Int. J. Production Economics 128 (2010) 144–152

Contents lists available at ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

A hybrid genetic algorithm for no-wait flowshop scheduling problem

Lin-Yu Tseng [a,b,*], Ya-Tai Lin [b]

[a] Institute of Networking and Multimedia, National Chung Hsing University, 250 Kuo Kuang Road, Taichung, Taiwan
[b] Department of Computer Science and Engineering, National Chung Hsing University, 250 Kuo Kuang Road, Taichung, Taiwan

ARTICLE INFO

Article history:
Received 13 September 2007
Accepted 4 June 2009
Available online 17 June 2010

Keywords:
Genetic algorithm
Local search
Hybrid genetic algorithm
No-wait flowshop scheduling

ABSTRACT

In this paper, a hybrid genetic algorithm is proposed to solve the no-wait flowshop scheduling problem with the makespan objective. The proposed algorithm hybridizes the genetic algorithm and a novel local search scheme. The OA-crossover operator is designed to enhance the capability of intensification in the genetic algorithm. The proposed local search scheme combines two local search methods: the Insertion Search (IS) and a novel local search method called the Insertion Search with Cut-and-Repair (ISCR). These two local search methods play different roles in the search process. The Insertion Search is responsible for searching a small neighborhood while the Insertion Search with Cut-and-Repair is responsible for searching a large neighborhood. The experimental results show the advantage of combining the two local search methods. Extensive experiments were conducted to evaluate the proposed hybrid genetic algorithm and the results revealed that the proposed algorithm is very competitive. It obtained the same best solutions that were reported in the literature for all problems in the benchmark provided by Carlier (1978). Also, it improved 5 out of the 21 current best solutions reported in the literature and achieved the current best solutions for 14 of the remaining 16 problems in the benchmark presented by Reeves (1995). Furthermore, the proposed algorithm was applied to effectively solve the 120 problems in the benchmark provided by Taillard (1990).

1. Introduction

The flowshop scheduling problem is an important scheduling problem and has been extensively studied since it was proposed in 1954 by Johnson. We consider the flowshop scheduling problem with the no-wait constraints in this paper. In a no-wait flowshop scheduling, once a job is started on the first machine, it has to be continuously processed through completion at the last machine without interuptions. In other words, the starting time of a job on the first machine may have to be delayed in order to meet the no-wait constraints. The problem is formally defined in Section 2.

The no-wait flowshop scheduling problem with makespan criterion was proved NP-hard by Rock (1984) (Reeves, 1995). Therefore, instead of trying to find the optimal solution, efforts have been devoted to designing the heuristic and metaheuristic methods in order to find high-quality solutions in a reasonable computation time.

Some heuristic algorithms were proposed to solve the no-wait flowshop scheduling problem. For example, Reddi and Ramamoorthy (1972), Wismer (1972) and Bonney and Gundry (1976) proposed their methods in the 1970s. King and Spachis (1980) proposed their method in the 1980s. In the 1990s, Gangadharan and Rajendran (1993) and Rajendran (1994) proposed two heuristic methods GAN–RAJ and RAJ. They showed by experiments that these methods outperformed the previous heuristic methods reported in the literature. Both GAN–RAJ and RAJ use the same method to generate the initial sequence. After the initial sequence is generated, GAN–RAJ picks the job one by one from the beginning of the sequence to the end of the sequence and inserts the pick-up job into every position behind the original position of the pick-up job trying to find a better scheduling. The only difference between RAJ and GAN–RAJ is that RAJ inserts the pick-up job into positions between $(h+1)/2$ and $h+1$, where h is the position of the pick-up job. The performance of RAJ is shown to be better than that of GAN–RAJ.

Recently, several metaheuristic methods were proposed to solve this problem. Aldowaisan and Allahverdi (2003) designed methods based on simulated annealing and genetic algorithm, respectively. The proposed methods use local search methods that combine NEH heuristic, the insertion operator and the exchange operator. Their methods outperform RAJ heuristic. Schuster and Framinan (2003) presented two metaheuristic methods: the

* Corresponding author at: Institute of Networking and Multimedia, National Chung Hsing University, 250 Kuo Kuang Road, Taichung, Taiwan.
Tel.: +886 4 22874020; fax: +886 4 22853869.
E-mail address: lytseng@cs.nchu.edu.tw (L.-Y. Tseng).

0925-5273/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ijpe.2010.06.006

variable neighborhood search (VNS) and the algorithm (GASA) that hybridizes the genetic algorithm (GA) and the simulated annealing (SA). Their methods also perform better than RAJ. Grabowski and Pempera (2005) proposed three methods based on tabu search (TS): TS, TS+M and TS+MP, In TS, a single insertion move is used as the neighborhood search method. In TS+M, multiple insertion moves are used, while in TS+MP, single insertion move is used and if the solution is not improved after a number of consecutive iteration, multiple insertion moves will be used. Revealed by experimental results, these tabu search methods outperform the VNS and the GASA. Liu et al. (2007) proposed algorithms based on the particle swarm optimization (PSO). They developed two local search methods: the NEH-based local search method and the SA-based local search method. Then, they hybridized the PSO and the NEH-based local search method as their first algorithm, and they hybridized the POS and the SA-based local search method as their second algorithm. Both methods outperform the VNS and the GASA but are slightly inferior to tabu search methods. In addition to above mentioned studies, recently Lin et al. (2008) developed new features of ant colony optimization for flowshop scheduling problems. Haouari and Hidri (2008) proposed a new lower bound for the hybrid flowshop scheduling problem. Liu and Kozan (2009) considered four inter-machine buffer conditions in the flowshop scheduling problem.

In the recent years, many hybrid genetic algorithms have been developed for various kinds of problems including the scheduling problem. In general, the genetic algorithm acting as a global search scheme is hybridized with a local search scheme in order to enhance both diversification and intensification (Reeves, 1994). Some hybrid genetic algorithms for scheduling problems are surveyed in the following. Liaw (2000) hybridized the genetic algorithm with the tabu search to solve the open shop scheduling problem and the hybrid GA found better solutions for some benchmark problems. Gonçalves et al. (2005) proposed a local search method based on the critical path and combined the genetic algorithm and this local search method to solve the job shop scheduling problem. Park (2001) (Reeves, 1995) developed the greedy interchange local optimization algorithm as the local search scheme in the hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines and obtained better results than previous research works. Valls et al. (2008) presented a hybrid genetic algorithm for the resource-constrained project scheduling problem. They used two-phase strategy in their algorithm. Also the peak crossover operator and the double justification operator specifically designed for the resource-constrained project scheduling problem were used. Their results outperform the previous research works. For the no-wait flowshop scheduling problem considered in this study, Gonazelez et al. (1995) proposed a hybrid genetic algorithm that hybridized the genetic algorithm with three problem oriented operators based on the heuristics developed by Gupta (1971), Palmer (1965) and Rajendran (1994), respectively. The results obtained are better than those obtained by the traditional genetic algorithm.

In this paper, an algorithm is proposed that hybridizes the genetic algorithm and two local search methods. The genetic algorithm acts as the global search scheme. Insertion Search (IS) is used to search small neighborhoods while Insertion Search with Cut-and-Repair (ISCR) is used to search large neighborhoods. The combination of IS and ISCR results in a novel powerful local search scheme. As the experimental results on benchmark problems show, the proposed hybrid GA outperforms the VNS (Schuster and Framinan, 2003), the GASA (Schuster and Framinan, 2003), the tabu search methods (Grabowski and Pempera, 2005) and the PSO-based methods (Liu et al., 2007).

2. The no-wait flowshop scheduling problem

The no-wait flowshop scheduling problem is formally defined in the following. n jobs $\{J_1, J_2, \ldots, J_n\}$ are to be processed on a series of m machines $\{M_1, M_2, \ldots, M_m\}$ sequentially. The processing time of job J_i on machine M_j is given as T_{ij}. At any time, each job can be processed on at most one machine and each machine can process at most one job. Also, once a job is processed on a machine; it cannot be terminated before completion. The sequence in which the jobs are to be processed is the same for each machine. The no-wait constraint requires that the starting time of job J_i on machine M_j be equal to the completion time of job J_i on machine M_{j-1} for each i and each j. And the objective is to find a permutation of jobs such that the makespan is minimized. Now let $\pi=\{\pi_1, \pi_2, \ldots, \pi_n\}$ be a permutation of jobs and let $C(\pi_i, j)$ be defined as follows:

$$C(\pi_1,1) = T_{\pi_1,1}$$
$$C(\pi_1,j) = C(\pi_1,j-1) + T_{\pi_1,j}, \quad j = 2,3,\ldots,m$$
$$C(\pi_i,j) = C(\pi_{i-1},1) + P(\pi_i) + \sum_{l=1}^{j} T_{\pi_i,l}, \quad i = 2,3,\ldots,n; \quad j = 2,3,\ldots,m$$

where

$$P(\pi_i) = \max\left\{0, \max_{2 \le j \le m}\left(C(\pi_{i-1},j) - \left(C(\pi_{i-1},1) + \sum_{l=1}^{j-1} T_{\pi_i,l}\right)\right)\right\}, \quad i = 2,3,\ldots,n$$

The makespan of the scheduling corresponding to π is defined as

$$C_{max}(\pi) = C(\pi_n, m)$$

And the objective of the no-wait flowshop scheduling problem is to find a permutation π^* in the set of all permutations Π such that

$$C_{max}(\pi^*) = \min_{\pi \in \prod} C_{max}(\pi)$$

is satisfied.

3. The proposed hybrid genetic algorithm

In this section, we described the proposed hybrid genetic algorithm for the no-wait flowshop scheduling problem. Our algorithm hybridized the genetic algorithm and two local search methods. The genetic algorithm acts as a global search method in our algorithm because it is good at searching the whole solution space globally. The hybridization of the genetic algorithm and the local search methods makes the search more effective and more efficient as shown by experimental results. Moreover, an orthogonal-array-based crossover operator (OA-crossover) was utilized in our algorithm to improve the performance. In the following the proposed hybrid genetic algorithm is described. The details of the OA-crossover and two local search methods will be described in the subsections thereinafter.

/*initialization*/
Step 1: Set the values of the population size (P_s), the crossover rate (P_c), the mutation rate (P_m) and the termination condition (*Max_Stuck*). Set $S_l = 0$.
Step 2: Produce the initial population that consists of P_s randomly generated chromosomes.
Step 3: Evaluate the makespan of each chromosome in the population. Deposit the chromosome with the best makespan in *BEST* and its makespan in C^*.
/*crossover, local search and selection*/

Step 4: Repeat Step 5 to Step 6 $P_s \times P_c$ times.
Step 5: Randomly choose two chromosomes P_1 and P_2. Apply OA-crossover to the parent chromosomes P_1 and P_2 to produce the child chromosome *Child*.
Step 6: Apply Insertion Search to *Child*. P_1 and P_2 are replaced by the best two of P_1, P_2 and *Child*.
Step 7: Find the chromosome π_b with the best makespan C_b in the population. If $C_b < C^*$ then $BEST \leftarrow \pi_b$, $C^* \leftarrow C_b$, and set $S_l \leftarrow 0$, and apply Insertion Search with Cut-and-Repair to *BEST*. Otherwise, $S_l \leftarrow S_l + 1$.
/*mutation*/
Step 8: Randomly choose $P_s \times P_m$ chromosomes from the population and apply the mutation operator to these chromosomes.
Step 9: If $S_l > Max\_Stuck$ then stop. Otherwise, go to Step 4.

The loop (Steps 5–6) performs the OA-crossover and the IS local search $P_s \times P_c$ times. After that, the ISCR local search will be applied if the current best solution had been improved (Step 7). And then $P_s \times P_m$ mutations were performed (Step 8). Step 4 to Step 8 can be viewed as a generation. In Step 6, the local search method named Insertion_Search (IS) is applied to *Child*. The Insertion Search is used as the main local search operator. Every time the current best solution is improved, the other local search method named Insertion Search with Cut-and-Repair (ISCR) is applied in order to search a larger neighborhood of the current best solution. In Step 6, the selection is an eugenic one. The *Child* will replace the parent only if it is better than the worse parent. Finally, if the number of stuck generations is more than *Max_Stuck*, the algorithm terminates.

3.1. Representation of chromosome and definition of fitness function

In genetic algorithms, a chromosome represents a solution in the solution space. For the permutation flowshop scheduling problem, we use a permutation π of jobs as a chromosome. For example, suppose there are six jobs and four machines in a flowshop scheduling problem. A permutation $\pi = [2, 3, 1, 6, 5, 4]$ is a permutation of six jobs and this chromosome represent a scheduling in which the sequence of jobs on each machine is $J_2, J_3, J_1, J_6, J_5, J_4$.
The definition of fitness function is just the reciprocal of the objective function value, that is, the reciprocal of the makespan of the scheduling represented by the chromosome.

3.2. OA-crossover

The crossover operator is used in genetic algorithms to find better solutions by recombining good genes from different parent chromosomes. One cut point or two cut points were usually used in the crossover operator. As a generalization to this, multiple cut points are used in the proposed crossover. Father chromosome and mother chromosome are randomly divided into multiple subsequences. Several recombination of subsequences based on the orthogonal array are sampled and the Taguchi method Tsai et al. (2004) is used to select better subsequences for recombination. For details of the orthogonal array please refer to Appendix. The OA-crossover had been used in Tsai et al. (2004) to solve the global numerical optimization problem and used in Ho and Chen (2000) to solve the traveling salesman problem. In this study, we use three cut points for Carlier's benchmark (Carlier, 1978) and six cut points for both Reeves' benchmark problems (Reeves, 1995) and Taillard's benchmark problems (Taillard, 1990). According to our experience, it is a good choice to increase the number of cut points in the OA-crossover as the size of the instance increases. The OA-crossover is described in the following:

Step 1: Let N be the number of pieces into which the user wants to cut parent chromosomes P_1 and P_2 for recombination. Generate the orthogonal array $L_{N+1}(2^N)$.
Step 2: Randomly choose parent chromosomes P_1 and P_2. Randomly choose $N-1$ cut points to cut P_1 and P_2 into N subsequences.
Step 3: Consult the ith row of the OA $L_{N+1}(2^N)$ and generate a sampled child C_i, for $i = 1, 2, \ldots, N+1$. The jth subsequence of C_i is taken from the jth subsequence of P_1 if the level of the jth factor in row i of the OA is 0. Otherwise, the jth subsequence of C_i is taken from the jth subsequence of P_2. Repair C_i whenever it is necessary. (Repair scheme will be described later.)
Step 4: Calculate the evaluation value E_i of each sampled child C_i, for $i = 1, 2, \ldots, N+1$. The evaluation value E_i is the fitness value (i.e. the reciprocal of the makespan) of the chromosome C_i.
Step 5: Calculate the main effect F_{jk} of factor j with level k, for $j = 1, 2, \ldots, N$ and $k = 0, 1$.
Step 6: Find the best level for each factor. The best level of factor j is k if $F_{jk} = \max\{F_{j0}, F_{j1}\}$. Use the best levels of all factors to generate another child C_{N+2} (Taguchi method). Repair C_{N+2} whenever it is necessary. Calculate the evaluation value E_{N+2}.
Step 7: From $C_1, C_2, \ldots, C_{N+2}$, choose the one with the best evaluation value to be the child chromosome.

The sampled child C_i generated in Step 3 and the child C_{N+2} generated by Taguchi method may need to be repaired. The following example illustrates the repair scheme. Fig. 1 shows two parent chromosomes Parent 1 and Parent 2 together with an OA $L_4(2^3)$. When generating the sampled child C_2, the second row of the OA $L_4(2^3)$ is consulted. Since the content of this row is 011, the first subsequence of C_2 is taken to be the first subsequence of Parent 1, which is 3, 5. The second subsequence of C_2 is taken to be the second subsequence of Parent 2, which is 7, 1. Up to now, it is all right and nothing needed to be repaired. The third subsequence of C_2 is taken to be the third subsequence of Parent 2, which is 4, 0, 5, 3. Now it needs to be repaired because 5, 3 already appeared in C_2. So the last two places of C_2 are cleared and the jobs not yet appear in C_2 are taken from Parent 1 to put into these two places in the order they appear in Parent 1.

3.3. Two local search methods

The purpose of the local search is to find a better solution from the neighborhood of a solution. Let π be a solution in the solution space. The neighborhood of π, $N(\pi)$ is defined as the set of all solutions that can be reached by applying some operator to solution π. Two operators are often used in algorithms designed for solving the flowshop scheduling problem. They are the insertion operator $Ins(x, y)$ and the exchange operator $Exch(x, y)$. Both operators were used by Taillard (1990). The insertion

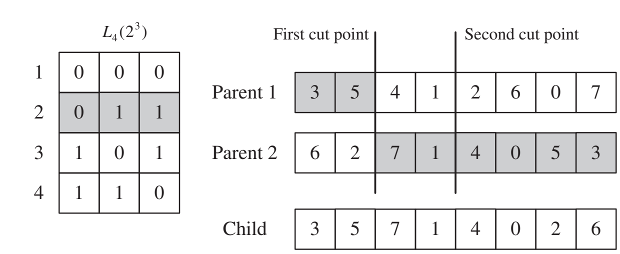

Fig. 1. An example that illustrates the repair scheme.

operator was also used by Grabowski and Pempera (2005). *Ins*(x, y) picks out the job at position x and inserts it into position y. *Exch*(x, y) exchanges the positions of the job at position x and the job at position y. After doing some experiments, we observed that the insertion operator is superior to the exchange operator in local search performance. Therefore, we use the insertion operator in the local search. As for the exchange operator, we use it in mutation. For a chromosome consists of n jobs, as many as $n \times (n\text{-}1)$ insertion operators can be applied to this chromosome. But applying all $n \times (n\text{-}1)$ insertion operators is not efficient, so a parameter α is used to control the range of positions into which a pick-out job can be inserted. The first local search method (Insertion Search) utilized in our algorithm is described in the following. In this procedure, a permutation π represents a chromosome (a solution) and $\pi(i)$ represents the job at the ith position of π. C represents the makespan of π.

Procedure Insertion-Search (π, c, α)

Step 1: Set the search List $SL \leftarrow \{1, 2, \ldots, n\}$.
Step 2: If SL is not empty, randomly choose p from SL and remove it from SL, then go to Step 3. Otherwise, stop and return π and C.
Step 3: Execute insertion operators *Ins*(p, k) for $k=p-1$, $p-2$, …, max(p-α, 1) and $p+1$, $p+2$,…, min($p+\alpha$, n). Calculate the makespan after executing each insertion operator and choose the best one. If the makespan of the best solution is better than C, let π be the best solution and C be the makespan of the best solution and go to Step 1. Otherwise, go to Step 2.

It is easily seen from the above procedure that Insertion Search searches only a small neighborhood. Hence, Insertion Search may sometimes be trapped in a local optimum. So we proposed another local search method called Insertion Search with Cut-and-Repair (ISCR) that has larger diversification ability and helps the search jump out of the local optima. ISCR uses a procedure named Cut-and-Repair. So we introduce procedure Cut-and-Repair first. Procedure Cut-and-Repair randomly chooses two pairs of adjacent jobs in π as cut points. Then it cuts π between each of these two pairs of adjacent jobs and picks another job to insert into the cutting position. This procedure is described in the following:

Procedure Cut-and-Repair (π, c)
Step 1: Set Cutting List $CL \leftarrow \phi$ and Moving List $ML \leftarrow \phi$.
Step 2: Randomly choose two pairs of adjacent jobs as cut points. Put the two cut points in CL.
Step 3: If CL is not empty, randomly choose one cut point (cp, $cp+1$) from CL and go to Step 4. Otherwise, stop and return π and C.
Step 4: For $t \in \{1, 2, \ldots, cp-1\}$, execute *Ins*($t$, cp). For $t \in \{cp+2, cp+3, \ldots, n\}$, execute *Ins*($t$, $cp+1$). From above insertion operators, find the eight operators that result in the smallest amount of makespan. Put these eight operators in ML.
Step 5: Generate a random number θ in [0, 1). If θ is greater than 0.5, choose the best operator in ML; otherwise, randomly choose an operator from ML. Apply this operator to π and update C accordingly. Set $ML \leftarrow \phi$ and go to Step 3.

A simple example is given in Fig. 2 that illustrates the Cut-and-Repair operation. Let the solution π be {3, 5, 2, 1, 4, 0}. Suppose Cut Point (3, 4) is chosen from the Cutting List CL (Step 3), every job in other position will be taken and inserted into this cut point (as illustrated by Fig. 2), and the best eight operations will be put in the Moving List (Step 4). Finally, one of the eight operations will be applied to π.

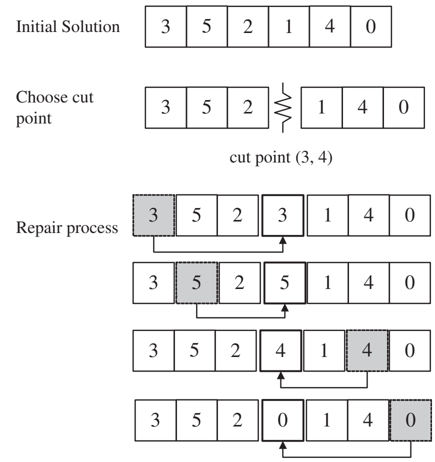

Fig. 2. An example that illustrates the Cut-and-Repair procedure.

Now we describe the second local search method, which searches a larger neighborhood, in the following:

Procedure Insertion-Search-with-Cut-and-Repair (π, c, α, *Max_Loop*)
Step 1: $\pi^* \leftarrow \pi$, $c^* \leftarrow c$ and iter $\leftarrow 0$.
Step 2: Execute Insertion-Search (π, c, α) with return values in π and c. If $c < c^*$ then $\pi^* \leftarrow \pi$ and $c^* \leftarrow c$.
Step 3: Execute Cut-and-Repair (π, c) with return values in π and c.
Step 4: $iter \leftarrow iter+1$. If $iter \geq Max\_Loop$ then stop and return π^* and c^*. Otherwise, go to Step 2.

3.4. Mutation

The exchange operator is used as the mutation operator. $P_s \times P_m$ chromosomes are randomly chosen from the population. For each chosen chromosome, *Exch*(x, y) is executed t times, where x, y are randomly chosen positions and t is an integer randomly drawn from 1 to T with T a predefined integer parameter.

4. Experimental results

The proposed hybrid genetic algorithm was implemented using C++ language on a personal computer of which the CPU is Intel Pentium III 1266 MHz and the memory size is 1 GB and the operating system is Windows XP. In order to compare the performance of our method with those of other methods (Grabowski and Pempera, 2005; Liu et al., 2007; Schuster and Framinan, 2003), we conducted experiments on 29 benchmark problems from OR-Library, which consists of eight problems (car1, car2, …, car8) provided by Carlier (Carlier, 1978) and 21 problems (rec01, rec03, …, rec41) provided by Reeves (1995).

We first did an experiment to compare the performance of two local search methods. Then we devised an experiment to compare the performance of the primitive GA, the GA with Insertion Search, the GA with Insertion Search with Cut-and-Repair, and the proposed hybrid GA. Finally, we compared the performance of the proposed hybrid genetic algorithm with those of other algorithms reported in the literature by running the proposed algorithm on the 29 benchmark problems.

4.1. Comparison of two local search methods

In this comparison test, seven problems were used. These seven problems consisted of the first problem taken from each of the sets 20×5, 20×10, 20×15, 30×10, 30×15, 50×10 and 75×20 of the benchmark provided by Reeves (1995). Because the performances of local search methods were sensitive to initial solutions, we randomly generated 10 solutions for each problem and these solutions were used as the initial solutions for the two local search methods. The comparison of the performances of two local search methods is shown in Table 1. AvgTime denotes the average CPU time (in seconds) of 10 runs. AvgPRD denotes the average percentage relative difference, which is defined as $(C_{avg} - C^*)/C^* \times 100\%$ with C^* being the makespan of the best solution found by RAJ heuristic. With the first glance at Table 1, one gets the impression that Insertion Search spends less computation time but obtains inferior quality solutions and on the contrary, Insertion Search with Cut-and-Repair obtains better quality solutions by using more computation time. This is true because the latter uses the former as a subroutine. Next, let us examine the value of α. The quality of solution is improved significantly when the value of α is raised from $n/5$ to $n/2$. On the other hand, the quality of solution is improved less significantly when the value of α is raised from $n/2$ to n. For Insertion Search with Cut-and-Repair, the quality of solution is improved as the value of *Max_Loop* increases.

4.2. Comparison of primitive GA with hybrid GA

In this experiment, the same seven problems used in Section 4.1 were used. Four algorithms, namely the primitive GA, the GA with Insertion Search, the GA with Insertion Search with Cut-and-Repair, and the proposed hybrid GA were run on these seven problems 10 times. The average CPU time (AvgTime) and the average percentage relative difference (AvgPRD) were reported in Table 2. For all four algorithms, the crossover probability P_c is set to 50%, the orthogonal array used for crossover is $L_8(2^7)$, the mutation probability is set to 5%, and the parameter T used in mutation is set to 5. Other parameter settings are shown in Table 2. P_s represents the population size, *Max_Stuck* represents the termination condition, α_1 represents the search range for Insertion Search, α_2 and *Max_Loop* represent the search range and the termination condition for Insertion Search with Cut-and-Repair. AvgTime and AvgPRD are the same as defined in Section 4.1.

Table 1
Comparison of two local search methods.

Methods	Parameters		Result	
	Max_Loop	α	AvgPRD	AvgTime
Insertion Search		$n/5$	4.980456	0.000671
		$n/2$	-0.708563	0.001571
		n	-1.997173	0.002231
	2	$n/5$	3.761958	0.000893
	5	$n/5$	1.772379	0.001800
	10	$n/5$	-0.354806	0.003348
Insertion Search	20	$n/5$	-1.698037	0.006696
With Cut-and-Repair	2	$n/2$	-1.183419	0.002009
	5	$n/2$	-2.604320	0.004018
	10	$n/2$	-3.447245	0.007143
	20	$n/2$	-4.158473	0.012500
	2	n	-2.686445	0.003348
	5	n	-3.519977	0.005134
	10	n	-4.492143	0.009821
	20	n	-5.028473	0.018973

Table 2
Comparison of primitive GA and hybrid GAs.

Methods	Parameters					Result	
	P_s	*Max_Stuck*	*Max_Loop*	α_1	α_2	AvgPRD	AvgTime
	$n/2$	10				10.7819	0.0065
	$n/2$	20				10.2037	0.0096
	$n/2$	30				9.9156	0.0154
	n	10				4.8778	0.0188
	n	20				4.5064	0.0257
Primitive GA	n	30				4.2379	0.0348
	$2n$	10				1.7330	0.0433
	$2n$	20				0.8915	0.0618
	$2n$	30				1.1057	0.0830
	$3n$	10				0.1348	0.0772
	$3n$	20				-0.0098	0.1016
	$3n$	30				-0.5770	0.1212
	$10n$	30				-3.2127	0.4205
GA+IS	$n/2$	10		$n/2$		-5.5974	0.2949
	$n/2$	10		n		-5.9784	0.4172
GA+ISCR	$n/2$	10	2		$n/2$	-6.1497	0.6234
	$n/2$	10	2		n	-6.3012	0.8980
	$n/2$	10	2	$n/5$	n	-4.3684	0.0833
	$n/2$	10	2	$n/2$	n	-5.5774	0.2511
	$n/2$	10	2	n	n	-5.9840	0.4188
	$n/2$	10	5	$n/5$	n	-4.7756	0.0810
Hybrid GA	$n/2$	10	5	$n/2$	n	-5.7078	0.3049
	$n/2$	10	5	n	n	-6.0426	0.3797
	$n/2$	10	10	$n/5$	n	-5.5953	0.1089
	$n/2$	10	10	$n/2$	n	-6.0082	0.2551
	$n/2$	10	10	n	n	-6.1909	0.3627
	n	20	10	$n/2$	n	-6.3365	0.8652
	$2n$	30	10	n	n	-6.5802	3.8460

From Table 2, it is observed that with comparable computation time, the hybrid GA obtains best quality solutions, then come the GA with Insertion Search with Cut-and-Repair and the GA with Insertion Search, and finally comes the primitive GA. The primitive GA does not perform well even when the population size is enlarged to $10n$ because the GA is suitable for global search but may not be appropriate for local search. With the help of local search methods, the quality of solutions improved. Containing both IS and ISCR, the proposed hybrid GA performs best as might be expected.

4.3. Comparison with other methods

The performance of the proposed hybrid GA was compared with the performances of those metaheruistic methods proposed by Schuster and Framinan (2003), Grabowski and Pempera (2005) and Liu et al. (2007). The comparison is shown in Table 3. “Instance” denotes the problem name, “n” denotes the number of jobs, “m” denotes the number of machines, “Opt” gives the makespans of the optimal solution of Carlier's benchmark problems, “RAJ” represents the results by RAJ heuristic (Rajendran, 1994), “VNS” and “GASA” represents the results of Schuster and Framinan (2003), “Tabu Search” represents the results of Grabowski and Pempera (2005), “HPSO” represents the results of Liu et al. (2007), and “HGA” represents the proposed hybrid GA. Three methods: TS, TS+M and TS+MP were proposed in Tabu Search (Grabowski and Pempera, 2005), so the column “Method” denotes which method that obtains the best solution. The parameter settings for the proposed hybrid GA are listed in Table 4.

Table 3
Performance comparison with five other methods.

Instance	$n\times m$	Opt	RAJ	VNS			GASA			Tabu Search				HPSO		HGA		
				Min	PRD	Time	Min	PRD	Time	Method	Min	PRD	Time	PRD	Time	Min	PRD	Time
car1	11×5	8142	8288	8201	0.70	0	**8142**	**0.00**	1	All	**8142**	**0.00**	0.1	**0.00**	0.4	**8142**	**0.00**	0.002
car2	13×4	8242	8610	8256	0.20	0	**8242**	0.00	1	All	**8242**	**0.00**	0.1	**0.00**	0.7	**8242**	**0.00**	0.002
car3	12×5	8866	9226	**8866**	**0.00**	0	**8866**	**0.00**	1	All	**8866**	**0.00**	0.1	**0.00**	0.9	**8866**	**0.00**	0.002
car4	14×4	9195	10,119	9348	1.60	0	**9195**	**0.00**	2	All	**9195**	**0.00**	0.1	**0.00**	1.4	**9195**	**0.00**	0.000
car5	10×6	9159	10,039	9496	3.50	0	**9159**	**0.00**	1	All	**9159**	**0.00**	0.1	**0.00**	0.6	**9159**	**0.00**	0.002
car6	8×9	9690	10,161	**9690**	**0.00**	0	**9690**	**0.00**	1	All	**9690**	**0.00**	0.1	**0.00**	0.3	**9690**	**0.00**	0.000
car7	7×7	7705	7903	**7705**	**0.00**	0	**7705**	**0.00**	0	All	**7705**	**0.00**	0.1	**0.00**	0.2	**7705**	**0.00**	0.000
car8	8×8	9372	9515	**9372**	**0.00**	0	**9372**	**0.00**	1	All	**9372**	**0.00**	0.1	**0.00**	0.3	**9372**	**0.00**	0.000
rec01	20×5		1590	1546	−2.77	0	1527	−3.96	6	TS	**1526**	**−4.03**	0.2	−3.77	3.9	**1526**	**−4.03**	0.009
rec03	20×5		1457	1394	−4.32	0	1392	−4.46	6	All	**1361**	**−6.59**	0.2	**−6.59**	4.8	**1361**	**−6.59**	0.006
rec05	20×5		1637	1522	−7.03	0	1524	−6.90	7	TS+MP	**1511**	**−7.70**	0.2	−7.39	4.1	**1511**	**−7.70**	0.008
rec07	20×10		2119	2070	−2.31	0	2046	−3.45	12	All	**2042**	**−3.63**	0.2	**−3.63**	6.6	**2042**	**−3.63**	0.008
rec09	20×10		2141	2090	−2.38	0	2045	−4.48	11	TS	**2042**	**−4.62**	0.3	−4.58	6.7	**2042**	**−4.62**	0.008
rec11	20×10		1946	1916	−1.54	0	**1881**	**−3.34**	10	All	**1881**	**−3.34**	0.2	**−3.34**	4.7	**1881**	**−3.34**	0.008
rec13	20×15		2709	2553	−5.76	0	2556	−5.65	17	All	**2545**	**−6.05**	0.3	**−6.05**	11.0	**2545**	**−6.05**	0.009
rec15	20×15		2691	2532	−5.91	0	**2529**	**−6.02**	17	TS+M	**2529**	**−6.02**	0.3	**−6.02**	8.6	**2529**	**−6.02**	0.008
rec17	20×15		2740	2599	−5.15	0	2590	−5.47	16	All	**2587**	**−5.58**	0.3	**−5.58**	8.6	**2587**	**−5.58**	0.008
rec19	30×10		3157	2918	−7.57	1	2895	−8.30	34	TS	**2850**	**−9.72**	0.4	−9.15	23.0	**2850**	**−9.72**	0.034
rec21	30×10		3015	2888	−4.21	1	2948	−2.22	35	TS	**2823**	**−6.37**	0.4	−5.70	24.0	2829	−6.17	0.030
rec23	30×10		3030	2704	−10.76	0	2827	−6.70	35	TS+MP	**2700**	**−10.89**	0.4	−10.80	24.0	**2700**	**−10.89**	0.025
rec25	30×15		3835	3626	−5.45	1	3732	−2.69	55	TS+M	**3593**	**−6.31**	0.5	−5.71	32.0	**3593**	**−6.31**	0.031
rec27	30×15		3655	3442	−5.83	1	3560	−2.60	51	TS+M	3432	−6.10	0.5	**−6.13**	39.0	**3431**	**−6.13**	0.031
rec29	30×15		3583	3324	−7.23	1	3440	−3.99	54	TS+M	**3291**	**−8.15**	0.5	−7.81	31.0	**3291**	**−8.15**	0.031
rec31	50×10		4631	4413	−4.71	5	4757	2.72	147	TS+MP	4343	−6.22	1.1	−5.92	122.0	**4334**	**−6.41**	0.267
rec33	50×10		4770	4515	−5.35	7	4998	4.78	145	TS+MP	4466	−6.37	1.1	−5.51	116.0	**4458**	**−6.54**	0.252
rec35	50×10		4718	4458	−5.51	7	4891	3.67	146	TS+M	4427	−6.17	1.1	−6.02	105.0	**4424**	**−6.23**	0.225
rec37	75×20		8979	**8081**	**−10.00**	122	9508	5.89	907	TS+M	8127	−9.49	2.6	−8.89	635.0	8121	−9.56	1.447
rec39	75×20		9158	8671	−5.32	106	9964	8.80	890	TS	8517	−7.00	2.5	−6.79	897.0	**8505**	**−7.13**	1.283
rec41	75×20		9344	8652	−7.41	110	9978	6.79	940	TS+MP	8520	−8.82	2.6	−7.94	883.0	**8505**	**−8.98**	1.073

Table 4
Parameter settings for the hybrid GA.

Benchmark	P_s	P_c	P_m	*Max_Stuck*	OA	α_1	α_2	*Max_Loop*
Carlier	5	50%	5%	10	$L_4(2^3)$	$n/2$	n	5
Reever	$n/2$	50%	5%	10	$L_8(2^7)$	$n/2$	n	10

We had run the HGA on each problem 10 times and the column "Time" denotes the average CPU time. "Min" represents the makespan of the best solution found. "PRD" represents the percentage relative difference $(C_{min}-C^*)/C^*\times 100\%$, where C^* is the makespan of the known optimal solution for Carlier's benchmark problems but C^* is the makespan of the best solution found by RAJ heuristic for Reeves' benchmark problems. VNS and GASA (Schuster and Framinan, 2003) were run on a personal computer with Athlon 1.4 GHz CPU and they were run 30 times on each problem. Tabu Search methods (Grabowski and Pempera, 2005) were run on a personal computer with Pentium 1.0 GHz CPU and how many times they were run on each problem is unknown. HPSO (Liu et al., 2007) was run on a personal computer with Mobile Pentium IV 2.2 GHz CPU and it was run 20 times on each problem.

It is observed from Table 3 that the HGA found the optimal solutions for all eight Carlier's benchmark problems with very little computation time. For the 21 problem provided by Reeves, the HGA improved five out of the 21 current best solutions reported in the literature and achieved the current best solutions for 14 of the remaining 16 problems with less computation time than other methods. The proposed HGA efficiently finds good quality solutions.

Table 3 shows only the short-term search capability of the HGA. In order to find out whether the HGA will be trapped in a local optimum and makes no progress even if more computation time is allowed, another experiment was conducted. We ran the HGA for the mid-term search and the long-term search. For the mid-term search, the population size is increased to n and *Max_Stuck* is increased to 20. For the long-term search, the population size and *Max_Stuck* are increased to $2n$ and 30, respectively, and the range for Insertion Search is increased to n. We ran the HGA 10 times on each of Reeve's problems. Results are shown in Table 5. "Avg" denotes the average of the makespan of each run's best solution. From Table 5, it is noted that quality of solution improves as the search time increased for most problems. In the long-term search, the HGA further improved seven out of the 21 best solutions.

The largest number of jobs in Reeves' benchmark problems is 75 while Taillard (1990) provided benchmark problems whose largest number of jobs is 500. So we conducted still another experiment to test the performance of the HGA on the 120 problems presented by Taillard. The HGA was run 10 times on each of these 120 problems and the results are given in Table 6. The experimental results reveal that the HGA has the capability to solve large scale problems.

5. Conclusions and future works

A hybrid genetic algorithm for the no-wait flowshop scheduling problem was proposed in this paper. In this algorithm, the GA is used as the global search scheme and is hybridized with a novel local search scheme. An OA-based crossover operator is designed to enhance the intensification ability of the GA. The proposed local search scheme combines two local search methods: IS and ISCR. IS (Insertion Search) is the local search method that searches a small neighborhood. ISCR (Insert Search with Cut-and-Repair), with IS as its subroutine, searches a large neighborhood. The hybridization of the GA with these two local search methods that have

Table 5
The mid-term and long-term search capability of the hybrid GA.

Instance	$n \times m$	Short-term test			Mid-term test			Long-term test		
		Min	Avg	Time	Min	Avg	Time	Min	Avg	Time
rec01	20 × 5	1526	1530.2	0.009	1526	1527.4	0.019	1526	1526.0	0.047
rec03	20 × 5	1361	1371.2	0.006	1361	1370.2	0.016	1361	1362.7	0.038
rec05	20 × 5	1511	1517.1	0.008	1511	1512.7	0.016	1511	1512.9	0.053
rec07	20 × 10	2042	2050.0	0.008	2042	2043.5	0.017	2042	2042.5	0.048
rec09	20 × 10	2042	2049.9	0.008	2042	2045.3	0.016	2042	2042.3	0.053
rec11	20 × 10	1881	1891.7	0.008	1881	1889.9	0.017	1881	1884.9	0.044
rec13	20 × 15	2545	2555.6	0.009	2545	2550.5	0.020	2545	2545.7	0.039
rec15	20 × 15	2529	2539.0	0.008	2529	2531.2	0.017	2529	2529.6	0.045
rec17	20 × 15	2587	2594.4	0.008	2587	2591.0	0.017	2587	2587.0	0.045
rec19	30 × 10	2850	2876.0	0.034	2850	2861.4	0.086	2850	2860.3	0.234
rec21	30 × 10	2829	2841.3	0.030	**2821**	2829.4	0.084	**2821**	2823.7	0.245
rec23	30 × 10	2700	2730.3	0.025	2700	2719.5	0.095	2700	2705.7	0.278
rec25	30 × 15	3593	3616.4	0.031	3597	3607.1	0.083	3593	3600.8	0.219
rec27	30 × 15	3431	3469.1	0.031	3431	3442.6	0.097	3431	3435.1	0.263
rec29	30 × 15	3291	3309.9	0.031	3291	3308.7	0.073	3291	3298.6	0.220
rec31	50 × 10	4334	4378.8	0.267	**4318**	4340.9	0.997	**4313**	4328.4	2.833
rec33	50 × 10	4458	4510.1	0.252	**4452**	4472.5	0.809	**4431**	4455.6	3.469
rec35	50 × 10	4424	4482.8	0.225	**4411**	4442.1	0.988	**4397**	4417.9	3.052
rec37	75 × 20	8121	8204.7	1.447	**8052**	8131.9	4.834	**8025**	8048.9	23.502
rec39	75 × 20	8505	8633.6	1.283	**8465**	8538.7	4.967	**8446**	8482.9	22.559
rec41	75 × 20	8505	8673.4	1.073	**8491**	8591.0	4.328	**8482**	8491.8	26.691

Table 6
The results of the HGA on Taillard's benchmark problems.

Instance	$\boldsymbol{n \times m}$	**Max**	**Min**	**Avg**	**AvgTime**
ta001	20 × 5	1485	1449	1472.7	0.041
ta002	20 × 5	1505	1460	1477.6	0.044
ta003	20 × 5	1431	1386	1406.8	0.045
ta004	20 × 5	1573	1521	1546.4	0.041
ta005	20 × 5	1445	1403	1426.6	0.047
ta006	20 × 5	1471	1430	1446.6	0.038
ta007	20 × 5	1496	1461	1479.4	0.050
ta008	20 × 5	1475	1433	1459.2	0.053
ta009	20 × 5	1429	1398	1409.2	0.038
ta010	20 × 5	1368	1324	1345.5	0.042
ta011	20 × 10	1998	1955	1972.6	0.044
ta012	20 × 10	2166	2123	2154.6	0.044
ta013	20 × 10	1942	1912	1931.1	0.045
ta014	20 × 10	1811	1782	1794.3	0.042
ta015	20 × 10	1947	1933	1934.4	0.045
ta016	20 × 10	1879	1827	1850.0	0.044
ta017	20 × 10	1971	1944	1951.5	0.059
ta018	20 × 10	2066	2006	2038.5	0.047
ta019	20 × 10	1973	1908	1953.9	0.039
ta020	20 × 10	2032	2001	2019.3	0.047
ta021	20 × 20	2972	2912	2938.5	0.044
ta022	20 × 20	2835	2780	2814.2	0.048
ta023	20 × 20	2984	2922	2962.4	0.042
ta024	20 × 20	2994	2967	2982.5	0.045
ta025	20 × 20	3017	2953	2995.1	0.045
ta026	20 × 20	2964	2908	2932.8	0.042
ta027	20 × 20	3028	2970	3004.8	0.039
ta028	20 × 20	2826	2763	2789.5	0.047
ta029	20 × 20	3009	2972	3005.3	0.039
ta030	20 × 20	2979	2919	2956.3	0.042
ta031	50 × 5	3229	3127	3198.3	2.406
ta032	50 × 5	3475	3438	3453.1	3.484
ta033	50 × 5	3277	3182	3242.5	2.669
ta034	50 × 5	3384	3289	3349.8	3.156
ta035	50 × 5	3404	3315	3369.1	3.075
ta036	50 × 5	3377	3324	3356.4	3.792
ta037	50 × 5	3280	3183	3246.4	2.653
ta038	50 × 5	3288	3243	3264.9	3.342
ta039	50 × 5	3121	3059	3088.2	3.588
ta040	50 × 5	3383	3301	3341.5	3.353
ta041	50 × 10	4306	4251	4289.1	3.173
ta042	50 × 10	4235	4139	4193.5	2.841
ta043	50 × 10	4124	4083	4108.6	2.791
ta044	50 × 10	4549	4480	4517.4	3.147
ta045	50 × 10	4367	4316	4333.2	3.727

Table 6 (*continued*)

Instance	$n \times m$	Max	Min	Avg	AvgTime
ta046	50 × 10	4332	4282	4301.6	2.930
ta047	50 × 10	4444	4376	4412.6	3.253
ta048	50 × 10	4347	4304	4331.9	3.842
ta049	50 × 10	4188	4162	4173.0	3.645
ta050	50 × 10	4304	4232	4279.0	2.878
ta051	50 × 20	6172	6138	6149.7	3.727
ta052	50 × 20	5790	5721	5751.5	2.536
ta053	50 × 20	5929	5847	5884.4	2.978
ta054	50 × 20	5827	5781	5804.7	3.683
ta055	50 × 20	5950	5891	5909.7	2.886
ta056	50 × 20	5911	5875	5890.6	3.708
ta057	50 × 20	6001	5937	5974.2	3.656
ta058	50 × 20	5971	5919	5951.0	3.166
ta059	50 × 20	5899	5839	5873.5	2.405
ta060	50 × 20	5979	5935	5963.5	3.175
ta061	100 × 5	6594	6492	6557.2	45.761
ta062	100 × 5	6469	6353	6409.4	56.869
ta063	100 × 5	6320	6148	6260.4	54.755
ta064	100 × 5	6198	6080	6159.0	45.809
ta065	100 × 5	6397	6254	6325.6	56.858
ta066	100 × 5	6296	6177	6225.7	49.164
ta067	100 × 5	6460	6257	6409.0	46.133
ta068	100 × 5	6359	6225	6308.6	61.564
ta069	100 × 5	6561	6443	6516.3	59.500
ta070	100 × 5	6592	6441	6542.5	45.503
ta071	100 × 10	8230	8115	8173.5	77.302
ta072	100 × 10	8101	7986	8048.1	56.067
ta073	100 × 10	8215	8057	8142.2	81.019
ta074	100 × 10	8505	8327	8437.5	70.181
ta075	100 × 10	8096	7991	8046.4	66.008
ta076	100 × 10	7947	7823	7883.7	85.295
ta077	100 × 10	8081	7915	8007.0	57.316
ta078	100 × 10	8105	7939	8049.2	66.153
ta079	100 × 10	8349	8226	8290.4	48.631
ta080	100 × 10	8340	8186	8255.3	61.822
ta081	100 × 20	10,932	10,745	10,826.6	91.084
ta082	100 × 20	10,847	10,655	10,762.5	62.809
ta083	100 × 20	10,821	10,672	10,740.4	78.780
ta084	100 × 20	10,797	10,630	10,679.7	102.419
ta085	100 × 20	10,777	10,548	10,658.2	81.913
ta086	100 × 20	10,853	10,700	10,753.0	80.611
ta087	100 × 20	11,031	10,827	10,913.6	86.192
ta088	100 × 20	10,992	10,863	10,905.2	84.967
ta089	100 × 20	10,963	10,751	10,859.0	78.608
ta090	100 × 20	11,067	10,794	10,928.8	86.792
ta091	200 × 10	15,916	15,739	15,843.5	669.517
ta092	200 × 10	15,764	15,534	15,645.3	651.884
ta093	200 × 10	16,026	15,755	15,882.4	665.973
ta094	200 × 10	16,111	15,842	15,927.0	544.425
ta095	200 × 10	15,829	15,692	15,763.8	600.841
ta096	200 × 10	15,731	15,622	15,669.9	493.281
ta097	200 × 10	16,029	15,877	15,962.3	609.767
ta098	200 × 10	15,933	15,733	15,833.2	546.283
ta099	200 × 10	15,759	15,573	15,626.2	692.870
ta100	200 × 10	15,934	15,803	15,869.1	669.817
ta101	200 × 20	20,458	20,148	20,331.3	821.692
ta102	200 × 20	20,889	20,539	20,763.5	845.433
ta103	200 × 20	20,636	20,511	20,583.8	676.202
ta104	200 × 20	20,753	20,461	20,594.6	757.134
ta105	200 × 20	20,601	20,339	20,544.4	718.309
ta106	200 × 20	20,780	20,501	20,661.9	756.392
ta107	200 × 20	20,915	20,680	20,804.3	519.278
ta108	200 × 20	20,814	20,614	20,665.7	758.638
ta109	200 × 20	20,757	20,300	20,574.9	1078.077
ta110	200 × 20	20,712	20,437	20,587.6	704.683
ta111	500 × 20	49,580	49,095	49,289.4	2115.883
ta112	500 × 20	50,354	49,461	49,948.2	1967.702
ta113	500 × 20	49,399	48,777	49,139.0	1944.677
ta114	500 × 20	50,004	49,283	49,657.8	2143.773
ta115	500 × 20	49,847	48,950	49,423.1	2204.473
ta116	500 × 20	50,046	49,533	49,848.4	1989.594
ta117	500 × 20	49,591	48,943	49,366.5	2110.389
ta118	500 × 20	49,942	49,277	49,675.8	1962.286
ta119	500 × 20	49,697	49,207	49,429.3	2547.245
ta120	500 × 20	50,002	49,092	49,545.6	2059.581

different characteristics and capabilities makes the hybrid algorithm an effective and efficient one. For short-term search, the proposed algorithm found the optimal solutions for all eight Carlier's benchmark problems. Also, for Reeves' benchmark problems, the proposed algorithm improved five out of the 21 current best solutions reported in the literature and achieved the current best solutions for 14 of the remaining 16 problems. Also, the proposed algorithm has the capability for long-term search and the capability to solve large scale problems. For long-term search, it further improved seven out of the 21 best solution. As future works, we plan to investigate the applicability of the proposed hybrid genetic algorithm to other scheduling problems.

Acknowledgements

The authors gratefully acknowledge the partly support of National Science Council of ROC under the contract NSC 96-2628-E-005-074-MY3 and the partly support of the Ministry of Education, Taiwan, ROC under the ATU plan.

Appendix. The orthogonal arrays

In this Appendix, we briefly introduce the concept of orthogonal arrays which are used in experimental design methods. For more details, the reader may refer to Montgomery (1991) (Liaw, 2000). Suppose in an experiment, there are k factors and each factor has q levels. In order to find the best setting of each factor's level, q^k experiments must be done. Very often, it is not possible or cost effective to test all q^k combinations. It is desirable to sample a small but representative sample of combinations for testing. The orthogonal arrays were developed for this purpose. In an experiment that has k factors and each factor has q levels, an orthogonal array $OA(n, k, q, t)$ is an array with n rows and k columns which is a representative sample of n testing experiments that satisfies the following three conditions. (1) For the factor in any column, every level occur the same number of times. (2) For the t factors in any t columns, every combination of q levels occur the same number of times. (3) The selected combinations are uniformly distributed over the whole space of all the possible combinations. In the notation $OA(n, k, q, t)$, n is the number of experiments, k is the number of factors, q is the number of levels of each factor and t is called the strength. Another often used notation for the orthogonal array is $L_n(q^k)$. In this notation t is omitted and is always set to 2. A $L_8(2^7)$ orthogonal array is shown in Table 7 as an illustrating example.

For an experiment, there are various orthogonal arrays available. After an orthogonal array is chosen, one may apply the following criterion to determine the best combinations of each factor's level in this experiment. Let E_i be the evaluation value of the ith experiment in the array. The main effect of factor j with level k, F_{jk} is defined as $F_{jk} = \Sigma_{i=1}^{n} E_i A_{ijk}$, where A_{ijk} is 1 if factor j's level is k in the ith experiment and A_{ijk} is 0 otherwise. After all F_{jk} had been computed, the level of factor j is chosen to be l if $F_{jl} = \max_{1 \le k \le q} F_{jk}$.

Table 7
$L_8(2^7)$ orthogonal array.

Test no.	Factors							Evaluation value (E_i)
	1	2	3	4	5	6	7	
1	0	0	0	0	0	0	0	E_1
2	0	0	0	1	1	1	1	E_2
3	0	1	1	0	0	1	1	E_3
4	0	1	1	1	1	0	0	E_4
5	1	0	1	0	1	0	1	E_5
6	1	0	1	1	0	1	0	E_6
7	1	1	0	0	1	1	0	E_7
8	1	1	0	1	0	0	1	E_8

References

Aldowaisan, T., Allahverdi, A., 2003. New heuristics for no-wait flowshops to minimize makespan. Computers and Operations Research 30, 1219–1231.
Bonney, M.C., Gundry, S.W., 1976. Solutions to the constrained flowshop sequencing problem. Operational Research Quarterly 24, 869–883.
Carlier, J., 1978. Ordonnancements a contraintes disjonctives. RAIRO Recherche Operationnelle 12, 333–351.
Gonazelez, B., Torres, M., Moreno, J.A., 1995. A hybrid genetic algorithm approach for the "no-wait" flowshop scheduling problem. In: Proceedings of the First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, London, UK, pp. 59–64.
Gupta, J., 1971. A functional heuristic algorithm for the flowshop scheduling problem. Operational Research Quarterly 22, 39–47.
Grabowski, J., Pempera, J., 2005. Some local search algorithms for no-wait flow-shop problem with makespan criterion. Computers and Operations Research 32, 2197–2212.
Gonçalves, J.F., Mendes, J.J.D.M., Resende, M.G.C., 2005. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167 (1), 77–95.
Gangadharan, R., Rajendran, C., 1993. Heuristic algorithms for scheduling in the no-wait flowshop. International Journal of Production Economics 32, 285–290.
Haouari, M., Hidri, L., 2008. On the hybrid flowshop scheduling problem. International Journal of Production Economics 113, 495–497.
Ho, S.Y., Chen, J.H., 2000. A GA-based systematic reasoning approach for solving traveling salesman problems using an orthogonal array crossover. In: Proceedings of the Fourth International IEEE Conference/Exhibition on High Performance Computing in Asia-Pacific Region, Beijing, China, pp. 659–663.
King, J.R., Spachis, A.S., 1980. Heuristics for flowshop scheduling. International Journal of Production Research 18, 343–357.
Lin, B.M.T., Lu, C.Y., Shyu, S.J., Tsai, C.Y., 2008. Development of new features of ant colony optimization for flowshop scheduling. International Journal of Production Economics 112, 742–755.
Liu, B., Wang, L., Jin, Y.-H., 2007. An effective hybrid particle swarm optimization for no-wait flow shop scheduling. International Journal of Advanced Manufacturing Technology 31, 1001–1011.
Liu, S.Q., Kozan, E., 2009. Scheduling a flow shop with combined buffer conditions. International Journal of Production Economics 117, 371–380.
Liaw, C.F., 2000. A hybrid genetic algorithm for the open shop scheduling problem. European Journal of Operational Research 124 (1), 28–42.
Montgomery, D.C., 1991. Design and Analysis of Experiments 3rd ed. Wiley, New York.
Palmer, D., 1965. Sequencing jobs through a multi-stage process in the minimum total time—a quick method of obtaining a near optimum. Operations Research Quarterly 16, 101–107.
Reeves, C.R., 1994. Genetic algorithms and neighborhood search, Evolutionary Computing. Springer, Berlin pp. 115–130.
Reeves, C.R., 1995. A genetic algorithm for flowshop sequencing. Computers and Operations Research 22 (1), 5–13.
Park, Y.B., 2001. A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines. International Journal of Production Economics 73 (2), 175–188.
Rock, H., 1984. The three-machine no-wait flowshop problem is NP-complete. Journal of the Association for Computing Machinery 31, 336–345.
Reddi, S.S., Ramamoorthy, C.V., 1972. On the flowshop sequencing problems with no wait intermediate queues. Operational Research Quarterly 23, 323–331.
Rajendran, C., 1994. A no-wait flowshop scheduling heuristic to minimize makespan. Journal of the Operational Research Society 45, 472–478.
Schuster, C.J., Framinan, J.M., 2003. Appreciative procedures for no-wait job shop scheduling. Operations Research Letters 31, 308–318.
Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operations Research 47, 65–74.
Tsai, J.T., Liu, T.K., Chou, J.H., 2004. Hybrid Taguchi-genetic algorithm for global numerical optimization. IEEE Transactions on Evolutionary Computation 8 (4), 365–377.
Valls, V., Ballestin, F., Quintanilla, S., 2008. A hybrid genetic algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 185, 495–508.
Wismer, D.A., 1972. Solution of the flowshop sequencing problem with no intermediate queues. Operations Research 20, 689–697.