A High Performing Memetic Algorithm for the Flowshop Scheduling Problem With Blocking

Quan-ke Pan, Ling Wang, Hong-yan Sang, Jun-qing Li, and Min Liu

***Abstract*—This paper considers minimizing makespan for a blocking flowshop scheduling problem, which has important application in a variety of modern industries. A constructive heuristic is first presented to generate a good initial solution by combining the existing profile fitting (PF) approach and Nawaz–Enscore–Ham (NEH) heuristic in an effective way. Then, a memetic algorithm (MA) is proposed including effective techniques like a heuristic-based initialization, a path-relinking-based crossover operator, a referenced local search, and a procedure to control the diversity of the population. Afterwards, the parameters and operators of the proposed MA are calibrated by means of a design of experiments approach. Finally, a comparative evaluation is carried out with the best performing algorithms presented for the blocking flowshop with makespan criterion, and with the adaptations of other state-of-the-art MAs originally designed for the regular flowshop problem. The results show that the proposed MA performs much better than the other algorithms. Ultimately, 75 out of 120 upper bounds provided by Ribas *et al.* ["An iterated greedy algorithm for the flowshop scheduling with blocking", *OMEGA*, vol. 39, pp. 293–301, 2011.] for Taillard flowshop benchmarks that are considered as blocking flowshop instances are further improved by the presented MA.**

***Note to Practitioners*—A blocking flowshop problem has important applications in plastic, steel, chemical, and many other industries. In the blocking flowshop, no intermediate buffer exists between machines due to technical requirements or the process characteristics, and the zero buffer setting is also different from no-wait setting for more than two machines. This paper aims to minimize makespan for the flowshop scheduling problem with blocking, which can lead to a high throughput, and it is very important in situations where a simultaneously received batch of jobs is required to be completed as soon as possible. We propose an effective memetic algorithm (MA) including advanced techniques like a heuristic-based initialization, a path-relinking-based crossover operator, a referenced local search, and a procedure to control the diversity of the population. The effectiveness of the proposed MA is demonstrated by extensive comparisons against the best existing methods for the considered problem, as well as with several adapted hybrid genetic algorithms originally designed for the regular flowshop problems. However, in some situations, setup times of machines and transporting times of jobs cannot be negligible. This work can be extended to these practical problems by considering the constraints in the objective function. In addition, the application of the proposed MA can also be generalized for other combinatorial optimization problems including no-wait flowshop problems, hybrid flowshop problems, flexible job-shop problems, and many others.**

***Index Terms*—Blocking, flowshop scheduling, makespan, memetic algorithm.**

NOTATIONS LIST

$J = \{1, 2, \ldots, n\}$	Set of all the jobs, and n is the total number of jobs.
$M = \{1, 2, \cdots, m\}$	Set of all the machines, and m is the total number of machines.
$o_{j,k}$	Operation corresponding to the processing of job $j \in J$ on the machine $k \in M$.
$p_{j,k}$	Processing time of the operation $o_{j,k}$.
π, β	Job permutation containing all or a part of the n jobs from $J = \{1, 2, \ldots, n\}$.
Π	Set of all the job permutations which include n jobs.
$C_{\max}(\pi)$	Makespan of a job permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, where $\pi_i \in J$.
$d_{[i],k}$	Departure time of operation $o_{[i],k}$, where $[i]$ represents the ith job of permutation π, i.e., π_i.
U	Set of the unscheduled jobs.
Λ	Control parameter of the PF+NEH method, where $0 \leq \lambda \leq n$.

Manuscript received May 12, 2012; revised July 12, 2012; accepted September 12, 2012. Date of publication November 15, 2012; date of current version June 27, 2013. This paper was recommended for publication by Associate Editor C.-H. Chu and Editor M. C. Zhou upon evaluation of the reviewers' comments. This work was supported in part by the National Key Basic Research and Development Program of China (No. 2013CB329503, 2009CB320601 and 2009CB320602), the National Science Foundation of China (No. 61174187, 61174189, 61104179, and 60834004), the Foundation of Shandong Province in China (BS2010DX005), the Basic Scientific Research Foundation of Northeast University (No. N110208001), the starting Foundation of Northeast University (No. 29321006), the National Science and Technology Major Project of China (No.2011ZX02504-008), and the Doctoral Program Foundation of Institutions of Higher Education of China (20100002110014).

Q. Pan is with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang, 110819, China, and also with the School of Computer Science, Liaocheng University, Liaocheng 252059, China (e-mail: panquanke@gamil.com).

L. Wang and M. Liu are with the Tsinghua National Laboratory for Information Science and Technology (TNList), Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: wangling@mail.tsinghua.edu.cn; lium@mail.tsinghua.edu.cn).

H. Sang is with the School of Mathematics Science, Liaocheng University, Liaocheng 252059, China (e-mail: sanghongyan@gamil.com).

J. Li is with the School of Computer Science, Liaocheng University, Liaocheng 252059, China (e-mail: Lijunqing@lcu.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2012.2219860

PS	Population size.
P_c	Crossover rate.
P_m	Mutation rate.
γ	Threshold value. A restart mechanism is applied when the best solution found so far is not improved in γ generations.

I. Introduction

THE FLOWSHOP scheduling problem, which consists in determination of the best sequence for processing n jobs on m machines in the same order, has been a keen area of research work for many years [1]. Extensive research work has been conducted for the regular flowshop scheduling problem with the assumption that there exists an infinite buffer space between machines and a job can be stored for unlimited amount of time [2]. However, in practice, there are many production processes where no intermediate buffer exists due to the technical requirements or process characteristics. For example, in the steelmaking and continuous casting process in the iron and steel industry, a ladle furnace remains blocked by a charge until a caster is available for casting it. Another example [3] comes from the chemical industry where partially processed jobs (physical and chemical changes in the material) are sometimes kept in the machines because of the lack of intermediary storage. In such cases, intermediate queues of jobs waiting in the production system for their next operation are not allowed. A job having completed processing on a machine cannot leave this machine until its next machine is available for processing. That is, the job will block the current machine if its next machine is not available. The flowshop scheduling problem without intermediate buffers, or the blocking flowshop scheduling problem, can be found in a variety of industrial systems [3]–[7]. A comprehensive survey of the applications of the blocking flowshop problem in modern manufacturing systems can be found in [8].

In the literature, the most common criterion for the blocking flowshop problem is the minimization of the makespan or the maximum completion time (denoted as $C_{\max}$) of the schedule. This criterion can lead to a high throughput, and it is very important in the situations where a simultaneously received batch of jobs is required to be completed as soon as possible. We also consider the blocking flowshop scheduling problem with makespan criterion in this paper. The problem can be denoted as $Fm/block/C_{\max}$ using the well-known $\alpha/\beta/\gamma$ notation presented by Graham *et al.* [9]. It was shown by Reddi and Rananoorthy [10] that the $Fm/block/C_{\max}$ problem with $m = 2$ can be reduced to a special case of the traveling salesman problem and be solved in $O(n \log n)$ time using Gilmore *et al.* algorithm [11]. However, for $m \geq 3$, the problem is unfortunately NP-hard in the strong sense [7]. Therefore, it is more practical to develop heuristics and metaheuristics for the general blocking flowshop scheduling problem.

Heuristic methods are mainly proposed based on the specific characteristics of the problem. For example, McCormich *et al.* [12] developed a heuristic, known as Profile Fitting (PF), for solving sequencing problems in an assembly line with blocking to minimize cycle time, where the PF tried to sequence jobs leading to the minimization of the idle and blocking times on machines. Leisten [13] presented a more comprehensive approach for dealing with permutation and non-permutation flowshops with finite and unlimited buffers to maximize the use of the buffers and to minimize the machine blocking. However, after the extensive experiments, Leisten [13] concluded that the presented heuristic was not comparable with the Nawaz–Enscore–Ham (NEH) heuristic [14], which was initially proposed for the regular flowshop problem. Based on the makespan properties of the blocking flow shop problem proposed by Ronconi and Armentano [15], Ronconi [4] proposed three constructive heuristics. Among them, the two combination methods, namely MME and PFE provided better results than the NEH algorithm in problems with up to 500 jobs and 20 machines. According to the connection between no-wait flowshop and blocking flowshop, Abadi *et al.* [16] developed a heuristic to minimize cycle time in the blocking flowshop scheduling. Recently, Ronconi and Henriques [17] designed several constructive heuristics to minimize the total tardiness in a blocking flowshop and provided promising results. Pan and Wang [18] presented a total of eight constructive heuristics and composite heuristics for the objective of minimizing makespan. The authors' experiments showed that the presented constructive heuristics performed significantly better than the existing ones, and the proposed composite heuristics further improved the presented constructive heuristics by a considerable margin. In addition, 17 new best-known solutions for Taillard benchmarks with large scale were found by the presented heuristics.

With the development of the computer technology, metaheuristics presented for the blocking flowshop problems have grown quickly. For the makespan criterion, Caraffa *et al.* [19] developed a genetic algorithm (GA) for the large size restricted slowdown flowshop problem where the blocking flowshop was a special case. The author's computational experiments showed that the GA performed significantly better than the heuristic method developed by Abadi *et al.* [16]. Ronconi [20] proposed a heuristic based on a branch-and-bound method using the lower bounds that exploited the blocking nature. Grabowski and Pempera [6] designed a fast tabu search (TS) approach, where the concept of multimoves were presented and applied to guide the search process towards promising areas in the solution space. The authors empirically demonstrated the superiority of the TS to the GA [19] and Ronconi's method [20], and they also reported the upper bounds for 120 Taillard benchmarks [21]. Wang *et al.* [22] presented a hybrid genetic algorithm ($\mathrm{HGA_W}$ for short) and Liu *et al.* [23] proposed a hybrid particle swarm optimization (HPSO) method for a flowshop scheduling problem with limited buffers, where the blocking flowshop problem is a special case. Recently, Wang *et al.* [24] proposed a hybrid discrete differential evolution (HDDE) algorithm with better performance than the TS method. The HDDE applied a speed-up technology to reduce computational effort and improved 112 out of 120 upper bounds provided by Grabowski and Pempera [6] in 5 nm milliseconds CPU time. More recently, Ribas *et al.* [5] proposed a simple but effective iterated greedy (IG) algorithm. After long time numerical experiments, the authors claimed that the IG outperformed the HDDE algorithm and provided new best-known solutions for

most of Taillard benchmarks. Wang and Tang [25] presented a discrete particle swarm optimization (DPSO) algorithm, where a self-adaptive diversity control strategy was adopted to diversify the population and a stochastic variable neighborhood search was used to improve the search intensification. For the total flowtime criterion, Wang *et al.* [26] presented three hybrid harmony search algorithms and showed the effectiveness of these algorithms by computational experiments.

Memetic algorithms (MAs) represent one of the recent growing areas of the research work in evolutionary computation [27]–[32]. The MA is also referred in some literature as hybrid genetic algorithms (HGAs) or genetic local search. There is a rich literature where the MAs have been successfully applied to scheduling problems (see, for example, in [29], [30], [33]–[36], and among many others). Therefore, following their successful applications, we present an effective MA for the blocking flowshop problem with makespan criterion. The computational comparison shows that the proposed MA is a new state-of-the-art approach for the problem under consideration.

The remaining contents of this paper are organized as follows. In Section II, the blocking flowshop problem is stated and formulated. In Section III, the proposed MA is described in detail. The calibration of the MA is given in Section IV. A comprehensive comparison of the MA with the existing metaheuristics is provided in Section V. Finally, we conclude this paper in Section VI.

II. The Blocking Flowshop Scheduling Problem

There are n jobs $J = \{1, 2, \ldots, n\}$ and m machines $M = \{1, 2, \cdots, m\}$ in a flowshop with no intermediate buffer between machines. Each job $j \in J$ is to be sequentially processed on machine 1, machine 2, and so on until last machine m. The operation $o_{j,k}$ corresponds to the processing of job $j \in J$ on machine $k \in M$ during an deterministic, non-negative, and uninterrupted processing time $p_{j,k}$. Since the flowshop has no intermediate buffers, a job having finished its operation $o_{j,k}$ has to remain on the current machine k until the next machine $k+1$ is free. We assume that all the jobs are independent and available for processing at time 0; at any time, no job can be processed on more than one machine, and no machine can process more than one job simultaneously. Given that the sequence in which the jobs are to be processed is the same for each machine. The objective is then to find a sequence for processing all the jobs with minimal makespan (i.e., maximum completion time).

Each schedule of jobs can be denoted by a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, $\pi_i \in J$. Let $d_{[i],k}$ be the departure time of the operation $o_{[i],k}$, where $[i]$ represents the job π_i. It is clear that $d_{[i],k}$ provides a feasible schedule if and only if the following constraints are satisfied for $i \in \{1, 2, \ldots, n\}$ and $k \in M$:

$$d_{[i],k} \geq d_{[i],k-1} + p_{[i],k} \tag{1}$$

$$d_{[i],k} \geq d_{[i-1],k+1} \tag{2}$$

where $d_{[i],0} = 0$, $d_{[0],k} = 0$, and $d_{[i],m+1} = 0$.

Condition (1) represents the route of jobs through machines. Since the flowshop has no intermediate buffers, a job should immediately enter the next machine and start processing once it departs the current machine. So, $d_{[i],k-1}$ is equal to the start time of job π_i on machine k, and $d_{[i],k-1} + p_{[i],k}$ is the completion time of job π_i on machine k. Condition (2) follows from the given sequence π of processing jobs on each machine. It specifies that job π_i has to remain on machine k until the next machine $k+1$ is free, i.e., job π_{i-1} departs from machine $k+1$. This condition represents the critical core of the blocking flowshop scheduling problem.

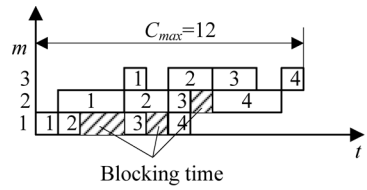

Fig. 1. An example of calculation for makespan.

Then, the schedule can be found using the following recursive formulae:

$$d_{[i],k} = \max\left\{d_{[i],k-1} + p_{[i],k}, d_{[i-1],k+1}\right\}$$
$$i = 1, 2, \cdots, n. \quad k = 1, 2, \cdots, m. \tag{3}$$

And the makespan is given by

$$C_{\max}(\pi) = d_{[n],m}. \tag{4}$$

Let Π denote the set of all the job permutations. Then, the objective of scheduling is to find a job permutation π^*, such that

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi). \tag{5}$$

The following example with four jobs and three machines illustrates the calculation of makespan in detail. Given a permutation $\pi = (1, 2, 3, 4)$ and the processing time $p_{j,k}$

$$[p_{j,k}]_{4\times 3} = \begin{bmatrix} 1 & 3 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \\ 1 & 3 & 1 \end{bmatrix}$$

Then, the departure time $d_{[i],k}$ is calculated as follows (see in Fig. 1):

$$\begin{aligned}
d_{[1],1} &= p_{[1],1} = 1 \\
d_{[1],2} &= d_{[1],1} + p_{[1],2} = 4 \\
d_{[1],3} &= d_{[1],2} + p_{[1],3} = 5 \\
d_{[2],1} &= \max\left\{d_{[1],1} + p_{[2],1}, d_{[1],2}\right\} = 4 \\
d_{[2],2} &= \max\left\{d_{[2],1} + p_{[2],2}, d_{[1],3}\right\} = 6 \\
d_{[2],3} &= d_{[2],2} + p_{[2],3} = 8 \\
d_{[3],1} &= \max\left\{d_{[2],1} + p_{[3],1}, d_{[2],2}\right\} = 6 \\
d_{[3],2} &= \max\left\{d_{[3],1} + p_{[3],2}, d_{[2],3}\right\} = 8 \\
d_{[3],3} &= d_{[3],2} + p_{[3],3} = 10 \\
d_{[4],1} &= \max\left\{d_{[3],1} + p_{[4],1}, d_{[3],2}\right\} = 8 \\
d_{[4],2} &= \max\left\{d_{[4],1} + p_{[4],2}, d_{[3],3}\right) = 11 \\
d_{[4],3} &= d_{[4],2,} + p_{[4],3} = 12
\end{aligned}$$

Thus, the makespan is: $C_{\max} = d_{[4],3} = 12$.

It can be seen from Fig. 1 that job 3 has to remain on machine 2 after it is finished at time 7 since machine 3 is not free. Thus, job 4 can be started only after job 3 leaves machine 2 at

time 8. The blocking time caused by job 3 is equal to 1, which affects the start time of job 4 on machine 2, and thus it affects the start time of all the successive operations and the makespan. If we consider the above examples without blocking constraints, i.e., in the regular flowshop environments, job 4 can be started earlier at time 7 since job 3 enters unlimited intermediate storages once it is finished. And the makespan value is 11. Thus, the blocking flowshop problem is different from the regular flowshop problem.

III. The Proposed Memetic Algorithm (MA)

Memetic algorithm (MA) is inspired by Darwinian principles of nature selection and Dawkins' notion of meme defined as a unit of cultural evolution that is capable of local/individual refinements. MA repeats evaluations, selection, crossover, mutation, and local refinements after initialization, until a given stopping criterion is met. This paper presents an effective MA for the blocking flowshop problem to minimize makespan. We first detail the components including solution representation, population initialization, selection operator, crossover operator, mutation operator, local search procedure, population updating and a diversity controlling mechanism, and then give the pseudocode of the proposed MA.

A. Solution Representation

A solution representation is one of the key issues to design MA. We adopt the job permutation based representation. That is, each chromosome or individual is represented as an n-job permutation. In this permutation, there should be no missing jobs and no job is repeated. The relative order of the jobs indicates the processing order of the jobs on the machines in the shop. This encoding method has been widely used in the flowshop literature [33]–[41] due to its simplicity and effectiveness.

B. Population Initialization

A set of PS initial individuals or chromosomes form an initial population, where PS represents population size. The initial population with a high level of quality and diversity often results in a faster convergence towards good solutions. A common trend in recent scheduling literature is to construct a few good initial individuals by effective heuristics and to produce others randomly. The profile fitting (PF) method and the NEH approach are two well-known heuristics for the blocking flowshop and regular flowshop, respectively. The PF heuristic [12] tries to sequence jobs for resulting in a minimum sum of the idle and blocking time on machines. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_i)$ denote a partial sequence containing the jobs already scheduled, U be the set of the unscheduled jobs. In order to append a job to the permutation π, for each job $j \in U$, we compute the idle time that a machine spends between processing jobs π_i and j, and the blocking time that job j causes, as if job j was appended to π. The job with the smallest sum of idle and blocking time is chosen. Following Ribas *et al.* [5] and Ronconi [4], we set the job with the smallest total processing time as the first job of the permutation π. The procedure of the PF heuristic is explained as follows.

As seen in Fig. 2, to append a job to $\pi = (\pi_1, \pi_2, \ldots, \pi_i)$, we have to compute the sum of the idle time and blocking time for each job $j \in U$. This results in a computational complexity $O(nm)$. There are a total of $n-1$ replications for the appending process, so the PF is with a computational complexity $O(n^2m)$.

Procedure PF
Select the job with the smallest sum of processing time on all the machines as the first job of a sequence π. Denote this sequence as $\pi = (\pi_1)$. Let $U = J - \{\pi_1\}$.
for $i := 1$ **to** $n-1$ **do** % (construct a whole sequence)
Compute the departure time $d_{[i],k}$, $k = 1, 2, ..., m$, for the last job π_i in the sequence $\pi = (\pi_1, \pi_2, ..., \pi_i)$.
For each job $j \in U$, as if it was appended and became the $(i+1)^{th}$ job of π, compute its departure times $d'_{[i+1],k}$, for $k = 1, 2, ..., m$, and the sum of the idle and blocking times $\delta_j = \sum_{i=1}^{m} (d'_{[i+1],k} - d_{[i],k} - p_{j,k})$.
Select the job resulting in smallest δ_j value as the $(i+1)^{th}$ job of π, and remove the selected job from U.
endfor
return π

Fig. 2. PF heuristic.

Procedure NEH
Generate a seed sequence $\beta = (\beta_1, \beta_2, ..., \beta_n)$ by sorting jobs according to their non-increasing total processing times.
$\pi := (\beta_1)$
for $i := 2$ **to** n **do** % (the NEH enumeration procedure)
Take job β_i from β and test it in all the i possible slots of π.
Inset job β_i in π at the slot resulting in the lowest objective value.
endfor
return π

Fig. 3. NEH heuristic.

The NEH method [14] is based on the idea that jobs with high total processing times (i.e., sum of processing time on all machines) should be scheduled as early as possible. Initially, the NEH heuristic yields a seed sequence $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ according to the non-increasing total processing times. Then, a partial sequence π is generated using β_1 as its first job, i.e., $\pi := (\beta_1)$. Afterwards, the remaining jobs in β are taken out one by one and inserted into the best slot of π. The procedure of the NEH heuristic is described as follows (Fig. 3).

There are a total of $(n(n+1)/2) - 1$ insertions, and each insertion generates a partial sequence. NEH has to evaluate all the generated sequences, so its computation complexity is $O(n^3m)$.

To obtain a higher performing heuristic for the blocking flowshop problem with makespan criterion, we present a combination of the PF and NEH. The presented heuristic first utilizes the PF heuristic to generate a partial sequence π with $n - \lambda$ jobs, where λ is an integer in the range of $[0, n]$. Then, the remaining λ jobs are inserted in the partial sequence π by the NEH enumeration procedure. We call the presented heuristic as PF+NEH, and give its procedure as follows.

In the PF+NEH (see in Fig. 4), the PF performs $n - \lambda$ replications to the appending process to generate a partial sequence $\pi = (\pi_1, \pi_2, \ldots, \pi_{n-\lambda})$. So the computational complexity in the PF phase is $O(n(n-\lambda)m)$. The NEH procedure locates the remaining λ jobs and evaluates a total of $\lambda(2n-\lambda+1)/2$ partial sequences. The computational complexity in the NEH phase is $O(\lambda n^2 m)$. Hence, the complexity of the PF+NEH heuristic is

Procedure PF+NEH(λ)
Generate a partial sequence $\pi = (\pi_1, \pi_2, ..., \pi_{n-\lambda})$ using the PF heuristic.
Generate another partial sequence $\beta = (\beta_1, \beta_2, ..., \beta_\lambda)$ by sorting the remaining jobs according to their non-decreasing total processing times.
for $i := 1$ **to** λ **do** % (the NEH enumeration procedure)
Take job β_i from β and test it in all the possible slots of π.
Inset job β_i in π at the slot resulting in the lowest makespan value.
endfor
return π

Fig. 4. PF+NEH heuristic.

$O(\lambda n^2 m)$. However, if the speed-up technology [24] designed to minimize makespan for the blocking flowshop problem is adopted, the complexity of the NEH enumeration procedure can be reduced to $O(\lambda nm)$. Thus, the PF+NEH heuristic can be accelerated in time $O(n^2 m)$.

A speed-up technology for a serial of consecutive insertion moves was first presented for the regular flowshop scheduling problem by Taillard [42]. Wang *et al.* [24] extended the method to the blocking flowshop problem. Given a partial sequence $\pi = (\pi_1, \pi_2, \ldots, \pi_l)$ with l jobs, we insert job j into all the $l+1$ possible slots of $\pi = (\pi_1, \pi_2, \ldots, \pi_l)$, respectively, and obtain $l+1$ new sequences. The evaluation of all these sequences can be completed in time $O(lm)$ as follows.

Step 1) Generate a schedule by first scheduling job π_1, then job π_2, and so on, until job π_l.

Step 2) Calculate departure time $d_{[i],k}$ for $o_{[i],k}$, $i = 1, 2, \ldots, l$, $k = 1, 2, \ldots, m$, using the recursive expressions (3).

Step 3) Put off the start time of $o_{[i],k}$, $k = m, m-1, \ldots, 1$, $i = l, l-1, \ldots, 1$, as late as possible, without changing the precedence of jobs and the departure time $d_{[l],k}$. Calculate the new start time $s'_{[i],k}$ for $o_{[i],k}$.

Step 4) Compute the time interval $I_{[i],k}$ between $s'_{[i],k}$ and $d_{[l],k}$, i.e., $I_{[i],k} = d_{[l],k} - s'_{[i],k}$, $i = 1, 2, \ldots, l$.

Step 5) Compute the makespan value for the new sequence generated by inserting job j at position h of $\pi = (\pi_1, \pi_2, \ldots, \pi_l)$, $h = 1, 2, \ldots, l, l+1$, as follows.

Step 5.1. Compute the departure time $d_{j,k}$ for job j, $k = 1, 2, \ldots, m$, according to the departure time $d_{[h-1],k}$.

Step 5.2. The makespan of the new sequence is given by $C_{\max} = \max_{k=1,2,\ldots,m} \{d_{j,k} + I_{[h],k}\}$.

With the PF+NEH, one chromosome is generated, and then it combines with $PS-1$ different chromosomes generated randomly to form the initial population of the proposed MA. To facilitate for the population updating procedure presented in the following sections, we always store the PS individuals in an ascending order according of their makespan values.

C. Selection, Crossover, and Mutation

A selection is a key operator in MA search, which is used to select some good chromosomes as seeds. For the selection of parents, we adopt the classical tournament selection where two chromosomes are picked up randomly from the population, and compared to each other, and then the one with better makespan value is chosen.

Procedure Path-relinking-shift(β, π)
% (transform β into π)
for $i := 1$ **to** n **do**
if $\pi_i \neq \beta_i$ **then**
Find job β_k in permutation β which is identical to job π_i.
Remove job β_k from its original position and insert it into position i of permutation β. If the generated permutation is not identical to π, put the generated permutation into set S;
endif
endfor
if S is empty, randomly pick a job from π and reinsert it into another randomly selected position in π. Put the generated permutation into S.
Evaluate all the solutions in S and output the one with best makespan value.
end

Fig. 5. The procedure of path relinking with shift moves.

After the selection, a crossover operator exchanges the information of the selected parents to generate promising offspring or sequences. Path relinking is a search technique to explore the search space or "path" between a given set of solutions [38]. It can be used to generate a set of new solutions or offspring between two solutions from the set. In the GA literature, the path relinking technique is most used as a local search procedure. Recently, Vallada and Ruiz [38] tested the path relinking technique either as a crossover operator or as a local search procedure for a minimum tardiness permutation flowshop problem. The computational experiments conducted by the authors demonstrated that the GA with the path relinking technique substituting crossover operator is statistically better. Thus, we consider the path relinking technique as the crossover operator, which is applied with a probability P_c to two distinct individuals selected by the tournament selection.

The path relinking technique can be described as follows. For two selected individuals β and π, we perform a series of shift or swap moves to transform β to π. Each time we carry out a move and obtain an intermediate solution. As the number of moves increase, the obtained intermediate solution looks less like β but more like π. At the end, we evaluate all the intermediate solutions and select the one with the lowest makespan value as a child of parents β and π. In our MA, we keep the individuals or chromosomes different from each other, so the selected parents β and π should not be identical. However, there exists the case that transforming β to π needs only one move. That is, no intermediate solution is obtained in the transforming process. In this case, we perform a random insertion or swap move to π to ensure that a new solution is generated, and the generated solution is set as the child. Note that the move set from individual β to π is not the same as those from π to β. Therefore, we yield another child by performing the path relinking from π to β. We test the path relinking with both shift and swap moves in Section IV, and select the better one as the crossover operator in the proposed MA. The procedures of the path relinking with shift and swap moves are described in Fig. 5 and Fig. 6, respectively

Let us consider an example with two permutations $\beta = (1, 2, 3, 5, 4)$ and $\pi = (2, 4, 3, 5, 1)$. By performing

```
Procedure Path-relinking-swap(β, π)
% (transform β into π)
  for i := 1 to n do
    if π_i ≠ β_i then
      Find job β_k in permutation β which is identical to job π_i.
      Exchange the positions of job β_k and β_i. Put the generated
      permutation into set S
    endif
  endfor
  if S is empty, randomly pick two jobs from π and exchange their
  positions. Put the generated permutation into S.
  Evaluate all the solutions in S and output the one with best makespan
  value.
end
```

Fig. 6. The procedure of path relinking with swap moves.

```
Procedure RLS(π, π^ref)
  repeat
    for i := 1 to n do
      π' := π
      Find job π_i^ref in π' and remove it.
      Test π_i^ref in all the possible slots of π'.
      Inset π_i^ref in π' at the slot resulting in the lowest makespan
  value.
      if C_max(π') < C_max(π) then
        π := π'
      endif
    endfor
  until π is not improved
return π
```

Fig. 7. Referenced local search.

Path-relinking-shift(β, π), we generate three intermediate solutions (2,1,3,5,4), (2,4,1,3,5), and (2,4,3,1,5). However, by performing Path-relinking-swap(β, π), we only yield an intermediate solution (2,1,3,5,4).

A mutation serves to introduce random variations into the population, essentially a mechanism to circumvent population stagnation [27], [28]. The swap mutation and shift mutation are two simple mutations commonly used in the flowshop literature. The swap mutation randomly chooses two jobs in the sequence and exchanges their positions; whereas the shift mutation randomly removes a job from its original position and relocates it in another randomly selected position. According to [36], the shift mutation is clearly better than the swap mutation. Therefore, we adopt shift mutation here. In the MA, this mutation operator is applied after crossover operator with a mutation probability P_m, that is, the offspring will undergo a mutation operator if a random number uniformly generated in the range of [0,1] is less than P_m.

D. Local Search

A local refinement heuristic plays a key role in the evolution process in MAs. For the blocking flowshop scheduling problem with makespan criterion, several local search methods are utilized and present good performance in their original papers. These are: the stochastic variable neighborhood search (SVNS) of Wang and Tang [25], the non-exhaustive descent algorithm (NEDA) of Ribas *et al.* [5], and the referenced local search (RLS) of Wang *et al.* [24]. These local searches need to evaluate lots of neighbors in the insertion or exchange neighborhood of the current solution, and the computational efforts are very expensive. However, the speed-up technology in Section III-B can be used to save the computational time for the RLS method. To present an efficient MA that can find good solutions in limited computational time, we adopt the RLS here. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a sequence to be improved and $\pi^{ref} = (\pi_1^{ref}, \pi_2^{ref}, \ldots, \pi_n^{ref})$ be a referenced sequence. The procedure of the RLS is given as follows.

It can be seen from Fig. 7 that the RLS method aims to find a better permutation than π in its neighborhood. According to the theorem presented by Grabowski and Pempera [6] based on the graph model, some insertion moves cannot produce neighbors with better makespan. We can remove these non-improving moves from the insertion move set to further save computational time (see in Wang *et al.* [24]). The computational time requirement of the RLS method is further decreased.

The RLS is applied to each of the offspring individuals after the selection, crossover and mutation if the individual is different from its parents. We also perform the RLS to the best individuals after initialization of the population.

E. Population Updating

Another aspect considered in the MA is the way to select the individuals for the next generation. To maintain the diversity of the population to avoid both cycling search and getting trapped in a local optimum, we update the population as follows. The best PS different individuals among all the parent and offspring individuals are reserved as the population for the next generation. As a result, the population contains different individuals, which will steadily evolve to a promising region. This population updating method is similar to the generational scheme presented in [36], which results in a very effective genetic algorithm.

F. Diversity Controlling Mechanism

As the population evolves over generations, the diversity of the populations falls, and the individuals in the population may become very similar. The algorithm is or will soon be stalled around a local optimum. To overcome this problem, according to literature [36], [38], a restart mechanism is applied when the best solution found so far is not improved in γ generations. The restart mechanism generates 50% new individuals by performing two shift mutations to the 50% best individuals in the current population, and yields the remaining 50% individuals randomly. With this restart mechanism, the MA is expected to reintroduce diversity in the population and to escape from a local optimum.

G. Computational Procedure of the Proposed MA

With all the above designs, the procedure of the proposed MA is described in Fig. 8.

In the above MA, the heuristic-based initialization procedure provides a good initial population with a certain level of quality and diversity. The path-relinking-based crossover operator effectively explores the search space between two parents. The diversity controlling mechanism and population updating method remains the diversity of the population. The problem-dependent

```
Procedure MA
    Population:=InitializePopulation();%(Population initialization)
    Bestsofar:=FindBestIndividual(population); %(Store best sequences
  of the population)
    Bestsofar:=RLS(Bestsofar, Bestsofar);        %(Perform local search)
    while NOT Termination Criterion do      %(Main loop)
      while Size of Offspring population<PS do
        %(Selection)
        Parent1:= TournamentSelection(population);
        Parent2:=TournamentSelection(population);
        %(Crossover)
        if rand()<Pc then
          Offspring1:=Path-relinking-Swap(Parent1, Parent2);
          Offspring2:=Path-relinking-Swap(Parent2, Parent1); }
        else
          Offspring1:=Parent1;
          Offspring2=:Parent2;
        endif
        %(Mutation)
        if rand()<Pm then
          Offspring1:=Mutation(Offspring1);
        endif
        if rand()<Pm then
          Offspring2:=Mutation(Offspring2);
        endif
        %(local search, and add offspring to offspring population)
        if Offspring1≠Parent1 or Parent 2 then
          Offspring1:= RLS (Offspring1, Offspring1);
          OffspringPopulation:=Add(Offspring1);
        endif
        if Offspring2≠Parent2 or Parent 1 then
          Offspring2:= RLS (Offspring2, Offspring2);
          OffspringPopulation:=Add(Offspring2);
        endif
      end while
      Population:=PopulationUpdating(Population,
      OffspringPopulation); %(Update population)
      Bestsofar:=UpdateBestsofar(Population);
      if Bestsofar is not improved in γ generations then
          Restart Population;
      end if
    end while
  return Bestsofar;
```

Fig. 8. Pseudocode for the proposed MA.

local search performs local refinements so as to balance exploration and exploitation. Since the balance of generality and problem specificity is stressed, it is expected to achieve good results for the blocking flow shop scheduling problems with makespan criterion.

IV. Calibration of the Proposed MA

A. Calibration of the PF+NEH Heuristic

The PF+NEH has a control parameter λ. It is equal to the PF if $\lambda = 0$, and to the NEH if $\lambda = n$. To determine an appropriate λ value for the PF+NEH, we conduct a simple experiment based on the following instances and measure.

The test set of instances is generated with $n \in \{50, 75, 100, 125, 150, 175, 200\}$ and $m \in \{5, 10, 15, 20\}$. One instance is generated for each combination of n and m. This leads to a total of 28 instances. The processing times are given by a discrete uniform distribution in the interval [1, 99]. The above distribution of processing times, the number of jobs, n, and the number of machines, m, have been widely used in the scheduling literature. In addition, a set of 28 instances is

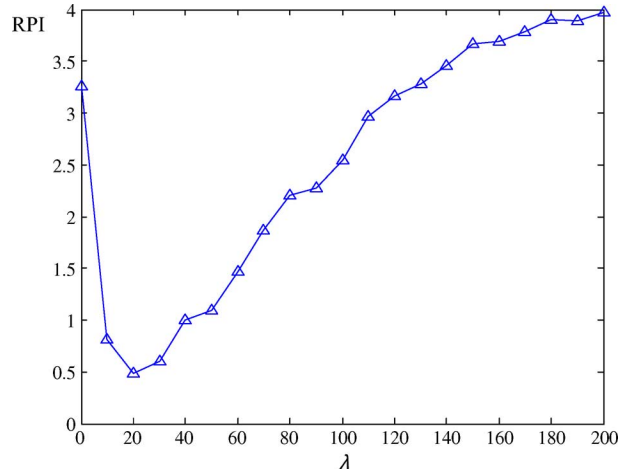

Fig. 9. Behavior of RPI as a function of λ for the PF+NEH heuristic.

enough to show the difference of the performances between the compared algorithms.

The PF+NEH is coded in VC++ 6.0 and run on a Pentium (R) 4 CPU 2.80 GHz with 512 MB Main Memory in Windows XP Operation System. The relative percentage increase (RPI) is calculated as a response variable for the experiments as follows:

$$RPI(c_i) = (c_i - c^{\min})/c^{\min} \times 100 \tag{6}$$

where c_i is the makespan value generated by a given algorithm configuration, and $c^{\min}$ is the minimum makespan value found among all the configurations.

In the experiment, we set λ from 0 to 200 with a step equal to 10. That is, 21 different algorithm configurations are considered. If the number of jobs, n, is less than λ, we perform the NEH to this instance. Since the instance with the largest size has $n = 200$ jobs in the experiment, we perform the NEH to all the instances when $\lambda = 200$. Fig. 9 shows the performance of the PF+NEH heuristic with the various values of λ.

According to Fig. 9, the PF+NEH with $\lambda \in [10, 30]$ produces much lower RPI value than with $\lambda = 0$ and $\lambda = 200$, suggesting that the PF+NEH outperforms both the PF and NEH at a substantial margin. It can be also observed that the PF+NEH presents its best performance at $\lambda = 20$, the difference of RPI is not large when λ is in the range of $\lambda \in [10, 30]$, which shows some robustness regarding the choice of the parameter λ. Finally, we set $\lambda = 20$ for the PF+NEH.

We investigate whether other heuristics combined with the NEH lead to better results. Five heuristics are considered. These are: two commonly used dispatch rules in the scheduling literature, i.e., the shortest processing time (SPT) and the largest processing time (LPT) rules, the MM heuristic proposed in [4] for the blocking flowshop problem, the shortest blocking time (SBT) rule, and the shortest idle time (SIT) rule. The basic PF addresses to minimizing both idle and blocking times. We generate SBT/SIT rule by only addressing to minimizing the blocking/idle time in the PF. We combine the above five methods with the NEH, as does in the PF+NEH, and obtain five approaches denoted as SPT+NEH, LPT+NEH, MM+NEH, SBT+NEH, and SIT+NEH, respectively. These methods are run for the 28 instances on the PC mentioned above with the

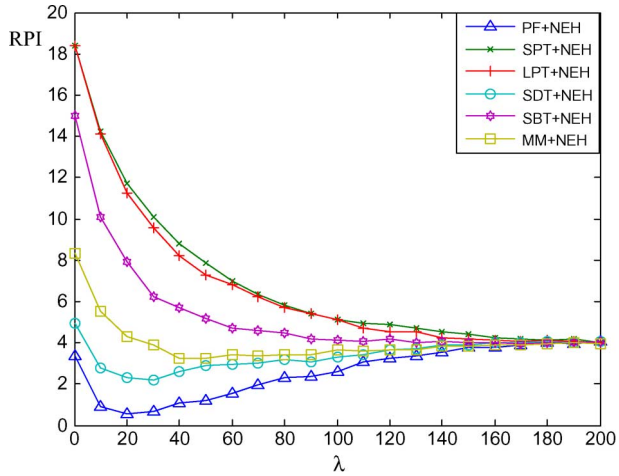

Fig. 10. Behavior of RPI as a function of λ for the six heuristics.

parameter λ varied from 0 to 200 with a step 10. We report the performance of these heuristics and the PF+NEH in Fig. 10.

As we can see from Fig. 10 that the PF+NEH performs much better than the other methods at all the λ values except $\lambda = 200$, where all the six heuristics are equal to the NEH approach. This suggests the effectiveness of trying to minimize the idle and blocking time on machines in the construction process of an initial sequence.

We next examine the method of combining the PF and NEH. We first generate a sequence of all the n jobs using the PF, and then randomly extract λ jobs from the generated sequence and form a partial sequence β (see in Fig. 4) by sorting the extracted jobs according to their total non-increasing process times. Finally, a result permutation is yielded using the NEH to relocate these λ jobs. We call this method as RAND. Since the RAND is non-deterministic, we run it five replications and denote them as RAND1, RAND2, RAND3, RAND4, and RAND5, respectively. We compare the RAND methods and the PF+NEH based on the 28 instances, and report the experimental results in Fig. 11. It is clear from Fig. 11 that the PF+NEH produces much better results than the RAND methods, suggesting the effectiveness of the strategy used to combine the PF and NEH in the PF+NEH.

B. Calibration of the Proposed MA

We calibrate the presented MA by a series of pilot experiments. We consider a full factorial design where the different choices for the following parameter values and operators are considered. Population size (PS) is tested at seven levels: 3, 5, 8, 10, 20, 30, and 40; Mutation probability (Pm) is tested at six levels: 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0; Crossover probability (Pc) is tested at six levels: 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0; Restart (γ) is tested at four levels: 20, 40, 60, and 80; Crossover type is tested at two levels: Path-relinking-shift and Path-relinking-swap. All the possible combinations of the above factors result in a total of $7 \times 6 \times 6 \times 4 \times 2 = 2016$ different configurations for the proposed MA.

We code the presented MA in VC++ 6.0, and run all the above 2016 configurations on the aforementioned PC with a termination criterion set as maximum elapsed CPU time $t = 5\ \mathrm{nm}$ milliseconds. We employ the test set generated in Section IV-A, and five independent replications are carried out for each of the 28 instances. The RPI is calculated as a response variable. Note that in the experiment, we have a total of 2016 algorithm configurations which are executed five times and give a total of $2016 \times 5 = 10080$ results for each instance. The minimum makespan $c^{\min}$ used when computing RPI is the best one among the 10080 results.

Following Ruiz *et al.* [36], Ruiz and Stuzle [37], Vallada and Ruiz [38], and many others, we analyze the experimental results by means of a multifactor analysis of variance (ANOVA) technique. This is a very powerful statistical approach to set the different parameters at statistically significant values among the test ones. We carefully check three main hypothesis of ANOVA, i.e., normality, homogeneity of variance, and independence of residuals. The results from the experiments show that all of the hypothesis could be accepted. The resulting ANOVA table for the experiment is shown in Table I.

The experimental results are analyzed by using F-ratio for those factors whose p-value is close to zero. The p-value is a measure of statistical significance, and p-value close to zero suggests that there is a statistically significant difference between the levels of the factor or interaction considered. F-ratio is a ratio between the variance explained by a factor and the unexplained variance. A large F-ratio indicated that the factor has great effect on the response variable. We do not consider the interactions of more than two factors since their F-ratios are negligible.

The greatest F-ratio, according to the results from Table I, corresponds to the mutation rate or P_m. This suggests that the P_m has the most important effect over the response variable among the considered factors. The P_m has six levels: 0, 0.2, 0.4, 0.6, 0.8, and 1.0. The six means with the Tukey HSD intervals (at the 95% confidence level) are plotted in Fig. 12.

As we can see from Fig. 12, the mutation rate $P_m = 0$ results in the worst results, suggesting the effectiveness of using mutation operator in the MA. For the MA with mutation operator, mutation rate of 0.8 produces the lowest RPI value which is significantly better than those generated by mutation rate of 0.2, 0.4 and 0.6. However, no significant difference is observed for the response variable among the mutation rate of 0.8 and 1.0. This indicates that the proposed MA is robust with regards to the choice of mutation rate.

The population size or PS corresponds to the second biggest F-ratio. There are seven levels considered for this factor. The means plot for this factor can be observed in Fig. 13. It can be observed from Fig. 13 that a population size value of ten results in significantly better solutions than that of 3, 5, 8, 20, 30, and 40. Therefore, we set $PS = 10$ for the proposed MA.

We next examine the probability of crossover or P_c. From the means plot in Fig. 14, it is clear that not using crossover leads to a very bad MA. On the other hand, a MA with $P_c = 1$ also produces worse RPI values than with $P_c = 0.2$, 0.4, 0.6, or 0.8. For the crossover rate equal to 0.2, 0.4, 0.6, and 0.8, the MA with $P_c = 0.2$ produces best results. The means plot of the interaction of crossover rate P_c and mutation rate P_m also shows that $P_c = 0.2$ and $P_m = 0.8$ lead to a best performing MA. This setting is quite different from that in the GA literature, where a large crossover rate and a small mutation rate are commonly used. We find, in the experiments, that there are many

TABLE I
ANOVA TABLE FOR THE MA EXPERIMENT

Source	Sum of Squares	Df	Mean Square	*F*-Ratio	*p*-Value
MAIN EFFECTS					
A: Crossover type	145.38	1	145.38	739.04	0.0000
B: γ	10.2883	3	3.42943	17.43	0.0000
C:*Pc*	1718.09	5	343.618	1746.79	0.0000
D:*Pm*	12443.7	5	2488.73	12651.52	0.0000
E:*PS*	4001.77	6	666.962	3390.52	0.0000
INTERACTIONS					
AB	0.0929716	3	0.0309905	0.16	0.9249
AC	30.0794	5	6.01587	30.58	0.0000
AD	142.041	5	28.4083	144.41	0.0000
AE	19.2263	6	3.20438	16.29	0.0000
BC	3.8658	15	0.25772	1.31	0.1857
BD	35.1858	15	2.34572	11.92	0.0000
BE	11.7588	18	0.653268	3.32	0.0000
CD	10368.6	25	414.744	2108.36	0.0000
CE	233.462	30	7.78205	39.56	0.0000
DE	2433.21	30	81.107	412.31	0.0000
RESIDUAL	55486.5	282067	0.196714		
TOTAL(CORRECTED)	87083.2	282239			

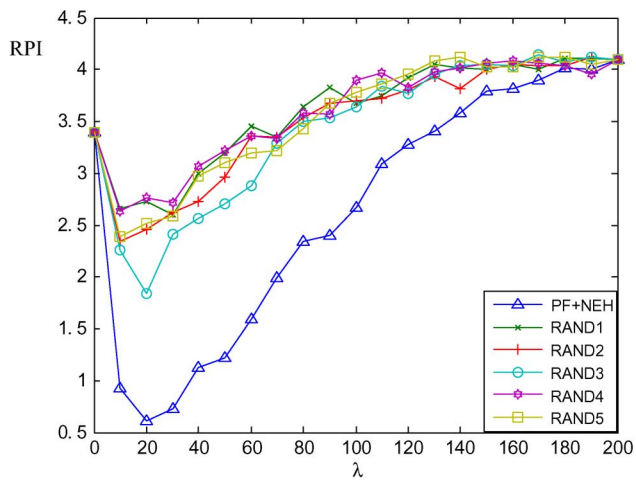

Fig. 11. Comparison of different combining methods of the PF and NEH.

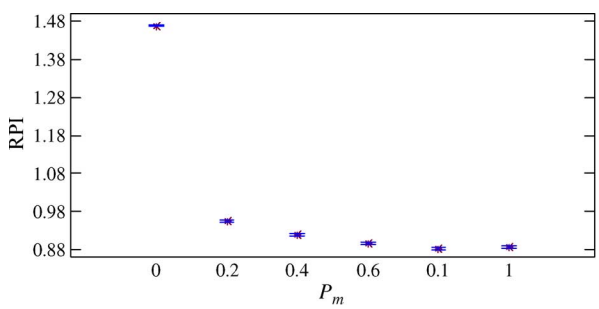

Fig. 12. Means plot for mutation rate Pm.

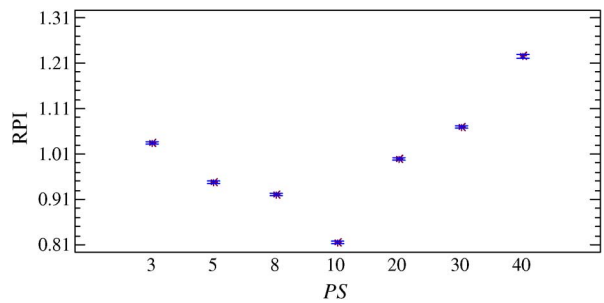

Fig. 13. Means plot for Population size PS.

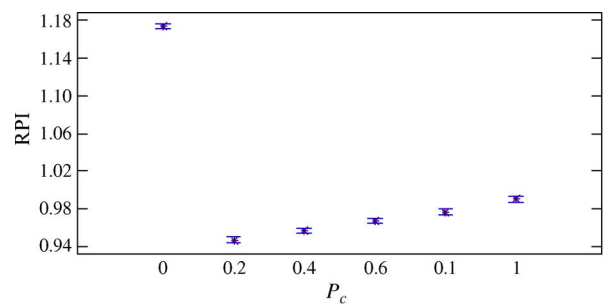

Fig. 14. Means plot for crossover rate Pc.

local optima close to each other in the solution space of the considered problem. The shift mutation causing local changes to an individual has a higher chance to find a local optimum than the crossover operator.

We proceed with the analysis and report the means plot of crossover type in Fig. 15. There is a statistically significant difference between the two crossover operators. Clearly, Path-relinking-swap leads to a better performing MA than Path-relinking-shift. Considering the shift mutation is used, this can be explained as an effective synergy between the swap-based crossover and shift mutation.

Finally, we set the parameters as follows according to the above analysis. Population size $PS = 10$; Mutation probability $P_m = 0.8$; Crossover probability $P_c = 0.2$; Crossover operator is set as Path-relinking-swap. For the diversity threshold, we simply set $\gamma = 20$ since there is no significant difference observed among the tested four levels.

C. On the Effectiveness of the Components in MA

The proposed MA includes some effective techniques like the heuristic-based initialization, path-relinking-based

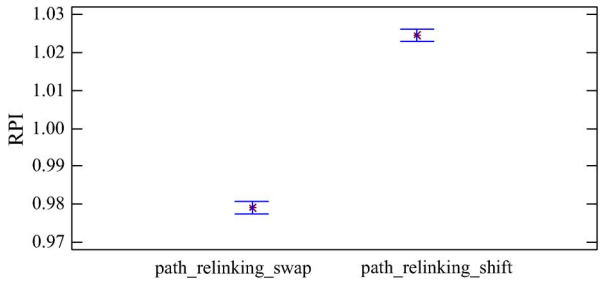

Fig. 15. Means plot for the type of crossover operator.

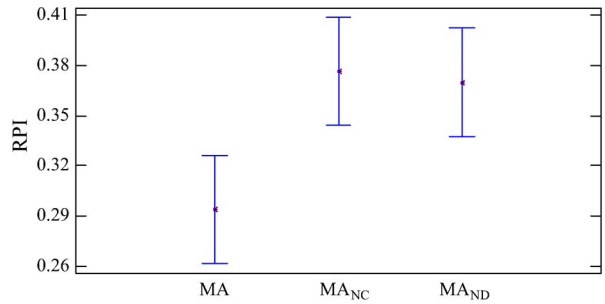

Fig. 16. Means plot for the MA, $\mathrm{MA_{NC}}$, and $\mathrm{MA_{ND}}$.

TABLE II
COMPARE THE COMPONENTS OF THE PROPOSED MA

Instances	$\mathrm{MA_{RI}}$	MA	$\mathrm{MA_{NC}}$	$\mathrm{MA_{ND}}$	GA
50×5	0.96	0.74	0.79	0.78	2.75
50×10	0.38	0.39	0.39	0.39	1.89
50×15	0.62	0.32	0.81	0.72	1.42
50×20	0.24	0.27	0.35	0.40	1.58
75×5	0.57	0.46	0.40	0.47	2.45
75×10	0.72	0.45	0.38	0.30	2.06
75×15	0.53	0.29	0.47	0.43	1.99
75×20	0.85	0.37	0.70	0.83	1.60
100×5	1.55	0.30	0.31	0.26	1.51
100×10	0.53	0.30	0.38	0.39	2.03
100×15	0.94	0.25	0.42	0.35	1.77
100×20	0.59	0.38	0.36	0.35	1.60
125×5	1.92	0.08	0.31	0.26	1.39
125×10	1.02	0.48	0.45	0.45	1.50
125×15	0.76	0.22	0.21	0.25	1.51
125×20	1.04	0.26	0.34	0.34	1.56
150×5	2.70	0.22	0.26	0.24	0.84
150×10	1.54	0.17	0.31	0.28	1.37
150×15	1.39	0.24	0.47	0.47	1.56
150×20	1.04	0.26	0.31	0.31	1.33
175×5	1.97	0.22	0.29	0.29	1.13
175×10	2.07	0.22	0.33	0.33	1.11
175×15	1.02	0.18	0.21	0.20	1.25
175×20	2.03	0.26	0.24	0.24	1.18
200×5	2.56	0.16	0.25	0.25	1.13
200×10	1.91	0.23	0.20	0.20	1.32
200×15	2.32	0.27	0.34	0.34	0.98
200×20	1.74	0.16	0.27	0.27	1.36
Average	1.27	0.29	0.38	0.37	1.54

crossover operator, referenced local search procedure, and diversity controlling mechanism. This section first examines the contribution of each component to the proposed MA. We obtain three algorithms by omitting a component from the MA at a time. These are: the MA without local search or pure GA, the MA without diversity controlling mechanism ($\mathrm{MA_{ND}}$ for short), and the MA without crossover operator or $\mathrm{MA_{NC}}$ (i.e., set the crossover rate $P_c = 0.0$ in the MA). We also implement a random initialization MA ($\mathrm{MA_{RI}}$) by starting the proposed MA from a population of initial individuals randomly generated. The above four algorithms and the proposed MA are utilized to solve the 28 instances on the aforementioned PC with an elapsed CPU time limit $t = 5\ \mathrm{mn}$ milliseconds. Five independent replications are carried out for each instance. The RPI values of each algorithm are reported in Table II.

It can be easily seen from Table II that all the four algorithms (i.e., $\mathrm{MA_{RI}}$, $\mathrm{MA_{NC}}$, $\mathrm{MA_{ND}}$, and GA) produce the worse overall RPI values than the MA, suggesting the contributions of the components tested to the proposed MA. The GA and $\mathrm{MA_{RI}}$ are the two worst algorithms with the largest overall RPI values (1.27% and 1.54%, respectively) even four times larger than that of the MA. This implies that the referenced local search and heuristic-based initialization play key contributions. Recall that both the NEH+PF and RLS procedures are problem-dependent. This further verifies the conclusion by Chen *et al.* [27] that distinct methods when combined together in a synergistic manner with the incorporation of domain knowledge can greatly enhance the problem-solving capability of the derived hybrid.

We next check whether the contributions of the proposed crossover operator and diversity controlling mechanism are indeed statistically significant. A multifactor ANOVA is carried out to analyze the RPI values generated by the MA, $\mathrm{MA_{NC}}$, and $\mathrm{MA_{ND}}$. We consider the type of algorithms, the number of jobs, n, and the number of machines, m, as factors, and report the three means with 95% Tukey HSD confidence intervals in Fig. 16. Note that an overlapping interval denotes an insignificant difference between the plotted means. From Fig. 16 it is clear that the MA performs significantly better than the $\mathrm{MA_{NC}}$ and $\mathrm{MA_{ND}}$. So, it can be concluded that the high performance of the proposed MA is due to the effective synergy of the heuristic-based initialization, path-relinking-based crossover operator, referenced local search, and diversity controlling mechanism.

We continue our experiments to investigate whether the degree of the GA-based global search versus the RLS-based local search is well balanced. We introduce a new parameter, the probability of local search or P_{ls}, to the MA. The probability defines the frequency of a child that should undergo the RLS procedure. Here, we vary the P_{ls} value from 0.0 to 1.0 with a step 0.25. Note that in the proposed MA, we fix $P_{ls} = 1.0$, i.e., the RLS is applied to each of the offspring individuals. We report the RPI values generated by the MA with different P_{ls} values in Table III.

It is clear from Table III that $P_{ls} = 1.0$ leads to the best results, indicating that the proposed MA balances the degree of global search versus local search well to make good use of the computational time available. In other words, the high performance of the proposed MA is mainly due to the effective combination of global search and local search, i.e., the balance of exploration and exploitation.

Finally, we investigate the effect of the different heuristics including the PF, NEH, and PF+NEH, on the proposed MA. We use the PF or NEH to produce an initial solution instead of the PF+NEH, and obtain two algorithms, i.e., the MA with the PF-based initialization ($\mathrm{MA_{PF}}$) and the MA with the NEH-based initialization ($\mathrm{MA_{NEH}}$). We execute the $\mathrm{MA_{PF}}$, $\mathrm{MA_{NEH}}$, and MA and report their RPI values in Table IV. It

TABLE III
COMPARE THE MA WITH DIFFERENT PROBABILITIES OF LOCAL SEARCH

Instances	P_{ls}=0.0	P_{ls}=0.25	P_{ls}=0.5	P_{ls}=0.75	P_{ls}=1.0
50×5	2.64	0.63	1.16	1.11	0.64
50×10	2.25	0.66	0.78	0.78	1.01
50×15	1.42	0.70	0.57	0.52	0.32
50×20	1.49	0.35	0.33	0.27	0.17
75×5	2.69	0.81	0.68	0.77	0.73
75×10	2.06	0.47	0.37	0.42	0.45
75×15	1.95	0.36	0.40	0.62	0.25
75×20	1.60	0.41	0.53	0.87	0.37
100×5	1.78	1.19	0.64	0.46	0.56
100×10	2.03	0.59	0.55	0.39	0.30
100×15	1.77	0.59	0.41	0.27	0.25
100×20	1.72	0.68	0.56	0.34	0.51
125×5	1.53	0.58	0.47	0.24	0.22
125×10	1.52	0.33	0.48	0.40	0.49
125×15	1.82	0.57	0.69	0.57	0.53
125×20	1.56	0.53	0.39	0.42	0.26
150×5	0.84	0.44	0.21	0.32	0.22
150×10	1.50	0.49	0.25	0.35	0.30
150×15	1.63	0.38	0.49	0.32	0.30
150×20	1.33	0.35	0.52	0.48	0.27
175×5	1.13	0.46	0.16	0.17	0.22
175×10	1.19	0.54	0.43	0.28	0.30
175×15	1.51	0.45	0.52	0.35	0.45
175×20	1.18	0.34	0.31	0.44	0.26
200×5	1.35	0.49	0.46	0.18	0.37
200×10	1.40	0.40	0.38	0.30	0.31
200×15	0.98	0.21	0.33	0.22	0.27
200×20	1.36	0.40	0.36	0.35	0.16
Average	1.62	0.51	0.48	0.44	0.37

TABLE IV
COMPARE THE MA WITH DIFFERENT HEURISTICS

Instances	MA_{PF}	MA_{NEH}	MA
50×5	0.49	1.24	0.74
50×10	1.02	0.99	1.23
50×15	0.78	0.46	0.32
50×20	0.34	0.72	0.47
75×5	0.67	0.89	0.49
75×10	0.33	0.79	0.53
75×15	0.46	0.54	0.27
75×20	0.76	0.65	0.37
100×5	0.60	1.62	0.19
100×10	0.72	1.00	0.30
100×15	0.93	0.45	0.25
100×20	0.40	0.61	0.20
125×5	0.56	2.41	0.08
125×10	0.44	1.01	0.34
125×15	0.45	0.73	0.20
125×20	0.63	0.89	0.26
150×5	0.57	2.84	0.22
150×10	0.50	1.41	0.14
150×15	0.44	1.13	0.24
150×20	0.35	0.90	0.30
175×5	0.65	1.77	0.22
175×10	0.44	1.55	0.22
175×15	0.35	1.13	0.44
175×20	0.53	1.79	0.26
200×5	0.18	2.62	0.25
200×10	0.50	1.41	0.23
200×15	0.51	1.74	0.27
200×20	0.25	1.50	0.31
Average	0.53	1.24	0.33

can be observed that the MA produces much better results than the $\mathrm{MA_{PF}}$ and $\mathrm{MA_{NEH}}$, suggesting the effectiveness of using the PF+NEH in the population initialization.

V. COMPARATIVE EVALUATIONS OF HEURISTICS

In this section, we evaluate the proposed MA by extensive computational experiments. We re-implement five effective meta-heuristics presented for the blocking flowshop problem with makespan criterion, including the HDDE [24], IG [5], $\mathrm{HGA_W}$ [22], HPSO [23], and the DPSO [25]. We also adapt three hybrid genetic algorithms recently presented for the regular flowshop problem to the problem considered here. These are the hybrid genetic algorithm of Tseng and Lin [33] (denote $\mathrm{HGA_{T1}}$), the hybrid genetic algorithm of Tseng and Lin [34] (denoted as $\mathrm{HGA_{T2}}$), and the hybrid genetic algorithm of Zhang *et al.* [35] (denoted as $\mathrm{HGA_Z}$). All the algorithms have been coded in VC++ 6.0 and run on a Pentium (R) 4 CPU 3.00 GHz with 1.0 GB Main Memory in Windows XP Operation System. We try to apply the speed-up technology presented in [24] and the partial speed-up technology presented in [41] to the above algorithms when it is possible. To make a fair comparison at limited computational time, all the algorithms adopt the same maximum elapsed CPU time of $t = \rho n(m/2)$ milliseconds as a termination criterion, where ρ has been tested at three values: 10, 20, and 30.

To the best of our knowledge, no common data sets were originally presented for the considered problem. The well-known regular flowshop benchmarks presented by Taillard [21] are considered as blocking flowshop instances with makespan criterion and used to test solution methods in [5], [6], and many others. The test bed consists of a total of 120 instances (denoted as TA01 $\sim$ TA120) of various sizes, having 20, 50, 100, 200, or 500 jobs, and 5, 10, or 20 machines. Each of subsets consists of 10 instances with the same size. We compare all the above algorithms based on the Taillard test bed. For each of the 120 instances, five independent replications are carried out and for each replication, and the RPI is calculated. For convenience for the readers to compare with our algorithm, we adopt the upper bounds taken from [5] as $c^{\min}$. The computed results, averaged across the 5 replications for each instance and grouped for each subset, are reported in Tables V–VII (the minimum RPI values are in bold).

It is clear from Tables V–VII that the presented MA is the winner in terms of overall solution quality. For the shortest CPU time $\rho = 10$, the proposed MA generates the lowest overall average RPI equal to 0.00%, which is much smaller than those generated by the HDDE (1.27%), DPSO (2.35%), IG (2.18%), HPSO (2.43%), $\mathrm{HGA_W}$ (4.75%), $\mathrm{HGA_Z}$ (1.27%), $\mathrm{HGA_{T1}}$ (8.25%), and $\mathrm{HGA_{T2}}$ (17.71%). For 9 out of 12 instance sizes, the presented MA produces the minimum RPI values. These RPI values are much lower than those generated by the other eight algorithms. Especially, for the three large-scale instance sizes, i.e., 200 × 10, 200 × 20, and 500 × 20, the presented MA yields RPI values less than zero, suggesting that the MA improves the upper bounds found by Ribas *et al.* [5]. Note that Ribas *et al.* [5] found these upper bounds at a very long computational time 30000 $\mathrm{n^2m}$ milliseconds, whereas our computational time is only $t = 10 \cdot n(m/2) = 5$ mn milliseconds. However, the proposed MA does not produce the minimum RPI values for three smallest instance sizes 20 × 5, 20 × 10, and 20 × 20. For the instance sizes 20 × 10 and 20 × 20, it is outperformed by the HDDE, and for the 20 × 5, it is surpassed by the $\mathrm{HGA_Z}$, although the difference of the RPI values generated by the MA and that by the $\mathrm{HDDE/HGA_Z}$ is

TABLE V
COMPUTATIONAL RESULTS OF THE ALGORITHMS ($\rho = 10$)

Instances	HGA_{T1}	HGA_{T2}	HGA_Z	HGA_W	HPSO	IG	DPSO	HDDE	MA
20×5	1.65	6.91	**0.04**	1.70	2.14	0.58	2.50	0.06	0.05
20×10	1.01	6.29	0.06	1.49	1.66	0.53	2.39	**0.02**	0.06
20×20	0.79	4.89	0.03	1.16	1.40	0.41	2.41	**0.01**	0.02
50×5	10.55	26.93	1.69	5.26	2.44	2.70	3.16	1.23	**0.61**
50×10	8.23	22.09	2.02	5.54	2.83	2.57	3.21	0.92	**0.49**
50×20	5.97	18.93	1.95	5.39	3.28	2.18	3.11	0.76	**0.39**
100×5	15.07	29.25	1.70	7.42	1.63	3.11	2.00	2.36	**0.30**
100×10	11.57	22.74	2.26	6.73	1.94	2.92	2.51	1.58	**0.06**
100×20	8.11	17.45	2.51	4.88	2.18	2.23	2.24	1.23	**0.14**
200×10	14.71	23.87	1.58	7.48	3.13	3.85	2.15	2.97	**-0.02**
200×20	10.24	17.32	2.39	5.43	3.16	2.70	1.96	2.10	**-0.06**
500×20	11.08	15.88	-1.01	4.53	3.40	2.35	0.54	1.97	**-2.08**
Average	8.25	17.71	1.27	4.75	2.43	2.18	2.35	1.27	**0.00**

TABLE VI
COMPUTATIONAL RESULTS OF THE ALGORITHMS ($\rho = 20$)

Instances	HGA_{T1}	HGA_{T2}	HGA_Z	HGA_W	HPSO	IG	DPSO	HDDE	MA
20×5	1.41	6.04	0.03	1.48	2.14	0.51	2.50	**0.01**	0.03
20×10	0.88	5.23	0.06	1.35	1.66	0.41	2.39	0.02	**0.01**
20×20	0.68	4.09	0.03	1.10	1.40	0.31	2.41	**0.01**	**0.01**
50×5	9.11	26.88	1.20	4.53	2.44	2.44	3.16	0.84	**0.34**
50×10	6.74	21.45	1.45	4.86	2.83	2.29	3.21	0.62	**0.26**
50×20	4.66	18.87	1.38	4.89	3.26	1.90	3.11	0.49	**0.25**
100×5	13.60	29.20	1.63	6.78	1.60	2.60	2.00	1.95	**0.04**
100×10	10.22	22.72	2.10	6.38	1.89	2.40	2.51	1.21	**-0.29**
100×20	7.09	17.43	2.33	4.68	2.08	1.90	2.24	0.88	**-0.21**
200×10	13.56	23.86	1.46	7.37	2.03	3.47	2.15	2.63	**-0.25**
200×20	9.19	17.31	2.22	5.34	2.19	2.47	1.96	1.83	**-0.33**
500×20	10.16	15.88	-1.01	4.51	3.40	1.99	0.54	1.59	**-2.17**
Average	7.28	17.41	1.07	4.44	2.24	1.89	2.35	1.01	**-0.19**

very small. When compared with the remaining six algorithms, the MA performs substantially better for all the 12 instance sizes. From the above discussion, we can conclude that the presented MA outperforms the other compared algorithms at a considerable margin for middle- or large-scale instance sizes, and for the small-scale instance sizes, it achieves comparable results with the HDDE and $\mathrm{HGA_Z}$ algorithms.

The HDDE and $\mathrm{HGA_Z}$ produce the second lowest overall average RPI value of 1.27%, which is much better than the third lowest overall average RPI values of 2.18% provided by the IG algorithm. Recall that in the MA, HDDE, and $\mathrm{HGA_Z}$, the PF+NEH and/or the speed-up technology are adopted. So, it is essential to take advantage of problem-specific characteristics for an effective solution method for the blocking flowshop problem with makespan criterion. However, different from the MA, the HDDE applied the NEH-based initialization and DE-based global search. These key differences lead to the relative worse results. The recently presented DPSO algorithm ranks fourth with 2.35% RPI value. The worst two performing algorithms are the $\mathrm{HGA_{T1}}$ and $\mathrm{HGA_{T2}}$ with the overall average RPI values equal to 8.25% and 17.71%, respectively.

For the CPU time $\rho = 20$, we find from Tables VI that all the algorithms except DPSO improve their results with additional elapsed CPU time. It seems that the DPSO is trapped into local optima and stagnates around the solutions at this time interval. The presented MA further improves the upper bounds found by Ribas *et al.* [5] for five large-scale instance sizes, and yields the lowest overall average RPI values equal to -0.19%. At this termination criterion, the MA achieves the minimum RPI values for 11 out of 12 instance sizes, and only obtains slightly worse results than the HDDE for instance size 20×5. The MA performs much better than the IG, DPSO, HPSO, $\mathrm{HGA_W}$, $\mathrm{HGA_{T1}}$, and $\mathrm{HGA_{T2}}$ for all of the instance sizes. The HDDE ranks second with slightly better overall RPI values than the $\mathrm{HGA_Z}$ which produces the third lowest RPI value of 1.07%. The HPSO surpasses the DPSO and ranks fourth. The worst two performing algorithms are still the $\mathrm{HGA_{T1}}$ and $\mathrm{HGA_{T2}}$.

For the CPU time $\rho = 30$, we again find from Table VII that presented MA beats all the other algorithms at a considerable margin. Hence it is concluded that the proposed MA is very effective, and on average, it outperforms the HDDE, IG, DPSO, HPSO, $\mathrm{HGA_W}$, $\mathrm{HGA_Z}$, $\mathrm{HGA_{T1}}$, and $\mathrm{HGA_{T2}}$ for the problem under consideration.

We further carry out a multifactor ANOVA to check whether the observed differences from Tables V–VII are indeed statistically different. In the AVONA, we compare the six best performing algorithms, i.e., HDDE, DPSO, HPSO, $\mathrm{HGA_Z}$, IG, and HGA, and consider ρ and the type of algorithm as factors. We take the $\mathrm{HGA_{T1}}$, $\mathrm{HGA_{T2}}$, and $\mathrm{HGA_W}$ out of experiments since their performances are substantially worse. Fig. 17 reports the means and 95% Tukey HSD confidence intervals of the interaction between the type of algorithms and the allowed CPU time ρ. Note that an overlapping interval denotes an insignificant difference between the plotted means. From the figure it is clear that the proposed MA performs significantly better than the other algorithms at different elapsed CPU times.

TABLE VII
COMPUTATIONAL RESULTS OF THE ALGORITHMS ($\rho = 30$)

Instances	HGA_{T1}	HGA_{T2}	HGA_Z	HGA_W	HPSO	IG	DPSO	HDDE	MA
20×5	1.30	5.71	0.03	1.43	2.14	0.40	2.50	**0.01**	0.03
20×10	0.81	4.86	0.06	1.30	1.66	0.34	2.39	0.02	**0.01**
20×20	0.54	3.91	0.03	1.10	1.40	0.24	2.41	**0.00**	0.01
50×5	8.04	18.88	0.90	4.31	2.44	2.20	3.16	0.67	**0.25**
50×10	5.82	15.86	1.16	4.66	2.83	2.05	3.21	0.43	**0.18**
50×20	3.86	13.39	1.04	4.81	3.26	1.82	3.11	0.34	**0.16**
100×5	12.51	29.19	1.41	6.56	1.60	2.25	2.00	1.74	**-0.06**
100×10	9.40	22.70	1.89	6.23	1.88	2.16	2.51	0.99	**-0.42**
100×20	6.34	17.42	2.07	4.59	2.06	1.74	2.24	0.69	**-0.35**
200×10	12.75	23.86	1.40	7.29	1.78	3.24	2.15	2.35	**-0.38**
200×20	8.57	17.30	2.14	5.29	1.93	2.31	1.96	1.65	**-0.45**
500×20	9.60	15.88	-1.01	4.50	3.40	1.83	0.54	1.43	**-2.30**
Average	6.63	15.75	0.93	4.34	2.20	1.71	2.35	0.86	**-0.28**

TABLE VIII
SOLUTIONS FOR TAILLARD'S BENCHMARK (SOLUTIONS FOUND BY THE MA ARE IN BOLD)

Instance	Best solution	Instance	Best solution	Instance	Best solution	Instance	Best solution	Instance	Best solution	Instance	Best solution
Ta01	1374	Ta21	2436	Ta41	3638	Ta61	**6143**	Ta81	**7796**	Ta101	14192
Ta02	1408	Ta22	2234	Ta42	**3486**	Ta62	6022	Ta82	**7845**	Ta102	**14876**
Ta03	1280	Ta23	2479	Ta43	**3483**	Ta63	5927	Ta83	7794	Ta103	**15057**
Ta04	1448	Ta24	2348	Ta44	3656	Ta64	**5756**	Ta84	**7797**	Ta104	**14975**
Ta05	1341	Ta25	2435	Ta45	3629	Ta65	**5957**	Ta85	**7817**	Ta105	**14733**
Ta06	1363	Ta26	2383	Ta46	**3596**	Ta66	**5812**	Ta86	**7826**	Ta106	**14861**
Ta07	1381	Ta27	2390	Ta47	**3692**	Ta67	**5989**	Ta87	**7923**	Ta107	**14988**
Ta08	1379	Ta28	2328	Ta48	**3562**	Ta68	**5856**	Ta88	**7984**	Ta108	**14926**
Ta09	1373	Ta29	2363	Ta49	**3527**	Ta69	**6066**	Ta89	**7877**	Ta109	**14885**
Ta10	1283	Ta30	2323	Ta50	**3622**	Ta70	**6142**	Ta90	**7913**	Ta110	**14921**
Ta11	1698	Ta31	**3000**	Ta51	**4479**	Ta71	**7016**	Ta91	**13348**	Ta111	**35677**
Ta12	1833	Ta32	**3199**	Ta52	4276	Ta72	**6740**	Ta92	**13242**	Ta112	**35953**
Ta13	1659	Ta33	3011	Ta53	**4261**	Ta73	**6878**	Ta93	**13318**	Ta113	**35732**
Ta14	1535	Ta34	3128	Ta54	**4366**	Ta74	**7116**	Ta94	**13290**	Ta114	**36084**
Ta15	1617	Ta35	**3162**	Ta55	**4261**	Ta75	6810	Ta95	**13247**	Ta115	**35774**
Ta16	1590	Ta36	**3166**	Ta56	4280	Ta76	**6614**	Ta96	**13079**	Ta116	**35948**
Ta17	1622	Ta37	3013	Ta57	**4304**	Ta77	**6783**	Ta97	**13517**	Ta117	**35631**
Ta18	1731	Ta38	**3067**	Ta58	**4317**	Ta78	**6790**	Ta98	**13483**	Ta118	**35943**
Ta19	1747	Ta39	2908	Ta59	**4315**	Ta79	**6981**	Ta99	**13277**	Ta119	**35658**
Ta20	1782	Ta40	**3111**	Ta60	**4413**	Ta80	**6914**	Ta100	**13325**	Ta120	**36016**

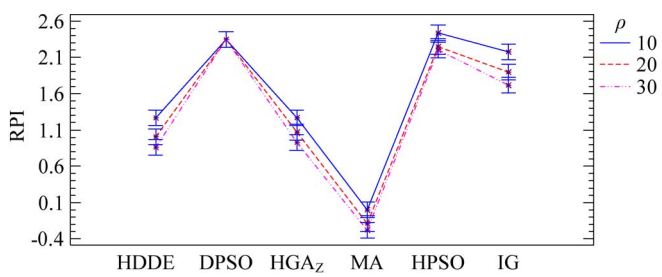

Fig. 17. The means plot and 95% Tukey HSD confidence intervals for the interaction between the algorithms and the allowed CPU time.

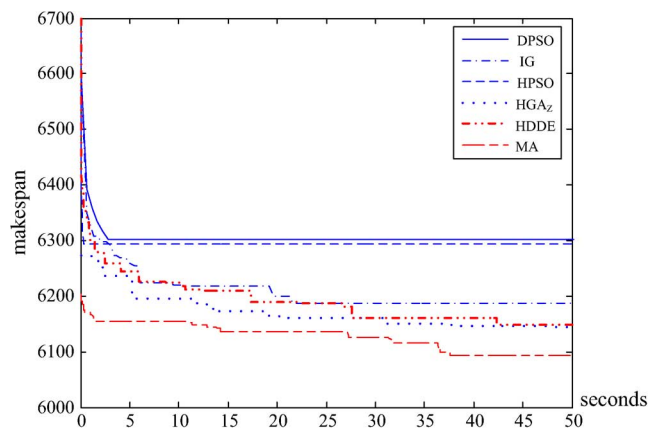

Fig. 18. The convergence curves of instance Ta61 (size 100 × 5).

In addition, the presented MA improves the upper bounds provided by Ribas *et al.* [5] in 75 out of 120 Taillard instances. We report these solutions in Table VIII, where the improved solutions by the MA are in bold and the other solutions are from Ribas *et al.* [5].

We next compare the six best performing algorithms including the MA, HDDE, HGA_Z, DPSO, HPSO, and IG by analyzing their convergence property. We run these algorithms with a very long CPU time $\rho = 200$ on the aforementioned PC. Several typical convergence curves are illustrated in Figs. 18–23 in terms of the best makespan value of each algorithm, showing that on average the MA algorithm descends much faster and reaches lower level than the HDDE, HGA_Z, DPSO, HPSO, and IG algorithms. The very similar

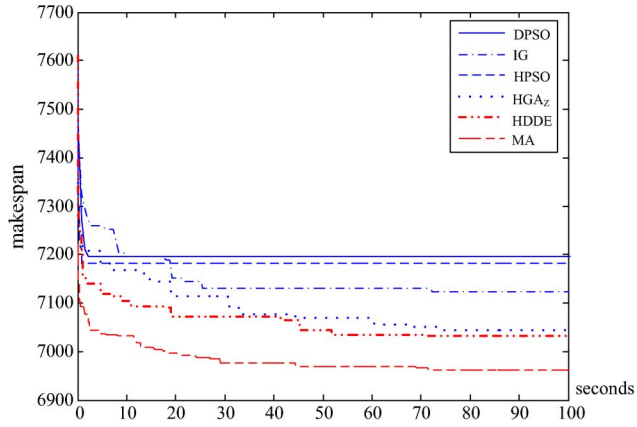

Fig. 19. The convergence curves of instance Ta71 (size 100 × 10).

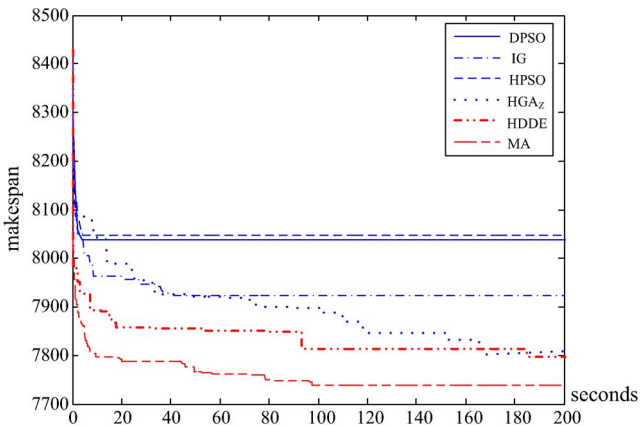

Fig. 20. The convergence curves of instance Ta81 (size 100 × 20).

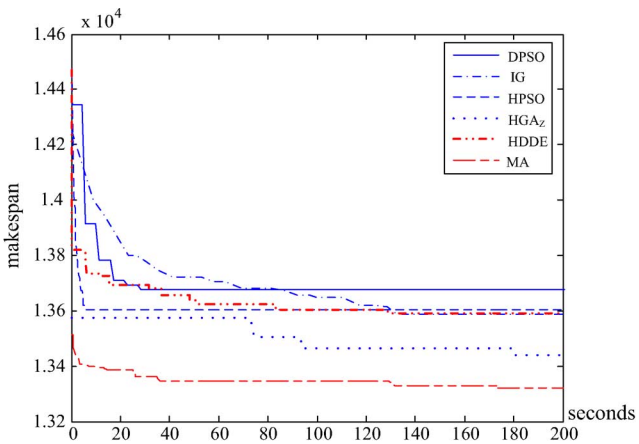

Fig. 21. The convergence curves of instance Ta91 (size 200 × 10).

convergence characteristics of the compared algorithms can be observed for other Taillard instances.

From the above comparison, it can be concluded that the proposed MA is a new state-of-the-art approach for solving the blocking flow shop scheduling problem with makesapn minimization.

The presented MA makes extensive use of some advanced techniques such as the effective population initialization, the

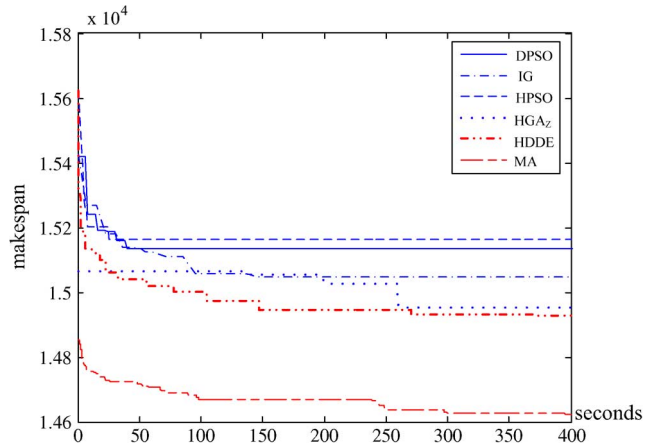

Fig. 22. The convergence curves of instance Ta101 (size 200 × 20).

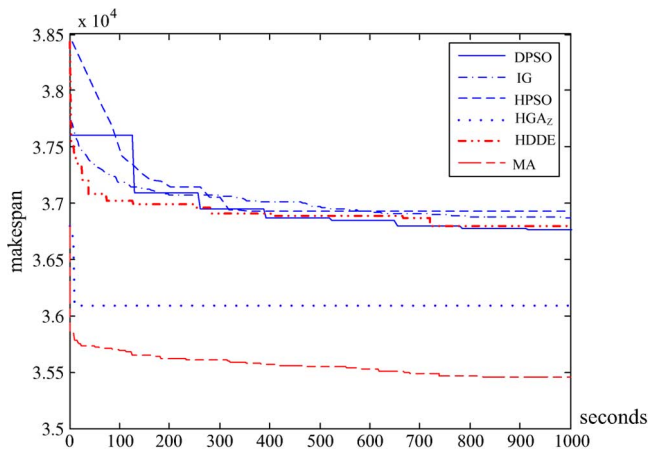

Fig. 23. The convergence curves of instance Ta111 (size 500 × 20).

crossover operator based on path relinking technique, the diversity controlling mechanism, and hybridization with local search. These techniques are in favor of the MA propagating good information in parents to offspring, maintaining diversity of population, and having higher local exploitation ability. In addition, taking advantage of the problem-specific characteristics including the PF+NEH and the speed-up technology, it makes the MA much more effective. Thus, the MA can achieve better performance than the others at several different levels. Basically, the high performance of the proposed MA is mainly due to the effective synergy of global search and local search (i.e., the balance of exploration and exploitation), and the fusion of the problem-specific characteristics. However, the proposed MA is oriented to the blocking flowshop with makespan criterion. It might not present the same high performance for solving other flowshop problems such as the no-wait flowshop, no-idle flowshop, and even the blocking flowshop with other criteria.

VI. Conclusion

This paper dealt with the blocking flowshop problem with makespan criterion, which has important applications in plastic, steel, chemical, and many other industries where no intermediate buffer exists between machines due to the technical requirements or the process characteristics. Recall that the zero buffer setting considered in this paper is different from no-wait

setting for more than two machines. To propose an effective memetic algorithm, we first presented a constructive heuristic which was utilized to generate a solution with high quality for the initial population, then used the path relinking technique substituting crossover operator to explore the search space between two selected parent individuals. Afterward, a referenced local search procedure was imbedded to stress intensification. To keep the diversity of the population, we kept each individual from the population unique, and a restart mechanism was invoked if the best solution found so far was not improved in certain generations. We obtained an effective MA by calibrating the parameters and operators by means of a design of experiments approach. Extensive comparisons were carried out for the proposed MA against the best existing methods for the considered problems, as well as with several adapted hybrid genetic algorithms originally designed for the regular flowshop. According to the computational results and statistical analysis, the proposed MA clearly outperformed all the other algorithms by a considerable margin. Furthermore, the MA improved the upper bounds provided by Ribas *et al.* [5] at 75 out of 120 Taillard instances.

Future work could focus on further exploring problem-specific characteristics, and developing more effective heuristic methods, crossover operators, and local search procedure for the blocking flowshop problem. Moreover, due to the effectiveness and simplicity of the presented MA, it could be interesting to analyze its performance using other objective criterion such as total flowtime and to generalize its application to other combinatorial optimization problems including the no-idle flwoshop, hybrid flowshop problem, no-wait job shop problem, and the flexible job shop problem [43]–[47]. Some realistic scheduling problems, such as the steelmaking and continuous casting problem from the modern steelmaking industry, seem a promising venue of research for the application of the techniques studied in this paper. Of course, each problem would need special tailoring and experimentation and this is the basis for future research.

REFERENCES

[1] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Englewood Cliffs, N.J.: Prentice-Hall, 1995.

[2] C. Rajendran and H. Ziegler, "An effective heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs," *Eur. J. Oper. Research*, vol. 103, pp. 129–138, 1997.

[3] T. Aldowaisan and A. Allahverdi, "New heuristics for m-machine no-wait flowshop to minimize total completion time," *OMEGA, Int. J. Manage. Sci.*, vol. 32, pp. 345–352, 2004.

[4] D. P. Ronconi, "A note on constructive heuristics for the flowshop problem with blocking," *Int. J. Prod. Econ.*, vol. 87, pp. 39–48, 2004.

[5] I. Ribas, R. Companys, and X. Tort-Martorell, "An iterated greedy algorithm for the flowshop scheduling with blocking," *OMEGA, Int. J. Manage. Sci.*, vol. 39, pp. 293–301, 2011.

[6] J. Grabowski and J. Pempera, "The permutation flow shop problem with blocking: A tabu search approach," *OMEGA, Int. J. Manage. Sci.*, vol. 35, pp. 302–311, 2007.

[7] H. Gong, L. Tang, and C. W. Duin, "A two-stage flow shop scheduling problem on batching machine and a discrete machine with blocking and shared setup times," *Comput. Oper. Res.*, vol. 37, pp. 960–969, 2010.

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

[9] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Ann. Discrete Math.*, vol. 5, pp. 287–362, 1979.

[10] S. S. Reddi and C. V. Rananoorthy, "On the flow-shop sequencing problem with no wait in process," *Oper. Res. Quart.*, vol. 23, pp. 323–331, 1972.

[11] P. C. Gilmore, E. L. Lawler, and D. B. Shmoys, "Well-solved special cases," in *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, E. L. Lawler, K. L. Lenstra, A. H. G. Rinooy Kan, and D. B. Shmoys, Eds. New York: Wiley, pp. 87–143.

[12] S. T. McCormich, M. L. Pinedo, S. Shenker, and B. Wolf, "Sequencing in an assembly line with blocking to minimize cycle time," *Oper. Res.*, vol. 37, pp. 925–936, 1989.

[13] R. Leisten, "Flowshop sequencing problems with limited buffer storage," *Int. J. Prod. Res.*, vol. 28, pp. 2085–2100, 1990.

[14] M. Nawaz, E. E. J. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow shop sequencing problem," *OMEGA, Int. J. Manage. Sci.*, vol. 11, pp. 91–95, 1983.

[15] D. P. Ronconi and V. A. Armentano, "Lower bounding schemes for flowshops with blocking in-process," *J. Oper. Res. Soc.*, vol. 52, pp. 1289–1297, 2001.

[16] I. N. K. Abadi, N. G. Hall, and C. Sriskandarajh, "Minimizing cycle time in a blocking flowshop," *Oper. Res.*, vol. 48, pp. 177–180, 2000.

[17] D. P. Ronconi and L. R. S. Henriques, "Some heuristic algorithms for total tardiness minimization in a flowshop with blocking," *OMEGA, Int. J. Manage. Sci.*, vol. 37, pp. 272–281, 2009.

[18] Q. K. Pan and L. Wang, "Effective heuristics for the blocking flowshop scheduling problem with makespan minimization," *OMEGA-Int. J. Manage. Sci.*, vol. 40, pp. 218–229, 2012.

[19] V. Caraffa, S. Ianes, T. P. Bagchi, and C. Sriskandarajah, "Minimizing makespan in a blocking flowshop using genetic algorithms," *Int. J. Prod. Econ.*, vol. 70, pp. 101–115, 2001.

[20] D. P. Ronconi, "A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking," *Ann. Oper. Res.*, vol. 138, pp. 53–65, 2005.

[21] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, pp. 278–285, 1993.

[22] L. Wang, L. Zhang, and D. Zheng, "An effective hybrid genetic algorithm for flow shop scheduling with limited buffers," *Comput. Oper. Res.*, vol. 33, pp. 2960–2971, 2006.

[23] B. Liu, L. Wang, and Y. Jin, "An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers," *Comput. Oper. Res.*, vol. 35, pp. 2791–2806, 2008.

[24] L. Wang, Q. K. Pan, P. N. Suganthan, W. H. Wang, and Y. M. Wang, "A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems," *Comput. Oper. Res.*, vol. 37, pp. 509–520, 2010.

[25] X. Wang and L. Tang, "A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking," *Appl. Soft Comput.*, vol. 12, pp. 652–662, 2012.

[26] L. Wang, Q. K. Pan, and M. F. Tasgetiren, "Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms," *Expert Syst. With Appl.*, vol. 37, pp. 7929–7936, 2010.

[27] X. Chen, Y. S. Ong, M. H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, pp. 591–607, 2011.

[28] Y. S. Ong, M. H. Lim, and X. Chen, "Memetic computation—Past, present & future," *IEEE Comput. Intell. Mag.*, vol. 5, pp. 24–31, 2010.

[29] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *IEEE Trans. Systems, Man, Cybern., Part C: Appl. Rev.*, vol. 28, pp. 392–403, 1998.

[30] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling," *IEEE Trans. Evol. Comput.*, vol. 7, pp. 204–223, 2003.

[31] J. Tang, M. H. Lim, and Y. S. Ong, "Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems," *Soft Computing—A Fusion of Foundations, Methodologies and Application*, vol. 19, pp. 873–888, 2007.

[32] Q. H. Nguyen, Y. S. Ong, and M. H. Lim, "A probabilistic memetic framework," *IEEE Trans. Evol. Comput.*, vol. 13, pp. 604–623, 2009.

[33] L. Y. Tseng and Y. T. Lin, "A hybrid genetic local search for the permutation flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 198, pp. 84–92, 2009.

[34] L. Y. Tseng and Y. T. Lin, "A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem," *Int. J. Prod. Econ.*, vol. 127, pp. 121–128, 2010.

[35] Y. Zhang, X. Li, and Q. Wang, "Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization," *Eur. J. Oper. Res.*, vol. 196, pp. 869–876, 2009.
[36] R. Ruiz, C. Maroto, and J. Alcaraz, "Two new robust genetic algorithms for the flowshop scheduling problem," *OMEGA, Int. J. Manage. Sci.*, vol. 34, pp. 461–476, 2006.
[37] R. Ruiz and T. Stutzle, "A simple and effective iterated greedy algorithm for the permutation flow shop scheduling problem," *Eur. J. Oper. Res.*, vol. 177, pp. 2033–2049, 2007.
[38] E. Vallada and R. Ruiz, "Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem," *OMEGA, Int. J. Manage. Sci.*, vol. 38, pp. 57–67, 2010.
[39] R. S. Farahman, R. Ruiz, and N. Boroojerdian, "New high performing heuristics for minimizing makespan in permutation flowshops," *OMEGA, Int. J. Manage. Sci.*, vol. 37, pp. 331–345, 2009.
[40] F. Glover and M. Laguna, *Tabu Search*. Boston, MA: Kluwer Academic Publishers, 1997.
[41] X. Li, Q. Wang, and C. Wu, "Efficient composite heuristics for total flowtime minimization in permutation flow shops," *OMEGA, Int. J. Manage. Sci.*, vol. 37, pp. 155–164, 2009.
[42] E. Taillard, "Some efficient heuristic methods for the flow shop sequencing problems," *Eur. J. Oper. Res.*, vol. 47, pp. 65–74, 1990.
[43] M. F. Tasgetiren, Y. C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem," *Eur. J. Oper. Res.*, vol. 177, pp. 1930–1947, 2007.
[44] B. B. Li and L. Wang, "A hybrid quantum-inspired genetic algorithm for multiobjective flow shop scheduling," *IEEE Trans. Syst., Man, Cybern.—Part B: Cybern.*, vol. 37, pp. 576–591, 2007.
[45] J. Zhu and X. Li, "An effective meta-heuristic for no-wait job shops to minimize makespan," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, pp. 189–198, 2012.
[46] N. B. Ho and J. C. Tay, "Solving multiple-objective flexible job shop problems by evolution and local search," *IEEE Trans. Syst., Man, Cybern.—Part C: Appl. Rev.*, vol. 38, pp. 674–685, 2008.
[47] K. Gokbayrak and O. Selvi, "Constrained optimal hybrid control of a flow shop system," *IEEE Trans. Autom. Control*, vol. 52, pp. 2270–2281, 2007.

Quan-ke Pan received the B.Sc. and Ph.D. degrees from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1993 and 2003, respectively.

Since 2003, he has been with School of Computer Science Department, Liaocheng University, where he became a Full Professor in 2006. He has been with the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, since 2011. He has authored more than 100 refereed papers. His current research interests include intelligent optimization and scheduling.

Ling Wang received the B.Sc. and Ph.D. degrees from Tsinghua University, Beijing, China, in 1995 and 1999, respectively.

Since 1999, he has been with the Department of Automation, Tsinghua University, where he became a Full Professor in 2008. He has authored five academic books and more than 200 refereed papers. His current research interests include intelligent optimization and scheduling.

Dr. Wang won the Outstanding Paper Award at the International Conference on Machine Learning and Cybernetics (ICMLC'02) in 2002, the Best Paper Award in the International Conference on Intelligent Computing (ICIC'11) in 2011, and the Top Cited Article Award by *Engineering Applications of Artificial Intelligence* (Elsevier). He also was the recipient of Science and Technology Award of Beijing City in 2008, and the National Natural Science Award (First Place in 2003, and Second Place in 2007) nominated by the Ministry of Education (MOE) of China. He became the Rising Star of Science and Technology of Beijing City in 2004, the Academic Young Talent of Tsinghua University in 2009, and the Program for New Century Excellent Talents in University by the MOE of China in 2009. He acts as the Co-Editor-in-Chief for *The Open Operational Research Journal*, and the Editorial Board Member for several journals, including *Memetic Computing, Swarm and Evolutionary Computation, International Journal of Automation and Control*, and *International Journal of Artificial Intelligence and Soft Computing*. He also has been Reviewer for many IEEE journals like the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, the *IEEE Computational Intelligence Magazine,* the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON NEURAL NETWORKS, the IEEE TRANSACTION ON SIGNAL PROCESSING, and the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS PART B and PART C.

Hong-yan Sang received the B.S. degree from Liaocheng University, China, in 2003. She is currently working towards the Ph.D. degree at the State Key Laboratory of Digital Manufacturing Equipment & Technology, Huazhong University of Science & Technology, China.

Since 2003, she has been with the School of Mathematics Science Department, Liaocheng University, where she became a Lecturer in 2008. Her current research interests include intelligent optimization and scheduling. She has authored more than ten refereed papers.

Jun-qing Li received the M.S. degree in computer science and technology in 2004 from Shandong Economic University, Shandong, China.

Since 2004, he has been with the Department of Computer Science, Liaocheng University, where he became an Associate Professor in 2008. He has authored more than 30 refereed papers. His current research interests include intelligent optimization and scheduling.

Min Liu (M'11) received the Ph.D. degree from Tsinghua University, Beijing, China, in 1999.

He is currently a Professor with the Department of Automation, Tsinghua University, Associate Director of Automation Science and Technology Research Department of Tsinghua National Laboratory for Information Science and Technology, Director of Control and Optimization of Complex Industrial Process, Tsinghua University, Director of China National Committee for Terms in Automation Science and Technology, and Director of Intelligent Optimization Committee of China Artificial Intelligence Association. He led more than 20 important research projects including the project of the National 973 Program of China, the project of the National Science and Technology Major Project of China, the project of the National Science Fund for Distinguished Young Scholars of China, the project of the National 863 High-Tech Program of China, and so on. He has published more than 100 papers and a monograph supported by the National Defense Science and Technology Book Publishing Fund. His main research interests are in optimization scheduling of complex manufacturing process and intelligent operational optimization of complex manufacturing process or equipment.

Dr. Liu won the National Science and Technology Progress Award.