Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 436701, 14 pages
http://dx.doi.org/10.1155/2013/436701

Research Article

Scheduling Semiconductor Multihead Testers Using Metaheuristic Techniques Embedded with Lot-Specific and Configuration-Specific Information

Yi-Feng Hung,[1] Chi-Chung Wang,[1] and Gen-Han Wu[2]

[1] *Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu 300, Taiwan*
[2] *Department of Business Administration and Graduate Institute of Logistics Management, National Dong Hwa University, Hualien 974, Taiwan*

Correspondence should be addressed to Yi-Feng Hung; yifeng@ie.nthu.edu.tw

Received 16 August 2013; Revised 22 October 2013; Accepted 23 October 2013

Academic Editor: Chin-Chia Wu

In the semiconductor back-end manufacturing, the device test central processing unit (CPU) is most costly and is typically the bottleneck machine at the test plant. A multihead tester contains a CPU and several test heads, each of which can be connected to a handler that processes one lot of the same device. The residence time of a lot is closely related to the product mix on test heads, which increases the complexity of this problem. It is critical for the test scheduling problem to reduce CPU's idle time and to increase tester utilization. In this paper, a multihead tester scheduling problem is formulated as an identical parallel machine scheduling problem with the objective of minimizing makespan. A heuristic grouping method is developed to obtain a good initial solution in a short time. Three metaheuristic techniques, using lot-specific and configuration-specific information, are proposed to receive a near-optimum and are compared to traditional approaches. Computational experiments show that a tabu search with lot-specific information outperforms all other competing approaches.

1. Introduction

Semiconductor manufacturing consists of front-end and back-end manufacturing. The front-end manufacturing contains the processes of wafer fabrication and wafer probe. In wafer fabrication, a pattern of circuitry is imprinted onto the surfaces of a wafer, on which hundreds of dice are fabricated, and dice are individually tested in the process of wafer probe. In recent years, vast research of scheduling has been done in the field of wafer fabrication [1–8]. In [9, 10] and the wafer probe process was scheduled as a mathematical model to minimize the makespan. Lot and process information are used to develop scheduling algorithm. The back-end manufacturing comprises the processes of assembly and final testing. In an assembly process, dice are sealed into *packaged devices* (or briefly, devices), which are then tested in a final testing process in order to ensure the functionality of the products before delivery to customers. In [11], the flow of the final test is formulated as a job shop model with various resource limitations. An assignment algorithm is developed to obtain the machine configuration of each job and allot specific resources. Moreover, a genetic algorithm called GASFTP is used to obtain a near-optimal schedule in real settings. In [12–15], back-end operations such as assembly and final test are considered as serial production stages or workstations and heuristic techniques such as ant colony optimization [12], multistep reinforcement learning algorithm [13], reactive greedy randomized adaptive search procedure (GRASP) [14] and dynamic machine/lot prioritization [15] are introduced. In this paper, we focus on the scheduling problem of multihead testers in order to solve the bottleneck of the final testing process.

A device is the smallest unit in the back-end manufacturing and a lot is a group of the same devices transported together. However, in a lot, each device is processed individually on a handler, through which each device is tested by a central processing unit (CPU). *Device testing time* (or briefly, testing time), which varies with device types, is the CPU's

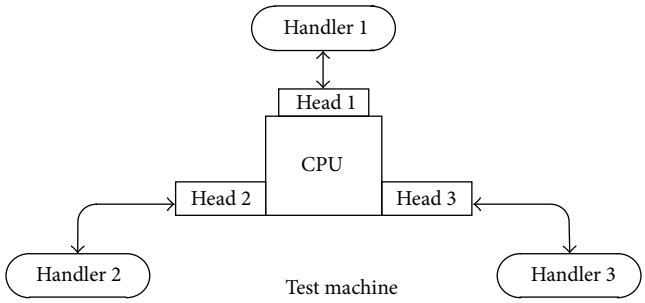

Figure 1: A three-head tester's CPU.

duration of inspecting a device's functionality. *Device* handling time (or briefly, handling time) is the duration that a handler takes to unload a finished device from a testing location, to load a waiting device onto the testing location, and, in some cases, to perform a temperature treatment.

Multihead tester operations were firstly introduced in detail to present the formulas for determining the residence times of lots [16]. A multihead tester contains a CPU and several test heads, each of which can be connected with a handler. At any particular time, a handler can only process one lot of the same device. However, a tester's CPU, as shown in Figure 1, is often connected to more than one handler. A tester can simultaneously place several lots to reduce CPU's idle time and to increase tester utilization. A CPU tests a device of a lot on each test head sequentially but skips the test head that is idle or is undergoing the handling time. The residence time of a lot, which is how long a lot stays in a test head, includes the testing time, the handling time, and the idle time of total devices in this lot. The cost of a handler is far less than the cost of a tester's CPU; hence, a tester normally carries several handlers to avoid the idle time of a CPU [16]. Thus, it becomes important to reduce CPU's idle time and to increase tester efficiency. We formulate this scheduling problem as an identical parallel machine scheduling problem with the objective of minimizing makespan ($C_{\max}$). The main difference of scheduling problems between a traditional parallel machine and a tester is the nature of residence times. In a tester scheduling problem, the residence time of a lot depends on the product mix on all the heads of a tester, which increases the complexity of this problem.

A *configuration* g is the status that a set of different lots, P_g, is simultaneously placed on the heads of a tester during a specific period of time, b_g. The configuration will be changed if the following conditions occur: (5) a lot finishes on a test head and a changeover then needs to be performed for the next lot. (2) A changeover is completed and then the next lot will start on the test head. Both conditions will affect the residence times of the lots placed on the other heads of the tester. Figure 2 is the Gantt charts of the three cases of a configuration in a three-head tester based on a microview of device types. The three test heads are connected to the three handlers, each of which processes the same devices in a lot. The time t_i is the unit testing time of a device type i and the time h_i is the unit handling time of a device type i. In each device test cycle, the CPU tests a device on each test head sequentially but skips the test head that is idle or is undergoing the handling time.

The device in a test head needs to wait until the testing completion of the CPU on another test head if a device's handling time is less than the sum of the device's testing times on the other heads. For example, under case 1, the test head 1 first tests the device type 1 with in a testing time t_1 and then prepares for the next device with a handling time h_1. After that, the test head 1 needs to wait until the testing completion of the CPU on test head 3. The CPU then switches to test on test head 1. Therefore, the sum of testing times is larger than the maximum of device testing and handling time among all the heads; that is, $\sum_{i=1}^{3} t_i > \max_{i=1,2,3}(t_i + h_i)$. On the other hand, the CPU needs to wait if the CPU turns back to a test head, which is still under the process of the handling time for the next device. That is to say, the CPU will be forced to be idle if a device's handling time is larger than the sum of the device's testing times on the other heads. For example, under case 2, the CPU needs to wait for handler 1 finishing the preparation for the next device after the CPU finishing the test of device type 1, 2, and 3. Thus, in such a case, the CPU is forced to be idle and the sum of testing times is less than the maximum of device testing and handling time among all the heads; that is, $\sum_{i=1}^{3} t_i < \max_{i=1,2,3}(t_i + h_i) = t_1 + h_1$. Both cases, including the idle test bed under case 1 and the idle CPU under case 2, prolong the residence times of the lots and cause a longer makespan. However, as shown in case 3, the most ideal case occurs when neither of the CPU and the test bed needs to wait for a period of time; that is, $\sum_{i=1}^{3} t_i = \max_{i=1,2,3}(t_i + h_i) = t_1 + h_1$.

In this paper, the testing parameters of any device type at any test head are assumed to be the same; that is, all the test heads are identical. At test plants, a certain device may only be tested on a given test head. The cost of a CPU is much more expensive than the cost of a handler. Hence, the limitation of handler availability is not considered. We herein assume that each running test in any test head cannot be terminated; that is, preemption and breakdown are not allowed. Moreover, between two consecutive lots, different operating parameters of test handlers exist and the test head and handler need to be reinstalled. This changeover time includes the installation time, loading time, and unloading time. For the sake of the simplicity, the changeover time between two consecutive lots is assumed to be independent of the device types involved and is a constant value.

This scheduling problem of multihead testers is similar to an identical parallel machine scheduling problem, which has been proved to be a NP-hard problem [17]. In this paper, this multihead tester scheduling problem is solved by a more complicated parallel machine scheduling problem because the residence time of a lot on a test head is dependent on the lots being processed on the other test heads. Due to the problem difficulty, most researchers have used search algorithms and an advanced initial solution to solve parallel machine scheduling problems [18–25]. As a result of the rapid development in personal computer speed, it would be worthwhile to invest in a relatively cheaper computer for solving this scheduling problem of a much more expensive

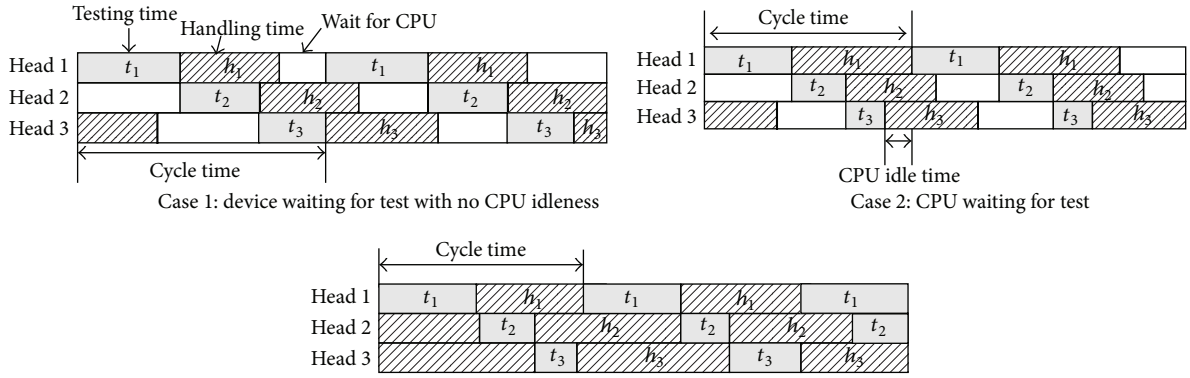

Figure 2: Three cases of a configuration in a three-head tester.

tester and to increase its productivity. Simulated annealing was first proposed in [26] and was developed initially to emulate cooling and recrystallization process. Reference [27] reviewed this search technique. Tabu search was first proposed in [28] and a review of tabu search was introduced in [29]. Genetic algorithm was first introduced in [30] and relevant review were given in [31, 32]. This problem is similar to parallel machine scheduling problems but is more complicated in that the residence time of a lot on a test head is dependent on the lots being placed on the other test heads. Recent research on parallel machine scheduling problems solved by simulated annealing, tabu search, and genetic algorithm can be seen in [33–40].

In this paper, three metaheuristic techniques, including simulated annealing, tabu search, and genetic algorithm embedded with lot-specific and configuration-specific information, are proposed and are compared to the traditional approaches. In order to tackle such a complex problem, two sets of measures, developed by the concepts of the lot information and the configuration information, are used in heuristic rules for generating new solutions. These heuristic rules used in this paper are embedded in traditional approaches. Furthermore, since a better initial solution normally helps locate a good solution faster, a heuristic rule, called a grouping heuristic method, is proposed to efficiently generate a good initial solution.

The remainder of the paper is organized as follows. In Section 2, a procedure of evaluating a schedule is introduced and a heuristic grouping method is developed to locate a good initial solution. Three metaheuristic techniques, including simulated annealing, tabu search, and genetic algorithm embedded with lot-specific and configuration-specific information, are compared to traditional approaches. This methodology is also extended to the discussion on the scheduling problem of multiple multihead testers. Computing results are presented in Section 3. Finally, we conclude with a brief discussion of the results in Section 4.

2. Methodology

In the overall design of the methodology, Section 2.1 presents the procedure for computing the makespan of a given schedule (solution). A two-phase methodology is presented in our proposed approach. The first phase uses the heuristic grouping method, outlined in Section 2.2, to obtain a good initial solution. Three metaheuristic techniques introduced from Sections 2.3 to 2.5, including simulated annealing, tabu search, and genetic algorithm, are developed to improve the initial solution in the second phase. In addition, limited running time is executed as the stopping criterion.

The following notations are used for a single multihead tester scheduling problem discussed from Sections 2.1 to 2.5. Section 2.6 discusses the extensions to multiple multihead tester scheduling problems.

Indices

i: lot i; $i = \{1, 2, \ldots, \mathrm{I}\}$.

j: test head j; $j = \{1, 2, \ldots, J\}$.

k: the sequence number k of a lot on a test head.

g: configuration g, the status that a set of different lots is simultaneously placed on the heads of a tester during a specific period of time.

Parameters

t_i: the testing time of a device in a lot i.

h_i: the handling time of a device in a lot i, which includes an unloading time, a loading time, and the time for temperature treatment.

d_i: the number of devices in a lot i.

Sets

G_i: the order sequence of the configurations in which a lot i is processed.

P_g: the set of lots processed in a configuration g.

Variables

σ_{jk}: the index number of the lot, that is, the kth lot processed on a test head j.

Table 1: Information about configurations.

g	1	2	3
Ending events	Lot 4 completed	Lot 2 completed	Head 3 changeover completed
P_g	$1, 2, 4$	$1, 2$	1
c_g	$\max\{t_1 + t_2 + t_4, \max(t_1 + h_1, t_2 + h_2, t_4 + h_4)\}$	$\max\{t_1 + t_2, \max(t_1 + h_1, t_2 + h_2)\}$	$\max\{t_1, t_1 + h_1\} = t_1 + h_1$
f_g	$\max(t_1 + h_1, t_2 + h_2, t_4 + h_4) - (t_1 + t_2 + t_4)$	$\max(t_1 + h_1, t_2 + h_2) - (t_1 + t_2)$	$(t_1 + h_1) - t_1 = h_1$
u_g	n_4	$n_2 - n_4$	b_3/c_3
b_g	$c_1 \cdot u_1$	$c_2 \cdot u_2$	$e_3 - e_2$
e_g	b_1	$b_2 + e_1$	e_1+ changeover time

s: a solution, $s = \{\sigma_{jk} \mid k = 1, \ldots, n_j, j = 1, \ldots, J\}$.

$n_{\sigma_{jk}}$: the number of devices in a lot σ_{jk} scheduled on a test head j.

u_g: the number of devices which are tested per head during a configuration g.

c_g: the cycle time per device test cycle in a configuration g, $c_g = \max\{\sum_{i \in P_g} t_i, \max_{i \in P_g}\{t_i + h_i\}\}$.

b_g: the period length of a configuration g, $b_g = c_g \cdot u_g$.

e_g: the end time of a configuration g, $e_g = e_{g-1} + b_g$.

r_i: the residence time of a lot i, $r_i = \sum_{g \in G_i} u_g \cdot c_g$, where $\sum_{g \in G_i} u_g = d_i$.

f_g: the CPU idleness in a configuration g, $f_g = \max_{i \in P_g}\{t_i + h_i\} - \sum_{i \in P_g} t_i$.

It is worth noting that the CPU idleness (f_g) in a configuration g is allowed to be negative. In such a case, the absolute value of CPU idleness (f_g) is the minimum device waiting time among all the lots in a configuration g as shown in case 1 of Figure 2. Hence, the CPU idle time per device test cycle in a configuration g is equal to $\max\{0, f_g\}$ and the device waiting time per device test cycle of a lot i in a configuration g is equal to $\max\{0, \sum_{i' \in P_g} t_{i'} - (t_i + h_i)\}$. An example schedule for a three-head tester, as shown in Figure 3, represents the residence times and the order sequence of different lots and the changeover times between two lots separately in three test heads based on a macroview of lot types. Under this schedule, the set G_i of a lot i is $G_1 = \{1, 2, 3, 4\}$, $G_2 = \{1, 2\}$, and $G_4 = \{1\}$ whereas the information of configurations 1, 2, and 3 is computed in Table 1.

2.1. Computing Procedure for Evaluating a Schedule. As discussed above, the residence time of a lot depends on the device types on the other heads at the same tester because the device types on the other heads will affect the device idle times of a lot. Thus, a procedure is needed to evaluate the makespan of a particular schedule, which is obtained by one of the methods outlined from Sections 2.2 to 2.5. In this procedure, the following variables are mentioned.

t_now: current time.

t_next$_j$: the next event time of a test head j, which will change the configuration.

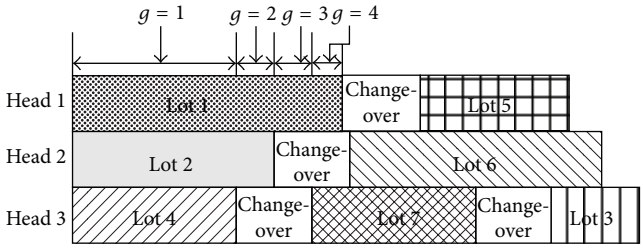

Figure 3: An example schedule of lot residence times for a three-head tester.

remain$_i$: the number of devices remaining untested in a lot i.

event$_j$: the next event type of a test head j. Event A represents the event when a lot is completed and Event B represents the event when a changeover is completed.

$i(j)$: the current index number of a lot i on a test head j.

After a solution (schedule) s is obtained, the following procedure, which is similar to a discrete event simulation, is applied to compute its makespan.

Step 1. Let remain$_i = d_i$, for all i.

Step 2. Let t_now $= 0$. Schedule the first lot of each test head.

Let event$_j = A$, for all j and $g = 1$. Determine c_g based on the device types on all test heads. Compute t_next$_j = c_g \cdot d_{i(j)}$, for all j.

Step 3. Find j', such that t_next$_{j'} = \min_j(t$_next$_j)$.

Compute remain$_{i(j)}$ = remain$_{i(j)} - ((t$_next$_{j'} - t$_now$)/c_g)$, for all j such that event$_j = A$. Let t_now $= t$_next$_j$. Let $g = g + 1$.

If (event$_{j'} = A$), go to Step 5; if (event$_{j'} = B$), go to Step 4.

Step 4. If no lot is remaining on head j', select the waiting lot with the largest number of devices from the other heads and reassign the lot to head j'.

Schedule the next lot onto head j'. Let event$_{j'} = A$. Determine c_g based on a new configuration.

Compute t_next$_j = c_g \times$ remain$_{i(j)}$, for all j, such that event$_j = A$. Go to Step 3.

Step 5. Determine c_g based on a new configuration. Let event$_{j'} = B$.

Compute t_next$_{j'} = t$_next$_{j'}$ + changeover time.

Compute $t_{\text{next}_j} = c_g \times \text{remain}_{i(j)}$, for all j, such that $\text{event}_j = A$.

If all lots are finished, the makespan of the schedule, t_now, is obtained and the procedure is stopped ; otherwise, go to Step 3.

2.2. Heuristic Grouping Method for Initial Solutions. Under a configuration g, the CPU idleness is computed by $f_g = \max_{i\in P_g}(t_i + h_i) - \sum_{i\in P_g} t_i$. If $\max_{i\in P_g}(t_i + h_i) > \sum_{i\in P_g} t_i$ (i.e., $f_g > 0$), the CPU will be idle for a time period of f_g per device test cycle in a configuration g. A lot with a large handling time (h_i) is likely to make f_g large and thus cause CPU to be idle. Therefore, the lots with large device testing times should be assigned to the other heads in order to make $\sum_{i\in P_g} t_i$ larger. If lots can be assigned to the other heads and the value $\max_{i\in P_g}(t_i + h_i)$ is larger than $\sum_{i\in P_g} t_i$, the CPU idleness can be avoided.

On the other hand, if $\max_{i\in P_g}(t_i + h_i) < \sum_{i\in P_g} t_i$ (i.e., $f_g < 0$), at least a device needs to wait for the CPU. Further, the minimum device waiting time per device test cycle among all heads is $|f_g|$. If the value $\sum_{i\in P_g} t_i$ is too large, the devices in a configuration g will need to wait for a long time and largely prolong the residence time of lots in a configuration g. Thus, under the condition of $\max_{i\in P_g}(t_i + h_i) < \sum_{i\in P_g} t_i$, the proposed heuristic grouping method is implemented to make $\sum_{i\in P_g} t_i$ as close to $\max_{i\in P_g}(t_i + h_i)$ as possible. The notations and the procedure of the heuristic grouping method are as follows. Let

T: the set of all lots, which are sorted by a descending order of device testing times;

H: the set of all lots, which are sorted by a descending order of device handling times;

x_i: the ith element in set T;

P: the set of the lots, which are currently under process and were selected from set T;

z: the lot which is currently under process and was selected from set H.

The Procedure Is as Follows.

Step 1. Sort the lots based on their device testing times and device handling times in order to generate the set T and the set H, respectively.

Step 2. Let z = the first element in H and remove z from both T and H.

Step 3. Find the largest k, such that $\sum_{i=k}^{k+J-2} t_{x_i} > h_z$, and put $x_k, x_{k+1}, \ldots, x_{k+J-2}$ into P. If no such k can be found, put x_1, $x_2, \ldots, x_{J-1}$ into P and remove the elements in P from both T and H.

Step 4. Schedule the selected lots (lot z and the lots in P) onto the test heads. If T and H are empty, go to Step 7; otherwise, go to Step 5.

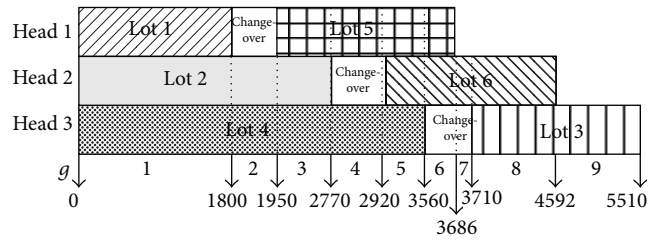

Figure 4: The solution obtained by the heuristic grouping method.

Table 2: Lot information (changeover time = 150).

Lot number	Testing time	Handling time	Lot size
1	6	11	100
2	7	9	150
3	4	5	200
4	5	8	200
5	8	6	100
6	3	4	150

Step 5. If z is the first lot to finish under the current configuration, let z = the first element in H, remove z from both T and H, and go to Step 4.

Step 6. If a lot in P is the first to finish under the current configuration, remove this lot from P. Find the largest k, such that $(t_{x_k} + \sum_{i\in P} t_i) > h_z$. Put x_k into P and remove x_k from both T and H. If no such k can be found, put x_1 into P and remove x_1 from T and H. Go to Step 4.

Step 7. Stop the procedure.

For example, Table 2 is the information of 6 lots waiting for test in a three-head tester. The steps of the heuristic grouping method are organized in Table 3. Figure 4 is the solution solved by the heuristic grouping method. The information of the lots and configurations is presented in Tables 4 and 5. Several examples of the calculation in Tables 4 and 5 are described as follows.

Configuration 1 ($g = 1$)

$P_1 = \{1, 2, 4\}$;

$c_1 = \max\{t_1 + t_2 + t_4, \max(t_1 + h_1, t_2 + h_2, t_4 + h_4)\} = \max\{6 + 7 + 5, \max(6 + 11, 7 + 9, 5 + 8)\} = 18$;

$f_1 = \max(t_1 + h_1, t_2 + h_2, t_4 + h_4) - (t_1 + t_2 + t_4) = \max(6 + 11, 7 + 9, 5 + 8) - (6 + 7 + 5) = -1$;

$u_1 = \min(d_1, d_2, d_4) = \min\ (100, 150, 200) = 100$;

$e_1 = b_1 = c_1 \times u_1 = 18\ \times\ 100 = 1800$.

Configuration 2 ($g = 2$)

$P_2 = \{2, 4\}$;

$c_2 = \max\{t_2 + t_4, \max(t_2 + h_2, t_4 + h_4)\} = \max\{7 + 5, \max(7 + 9, 5 + 8)\} = 16$;

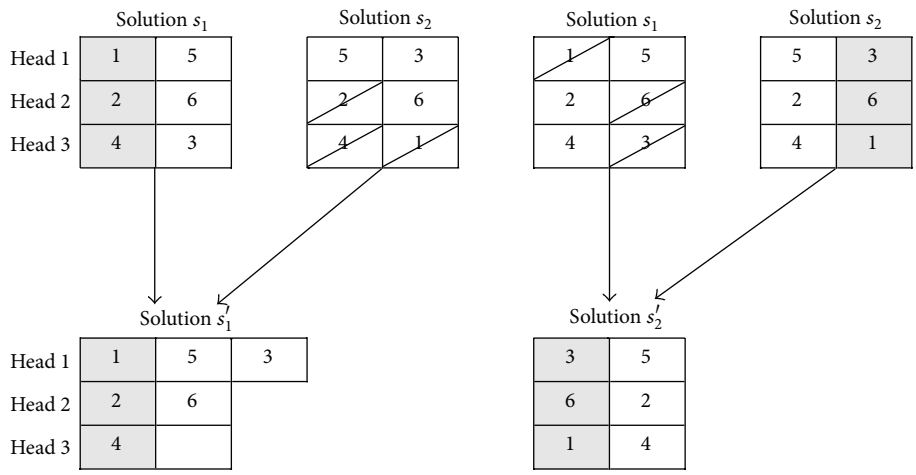

Figure 5: An example of crossover operator using configuration information.

Table 3: Steps of the heuristic grouping method.

Step	T	H	1st in H	z	P
1	5, 2, 1, 4, 3, 6	1, 2, 4, 5, 3, 6	1	ϕ	ϕ
	Description: generate T and H				
2	5, 2, 4, 3, 6	2, 4, 5, 3, 6	2	1	ϕ
	Description: put lot 1 onto the test head				
3	5, 3, 6	5, 3, 6	5	1	2, 4
	Description: select lots 2 and 4 from T and put onto the test head				
4	3, 6	3, 6	6	5	2, 4
	Description: lot 1 finishes testing; then select lot 5 for testing from T.				
5	3	3	3	5	4, 6
	Description: lot 2 finishes testing; then select lot 6 for testing from T.				
6	ϕ	ϕ	ϕ	5	3, 6
	Description: Lot 4 finishes testing; then select lot 3 for testing from T.				

Table 4: Lot information.

i	G_i	r_i
1	1	1800
2	1, 2, 3	2770
3	8, 9	1800
4	1, 2, 3, 4, 5	3560
5	3, 4, 5, 6	1736
6	5, 6, 7, 8	1672

Table 5: Configuration information.

g	c_g	f_g	P_g	u_g	b_g	e_g
1	18	−1	1, 2, 4	100	1800	1800
2	16	4	2, 4	9	150	1950
3	20	−4	5, 2, 4	41	820	2770
4	14	1	5, 4	10	150	2920
5	16	−2	5, 6, 4	40	640	3560
6	14	3	5, 6	9	126	3686
7	7	4	6	3	24	3710
8	9	2	6, 3	98	882	4592
9	9	5	3	102	921	5234

$f_2 = \max(t_2 + h_2, t_4 + h_4) - (t_2 + t_4) = \max(7 + 9, 5 + 8) - (7 + 5) = 4$;

$e_2 = e_1 + \text{changeover time} = 1800 + 150 = 1950$;

$b_2 = e_2 - e_1 = 1950 - 1800 = 150$;

$u_2 = b_2/c_2 = 150/16 = 9.375 \cong 9$.

Configuration 3 ($g = 3$)

$P_3 = \{5, 2, 4\}$;

$c_3 = \max\{t_5 + t_2 + t_4, \max(t_5 + h_5, t_2 + h_2, t_4 + h_4)\} = \max\{8 + 7 + 5, \max(8 + 6, 7 + 9, 5 + 8)\} = 20$;

$f_3 = \max(t_5 + h_5, t_2 + h_2, t_4 + h_4) - (t_5 + t_2 + t_4) = \max(8 + 6, 7 + 9, 5 + 8) - (8 + 7 + 5) = -4$;

$u_3 = \min(n_5, n_2, n_4) = \min(d_5, d_2 - u_1 - u_2, d_4 - u_1 - u_2) = \min(100, 150 - 100 - 9, 200 - 100 - 9) = 41$;

$b_2 = c_3 \times u_3 = 20 \times 41 = 820$;

$e_3 = e_2 + b_3 = 1950 + 820 = 2770$.

Lot 1 only contains configuration 1 but lot 2 contains configurations 1, 2, and 3. Hence, $r_1 = b_1 = c_1 \times u_1 = 18 \times 100 = 1800$ and $r_2 = b_1 + b_2 + b_3 = c_1 \times u_1 + b_2 + c_3 \times u_3 = 18 \times 100 + 150 + 20 \times 41 = 2770$. The schedule constructed here will be further improved by the simulated annealing, tabu search, and genetic algorithm outlined from Sections 2.3 to 2.5.

2.3. Simulated Annealing. Three simulated annealing methods developed in this study, each of which uses a different rule to generate neighbor solutions, are traditional simulated annealing (TSA), simulated annealing using configuration information (HSA-1), and simulated annealing using lot information (HSA-2). HSA-1 and HSA-2 are simulated annealing methods within which heuristic rules are used to guide the stochastic selection of neighbor solutions.

2.3.1. Traditional Simulated Annealing (TSA). An insertion rule and an exchange rule are developed to generate neighbor solutions for the use of simulated annealing [41]. In our TSA, these two rules are randomly chosen with equal probability of generating neighbor solutions. If the insertion rule is chosen, it randomly selects a lot and then randomly selects a slot for the insertion of the selected lot. If the exchanging rule is chosen, it randomly selects two lots and then swaps their positions.

2.3.2. Simulated Annealing Using Configuration Information (HSA-1). Let

$g_{\max}$: under current solution, the configuration g with the maximum f_g;

$g_{\min}$: under current solution, the configuration g with the minimum f_g.

Configuration $g_{\max}$ has the longest CPU idle time due to long handling times, whereas configuration $g_{\min}$ has the largest minimum device wait time among all heads due to long testing times. Based on the previous discussion in Section 2.2, a long handling time should be matched with long testing times to avoid waiting and reduce processing time. Thus, the lots in these two configurations are good candidates for exchanging positions. The exchange rule in HSA-1 randomly selects one lot from each of these two configurations and then exchanges them, whereas the insertion rule in HSA-1 randomly selects one lot from the set of the lots in $g_{\max}$ and then inserts the selected lot into a random slot.

2.3.3. Simulated Annealing Using Lot Information (HSA-2). Let

w_i: the waiting time measure for lot i in a particular solution; $w_i = (r_i/d_i) - (t_i + h_i)$, for all i;

p_i: the probability of lot i being selected; $p_i = (w_i / \sum_{i'} w_{i'})$, for all i.

In computing w_i, r_i/d_i is the average processing time per device in lot i under a current solution and $(t_i + h_i)$ is the minimum processing time per device. A larger w_i value indicates that the devices in lot i wait longer for CPU; thus, lot i seems to be a good candidate for a change. Thus, we calculate the probability of lot i being selected (p_i) by using the above formula.

The inserting method randomly selects a lot using the above probabilities and then randomly inserts it into a slot. The exchanging method uses the above probabilities to randomly select two lots and then swap their positions.

2.4. Tabu Search. A simple tabu search is successfully used to solve a certain class of NP-hard problems. However, for some problems, their neighborhood is too large or a neighbor solution is too expensive to evaluate. Evaluating the makespan of a given schedule in our problem is not trivial and requires a simulation-like procedure. Thus, a simple tabu search method may not be suitable. Certain modifications, such as neighborhood reduction or candidate list strategy, may be necessary to improve the efficiency of neighborhood examination in a tabu search [29].

Using a similar idea of reducing neighborhood, we developed three heuristic rules to select a better subset of all neighbors for evaluation.

2.4.1. Traditional Tabu Search (TTS). TTS implements a traditional simple tabu search. It evaluates each possible insertion of each lot. Therefore, the number of neighbor solutions in a tabu iteration is approximately equal to the square of the number of lots. Thus, it will take a long time for an iteration in a large problem.

2.4.2. Tabu Search with Deterministic Selection Using Configuration Information (HTS-1). According to the previous discussion, lots in $g_{\max}$ and lots in $g_{\min}$ are good candidates to swap positions; thus, HTS-1 examines the neighbor solution by each possible swap between these two configurations. Roughly speaking, if $g_{\max}$ has α lots and $g_{\min}$ has β lots, then there are $(\alpha \cdot \beta)$ neighbor solutions. Those methods deterministically select the two configurations $g_{\max}$ and $g_{\min}$.

2.4.3. Tabu Search with Stochastic Selection Using Configuration Information (HTS-2). Let

nf_g: the nonnegative CPU idleness measure for configuration g in a current solution;

q_g: the probability of configuration g, which will be used for selection.

$rg_{\max}$: a randomly selected configuration based on large CPU idleness;

$rg_{\min}$: a randomly selected configuration based on small CPU idleness.

This method randomly selects two configurations, then swaps the lots between these two configurations. First, compute

$$nf_g = f_g - \min_{g'} f_{g'}, \quad \forall g, \tag{1}$$

then compute the probability

$$q_g = \frac{nf_g}{\sum_{g'} nf_{g'}}, \quad \forall g. \tag{2}$$

In this way, a configuration g with a larger f_g will have higher probability q_g. Then, randomly select a configuration $rg_{\max}$ based on q_g.

Compute new

$$nf_g = \max_{g'} f_{g'} - f_g, \quad \forall g, \tag{3}$$

and then compute new

$$q_g = \frac{nf_g}{\sum_{g'} nf_{g'}}, \quad \forall g. \tag{4}$$

In this way, a configuration g with a smaller f_g will have higher probability q_g. Then, randomly select a configuration $rg_{\min}$ based on q_g.

In $rg_{\max}$, it is likely that the CPU waits due to long handling times, whereas in $rg_{\min}$, it is likely that devices wait due to long testing times. Based on the previous discussion, a long handling time should match with long testing times to reduce processing time. Thus, HTS-2 examines each possible lot swap between the two configurations $rg_{\max}$ and $rg_{\min}$.

2.4.4. Tabu Search Using Lot Information (HTS-3). HTS-3 randomly selects J (the number of test heads) lots based on the probabilities p_i. The higher p_i, is the worse position lot i has. Thus, HTS-3 makes it more likely to be selected for changing positions. Then, HTS-3 examines the neighbor solution by exchanging each pair of the selected lots; that is, there are $C(_2^J) = J!/(J-2)!2!$ neighbor solutions in a tabu iteration.

2.5. Genetic Algorithm. Reference [42] pointed out that the performance of genetic algorithm is determined by its crossover operator. An effective crossover is necessary for a successful genetic evolution. We use the heuristic measures introduced above and the *order crossover* operator proposed by [43] to design crossover operators and to preserve good genes from parent chromosomes.

We follow the method used by [41] to form the first generation. That is, the initial solution produced by the heuristic grouping method is duplicated once; then the other chromosomes in the first generation are randomly generated.

2.5.1. Genetic Algorithm Using Deterministic Selection to Preserve a Good Configuration (HGA-1). Neither a large positive f_g nor a small negative f_g is desirable. Let

$\tilde{f}_g$: the absolute value of f_g in a particular solution; $\tilde{f}_g = |f_g|$, for all g;

m_g: the number of lots under processing in configuration g;

$g \in s$: the configuration g belongs to chromosome s.

Let s_1 and s_2 be the two selected parent chromosomes for crossover. $\tilde{f}_g$ is the distance to the most ideal case; thus, we find the configuration g_1 such that $m_{g_1} = J$ and $f_{g_1} = \min_{g' \in s_1} \tilde{f}_{g'}$. Similarly, we obtain g_2 from solution s_2. Then, we put all the lots in g_1 at the first position of each test head in a child chromosome s_1'. In this way, we are maximizing the duration of using configuration g_1, since g_1 has the minimum distance to ideal case s_1. After that, we delete the lots in g_1 from s_2 and then append the remaining schedule of s_2 to s_1'. We can generate s_2' similarly. An example is shown in Figure 5. We assume that the best configuration in solution s_1 consists of lots 1, 2, and 4; thus, the configuration is preserved in the first positions of the test heads in s_1' to maximize its duration. Then, lots 1, 2, and 4 are deleted from s_2. The lots in the remaining schedule of s_2 are appended to the corresponding test head in s_1'. Assume that the best configuration in s_2 consists of lots 3, 6, and 1. Similarly, they are preserved in the first position of each test head in s_2'. Then, the other lots are appended to s_2' from the remaining schedule of s_1.

2.5.2. Genetic Algorithm Using Stochastic Selection to Preserve a Good Configuration (HGA-2). HGA-1 deterministically selects g_1 and g_2 based on the absolute values of the CPU idleness of configurations. However, HGA-2 randomly selects them based on the probabilities calculated from CPU idleness. We compute $nf_g = \max_{g'} \tilde{f}_{g'} - \tilde{f}_g$, for all g, then calculate the probability of g being selected by $q_g = nf_g / \sum_{g'} nf_{g'}$ for a particular chromosome. A configuration with CPU idleness closer to zero has a higher probability of being selected. Therefore, HGA-2 stochastically selects a configuration g_1 from s_1 and g_2 from s_2, respectively, then performs the crossover operation outlined in HGA-1.

2.5.3. Genetic Algorithm Using Deterministic Selection to Preserve a Good Head (HGA-3). Let

$j(i)$: the test head number, on which lot i is processed, in a particular chromosome;

$W_{j'}$: the aggregated measure of all the lots on head j' in a particular chromosome; $W_{j'} = \sum_{\{i | j' = j(i)\}} w_i / n_{j'}$, for all j'.

To preserve the head with a good schedule, we design the following crossover operation. Let s_1 and s_2 be the two selected parents for crossover operation. Find j_1, such that $W_{j_1} = \min_j \{W_j\}$ in chromosome s_1 and, similarly, j_2 in s_2. Then, we put the sequence of lots on head j_1 on the corresponding head of new chromosome s_1'. In this way, we maintain the sequence of lots on a head with a good schedule (a small W_j). After that, we delete the lots in j_1 from s_2 and then append the remaining schedule of s_2 to s_1'. Similarly, we can generate s_2'.

2.5.4. Genetic Algorithm Using Stochastic Selection to Preserve a Good Head (HGA-4). Let

nw_j: the nonnegative measure for head j in a particular solution; $nw_j = \max_{j'} W_{j'} - W_j$, for all j;

u_j: the probability of head j in a particular chromosome being selected for preserving in a new chromosome; $u_j = nw_j / \sum_{j'} nw_{j'}$, for all j.

A head with a lower value of W_j has a higher probability of being selected. Then, we randomly select j_1 and j_2 from s_1 and s_2, respectively. Next, we perform the crossover method outlined in HGA-3 using j_1 in s_1 and j_2 in s_2.

2.5.5. Traditional Genetic Algorithm by Randomly Preserving a Configuration (TGA-1). Instead of using the $\tilde{f}_g$ measures, the configuration to be preserved is randomly selected with each configuration having an equal probability. Then, the crossover operator presented in HGA-1 performs to generate new chromosomes.

2.5.6. Traditional Genetic Algorithm by Randomly Preserving a Head (TGA-2). Instead of using the W_i measures, the head to be preserved is randomly selected, with each head having an equal probability. Then, the crossover operator presented in HGA-3 performs to generate new chromosomes.

2.6. Extension to Multiple Multihead Testers with Identical Testers. All the search methods outlined above are used for a single multihead tester problem with identical tester heads can easily be extended to a multiple multihead tester problem with identical tester heads and identical testers. In the computing procedure for evaluating a schedule, the cycle time per device test cycle in a configuration depends only on the heads of the same tester. That is, the configuration of a tester is independent of the configurations of the other testers. When an event changes a configuration, we merely need to update the event time of the heads on the same tester. However, when advancing the time, we should consider all heads on all machines.

The heuristic grouping method can also be applied by initially moving one lot from set H to the first tester and moving the remaining lots from set T to the first tester, then moving one lot from set H to the second tester and finding the reaming lots from set T to the second tester, and continuing in the same way until all heads are full. After advancing time to the first event, the procedure is the same as that outlined in Section 2.2.

All the search methods can also be easily extended. We should consider every configuration of every machine when considering configuration information, every head on every machine when using head information, and every lot when using lot information. In this way, all the methods are similarly applied.

3. Computational Experiments

A series of computational experiments is designed to test the above methods and find out which one performs best in solving this problem. Microsoft Visual C++ is used to run our experiments. Experiments are conducted and divided in to two parts. In the first part, the problems with optimal solutions are designed by ensuring that the CPU idle time of all configurations is zero. The initial solution, solved by a heuristic grouping method, is compared to the results obtained from Long Processing Time (LPT) method [44] and Multifit method [45]. In addition, the solutions solved by three metaheuristic techniques using lot-specific and configuration-specific information and the traditional approaches are compared to the optimal solutions. In the second part, test problems are randomly generated by specified problem parameters. By controlling the parameters, the analysis of variance (ANOVA) is used to realize the factor effects which will affect the performance of the three metaheuristic techniques.

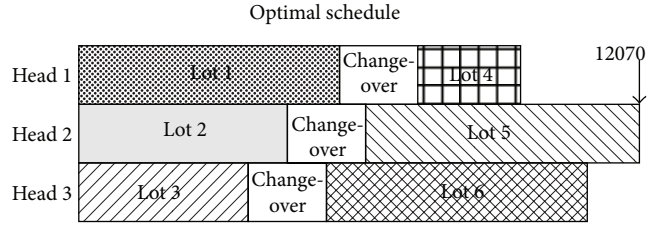

Figure 6: The optimal solution in a single tester problem.

3.1. Parameter of the Metaheuristic Techniques. In the algorithmic process of simulated annealing, three settings need to be determined. The initial temperature and the cooling procedure used in our experiments are called adaptive simulated annealing, which is suggested by [46]. The epoch length (the number of moves in a temperature) is the square of the number of lots in the problems, as suggested by [47].

To avoid a local optimum recurring in a tabu search, a tabu list is needed. The size of the tabu list is set to 7, as suggested by [48, 49].

Reference [50] proposed a so-called adaptive genetic algorithm, which dynamically adjusts the probability of crossover and mutation in genetic algorithms, and it shows that the adaptive genetic algorithm is better than a standard genetic algorithm. We use this adaptive approach for setting the crossover and mutation probabilities in our experiments. The population size used in our experiment is set to 100 [42].

3.2. Experiments Using Problems with Known Optimal Solutions. In order to find a problem with a known optimal solution, the inequality $\max_{i \in P_g}\{t_i + h_i\} < \sum_{i \in P_g} t_i$, for all g, needs to be satisfied in the test problem; that is, no CPU idle time occurred in each configuration. Figure 6 is the optimal schedule on one tester in one of the test problems. The device testing time and device handling time of each lot are shown in Table 6. The lots are duplicated on the machine twice to make it a three-head tester problem; that is, 18 lots are generated for this problem. Observing the experimental results of the initial solution obtained by different methods in Table 7, we can find out that the proposed heuristic grouping method is the best initial solution because other approaches do not consider the interaction between the test heads. The experimental results of all methods developed by three metaheuristic techniques are shown in Table 8 and the optimal solution can

Table 6: The parameters of lots of a known optimal problem experiment.

Lot number	Testing time	Handling time	Lot size
1	1.0	1.0	2000
2	1.2	0.9	1500
3	0.8	1.6	1000
4	1.2	2.0	1000
5	1.5	0.8	2500
6	1.0	0.6	2500

Table 7: The results of experiments using different algorithms solving initial solution with a known optimal problem.

Algorithms	Initial solution
Heuristic grouping	17266.00
Multifit	29960.00
LPT algorithm	24708.00

be found in most methods within 3000 CPU seconds, the time limitation of execution. HTS-3 takes the least amount of CPU time (0.5 seconds) to find the optimal solution. Generally speaking, tabu search is better than simulated annealing, and simulated annealing in turn is better than genetic algorithm. The metaheuristic techniques embedded with lot-specific or configuration-specific information are better than their corresponding traditional implementations.

3.3. Experiments Using Randomly Generated Problems. In this part of the experiments, the following factors are considered when generating the test problems:

(1) the number of machines: two factor levels are set to one machine and three machines in order to realize the performance of single multihead tester and multiple multihead tester scheduling problems;

(2) the number of test heads: three factor levels are set to two heads, three heads, and four heads in this single multihead tester problem; it is unusual to have a machine with more than four heads in practice;

(3) the ratio of the number of lots over the number of test heads: two factor levels are set as in (2) and (4);

(4) the ratio of handling to testing: this ratio might affect the performance of the algorithms; let

$\bar{t}$: the average testing time;

$\bar{h}$: the average handling time;

γ: the ratio of handling to testing and it is defined by

$$\gamma = \frac{\bar{h}}{(J-1)\cdot \bar{t}}. \tag{5}$$

Three levels for this ratio (γ) are set to 0.8, 1.0, and 1.2 in our random problems;

(5) the variation of device testing time and device handling time in the problem: these two values are randomly generated by the following uniform distributions:

$$\begin{aligned} t_i &\sim \text{uniform}\left[\bar{t}\times(1-r), \bar{t}\times(1+r)\right], \\ h_i &\sim \text{uniform}\left[\bar{h}\times(1-r), \bar{h}\times(1+r)\right], \end{aligned} \tag{6}$$

where r is the parameter to control the variation of testing time and handling time. Two factor levels for r are set to 0.1 and 0.3.

In sum, a total of 72 ($= 2 \times 3 \times 2 \times 3 \times 2$) factor combinations are considered and 10 random problems are tested for each factor combination. Thus, a total of 720 problems, each of which is solved by the above four tabu search methods, are tested. After several initial tests, the stopping search time is set to 250 CPU seconds in each search experiment. The average testing time ($\bar{t}$) is set to 2 CPU seconds and the average handling time is then determined by (5). The lot sizes are randomly generated by letting $d_i \sim$ uniform(1000, 2000). The changeover time is set to 1200 CPU seconds.

Since the optimal solutions are not obtained in this part of the experiments and the problem sizes are varied, we need to normalize the objective values of the solutions found by the search methods. The following notations are used to present how to normalize objective values and CPU times.

$(C_{\max})_v^{a\tau}$: the makespan found by method a at time τ for problem v.

$(C_{\max})_v^*$: the best solution (minimum makespan) found within 250 CPU seconds among all the methods for problem v.

$M_v^{a\tau} = (C_{\max})_v^{a\tau}/(C_{\max})_v^*$: the normalized objective value for problem v using method a at CPU time τ.

τ_v^*: the CPU time (in seconds) when the first method finds $(C_{\max})_v^*$ for problem v.

$L_v^* = \log_{10}(100\times\tau_v^*)$: the logarithm to base 10- of 100-fold of the CPU seconds at which the best solution is found. To avoid having the logarithm value of a CPU time being negative, we have a CPU time in seconds multiplied by 100. That is to say, the logarithm value of CPU time will only be negative when the actual CPU time finding the best solution is less than 0.01 second, which is impossible in our experiments.

$N_v^\tau = \log_{10}(100\times\tau)/L_v^*$: the normalization of CPU time τ for problem v.

3.3.1. Factorial Analysis Results and Algorithm Performance. We utilize multifactor testing of the general linear model to analyze our results. The method (METHOD) is treated as a fixed factor, and thus a total of six factors are considered in this experiment. The number of test heads (HEAD) is also a fixed factor, whereas the other factors—number of machine (MAC), the ratio of the number of lots to the number of testing heads (LOT_R), the ratio of testing to handling

Table 8: The results of the experiments using the problem with a known optimal problem.

Algorithms	Best solution	Ratio of best solution over optimal solution	CPU time used to find the best solution
Heuristic grouping	17266	1.43	—
TSA	12070	1	38.20
HSA-1	12074	1.0003314	2.50
HSA-2	12074	1.0003314	11.20
TTS	12986	1.075890638	0.00
HTS-1	15078	1.249212925	0.00
HTS-2	12070*	1	7.50
HTS-3	12070*	1	0.50
HGA-1	12074	1.0003314	121.42
HGA-2	12074	1.0003314	21.00
HGA-3	12074	1.0003314	32.83
HGA-4	12074	1.0003314	267.42
TGA-1	12074	1.0003314	8.3
TGA-2	12074	1.0003314	105.3

*The optimal solution found.

Table 9: The ANOVA table.

Source	Sum of squares	df	Mean square	F	Sig.
HEAD	0.392	2	$1.96E-01$	584.5335	0.000
METHOD	0.448	12	$3.70E-02$	111.0758	0.000
MAC	0.86	1	0.86	2559.29	0.000
LOT_R	$6.40E-02$	1	$6.40E-02$	192.7445	0.000
T_H_R	$1.19E-01$	2	$5.99E-02$	178.48	0.000
RANGE	$1.24E-02$	1	$1.90E-01$	567.7516	0.000
Error	3.0378	9010	$3.37E-04$		

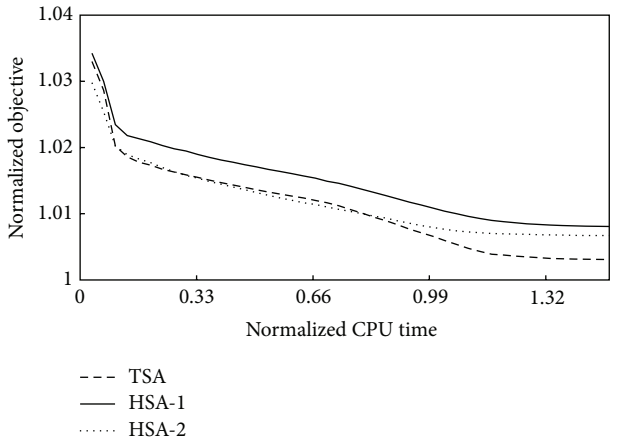

Figure 7: The normalized objective versus normalized CPU time for simulated annealing methods.

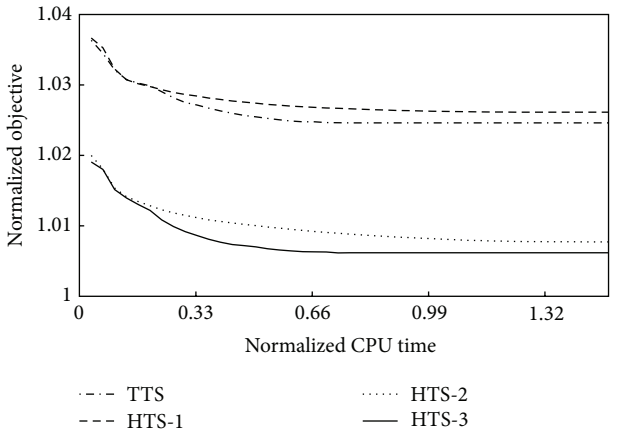

Figure 8: The normalized objective versus normalized CPU time for tabu search methods.

(T_H_R), and the variation of testing time and handling time (RANGE)—are random factors. In the statistical analysis, a significant level of $\alpha = 0.05$ is used throughout. The analysis of variance (ANOVA) results are shown in Table 9. All the factors have significant effects on search performance.

The plots of the average normalized objective versus normalized CPU time for the simulated annealing, tabu search, and genetic algorithm are shown in Figures 7, 8, and 9, respectively. The results of simulated annealing are shown in Figure 7. In the early stage, HSA-2 is better than

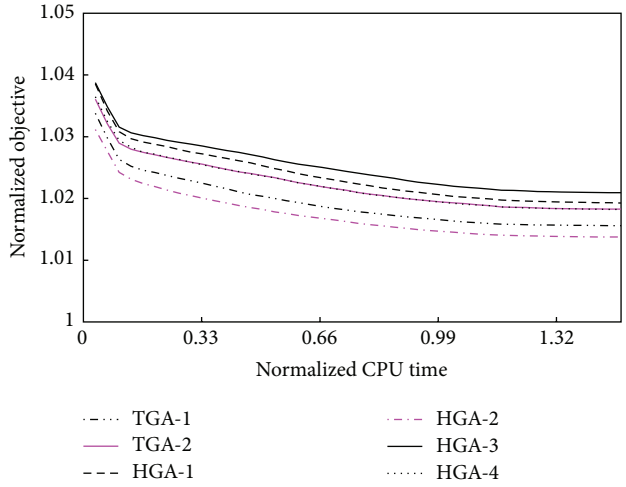

Figure 9: The normalized objective versus normalized CPU time for genetic algorithm methods.

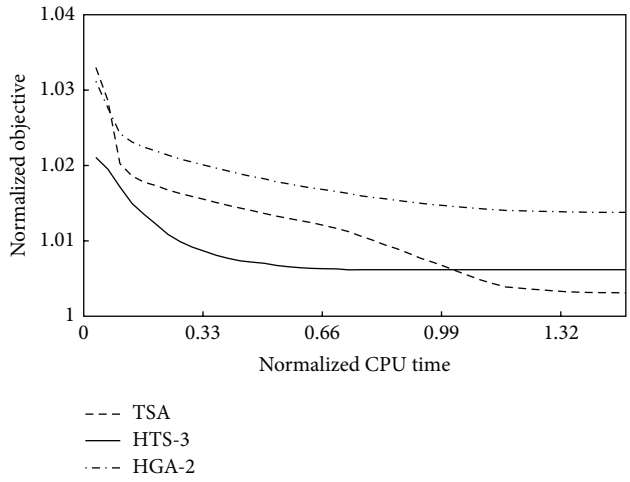

Figure 10: The normalized objective versus normalized CPU time for the best method from each meta algorithm group.

TSA. However, during the middle and later stages, these two methods are close. TSA is even better than HSA-2 in the final stage; that means that a simulated annealing with lot-specific information (HSA-2) finds a good solution faster. However, if the execution time is longer, traditional simulated annealing could catch up. Figure 8 shows that a tabu search with lot information (HTS-3) is always the best among all the tabu search methods. Overall, HTS-3 is always the best among all the 4 methods. This represents that the lot information used in HTS-3 is very useful in developing the heuristic rule to modify the traditional tabu search. Figure 9 shows that HGA-2 is the best while all of the other methods are not significantly different.

By choosing the best method from each metaheuristic group as shown in Figure 10, we can clearly see that HTS-3 is the best in the first 3/4 stage of whole time period. TSA, however, catches up and overtakes HTS-3 at the final 1/4 stage of whole time period. It is interesting that HTS-3 could find a good solution faster but the quality of traditional simulated annealing would be better than HTS-3 if the executive time is long enough. In addition, HGA-2 falls behind other two methods in searching a good solution in whole time period.

4. Conclusions

In semiconductor back-end testing facilities, it is very important to improve the efficiency of the tester which is the bottleneck at the plants. This paper focuses on the multihead tester scheduling problem with the objective of minimizing makespan, which tries to finish current waiting lots in a minimal time. The special features of such a parallel machine scheduling problem are utilized to propose a heuristic grouping method to generate a good initial solution efficiently. As a result of both CPU idle time and device waiting time prolonging the makespan, two performance measures are developed in this paper. One is calculated based on the CPU idleness of configurations; the other one is computed by using the device waiting times of lots. The three metaheuristic techniques are embedded with heuristic rules using these two measures.

Based on the comparative analysis of our experiments, a stochastic selection is better than a deterministic selection. For example, HTS-2 is better than HTS-1, HGA-2 is better than HGA-1, and HGA-4 is better than HGA-3. If the executive time is short, the tabu search with lot-specific information (HTS-3) performs best. When a longer executive time is allowed, the performance of traditional simulated annealing (TSA) would overtake the tabu search with lot-specific information (HTS-3) and becomes the best of all.

Acknowledgments

This research was supported by a research grant (NSC 100-2221-E-007-052-MY3) from the National Science Council of Taiwan, Taiwan, and research projects (101N2073E1 and 102N2075E1) from National Tsing Hua University.

References

[1] A. K. Gupta and A. I. Sivakumar, "Job shop scheduling techniques in semiconductor manufacturing," *The International Journal of Advanced Manufacturing Technology*, vol. 27, no. 11-12, pp. 1163–1169, 2006.

[2] R. Uzsoy, C.-Y. Lee, and L. A. Martinvega, "A review of production planning and scheduling models in the semiconductor industry part i: system characteristics, performance evaluation and production planning," *IIE Transactions*, vol. 24, no. 4, pp. 47–60, 1992.

[3] R. Uzsoy, C.-Y. Lee, and L. A. Martinvega, "A review of production planning and scheduling models in the semiconductor industry part ii: shop-floor control," *IIE Transactions*, vol. 26, no. 5, pp. 44–55, 1994.

[4] M. Mathirajan and A. I. Sivakumar, "A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor," *The International Journal of Advanced Manufacturing Technology*, vol. 29, no. 9-10, pp. 990–1001, 2006.

[5] L. Danping and C. K. M. Lee, "A review of the research methodology for the re-entrant scheduling problem," *International Journal of Production Research*, vol. 49, no. 8, pp. 2221–2242, 2011.

[6] R. C. Leachman, J. Kang, and V. Lin, "SLIM: short cycle time and low inventory in manufacturing at samsung electronics," *Interfaces*, vol. 32, no. 1, pp. 61–77, 2002.

[7] H. -C. Wu and T. Chen, "A fuzzy-neural ensemble and geometric rule fusion approach for scheduling a wafer fabrication factory," *Mathematical Problems in Engineering*, vol. 2013, Article ID 956978, 14 pages, 2013.

[8] L. Li, Z. Sun, M. C. Zhou, and F. Qiao, "Adaptive dispatching rule for semiconductor wafer fabrication facility," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 2, pp. 354–364, 2013.

[9] Y. H. Lee, M. Ham, B. Yoo, and J. S. Lee, "Daily planning and scheduling system for the EDS process in a semiconductor manufacturing facility," *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 5-6, pp. 568–579, 2009.

[10] K. P. Ellis, Y. Lu, and E. K. Bish, "Scheduling of wafer test processes in semiconductor manufacturing," *International Journal of Production Research*, vol. 42, no. 2, pp. 215–242, 2004.

[11] J.-Z. Wu and C.-F. Chien, "Modeling semiconductor testing job scheduling and dynamic testing machine configuration," *Expert Systems with Applications*, vol. 35, no. 1-2, pp. 485–496, 2008.

[12] Y. Song, M. T. Zhang, J. Yi, L. Zhang, and L. Zheng, "Bottleneck station scheduling in semiconductor assembly and test manufacturing using ant colony optimization," *IEEE Transactions on Automation Science and Engineering*, vol. 4, no. 4, pp. 569–578, 2007.

[13] Z. Zhang, L. Zheng, F. Hou, and N. Li, "Semiconductor final test scheduling with Sarsa(λ, k) algorithm," *European Journal of Operational Research*, vol. 215, no. 2, pp. 446–458, 2011.

[14] Y. Deng, J. F. Bard, G. R. Chacon, and J. Stuber, "Scheduling back-end operations in semiconductor manufacturing," *IEEE Transactions on Semiconductor Manufacturing*, vol. 23, no. 2, pp. 210–220, 2010.

[15] M. Fu, R. Askin, J. Fowler et al., "Batch production scheduling for semiconductor back-end operations," *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, no. 2, pp. 249–260, 2011.

[16] T. Freed and R. C. Leachman, "Scheduling semiconductor device test operations on multihead testers," *IEEE Transactions on Semiconductor Manufacturing*, vol. 12, no. 4, pp. 523–530, 1999.

[17] R. M. Karp, "Reducibility among Combinatorial Problems," in *Complexity of Computer Computation*, R. E. Miller and J. W. Thatcher, Eds., pp. 85–103, Plenum Press, New York, NY, USA, 1972.

[18] N. Piersma and W. van Dijk, "A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search," *Mathematical and Computer Modelling*, vol. 24, no. 9, pp. 11–19, 1996.

[19] M.-W. Park and Y.-D. Kim, "Search heuristics for a parallel machine scheduling problem with ready times and due dates," *Computers & Industrial Engineering*, vol. 33, no. 3-4, pp. 793–796, 1997.

[20] R. Cheng and M. Gen, "Parallel machine scheduling problems using memetic algorithms," *Computers & Industrial Engineering*, vol. 33, no. 3-4, pp. 761–764, 1997.

[21] C. A. Glass, C. N. Potts, and P. Shade, "Unrelated parallel machine scheduling using local search," *Mathematical and Computer Modelling*, vol. 20, no. 2, pp. 41–52, 1994.

[22] D. Prot, O. Bellenguez-Morineau, and C. Lahlou, "New complexity results for parallel identical machine scheduling problems with preemption, release dates and regular criteria," *European Journal of Operational Research*, vol. 231, no. 2, pp. 282–287, 2013.

[23] E. B. Edis, C. Oguz, and I. Ozkarahan, "Parallel machine scheduling with additional resources: notation, classification, models and solution methods," *European Journal of Operational Research*, vol. 230, no. 3, pp. 449–463, 2013.

[24] C. F. Liu, "A hybrid genetic algorithm to minimize total tardiness for unrelated parallel machine scheduling with precedence constraints," *Mathematical Problems in Engineering*, vol. 2013, Article ID 537127, 11 pages, 2013.

[25] Y.-K. Lin, "Particle swarm optimization algorithm for unrelated parallel machine scheduling with release dates," *Mathematical Problems in Engineering*, vol. 2013, Article ID 409486, 9 pages, 2013.

[26] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[27] P. J. M. van Laarhoven and E. M. L. Aarts, *Simulated Annealing: Theory and Application*, Springer, New York, NY, USA, 2nd edition, 2010.

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

[29] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, Mass, USA, 1997.

[30] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[31] Z. Michaelewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, New York, NY, USA, 3rd edition, 2013.

[32] M. Gen and R. Cheng, *Genetic Algorithms & Engineering Design*, John Wiley & Sons, New York, NY, USA, 1997.

[33] W.-C. Lee, C.-C. Wu, and P. Chen, "A simulated annealing approach to makespan minimization on identical parallel machines," *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 3-4, pp. 328–334, 2006.

[34] I. Sariçiçek and Ç. Çelik, "Two meta-heuristics for parallel machine scheduling with job splitting to minimize total tardiness," *Applied Mathematical Modelling*, vol. 35, no. 8, pp. 4117–4126, 2011.

[35] G. Waligóra, "Tabu search for discrete-continuous scheduling problems with heuristic continuous resource allocation," *European Journal of Operational Research*, vol. 193, no. 3, pp. 849–856, 2009.

[36] D. Y. Hu and Z. Q. Yao, "Parallel machines scheduling with sequence-dependent setup times constraints," *Advanced Science Letters*, vol. 4, no. 6-7, pp. 2528–2531, 2011.

[37] I. A. Chaudhry and P. R. Drake, "Minimizing total tardiness for the machine scheduling and worker assignment problems in identical parallel machines using genetic algorithms," *The International Journal of Advanced Manufacturing Technology*, vol. 42, no. 5-6, pp. 581–594, 2009.

[38] A. J. Ruiz-Torres, G. Paletta, and E. Pérez, "Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects," *Computers & Operations Research*, vol. 40, no. 8, pp. 2051–2061, 2013.

[39] Y.-F. Hung, C.-H. Liang, and J. C. Chen, "Sensitivity search for the rescheduling of semiconductor photolithography operations," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 1–4, pp. 73–84, 2013.

[40] W. Y. Jia, Z. B. Jiang, and Y. Li, "Closed loop control-based real-time dispatching heuristic on parallel batch machines with incompatible job families and dynamic arrivals," *International Journal of Production Research*, vol. 51, no. 15, pp. 4570–4584, 2013.

[41] C. A. Glass, C. N. Potts, and P. Shade, "Unrelated parallel machine scheduling using local search," *Mathematical & Computer Modelling*, vol. 20, no. 2, pp. 41–52, 1994.

[42] G. E. Liepins and M. R. Hilliard, "Genetic algorithms: foundations and applications," *Annals of Operations Research*, vol. 21, no. 1, pp. 31–57, 1989.

[43] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, New York, NY, USA, 1989.

[44] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing*, vol. 7, no. 1, pp. 1–17, 1978.

[45] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[46] M. Lundy and A. Mees, "Convergence of an annealing algorithm," *Mathematical Programming*, vol. 34, no. 1, pp. 111–124, 1986.

[47] C. K. Y. Lin, K. B. Haley, and C. Sparks, "A comparative study of both standard and adaptive versions of threshold accepting and simulated annealing algorithms in three scheduling problems," *European Journal of Operational Research*, vol. 83, no. 2, pp. 330–346, 1995.

[48] F. Glover, "Tabu search: a tutorial," *Interfaces*, vol. 20, no. 4, pp. 74–94, 1990.

[49] F. Glover, E. Taillard, and D. Werra, "A user's guide to tabu search," *Annals of Operations Research*, vol. 41, no. 1, pp. 3–28, 1993.

[50] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.