until all jobs are sequenced. Take job π_j, $j=n-2$, $n-3$,..., 1, and find the best sub-schedule by placing it in all possible positions of the sub-sequence of jobs that are already scheduled. The best sub-sequence is selected for the next generation.

We employ 31 benchmarks from [26–28] with different sizes to compare the performances of NEH with M_NEH. The results are listed in Tables 1 and 2, where the percent relative deviation (PRD) is calculated as follows:

$$\mathrm{PRD} = \frac{(M_i - M_{\mathrm{ref}}) * 100}{M_{\mathrm{ref}}} \tag{12}$$

where M_i is the makespan obtained by NEH or M_NEH and M_{ref} is the optimal value or upper bound of each instance [9].

From Tables 1 and 2, it can be seen that the performance of the M_NEH heuristic is slightly better than that of the NEH heuristic. Since M_NEH is of the same computational complexity as NEH, therefore, M_NEH will be employed to construct an initial sequence in IIGA. In addition, we design a construction phase in IIGA, which consists of a destruction procedure and a construction procedure. In the destruction procedure, d jobs are randomly removed from

Table 1 Comparison of the Nawaz-Enscore-Ham (NEH) and modified NEH (M_NEH) heuristics with respect to the upper bounds provided by RAJ [9]

Instance		RAJ	NEH	M_NEH
Name	M×J	C_{max}	PRD	PRD
Rec01	5×20	1,590	−1.32	0.06
Rec03	5×20	1,457	1.30	−0.34
Rec05	5×20	1,637	−4.52	−3.24
Rec07	10×20	2,119	3.21	−0.80
Rec09	10×20	2,141	−1.31	1.03
Rec11	10×20	1,946	0.62	−1.64
Rec13	15×20	2,709	−0.15	−0.48
Rec15	15×20	2,691	−4.46	−4.12
Rec17	15×20	2,740	−4.27	−0.99
Rec19	10×30	3,157	−5.54	−3.39
Rec21	10×30	3,015	−1.46	−1.69
Rec23	10×30	3,030	−5.48	−5.64
Rec25	15×30	3,835	−2.27	−1.33
Rec27	15×30	3,655	−2.49	−2.93
Rec29	15×30	3,583	−4.16	−6.25
Rec31	10×50	4,631	−2.33	−3.58
Rec33	10×50	4,770	−0.46	−1.72
Rec35	10×50	4,718	−3.03	−3.20
Rec37	20×75	8,979	−6.75	−6.43
Rec39	20×75	9,158	−2.31	−2.05
Rec41	20×75	9,344	−3.70	−5.84
Hel1	10×100	780	−4.10	−4.23
Hel2	10×20	189	−1.06	−1.06
Mean			−2.44	−2.6

the incumbent sequence π, and the remaining jobs construct a partial sequence π_1. These d jobs construct another sequence π_2 in the order in which they were chosen. In the construction procedure, the first job from π_2 is taken and Step 3 of NEH or M_NEH is randomly applied according a certain probability p to decide which position of π_1 should be inserted. This procedure is repeated until π_2 is empty. In particular, a uniform random number r is generated between 0 and 1. If r is less than p, then the NEH heuristic is applied to generate a new solution; otherwise, M_NEH is employed. Since the NEH and M_NEH heuristics use different methods to construct the solutions, it is helpful to enrich the constructing method and to avoid being trapped in certain solutions by randomly applying NEH or M_NEH.

4.2 Local search

After the construction phase, a local search procedure based on the insert neighborhood is applied to the constructed solution to generate a new solution.

For the sake of simple and easy implementation, we use a first-improvement type of pivoting rule outlined in Fig. 3. In order to reduce the CPU time requirements, the speed-up method presented in Sect. 3 is also applied.

4.3 Acceptance criterion

Inspired by the mechanism of simulated annealing (SA), which is of the ability to probabilistically escape from local optima, an SA-type of acceptance criterion is adopted. That is, the improved solution π' by local search is accepted with a probability of $\exp\{(C_{\max}(\pi) - \frac{C_{\max}(\pi')}{t})\}$, where π is the current solution and t is a control parameter that decreases as the algorithm runs. In this paper, we set the initial temperature as follows [29]:

$$t_0 = \frac{\sum_{j=1}^{n}\sum_{k=1}^{m} p(j,\ k)}{10 \cdot n \cdot m} \cdot h \tag{13}$$

Moreover, an exponential cooling schedule is applied ($t_k=\lambda t_{k-1}$, $0<\lambda<1$), which is often believed to be an excellent cooling recipe, since it provides a rather good compromise between a computationally fast schedule and the ability to reach a low-energy state [30].

4.4 Procedure of the IIGA

The procedure of the IIGA is described as follows:

Step 1: Initialize the parameters and generate an initial solution
Step 2: Apply the insert-based local search to improve the initial solution
Step 3: Randomly apply NEH or M_NEH according to a certain probability to construct a new solution based on the current one
Step 4: Apply the insert-based local search to the constructed solution to generate a new solution
Step 5: Apply an acceptance criterion to decide whether the new solution is accepted or not
Step 6: If a stopping criterion is met, then output the solution found best-so-far; otherwise, go back to Step 3.

Table 2 Comparison of the NEH and M_NEH heuristics with the optimal solutions

Instance		Optimum	NEH	M_NEH
Name	M×J	C_{max}	PRD	PRD
Car01	5×11	8,142	0.00	2.85
Car02	4×13	8,242	0.62	0.30
Car03	5×12	8,866	1.05	0.00
Car04	4×14	9,195	9.07	3.58
Car05	4×10	9,159	6.62	2.65
Car06	9×8	9,690	0.00	2.22
Car07	7×7	7,705	2.57	2.43
Car08	8×8	9,372	0.36	0.36
Mean			2.54	1.8

Fig. 3 The local search procedure

```
LocalSearch(π)
{
π*=π;
do{
   π=π*;
   generate a seed sequences π^0 of jobs randomly;
  for i=1 to n do {
     remove job π^0[i] from π* and obtained a partial sequence π";
     π'=best sequence obtained by inserting job π^0[i] in any possible positions of π";
     if Cmax(π')<Cmax(π*) π*=π';}
  } while(Cmax(π*)<Cmax(π))
return π.
}
```

5 Computational results

The proposed IIGA was coded in C++ and run on an Intel P4 3.0 GHz PC with 1 GB of memory. In total, 31 benchmarks taken from [26–28] were used for testing. The parameters were set as follows: h=1.5, λ=0.999, d=min (0.5n, 15), p=0.6, and the maximum number of generations was set to 1,000. For each instance, 20 runs were conducted, and each run was compared to the optimal solutions or the upper bounds (denoted RAJ) [9]. The average PRD over 20 runs is denoted by ARD and the average CPU time until termination is denoted by T_{AVG}. They are listed in Tables 3 and 4. Moreover, the ARD was computed as follows:

$$\mathrm{ARD} = \frac{1}{20}\sum_{i=1}^{20}\left(\frac{(M_i - M_{\mathrm{ref}})\cdot 100}{M_{\mathrm{ref}}}\right) \tag{14}$$

where M_i is the makespan generated by the proposed IIGA in each run and M_{ref} is the optimal value or upper bound of the instance reported in [9]. Furthermore, the best relative percent deviation (BRD), the worst relative percent deviation (WRD), and the standard deviation (SD) are also reported.

We compare the proposed IIGA with several existing metaheuristics from the literature for the problem considered in this paper, including TS, TS+M, TS+MP from [19] and DPSO from [2]. The BRD, ARD, WRD, SD, and T_{AVG} of each algorithm are listed in Tables 3 and 4.

The computational results presented in Table 3 show that IIGA performs slightly better than TS, TS+M, TS+MP, and DPSO in terms of the mean PRD value. In terms of the CPU time requirements, IIGA is much faster than DPSO, since the T_{AVG} of IIGA is 0.07 s, while DPSO is 0.44 s. Even though we employed a machine approximately 3.0 times (3000/1000=3.0) faster than the one used by Grabowski and Pempera [19], we can see that IIGA is faster than TS, TS+M, and TS+MP, as 0.07 s is much shorter than 0.9/3, 0.9/3, and 1.3/3, respectively. Moreover, IIGA is more robust than DPSO, since the mean SD generated by IIGA is much smaller than that by DPSO. In addition, Table 4 summarizes the results for the small instances whose optimal solutions were obtained by the branch and bound algorithm. These results confirm the favorable performance of IIGA in terms of ARD values and the CPU times as well. So, it can be concluded that the proposed IIGA is more effective and efficient than TS, TS +M, TS+MP, and DPSO for the no-wait flow shop scheduling problem with makespan criterion.

Table 3 Comparison of the algorithms with respect to upper bounds provided by RAJ [9]

Instance		RAJ	TS		TS+M		TS+MP		DPSO					IIGA				
Name	M×J	C_{max}	PRD	T_{avg}	PRD	T_{avg}	PRD	T_{avg}	BRD	WRD	ARD	SD	T_{avg}	BRD	WRD	ARD	SD	T_{avg}
Rec01	5×20	1,590	−4.03	0.2	−3.96	0.2	−3.96	0.2	−4.03	−4.03	−4.03	0.00	0.06	−4.03	−4.03	−4.03	0.00	0.02
Rec03	5×20	1,457	−6.59	0.2	−6.59	0.2	−6.59	0.2	−6.59	−6.59	−6.59	0.00	0.06	−6.59	−6.59	−6.59	0.00	0.02
Rec05	5×20	1,637	−7.39	0.2	−7.64	0.2	−7.70	0.2	−7.70	−7.64	−7.69	0.01	0.06	−7.70	−7.51	−7.69	0.04	0.02
Rec07	10×20	2,119	−3.63	0.2	−3.63	0.2	−3.63	0.2	−3.63	−3.63	−3.63	0.00	0.06	−3.63	−3.63	−3.63	0.00	0.02
Rec09	10×20	2,141	−4.62	0.2	−4.58	0.2	−4.58	0.2	−4.62	−4.62	−4.62	0.00	0.06	−4.62	−4.58	−4.62	0.02	0.02
Rec11	10×20	1,946	−3.34	0.2	−3.34	0.2	−3.34	0.2	−3.34	−3.34	−3.34	0.00	0.06	−3.34	−3.34	−3.34	0.00	0.02
Rec13	15×20	2,709	−6.05	0.3	−6.05	0.3	−6.05	0.3	−6.05	−6.05	−6.05	0.00	0.06	−6.05	−6.05	−6.05	0.00	0.02
Rec15	15×20	2,691	−5.91	0.3	−6.02	0.3	−5.91	0.3	−6.02	−6.02	−6.02	0.00	0.06	−6.02	−6.02	−6.02	0.00	0.02
Rec17	15×20	2,740	−5.58	0.3	−5.58	0.3	−5.58	0.3	−5.58	−5.58	−5.58	0.00	0.08	−5.58	−5.58	−5.58	0.00	0.02
Rec19	10×30	3,157	−9.72	0.4	−9.25	0.4	−9.38	0.4	−9.72	−9.12	−9.62	0.18	0.17	−9.72	−9.72	−9.72	0.00	0.04
Rec21	10×30	3,015	−6.31	0.4	−6.30	0.4	−6.17	0.4	−6.43	−6.04	−6.33	0.12	0.17	−6.43	−6.10	−6.27	0.11	0.04
Rec23	10×30	3,030	−10.76	0.4	−10.73	0.4	−10.89	0.4	−10.89	−10.07	−10.83	0.18	0.17	−10.89	−10.89	−10.89	0.00	0.04
Rec25	15×30	3,835	−5.97	0.5	−6.31	0.5	−6.21	0.5	−6.31	−5.48	−6.15	0.30	0.17	−6.31	−6.21	−6.30	0.03	0.04
Rec27	15×30	3,655	−5.64	0.5	−6.10	0.5	−5.83	0.5	−6.13	−5.64	−5.89	0.14	0.17	−6.13	−6.05	−6.12	0.02	0.05
Rec29	15×30	3,583	−7.94	0.5	−8.28	0.5	−7.94	0.5	−8.15	−7.56	−8.09	0.15	0.17	−8.15	−7.87	−8.14	0.06	0.06
Rec31	10×50	4,631	−5.90	1.1	−6.13	1.1	−6.22	1.1	−6.80	−6.35	−6.55	0.12	0.56	−6.91	−6.35	−6.65	0.17	0.11
Rec33	10×50	4,770	−5.51	1.1	−6.31	1.1	−6.37	1.1	−6.94	−6.02	−6.49	0.26	0.56	−6.79	−6.25	−6.55	0.16	0.10
Rec35	10×50	4,718	−6.08	1.1	−6.17	1.1	−5.91	1.1	−6.80	−6.00	−6.44	0.23	0.56	−6.80	−6.32	−6.66	0.13	0.10
Rec37	20×75	8,979	−9.41	2.5	−9.49	2.6	−9.36	2.6	−10.50	−9.46	−10.14	0.27	1.47	−10.46	−9.68	−10.13	0.26	0.18
Rec39	20×75	9,158	−7.00	2.5	−6.99	2.6	−6.91	2.6	−7.80	−6.88	−7.33	0.23	1.45	−7.73	−6.88	−7.40	0.21	0.18
Rec41	20×75	9,344	−8.78	2.5	−8.57	2.6	−8.82	2.6	−9.33	−8.54	−9.01	0.22	1.47	−9.32	−8.71	−8.98	0.17	0.18
Hel1	10×100	780	−8.08	3.8	−8.21	3.9	−8.33	3.9	−9.49	−8.85	−9.18	0.18	2.41	−9.62	−8.59	−8.96	0.27	0.29
Hel2	10×20	189	−5.29	0.2	−5.29	0.2	−5.29	0.2	−5.29	−5.29	−5.29	0.00	0.06	−5.29	−5.29	−5.29	0.00	0.02
Mean			−6.50	0.9	−6.59	0.9	−6.56	1.3	−6.88	−6.47	−6.73	0.11	0.44	−6.88	−6.62	−6.77	0.07	0.07

TS, TS+M, and TS+MP were run on a Pentium 1.0 GHz computer (Grabowski and Pempera [19]). DPSO and IIGA were run on a Pentium IV 3.0 GHz computer.

Table 4 Comparison of algorithms with the optimal solutions

Instance		TS		TS+M		TS+MP		DPSO				IIGA				
Name	M×J	PRD	T_{avg}	PRD	T_{avg}	PRD	T_{avg}	BRD	WRD	ARD	SD	T_{avg}	BRD	WRD	ARD	SD	T_{avg}
Car01	5×11	0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01
Car02	4×13	0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01
Car03	5×12	0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01
Car04	4×14	0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.01
Car05	4×10	0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01
Car06	9×8	0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Car07	7×7	0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Car08	8×8	0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Mean		0.00	0.10	0.00	0.1	0.00	0.1	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01

TS, TS+M, and TS+MP were run on a Pentium 1.0 GHz computer (Grabowski and Pempera [19]). DPSO and IIGA were run on a Pentium IV 3.0 GHz processor.

6 Conclusions

To the best of our knowledge, this is the first report on the application of an iterated greedy algorithm (IGA) to the no-wait flow shop scheduling problem with makespan criterion. We used the improved Nawaz-Enscore-Ham (NEH) heuristic to generate an initial solution with a certain quality, and employed the construction procedure based on the NEH (or modified NEH, M_NEH) heuristic and the simulated annealing (SA) type acceptance criterion to avoid the search getting trapped in local minima. We then applied the simple local search to stress local exploitation and developed a speed-up technology on evaluating the insert neighborhood solution to improve the efficiency. Computational results demonstrated the superiority of the proposed improved iterated greedy algorithm (IIGA) in terms of efficiency and effectiveness. The future work is to develop an IGA for other kinds of combinatorial optimization problems and to develop some multi-objective IGAs for multi-objective scheduling problems.

Acknowledgments The authors wish to thank the editor and the referees for their constructive comments on the earlier manuscript of this paper. This research is partially supported by the National Science Foundation of China, National 863 Hi-Tech R&D Plan, and the project sponsored by SRF for ROCS, SEM.

References

1. Stadtler H (2005) Supply chain management and advanced planning—basics, overview and challenges. Eur J Oper Res 163 (3):575–588
2. Pan Q-K, Tasgetiren MF, Liang Y-C (2005) A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem with makespan criterion. In: Proceedings of the 24th Annual Workshop of the UK Planning and Scheduling Special Interest Group (UK PlanSIG 2005), London, UK, December 2005, pp 31–41
3. Wang L, Zheng D-Z (2003) An effective hybrid heuristic for flow shop scheduling. Int J Adv Manuf Technol 21(1):38–44
4. Dimopoulos C, Zalazala AMS (2000) Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons. IEEE Trans Evolut Comput 4 (2):93–113
5. Liu B, Wang L, Jin Y-H (2007) An effective hybrid particle swarm optimization for no-wait flow shop scheduling. Int J Adv Manuf Technol 31:1001–1011
6. Pan Q-K, Tasgetiren MF, Liang Y-C (2006) Minimizing total earliness and tardiness penalties with a common due date on a single-machine using a discrete particle swarm optimization algorithm. Lect Notes Comput Sci 4150:460–467
7. Li B-B, Wang L (2007) A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling. IEEE Trans Syst Man Cybern B Cybern 37:576–591
8. Wang L (2003) Shop scheduling with genetic algorithms. Tsinghua University Press, Beijing, China

9. Rajendran C (1994) A no-wait flowshop scheduling heuristic to minimize makespan. J Oper Res Soc 45:472–478
10. Hall NG, Sriskandarayah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. Oper Res 44:510–525
11. Grabowski J, Pempera J (2000) Sequencing of jobs in some production system. Eur J Oper Res 125:535–550
12. Raaymakers WHM, Hoogeveen JA (2000) Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. Eur J Oper Res 126:131–151
13. Rock H (1984) The three-machine no-wait flow shop problem is NP-complete. J Assoc Comput Machinery 31:336–345
14. Bonney MC, Gundry SW (1976) Solutions to the constrained flowshop sequencing problem. Oper Res Q 27:869–883
15. King JR, Spachis AS (1980) Heuristics for flowshop scheduling. Int J Prod Res 18:343–357
16. Gangadharan R, Rajedran C (1993) Heuristic algorithms for scheduling in the no-wait flowshop. Int J Prod Econ 32:285–290
17. Aldowaisan T, Allahverdi A (2003) New heuristics for no-wait flowshops to minimize makespan. Comput Oper Res 30:1219–1231
18. Schuster CJ, Framinan JM (2003) Approximative procedures for no-wait job shop scheduling. Oper Res Lett 31:308–318
19. Grabowski J, Pempera J (2005) Some local search algorithms for no-wait flow-shop problem with makespan criterion. Comput Oper Res 32:2197–2212
20. Ruiz R, Stützle T (2007) A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 177:2033–2049
21. Jacobs LW, Brusco MJ (1995) A local-search heuristic for large set-covering problems. Nav Res Logist Q 42(7):1129–1140
22. Marchiori E, Steenbeek A (2000) An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. Lect Notes Comput Sci 1803:367–381
23. Nawaz M, Enscore EE Jr, Ham I (1983) A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem. Omega 11:91–95
24. Koulamas C (1998) A new constructive heuristic for the flowshop scheduling problem. Eur J Oper Res 105:66–71
25. Turner S, Booth D (1987) Comparison of heuristics for flow shop sequencing. Omega 15(1):75–85
26. Carlier J (1978) Ordonnancements à contraintes disjonctives. RAIRO Rech Oper 12:333–351
27. Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22:5–13
28. Heller J (1960) Some numerical experiments for an M×J flow shop and its decision-theoretical aspects. Oper Res 8:178–184
29. Osman IH, Potts CN (1989) Simulated annealing for permutation flow-shop scheduling. Omega 17(6):551–557
30. Wang L, Zheng D-Z (2001) An effective hybrid optimization strategy for job-shop scheduling problems. Comput Oper Res 28 (6):585–596