Available online at www.sciencedirect.com

ScienceDirect

Expert Systems with Applications 35 (2008) 485–496

Expert Systems with Applications

www.elsevier.com/locate/eswa

Modeling semiconductor testing job scheduling and dynamic testing machine configuration

Jei-Zheng Wu, Chen-Fu Chien [*]

Department of Industrial Engineering and Engineering Management, National Tsing Hua University, 101 Section 2 Kuang Fu Road, Hsinchu 30013, Taiwan, ROC

Abstract

The overall flow of the final test of integrated circuits can be represented by the job shop model with limited simultaneous multiple resources in which various product mixes, jobs recirculation, uncertain arrival of jobs, and unstable processing times complicate the problem. Rather than relying on domain experts, this study aims to develop a hybrid approach including a mathematical programming model to optimize the testing job scheduling and an algorithm to specify the machine configuration of each job and allocate specific resources. Furthermore, a genetic algorithm is also developed to solve the problem in a short time for implementation. The results of detailed scheduling can be graphically represented as timetables of testing resources in Gantt charts. The empirical results demonstrated viability of the proposed approach.
© 2007 Elsevier Ltd. All rights reserved.

Keywords: Semiconductor final testing; Machine configuration; Timetabling; Genetic algorithm; Intelligent manufacturing; Scheduling

1. Introduction

The semiconductor industry has grown rapidly and subsequent production planning problems have raised many important research issues (Lee, Uzsoy, & Martin-Vega, 1992; Leachman, 1993). The semiconductor companies compete with each other in cost structure, quality, and the delivery to maintain their competitive advantages (Chien, Wang, & Cheng, 2007). The semiconductor manufacturing scheduling problem is complicated by various product mixes, jobs recirculation, uncertain arrival of jobs, and unstable processing times. In particular, final testing of integrated circuits (IC) devices is the final operation in semiconductor manufacturing. The objectives of final test are to deliver passed IC devices of the required quality on time to customers while using resources efficiently. The overall flow of the final test can be represented by the job shop model with limited resources. Many optimization models or algorithms for short-term production planning at the shop floor level have been developed. However, the manufacturing environment of final test is unstable and uncertain. For example, the machine condition is uncertain affecting the yield rate and the throughput. Hence, the scheduling problem of final test is more complex than the conventional job shop scheduling problem. The testing site is typically the bottleneck of the final test operation because the equipment at the testing site is expensive and limited in amount. Also, final test directly suffers from the uncertain arrival of jobs from upstream, job recirculations, long job processing time, and long machine setup time. Engineers who rely on personal domain knowledge cannot find optimal machine configuration to respond to production needs rapidly and effectively. However, few studies have been done to address the present problem.

Focusing on real settings, this study aims to develop a hybrid approach to address semiconductor final test scheduling problem (SFTSP). In particular, the following manufacturing characteristics are considered. Firstly, the machines in the semiconductor final test facility are indeed

[*] Corresponding author. Tel.: +886 3 5742648; fax: +886 3 5722685.
E-mail address: cfchien@mx.nthu.edu.tw (C.-F. Chien).

0957-4174/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2007.07.026

the combinations of specific testers, handlers, and accessories. Such resources are limited and scheduling approaches must take the resource constraint into account (Chen, Chang, Chen, & Kao, 1995; Chien & Chen, 2007a). Secondly, while different types of test machines (different combinations of resources) can process the test for a given product at different speeds (different utilization rates), one machine configuration may be able to perform tests on more than one type of products. In addition to determine the testing schedule, the problem assigning a specific test machine configuration to a test job, given the limited resources, remains. Thirdly, changeover between different machine configurations requires sequence-dependent setup time including the disassembly time for original configurations and the assembly time for new machine configurations. Certainly, the sequence-dependent setup time reduces the utilization of testing resources and a good schedule avoids unnecessary setups (Chien & Chen, 2007b). Fourthly, the recirculations of testing jobs should be considered to ensure practical viability of the proposed scheduling method (Chien & Wu, 2003). Fifthly, some jobs that are released late but with higher priority and should be delivered on time may interrupt the earlier schedule. Consequently, developing consistently well-performed algorithms for scheduling testing jobs is difficult. In this study, we developed a mathematical programming model to optimize the testing job scheduling and proposed an algorithm to specify the machine configuration of each job and allocate specific resources. Furthermore, a genetic algorithm is also developed to solve the problem in a short time for practical viability. The results of detailed scheduling can be graphically represented as timetables of testing resources in Gantt charts. For validation, the results of the proposed approach are then compared with those of existing models.

The rest of this paper is organized as follows. Section 2 structures the semiconductor final test scheduling problem and reviews related research. Section 3 presents a mathematical model of job scheduling and the assignment algorithm to determine the machine configuration. Section 4 proposes a genetic algorithm to solve this problem. Section 5 compares the experimental results of the proposed model and genetic algorithm with those of existing studies. Conclusions and future research suggestions are finally made in Section 6.

2. Semiconductor final test and relative scheduling problems

The final test of IC devices is generally divided into the operations of Functional Test, Burn-in, Scan, Bake, Tape and Reel Station, and Package and Load Station. The Test site is the main operation site of the final test process and includes Room Test, Hot Test, and Cold test. The IC functional test is performed with testers, handlers, and accessories. Specific products can be tested only with appropriate machine configurations of testers, the handlers, and accessories. The tester, a central processing unit, loads test programs and is connected to the packaged IC devices via test heads to test their functions. Handlers, with built-in temperature control systems that enable tests in various temperatures, use suitable material handling devices called boats to load ICs by lots that packed in plastic antistatic trays or tubes. The boats release ICs from the trays or tubes and the suction devices (called nests) lift and load the ICs onto an electrical interface (called the load board) between the packaged circuits and the tester. After the functional test, the tested ICs are automatically removed from the handler and placed into the classified trays (or tubes) according to the results (i.e., passed or failed). The board and the nest are called the accessory (or kit). The jobs considered in the SFTSP are the IC devices in lots that were released from on-hand WIP or the output forecast of the assembly (i.e., the preceding operation to final test).

However, the machine configuration may not always be fixed. Different products can be tested by specific machine configurations (i.e., the corresponding combinations of testers, handlers, and accessories) with different processing times. It requires sequence-dependent setup time (SDST) to disassemble the original machine and assemble the new machine to test a different product. The quantities of testing resources such as testers, handlers, and accessories available in a final test facility are determined by long-term capacity planning. However, the daily availabilities of the resources are time-dependent and are influenced by the factors including current product-mix, equipment maintenance, machine breakdowns, and ongoing R&D experiments. Certain lots of IC devices would undergo functional testing more than once (i.e., recirculation). Thus, schedulers should trace the recirculated jobs and retain some machine capacity for them, which complicates the SFTSP. The present problem is to determine a testing job schedule and allocate appropriate combinations of resources to conduct the corresponding testing jobs.

A number of studies have been done on scheduling for semiconductor final test facilities. For example, Uzsoy, Martin-Vega, Lee, and Leonard (1991) and Ovacik and Uzsoy (1996) modeled and scheduled SFTSP including sequence-dependent setup time (SDST) based on the well-known shifting bottleneck approach developed by Adams, Balas, and Zawack (1988). After a number of computational experiments, they concluded proposed approaches significantly outperforming contemporary methods and CPU times being reasonable. The proposed decomposition methods were suggested applicable to SFTSP as well as the conventional job shop problems. Freed and Leachman (1999) further examined the multihead multiple tester scheduling problem where the head interdependency was taken into account and thus presented an enumerative solution technique. However, utilizing the contemporary tester scheduling methods for the multihead multiple test scheduling problems could result in infeasible or inferior solutions. In addition, Chien and Chen (2007b) developed a batch sequencing genetic algorithm embedded with a novel timetabling algorithm to solve the scheduling problem arising from Oxide–Nitride–Oxide (ONO) stacked film fabrication in semiconductor manufacturing that addressed the waiting time constraint and the frequency-based setup at the same time. Although the proposed enumerative procedure is restricted to small-scale problems, it provided optimal solutions for comparisons.

With consideration of dispatching rule-based heuristics, the computational experiment of Uzsoy, Church, and Ovacik (1992) with realistic data showed that their approach could yield a high quality solution to the SFTSP with little computational effort. Uzsoy, Lee, and Martin-Vega (1992) compared the performances of different dispatching rules in a dynamic final test manufacturing environment. They concluded that finding a dispatching rule that performed well for different scheduling objectives was difficult. Nevertheless, the performances of dispatching rules were robust to random variations in the variables including product routes, process time, setup time, lot sizes, and due-dates as well as to non-homogeneous job arrival patterns (i.e., the jobs arrive at an irregular rate over time as the preceding operations are completed). Furthermore, they found that Shortest Remaining Processing Time (SRPT) rule performed the best with respect to the objective of minimizing makepsan. When objectives were due-date related, Earliest Due-Date (EDD), Apparent Tardiness Cost with Setups (ATCS), and Shortest Setup plus Processing Time (SSPT) performed the best according to the performance measures of maximum lateness, average tardiness, and number of tardy jobs, respectively. Alternatively, Lin, Wang, and Lee (2004) utilized the concept of the theory of constraints to construct a capacity-constrained scheduling model. They designed hierarchical rules based on the importance, availability, flexibility of machines and characteristics of jobs, e.g., job flexibility and setup times to prioritize resource allocation. Although rule-based heuristics could provide reliable guidance with consideration of variation in a production system, they were not justifiable in terms of optimality or approximation. The verification was to compare with any other existing heuristics which are often rule-based in the literature. In words, a simulation-based approach could not guarantee solution quality for an arbitrary set of scheduling input data.

Regarding system dynamics, Perry and Uzsoy (1993) developed a methodology for resolving the dynamic job shop problem by combining a decomposition methodology for the static problem with an event-driven rescheduling (EDR) approach. The EDR approach rescheduled the final test jobs once a new job arrives or a machine fails. They highlighted the impact of dynamic job arrivals and other disruptions in a realistic final test facility and showed that the approach outperformed EDD-based heuristics according to computational results, when the objectives were minimizing maximum lateness. Chikamura, Nakamae, and Fujioka (1998) investigated the impact of dynamic job arrival on dispatching rule-based scheduling. They showed that the myopic and local behavior of dispatching rule-based scheduling might cause inefficient resource allocation in the long-term. Chen and Hsia (1994) employed branch and bound method with Lagrangian Relaxation (LR) to solve SFTSP and tested the feasibility of LR in two randomly created examples and two examples based on real factory data. They compared the performance of LR with three dispatching rules, i.e., SPT (Shortest Processing Time), EDD, and FIFO (Fist In First Out), according to five performance measures (three due-date related and two throughput related). According to their results, LR consistently outperformed the three dispatching rules when throughput related objectives were considered. Also, LR yielded a good feasible schedule within a reasonable computing time, though the heuristics based on dispatching rules took less time to yield such a feasible schedule. However, they did not investigate whether LR was still better than the other three dispatching rules when only due-date related objectives were considered.

3. Mathematical model and assignment algorithm

3.1. Mixed-integer linear programming model

Firstly, we constructed a mathematical mixed-integer linear programming model (MILP) for SFTSP to optimize the allocation of jobs in the evaluated time given the constraints. In particular, SFTSP is formulated as a combinatorial optimization problem with resource constraints. All parameters including processing time, due-dates, setup time, routing of jobs, and available quantities of each type of resources at each time are assumed to be given and deterministic. The MILP model was viewed as a timetable with the evaluated time on the horizontal axis and the jobs on the vertical axis. The time horizon consisting of T finite discrete time intervals is defined to be sufficiently long to process all jobs. The objective is to minimize the total weighted tardiness or makespan, denoted by $C_{\max}$, referring to the completion time of the latest finished job. Consider the following parameters and variables in the proposed MILP:

j	job j which is a split and pre-defined lot of product to be processed
o	functional test operation
m	machine configuration m which is a specific combination of a tester, a handler, and accessories
jo	operation (j,o), means that job j is to process functional test o
jom	operation (j,o,m), means that operation (j,o) is processed on machine m
c_j	completion time of job j
c_{jo}	completion time of operation (j,o)
d_j	due date of job j
e_{jo}	beginning time of operation (j,o)
p_{jom}	processing time of operation (j,o,m)
$s_{m_1m_2}$	sequence-dependent setup time between machines m_1 and m_2

x_{jomt}	equals to one if operation (j,o,m) is processed at time t or zero otherwise
z_{jom}	equals to one if operation (j,o) is selected to be processed on machine m or zero otherwise
N	number of jobs to be processed
T	time horizon under consideration
TD_j	Tardiness of job j
A_m	set of accessories with accessory type required for machine m
B_{jo}	set of machines that can be used to process operation (j,o)
FT_j	set of functional tests for job j
H_m	set of handlers with handler type required for machine m
M_R	set of machine types that require resource type R
O	set of all operations to be evaluated, i.e., $o = \{(j,o,m)\mid \forall j \in \{1,2,\ldots,N\}\ \forall o \in FT_j\ \forall m \in B_{jo}\}$
T_m	set of testers with tester type required for machine m
Δ_{Rt}	available quantities of resource R at time t
Ω_m	set of operations that can be processed on machine type m
α_{jomt}	equals to one if operation (j,o) begins processed on machine m at time t or zero otherwise
β_{jomt}	equals to one if operation (j,o) is processed on machine m at time t and not processed on machine m at time t+1 or zero otherwise
$\widetilde{A}$	set of all accessory types
$\widetilde{H}$	set of all handler types
$\widetilde{T}$	set of all tester types

The objective functions of the MILP model can be formulated are as follows.

$$\text{Minimize}\, J \equiv \sum_{j=1}^{N} w_j TD_j \quad \text{where}\ \mathrm{TD}_j = \max\{c_j - d_j, 0\} \tag{1}$$

$$\text{or Minimize}\ C_{\max}. \tag{2}$$

Eq. (1) forces the job scheduling to have a high priority to meet its due-date to satisfy the customers. Alternatively, objective Eq. (2) aims to reduce the cycle time of all jobs, thus increasing the productivity of the final test facility. Both objectives are mathematically tractable. This study considers processing time constraints, operation allocation constraints, resource constraints, non-preemption constraints, SDST constraints, precedence constraints, and additional constraints with details shown as follows.

3.1.1. Processing time constraints

$$\sum_{t=1}^{T} x_{jomt} - p_{jom} z_{jom} = 0 \quad \forall (j,o,m) \in O. \tag{3}$$

This constraint requires that the total time taken to process a certain operation must equal the processing time of the operation through the evaluated time horizon, if the operation is selected to process certain operation. That is, the sum of x_{jomt} with all t from 1 to T must equal the processing time of operation (j,o,m) for all $(j,o,m) \in O$ if machine m is selected to process operation (j,o).

3.1.2. Operation allocation constraints

$$\sum_{\forall m \in B_{jo}} z_{jom} = 1 \quad \forall j \in \{1,2,\ldots,N\}\ \forall o \in FT_j. \tag{4}$$

This constraint restricts the allocation of each job to a single machine.

3.1.3. Resource constraints

$$\begin{aligned}
&\sum_{\forall m \in M_{T_i}\ \forall (j,o) \in \Omega_m} x_{jomt} \leqslant \Delta_{T_i t} \\
&\quad \forall t \in \{1,2,\ldots,T\}\ \forall T_i \in \widetilde{T}, \\
&\sum_{\forall m \in M_{H_i}\ \forall (j,o) \in \Omega_m} x_{jomt} \leqslant \Delta_{H_i t} \\
&\quad \forall t \in \{1,2,\ldots,T\}\ \forall H_i \in \widetilde{H}, \\
&\sum_{\forall m \in M_{A_i}\ \forall (j,o) \in \Omega_m} x_{jomt} \leqslant \Delta_{A_i t} \\
&\quad \forall t \in \{1,2,\ldots,T\}\ \forall A_i \in \widetilde{A}.
\end{aligned} \tag{5}$$

Resource constraints ensure that the sum of the operations that use the same kinds of resources do not exceed the available quantities of the various resources in each time slot.

3.1.4. Non-preemption constraints

$$\begin{aligned}
&x_{jomt} - x_{jom,t-1} = \alpha_{jomt} - \beta_{jom,t-1} \\
&\quad \forall (j,o,m) \in O \text{ and } t \in \{2,3,\ldots,T\}, \\
&x_{jom1} = \alpha_{jom1}, \\
&x_{jomT} = \beta_{jomT}, \\
&0 \leqslant \alpha_{jomt}, \quad \beta_{jomt} \leqslant 1,
\end{aligned} \tag{6}$$

$$\begin{aligned}
&\sum_{t=1}^{T} \alpha_{jomt} = z_{jom}, \\
&\sum_{t=1}^{T} \beta_{jomt} = z_{jom}.
\end{aligned} \tag{7}$$

Eq. (6) describes the relationships among decision variables, x_{jomt}, α_{jomt}, and β_{jomt}. Notably, α_{jomt} and β_{jomt} can be relaxed as continuous variables bounded by the upper bound, 1 and the lower bound, 0. Eq. (7) restricts that operation (j,o,m) must be non-preemptive, otherwise more than one "one" variable will be present for both α_{jomt} and β_{jomt}. That is, preemption is not considered here.

3.1.5. SDST constraints

$$\begin{aligned}
&\sum_{\forall (j_1,o_1)\in\Omega_{m_1}}^{x_{j_1o_1m_1t}} + \sum_{\forall (j_2,o_2)\in\Omega_{m_2}}^{x_{j_2o_2m_2(t+i)}} \leqslant \Delta_{T_kt}\\
&\quad \forall t\in\{0,\ldots,T\}\ \forall i\in\{1,\ldots,s_{m_1m_2}\},\\
&t+i\leqslant T\ \forall m_1,m_2\in M_{T_k}\ \forall T_k\in\widetilde{T},\\
&\sum_{\forall (j_1,o_1)\in\Omega_{m_1}}^{x_{j_1o_1m_1t}} + \sum_{\forall (j_2,o_2)\in\Omega_{m_2}}^{x_{j_2o_2m_2(t+i)}} \leqslant \Delta_{H_kt}\\
&\quad \forall t\in\{0,\ldots,T\}\ \forall i\in\{1,\ldots,s_{m_1m_2}\},\\
&t+i\leqslant T\ \forall m_1,m_2\in M_{H_k}\ \forall H_k\in\widetilde{H},\\
&\sum_{\forall (j_1,o_1)\in\Omega_{m_1}}^{x_{j_1o_1m_1t}} + \sum_{\forall (j_2,o_2)\in\Omega_{m_2}}^{x_{j_2o_2m_2(t+i)}} \leqslant \Delta_{A_kt}\\
&\quad \forall t\in\{0,\ldots,T\}\ \forall i\in\{1,\ldots,s_{m_1m_2}\},\\
&t+i\leqslant T\ \forall m_1,m_2\in M_{A_k}\ \forall A_k\in\widetilde{A}.
\end{aligned} \tag{8}$$

Eq. (8) demands that two jobs using the same resource must be separate for a time period no less than the SDST.

3.1.6. Precedence constraint

$$\begin{aligned}
&c_{jo_1}\leqslant e_{jo_2}]\text{for } (j,o_1)\to(j,o_2)\\
&\quad\forall j\in\{1,2,\ldots,N\}\ \forall o_1,o_2\in FT_j.
\end{aligned} \tag{9}$$

In practice, jobs may be re-circulated (i.e., tested more than once) in the Testing site. Precedence constraint restricts that the recirculated operation can be processed only when the earlier operation has been finished.

3.1.7. Additional constraints

$$\begin{aligned}
&c_{jo}=\sum_{t=1}^{T}\sum_{\forall m\in B_{jo}} t\beta_{jomt}+1\quad \forall j\in\{1,2,\ldots,N\},\\
&e_{jo}=\sum_{t=1}^{T}\sum_{\forall m\in B_{jo}}^{t\alpha_{jomt}}\quad \forall j\in\{1,2,\ldots,N\},\\
&c_j\geqslant c_{jo}\quad \forall j\in\{1,2,\ldots,N\}\ \forall o\in FT_j.
\end{aligned} \tag{10}$$

$$\begin{aligned}
&TD_j\geqslant c_j-d_j,\\
&TD_j\geqslant 0\quad \forall j\in\{1,2,\ldots,N\}.
\end{aligned} \tag{11}$$

$$C_{\max}\geqslant c_j\quad \forall j\in\{1,2,\ldots,N\}. \tag{12}$$

Eq. (10) shows the relationships between the beginning and completion times of jobs, operations, and operations. Since Eq. (1) is in the form of min–max, it may be reformulated by adding a linear Eq. (11). Alternatively, when the objective is to minimize makespan, i.e., Eqs. (2), Eq. (12) is applied to retain the feasibility that makespan is larger than the time for completing any job.

The proposed MILP model can accommodate the situations in a dynamic environment in which new jobs arrive or other disruptions occur, such as machine breakdown or resource unavailability. In response to a certain disruption, the proposed scheduling method can dynamically reset the time horizon while fixing the jobs that have been processed and accordingly reschedule the remaining jobs for the remainder of the time.

The optimal solution of the MILP model includes the beginning and completion times of operations. However, it is difficult to accommodate machine configurations to the testing schedule and setting up appropriate combinations of specific testers, handlers, and accessories. Therefore, we proposed an assignment algorithm to determine specific combinations of testers, handlers, and accessories for the corresponding operations at each time in which the scheduling solution of MILP model can be visualized in Gantt charts.

3.2. Assignment algorithm

An assignment algorithm is developed to assign appropriate resources to each operation and to represent graphically the optimal scheduling solution of MILP model. Briefly, this algorithm searches the earliest unassigned operation and matches it with available tester, handler, and accessory that can be combined into an appropriate machine to process the operation at each iteration until all operations have been assigned. The proposed assignment algorithm involves the following steps with additional parameters:

e_{jom}	beginning time of operation (j,o,m). In particular, $e_{jom}=e_{jo}$ for $z_{jom}=1$ and $e_{jom}=0$ for $z_{jom}=0$
c_{jom}	completion time of operation (j,o,m). Particularly, $c_{jom}=c_{jo}$ for $z_{jom}=1$; $c_{jom}=0$, otherwise
A_{hk}	type h accessory with specific index (ID) k
C_r	current completion time of specific resource r
E_r	the earliest available time of specific resource r
H_{hk}	type h handler with specific ID k
T_{hk}	type h tester with specific ID k
V	set of all unassigned operations
$\widehat{A}_W$	set of accessories that can be used to process a specific set W of jobs
$\widehat{H}_W$	set of handlers that can be used to process a specific set W of jobs
$\widehat{T}_W$	set of testers that can be used to process a specific set W of jobs
$\overline{A}_{T_{hk}t}$	assign an accessory to tester T_{hk} at time t, e.g., if accessory a is assigned to T_{hk} at time t, we have $\overline{A}_{T_{hk}t}=a$
$\overline{H}_{T_{hk}t}$	assign a handler to tester T_{hk} at time t, e.g., if handler d is assigned to T_{hk} at time t, we have $\overline{H}_{T_{hk}t}=d$
$\overline{O}_{T_{hk}t}$	assign operation (j,o,m) to tester T_{hk} at time t, e.g., if operation (j,o,m) is assigned to T_{hk} at time t, we have $\overline{O}_{T_{hk}t}=(j,o,m)$

Step 1 Initialize V to be the set of optimal operations solved by the MILP model. Let $E_{T_{hk}}=0$, $E_{H_{hk}}=0$, $E_{A_{hk}}=0$, $M_{T_{hk}}=NULL$, $M_{H_{hk}}=NULL$,

$M_{A_{hk}} = NULL\ \forall T_{hk} \in \widehat{T}_V\ \forall H_{hk} \in \widehat{H}_V\ \forall A_{hk} \in \widehat{A}_V$. Go to step 2.

Step 2 Find (j^*, o^*, m^*) which is the unassigned operation in V with the earliest beginning time $e_{j^*o^*m^*}$. If more than a choice of (j^*, o^*, m^*) exists, choose arbitrarily. Find T_{m^*}, H_{m^*} and A_{m^*}. Go to step 3.

Step 3 Find $\overline{T}_{m^*} = \{T_{m_k^*} | E_{T_{m^*k}} + s_{M_{T_{m^*_k}}m^*} \leqslant e_{j^*o^*m^*}\ \forall T_{m_k^*} \in T_{m^*}\}$ which is the set of specific testers that can be assigned to process operation (j^*, o^*, m^*). Also, find $\overline{H}_{m^*} = \{H_{m_k^*} | E_{H_{m^*k}} + s_{M_{H_{m^*k}}m^*} \leqslant e_{j^*o^*m^*}\ \forall H_{m_k^*} \in H_{m^*}\}$ and $\overline{A}_{m^*} = \{A_{m_k^*} | E_{A_{m^*k}} + s_{M_{A_{m^*k}}m^*} \leqslant e_{j^*o^*m^*}\ \forall A_{m_k^*} \in A_{m^*}\}$. Go to step 4.

Step 4 Find $\overline{T}^*$, $\overline{H}^*$ and $\overline{A}^*$ that are the specific tester, handler and accessory with the largest earliest available times in $\overline{T}_{m^*}$, $\overline{H}_{m^*}$, and $\overline{A}_{m^*}$, respectively. If choices are available for $\overline{T}_{m^*}$, $\overline{H}_{m^*}$, and $\overline{A}_{m^*}$, choose arbitrarily. Go to step 5.

Step 5 From time $t = e_{j^*o^*m^*}$ to $c_{j^*o^*m^*} - 1$, set $\overline{O}_{T^*t} := (j^*, o^*, m^*)$, $\overline{H}_{T^*t} := \overline{H}^*$, and $\overline{A}_{T^*t} := \overline{A}^*$. Go to step 6.

Step 6 Update $E_{\overline{T}^*}$, $E_{\overline{H}^*}$ and $E_{\overline{A}^*} := c_{j^*o^*m^*}$ and $M_{\overline{T}^*}$, $M_{\overline{H}^*}$ and $M_{\overline{A}^*} := m^*$. Delete (j^*, o^*, m^*) from V. If $V = \emptyset$, stop; else go to step 2.

3.3. Numerical illustration

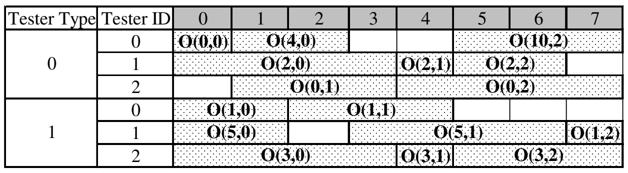
We illustrate the proposed MILP model and assignment algorithm with a numerical case. Six jobs, two types of testers (total 6 testers), two types of handlers (total 7 handlers), and four types of machine configurations are involved in this case. Some jobs must be processed more than once in the Testing site.

Table 1 illustrates an optimal scheduling result that includes the beginning and completion time of an example. ILOG CPLEX 9.0 (ILOG, 2003) was used to generate the optimal solution of the MILP model and determined that 8 time units would be required to finish all jobs. The assignment algorithm was applied to determine the resource allocation with respect to the optimal schedule. Furthermore, the combinations of various testing resources for each operation can be graphically visualized in Gantt charts. Figs. 1–3 illustrate the optimal schedule in various timetables generated by the assignment algorithm. The first column of the timetables refers to the tester type and the second to a specific tester of a given type. The other columns are the different time slots of the considered time horizon. Fig. 1 shows when and on which tester each operation should be processed. For example, $O(2,1)$ refers to operation $(2,1)$ that denotes job 2 processing functional test 1. Fig. 2 shows the times when a specific tester is combined with the other resources as a specific machine configuration. For example, M0 refers to machine configuration 0 that consists of a specific tester of type 0. Alternatively, Fig. 3 shows the times a specific handler is combined with the corresponding tester as the required machine configuration. For example, H02 denotes a specific handler of type 0 and ID 2.

However, in real settings, the proposed approach may not be able to generate the optimal schedules because of computational limitations. The number of binary variables of the MILP model almost equals the product of the time horizon and the number of jobs. In other words, a longer evaluated time horizon and more jobs present the MILP model with a much larger problem. The number of real variables is approximately double the product of the evaluated time horizon and the job number.

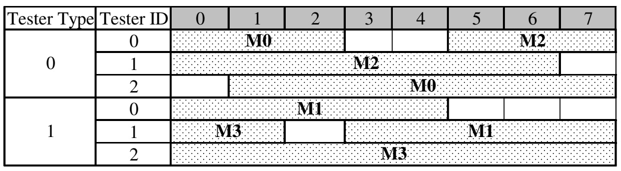
Table 1
An illustrative scheduling output data

Job ID	Test function	Machine type	Beginning time	Completing time
0	0	0	0	1
	1	0	1	4
	2	0	4	8
1	0	1	0	2
	1	1	2	5
	2	1	7	8
2	0	2	0	4
	1	2	4	5
	2	2	5	7
3	0	3	0	4
	1	3	4	5
	2	3	5	8
4	0	0	1	3
	1	1	5	8
5	0	3	0	2
	1	1	3	7



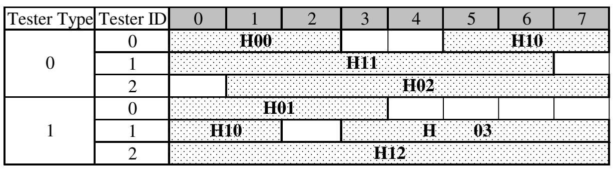
Tester Type	Tester ID	0	1	2	3	4	5	6	7
0	0	O(0,0)	O(4,0)	O(4,0)			O(10,2)	O(10,2)	O(10,2)
	1	O(2,0)	O(2,0)	O(2,0)	O(2,0)	O(2,1)	O(2,2)	O(2,2)	
	2		O(0,1)	O(0,1)	O(0,1)	O(0,2)	O(0,2)	O(0,2)	O(0,2)
1	0	O(1,0)	O(1,0)	O(1,1)	O(1,1)	O(1,1)			
	1	O(5,0)	O(5,0)		O(5,1)	O(5,1)	O(5,1)	O(5,1)	O(1,2)
	2	O(3,0)	O(3,0)	O(3,0)	O(3,0)	O(3,1)	O(3,2)	O(3,2)	O(3,2)

Fig. 1. Timetable of the optimal operation allocation.



Tester Type	Tester ID	0	1	2	3	4	5	6	7
0	0	M0	M0	M0			M2	M2	M2
	1	M2	M2	M2	M2	M2	M2	M2	
	2		M0	M0	M0	M0	M0	M0	M0
1	0	M1	M1	M1	M1	M1			
	1	M3	M3		M1	M1	M1	M1	M1
	2	M3	M3	M3	M3	M3	M3	M3	M3

Fig. 2. Timetable of the optimal machine allocation.



Tester Type	Tester ID	0	1	2	3	4	5	6	7
0	0	H00	H00	H00			H10	H10	H10
	1	H11	H11	H11	H11	H11	H11	H11	
	2		H02	H02	H02	H02	H02	H02	H02
1	0	H01	H01	H01	H01				
	1	H10	H10		H 03	H 03	H 03	H 03	H 03
	2	H12	H12	H12	H12	H12	H12	H12	H12

Fig. 3. Timetable of the optimal combination of testers and handlers.

4. Genetic algorithm

4.1. Fundamentals of genetic algorithms

The genetic algorithms is one of the most widely known stochastic searching techniques, based on the mechanism of natural selection and genetics (Gen & Cheng, 1997). Genetic algorithms have been widely applied to many optimization problems such as combinatorial optimization problems (Beasley & Chu, 1996; Hinterding, 1998; Kubota & Fukuda, 1996), job shop scheduling problems (Cheng, Gen, & Tsujimura, 1996; Cheng, Gen, & Tsujimura, 1999; Dagli & Sittisathanchai, 1995; Dorndorf & Pesch, 1995; Kobayashi, Ono, & Yamamura, 1995; Storer, Wu, & Vaccari, 1992), reliability optimization problems (Gen & Cheng, 1996; Dengiz, Altiparmak, & Smith, 1997), and manufacturing cell design problems (Moon, Kim, & Gen, 1999).

A genetic algorithm contains four parameters, i.e., crossover rate, mutation rate, population size, and generation size. A genetic algorithm initializes a set of solutions called a *population* for a specific problem. Each individual in the population is called a *chromosome* and represents a specific solution. The representations of chromosomes are problem-dependent and may influence the effectiveness and efficiency of genetic algorithms in solving problems. Iterations of the genetic algorithm are called *generations*. The chromosomes are evaluated during each generation according to specific measures of fitness. New chromosomes are called *offspring*. Offspring are generated either by merging two chromosomes (called parents) from the current generation using a crossover operator, or by modifying the chromosomes using a mutation operator. The crossover rate is defined as the ratio of the number of offspring produced in each generation to the population size. The mutation rate is defined as the percentage of the total number of mutated chromosomes in the population. After crossover and mutation, a new generation of chromosomes is created by selecting some of the parents and offspring, according to the fitness values while maintaining a constant population size. Following the evolution of several generations, the algorithms converge to the optimal or sub-optimal solutions to the problem. Increasing the population size, generation size, crossover rate, or mutation rate corresponds to exploring more of the solution space, reducing the chances of obtaining an infeasible solution, yet requiring greater computational effort.

4.2. Genetic algorithm for SFTSP

In practice, it is hard to ensure the combinatorial optimality of SFTSP within a reasonable time. A near-optimal solution is good enough for viable purpose. Therefore, a genetic algorithm for the semiconductor final test scheduling problem, denoted by GASFTSP, is proposed to efficiently solve the SFTSP with optimal or near-optimal schedule. Applying genetic algorithms to optimization problems has three major advantages. Firstly, genetic algorithms do not require much mathematical understanding about the optimization problems. Secondly, genetic algorithms may incorporate mechanisms that avoid staying in local optimal solutions. Thirdly, genetic algorithms can be flexibly combined with domain-dependent heuristics to address a specific problem efficiently. Fig. 4 illustrates the GASFTSP procedure. GASFTSP begins with initializing a set of chromosomes randomly generated from a number of sequences of dispatching rules. The length of a chromosome in GASFTSP is designed to equal the number of jobs to be scheduled. A chromosome is encoded as a sequence of dispatching rules that assign jobs to specific resources in a given period. The decoding method of GASFTSP is a greedy algorithm that will transform a list of dispatching rules, a chromosome, into a schedule. The proposed greedy algorithm first finds an available tester and subsequently assigns a released lot and required resources to the selected tester.

Denote U as the set of all unscheduled operations with all preceding operations having been scheduled and $U_{T_{hk}}$ as the set of unscheduled operations requiring tester T_{hk}. The greedy algorithm consists of the following steps.

Step 1 (*Initialize*) Initialize U. Let $E_{T_{hk}} = 0$, $C_{T_{hk}} = 0$, $C_{H_{hk}} = 0$, $C_{A_{hk}} = 0$, $M_{T_{hk}} = NULL$, $M_{H_{hk}} = NULL$, $M_{A_{hk}} = NULL$ $\forall T_{hk} \in \widehat{T}_U$ $\forall H_{hk} \in \widehat{H}_U$ $\forall A_{hk} \in \widehat{A}_U$. Go to step 2. (If resources are not available at the start of scheduling, it is required to keep some initial information of resources, e.g., if tester T_{hk} is assigned to process

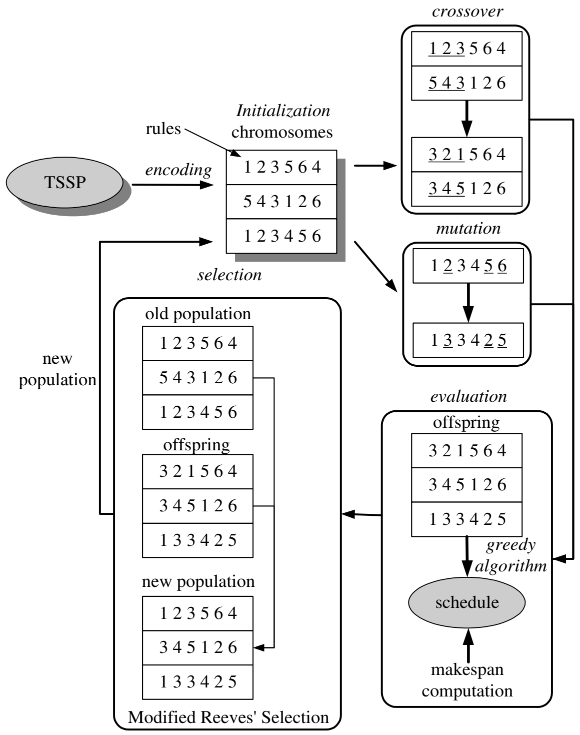


Fig. 4. Genetic algorithm for testing site scheduling problem.

certain operation on machine 1 with remaining 2 units of testing time at the start of scheduling, let$E_{T_{hk}} = 2$, $M_{T_{hk}} = 1$, and $M_{H_{hk}} = 1$.)

Step 2 (*Select a tester*) Let $G_{T_{hk}m} = \max\{C_{T_{hk}} + s_{M_{T_{hk}}m}, E_{T_{hk}}\}$, find $G^* = \min_{\forall T_{hk} \in \widehat{T}_U, m \in B_{jo}, \text{and} (j,o) \in U} \{G_{T_{hk}m}\}$ and the tester T^* on which G^* occurs. If alternatives are available for T^*, choose one with maximal $C_{T_{hk}}$. Subsequently, we refer to tester type of tester T^* as T^{**}. Go to step 3.

Step 3 (*Sort*) Let $O_{T^*} = \{(j,o,m) | \forall (j,o) \in U_{T^*}, T_m = T^{**}, m \in B_{jo}\}$ and sort O_{T^*} according to a given dispatching rule. Go to step 4.

Step 4 (*Find available handler*) Select the first operation (j_1, o_1, m_1) of O_{T^*}. Let $C^* = \max(C_{T^*} + s_{M_{T^*}m}, E_{T^*})$ and $\Xi = \{H_{hk} | C_{H_{hk}} + s_{M_{H_{hk}}m} \leqslant C^*, H_{hk} \in H_m\}$. Ξ is the set of those handlers that can be combined as machine m, before m is to begin processing job j. If $\Xi = \emptyset$ delete (j_1, o_1, m_1) from O_{T^*}. If $O_{T^*} = \emptyset$ go to step 6, else repeat step 4. If $\Xi \neq \emptyset$, let $E^*_H = \max_{\forall H_{hk} \in H_m} \{C_{H_{hk}} + s_{M_{H_{hk}}m}\}$ and let H^* be H_{hk} where E^*_H occurs. If alternatives for H^* exist, choose one with maximal $C_{H_{hk}}$. Go to step 5.

Step 5 (*Find available accessory*) Following the same operation (j_1, o_1, m_1) as in step 4, let$\Theta = \{A_{hk} | C_{A_{hk}} + s_{M_{A_{hk}}m} \leqslant C^*, A_{hk} \in A_m\}$. If $\Theta = \emptyset$, delete (j_1, o_1, m_1) from O_{T^*}. If $O_{T^*} = \emptyset$ go to step 6; else go to step 4. If $\Theta \neq \emptyset$, let $E^*_A = \max\{C_{A_{hk}} + s_{M_{A_{hk}}m}\}$ and let A^* be A_{hk} where E^*_A occurs. If alternatives for H^* exist, choose one with maximal $C_{H_{hk}}$. Go to step 7.

Step 6 (*Update earliest available time of a given tester*) Let $E_{T^*} := \max\{\min_{H_{hk} \in H_m}(C_{H_{hk}} + s_{M_{H_{hk}}m}), \min_{A_{hk} \in A_m} (C_{A_{hk}} + s_{M_{A_{hk}}m})\}$ and go to step 2.

Step 7 (*Assign and/or Stop*) From time $t = C^*$ to $C^* + p_{jom} - 1$, set$\overline{O}_{T^*t} := (j_1, o_1, m_1)$, $\overline{H}_{T^*t} := H^*$, and $\overline{A}_{T^*t} := A^*$. Set $E_{T^*}, C_{T^*}, C_{H^*}, C_{A^*} := C^* + p_{jom}$. Delete operation (j_1, o_1) from U and add the successor of operation (j_1, o_1) to U if a successor exists. If $U \neq \emptyset$, go to step 2; else let $T = \max_{\forall T_{hk} \in \widehat{T}_J} \{C_{T_{hk}}\}$ and stop.

Theoretically, the proposed greedy algorithm can generate a feasible schedule for SFTSP. Furthermore, a certain dispatching rule can be incorporated in step 3, the sort steps, to select the operation. For example, given a chromosome, [1 3 2 3 2] where 1 stands for rule SPT, 2 for SSPT, and 3 for SRPT, the SPT rule is used in step 3 at the first iteration of the proposed greedy algorithm to select an operation to a given tester, handler and accessories and the SRPT rule is used at the second iteration, and so on.

Crossover and mutation are then applied to selected chromosomes. The crossover operator of GASFTSP randomly selects two sub-strings from the parents and then exchanges them to produce two offspring. The mutation operator of GASFTSP randomly selects some genes of a given chromosome and mutates them by giving new dispatching rules randomly. The selection strategy of GASFTSP modifies and applies the simple ranking method proposed by (Reeves, 1995). In Reeves' selection, chromosomes of parents are ranked in ascending order of fitness function (or in descending order of makepsan) and are given probabilities $p_k = 2k/M(M+1)$ where k refers to the kth chromosome in ascending order and M refers to the fittest one. The modified Reeves' selection is a stochastic sampling of the $(\mu + \lambda)$ selection that randomly selects the μ best chromosomes as parents of the next generation from μ old parents and λ offspring (Gen & Cheng, 1997). Duplicated chromosomes are replaced by random chromosomes before selecting to prevent over-weighting them in each generation.

After that, all chromosomes are evaluated based on the objective of minimizing the makespan. Finally, chromosomes of the new population are selected from the old population and the offspring according to a modified version of Reeves' selection. The GASFTSP evolves by repeating the above procedures until the solution meets the terminating criteria.

5. Comparison experiments

The performance of GASFTSP was compared with six dispatching rules using the proposed greedy algorithm. The involved dispatching rules were SST (select operation with shortest setup time), RND (select operation in random order), SPT (select operation with shortest processing time), SSPT (select operation with shortest processing plus setup time), SRPT (select operation with shortest remaining processing time), and LRPT (select operation with largest remaining processing time from unscheduled operation). For example, the heuristic of combining the greedy algorithm and the SST dispatching rule is as follows:

Step 1 (*Initialize*)
Step 2 (*Choose a tester*)
Step 3 (*Sort*)
Let $O_{T^*} = \{(j,o,m) | \forall (j,o) \in U_{T^*}, T_m = T^{**}, m \in B_{jo}\}$ and sort O_{T^*} based on SST. Go to step 6
Step 4 (*Find available handler*)
Step 5 (*Find available accessory*)
Step 6 (*Update earliest available time of a given tester*)
Step 7 (*Assign and/or Stop*)

Similarly, applying the different dispatching rules involved in step 3 may derive the other heuristics. GASFTSP included 10,000 chromosomes, each with a sequence of random numbers generated from, for example, $\{0, 1, 2\}$ where 0 denoted RND, 1 denoted SSPT, and 2 denoted SST. The crossover and mutation rates of GASFTSP were set at 0.75 and 0.15, respectively.

Experimental data were tested on four different level problems, including simple, small-scale, large-scale, and wide-range problems. The performance measure of minimizing makespan was applied to evaluate the solution quality of all compared methods. The experiments were performed on a personal computer with an Athlon 1.2G processor and 785,904 KB RAM.

In each problem, the experiment included various quantities of resources, machines, and jobs with different routes. The simple problem merely included 15 jobs, each of which has one single operation that can be processed on either a single random machine or on two different types of machines, selected randomly with equal probability. The processing time of each operation was randomly generated from $\{1,2,\ldots,6\}$. The setup time between different operations was randomly generated from $\{1,2\}$. Two types of testers and two types of handlers were considered. Each type of tester or handler included three specific resources with the same function.

In the small-scale problem, all random generators were the same as the simple problem except that all jobs were able to be processed on only one random machine. Thirty experimental data were randomly tested on both problems. The simple and small-size problems were designed to be possibly optimized by a commercial mathematical programming package, ILOG CPLEX 9.0 (ILOG, 2003). We compared the optimal schedule with the schedules generated by GASFTSP and the greedy algorithms with different dispatching rules to show the gap between the approximate and optimal solutions and to examine the performance of GASFTSP.

The large-scale problem involved 150 jobs (lots) each of which had the same probability of including j operation where $j = 1,2,3$. Each operation had an equal probability to be processed on one, two, or three machines. The processing time of each operation was randomly generated from $\{1,2,\ldots,15\}$. The setup time between different operations was randomly generated from $\{1,2,\ldots,5\}$. Three types of testers, three types of handlers, and four types of accessories were employed. The numbers of Type I, Type II, and Type III testers were 10, 5, and 3, respectively. The numbers of Type I, II, and III handlers were 10, 8, and 4, respectively. The numbers of Type I, II, III, and IV accessories were 7, 7, 5, and 5, respectively. The size of this problem was similar to a daily scheduling problem in an anonymous semiconductor final test facility.

In the wide-range problem, resources, operations in routings, and machines were the same as in the large-scale problem. Differently, this problem comprised 60 jobs with processing time uniformly distributed on a wider range, i.e., $[1,50]$, and setup time uniformly distributed on $[1,15]$. Experimental data were independently replicated 30 times on both problems to evaluate the performance of GASFTSP and compare GASFTSP with the other heuristics. In addition, we experimented on the wide-range problem to identify the impact of increasing lot size (i.e., increasing processing time and setup time) on the scheduling performance of various algorithms. Table 2 presents the simulation parameters for each experimental problem.

Table 3 summarizes the experimental results of the simple and small-scale problems. It is shown that GASFTSP, labeled as GA, could find optimal solutions shown in the MILP columns, in all runs of the simple problem and in almost all runs of the small-size problem. Conversely, the results from the small-size problem reveal that few of the heuristics other than GASFTSP could approach to the optimal solutions. The results also suggest that the proposed GASFTSP, yielding a solution based on an intelligent combination of dispatching rules, be an approximate approach to the SFTSP. The average CPU times of GASFTSP on the simple and small-scale problems were 54.4 and 44 s, respectively. On the other hand, the other heuristics took less than 1 second of CPU time to find their solutions. Though the other heuristics were cheaper in computation, they did not guarantee solution quality and could approach to a good solution only by chance.

Table 4 summarizes the experimental results of the large-scale and wide-range problems. GASFTSP outperformed all the other methods in all experiment runs except in the third and fourth run on the large-scale problem where the SST rule obtained best solutions. The CPU times for GASFTSP in the former and latter problem were, respectively, 8106.3 and 5369.9 s, whereas the other heuristics took approximately 1 s. Indeed, the results of the heuristic of combining the greedy algorithm and the SST rule

Table 2
Simulation parameter description

Experimental problem	Number of resources	Number of jobs	Operations in routings	Machines to be processed	Processing time	Setup time
Simple	T: 3, 3 H: 3, 3 A: unlimited	15	1	1 or 2	$\{1,2,\ldots,6\}$	$\{1,2\}$
Small-scale	T: 3, 3 H: 3, 3 A: unlimited	15	1	1	$\{1,2,\ldots,6\}$	$\{1,2\}$
Large-scale	T: 10, 5, 3 H: 10, 8, 4 A: 7, 7, 5, 5	100	$[1,3]$	1, 2, or 3	$\{1,2,\ldots,15\}$	$\{1,\ldots,5\}$
Wide-rage	T: 10, 5, 3 H: 10, 8, 4 A: 7, 7, 5, 5	60	$[1,3]$	1, 2, or 3	$[1,50]$	$[1,15]$

Note: T denotes testers, H denotes handlers, and A denotes accessories.

Table 3
The simulation results for the simple and small-scale problems

Run	Simple problem								Small-scale problem							
	MILP	GA	SST	RND	SSPT	LRPT	SPT	SRPT	MILP	GA	SST	RND	SSPT	LRPT	SPT	SRPT
1	$\underline{9}$	$\underline{9}$	12	14	12	13	12	12	$\underline{9}$	$\underline{9}$	11	12	11	11	11	11
2	$\underline{9}$	$\underline{9}$	14	13	14	13	12	12	$\underline{11}$	$\underline{11}$	12	16	12	$\underline{11}$	12	12
3	$\underline{7}$	$\underline{7}$	11	14	11	10	9	9	$\underline{14}$	$\underline{14}$	16	17	17	17	18	18
4	$\underline{10}$	$\underline{10}$	12	16	12	15	12	12	$\underline{12}$	$\underline{12}$	13	$\underline{12}$	13	16	13	13
5	$\underline{9}$	$\underline{9}$	10	16	10	16	11	11	$\underline{11}$	$\underline{11}$	14	16	14	15	13	13
6	$\underline{8}$	$\underline{8}$	11	12	11	14	11	11	$\underline{15}$	$\underline{15}$	16	19	16	16	16	16
7	$\underline{8}$	$\underline{8}$	12	15	15	18	15	15	$\underline{8}$	$\underline{8}$	14	9	14	9	14	14
8	$\underline{9}$	$\underline{9}$	14	12	13	19	13	13	$\underline{16}$	$\underline{16}$	19	18	18	18	19	19
9	$\underline{10}$	$\underline{10}$	13	14	15	17	13	13	$\underline{13}$	$\underline{13}$	15	16	15	15	15	15
10	$\underline{11}$	$\underline{11}$	13	12	13	13	13	13	$\underline{10}$	$\underline{10}$	13	11	13	12	13	13
11	$\underline{10}$	$\underline{10}$	12	17	11	15	13	13	$\underline{14}$	$\underline{14}$	17	16	17	17	16	16
12	$\underline{10}$	$\underline{10}$	13	13	13	17	16	16	$\underline{13}$	$\underline{13}$	13	17	13	14	17	17
13	$\underline{9}$	$\underline{9}$	11	12	12	14	12	12	$\underline{10}$	$\underline{10}$	11	$\underline{10}$	11	10	11	11
14	$\underline{15}$	$\underline{15}$	16	17	16	16	16	16	$\underline{11}$	$\underline{11}$	12	13	12	11	12	12
15	$\underline{13}$	$\underline{13}$	14	15	15	17	15	15	$\underline{11}$	$\underline{11}$	14	15	14	14	14	14
16	$\underline{10}$	$\underline{10}$	13	15	11	14	14	14	$\underline{12}$	$\underline{12}$	16	13	16	16	16	16
17	$\underline{8}$	$\underline{8}$	11	11	11	14	11	11	$\underline{12}$	$\underline{12}$	15	17	15	13	19	19
18	$\underline{9}$	$\underline{9}$	10	11	10	15	10	10	$\underline{14}$	$\underline{14}$	15	15	15	17	15	15
19	$\underline{6}$	7	10	10	11	13	12	12	$\underline{13}$	$\underline{13}$	18	19	18	14	18	18
20	$\underline{13}$	$\underline{13}$	17	18	17	21	17	17	$\underline{9}$	10	12	13	12	11	12	12
21	$\underline{15}$	16	18	18	18	18	18	18	$\underline{11}$	$\underline{11}$	14	14	14	13	15	15
22	$\underline{9}$	$\underline{9}$	12	16	12	15	12	12	$\underline{13}$	$\underline{13}$	16	15	16	14	16	16
23	$\underline{11}$	$\underline{11}$	14	14	14	14	14	14	$\underline{15}$	$\underline{15}$	16	17	16	17	16	16
24	$\underline{10}$	$\underline{10}$	13	15	13	15	15	15	$\underline{9}$	$\underline{9}$	12	10	12	11	13	13
25	$\underline{9}$	$\underline{9}$	12	14	13	14	13	13	$\underline{10}$	$\underline{10}$	14	$\underline{10}$	14	11	14	14
26	$\underline{10}$	$\underline{10}$	12	16	13	13	13	13	$\underline{13}$	$\underline{13}$	14	15	14	14	14	14
27	$\underline{10}$	$\underline{10}$	12	15	14	16	14	14	$\underline{11}$	$\underline{11}$	11	12	11	14	11	11
28	$\underline{11}$	$\underline{11}$	15	16	15	15	15	15	$\underline{12}$	$\underline{12}$	15	16	16	13	16	16
29	$\underline{13}$	$\underline{13}$	16	18	16	16	16	16	$\underline{10}$	$\underline{10}$	14	13	14	12	15	15
30	$\underline{9}$	$\underline{9}$	13	14	13	14	10	10	$\underline{9}$	10	10	13	10	10	10	10

Note: $\underline{\text{Underline}}$ denotes the best solution for a given run.

were close to those of GASFTSP on the large-scale problem. However, GASFTSP was significantly preferred over the SST rule according to a pair-*t* test. Apart from GASFTSP, complete pair-*t* tests pointed out that the SST rule significantly outperformed the other rules on the large-scale problem whereas the SST and SSPT rules were significantly better than the RND, LRPT, SPT, and SRPT rules on the wide-range problem. Furthermore, the absence of best solutions for utilizing the SST rule to solve the wide-range problem implied that the benefit from merely emphasizing on the setup times between operations could not persist when the range of processing time became wider. In other words, the performance of a heuristic based a single rule correlates to not only the selected objective but also the profile of data.

In summary, GASFTSP was an approximate approach to SFTSP and consistently outperformed the other heuristics on the experimental problems. Although GASFTSP took 2–3 h in average to obtain a close-to-optimal solution for a problem with a structure and a data profile mimic to reality, it would be worthy to trade the computation time for solution quality. First of all, to our industrial knowledge, an engineer in charge of production planning and control can gather scheduling input data together in half a day before executing a daily schedule. In words, a two-hour computation ought to be reasonable. Secondly, the computation time of GASFTSP can be shortened via reducing number of generations for GA while sacrificing a little solution quality. Thirdly, the computation efficiency of GASFTSP can be further improved by incorporating several outperformed dispatching rules, e.g., SST and SSPT, as chromosomes into the initial population of GA. Finally, to redesign GASFTSP with respect to a different objective function requires merely a change of the fitness function in the GA. That is, the effort to maintain the proposed GASFTSP is limited. As a consequence, the proposed GASFTSP is viable to a realistic SFTSP.

6. Conclusion

In this study, we developed a MILP model of the testing scheduling problem to determine the optimal schedule in which most important characteristics of SFTSP were considered. The proposed MILP comprised a number of details which have not been explicitly modeled in the literature so that optimality of the problem or the approximation of a proposed heuristic can be examined. Furthermore, an assignment algorithm was developed to determine the corresponding machine configurations and graphically show the optimal schedule in timetables.

Table 4
The simulation results for the large-scale and wide-range problems

Run	Large-scale problem							Wide-range problem						
	GA	SST	RND	SSPT	LRPT	SPT	SRPT	GA	SST	RND	SSPT	LRPT	SPT	SRPT
1	119	169	186	136	155	168	133	255	301	386	289	387	309	292
2	110	123	120	155	179	139	134	250	288	333	297	369	339	299
3	124	121	133	187	198	166	161	208	229	301	240	353	266	253
4	138	137	161	202	195	174	179	215	255	303	251	342	292	274
5	113	129	124	164	192	137	140	217	273	291	250	311	269	255
6	122	128	137	182	177	159	157	249	298	365	312	390	290	309
7	139	151	155	193	198	181	190	237	270	361	288	379	306	333
8	125	141	152	166	177	171	158	208	224	293	250	372	274	294
9	103	125	118	164	173	126	145	246	308	311	253	400	306	304
10	125	130	130	167	212	149	148	225	245	309	282	382	300	303
11	113	114	131	173	177	148	143	207	248	280	242	331	240	282
12	124	140	134	188	205	156	159	195	234	278	232	362	259	246
13	125	141	140	181	200	161	157	235	240	327	263	323	282	288
14	122	131	123	179	211	145	145	206	230	320	241	365	271	275
15	118	132	134	166	171	148	146	278	334	400	300	431	343	345
16	148	163	159	209	204	187	192	181	237	255	213	274	237	219
17	115	121	123	163	188	141	139	222	239	282	249	336	295	311
18	103	113	117	143	174	132	134	206	252	303	232	335	233	277
19	114	116	128	166	196	142	150	237	272	308	257	387	290	298
20	118	125	132	175	186	156	161	233	306	352	279	380	304	330
21	107	112	113	162	185	127	132	285	322	408	379	425	368	359
22	122	137	134	176	189	158	153	217	253	314	245	383	309	286
23	127	133	144	183	196	164	152	240	277	325	281	378	324	295
24	112	124	120	158	181	135	146	266	282	363	315	353	336	325
25	113	120	125	168	202	140	143	250	273	324	278	410	282	312
26	100	104	113	143	173	130	122	252	293	385	299	388	343	299
27	134	141	153	197	186	172	174	214	254	296	232	359	260	261
28	129	148	144	207	196	178	162	279	364	433	307	425	391	404
29	129	147	149	191	204	174	162	218	246	330	255	361	268	280
30	123	139	139	168	172	158	170	193	227	288	230	304	234	263

Note: Underline denotes the best solution for a given run.

However, computer memory restriction and computation time limitation disabled the proposed model from generating an optimal schedule in real settings. Alternatively, a genetic algorithm (GASFTSP) minimizing the makepsan of the testing scheduling problem was proposed to reduce computation effort. The performance of GASFTSP was compared with the heuristics applying various single dispatching rules. After a number of simulation runs, GASFTSP was found consistently to outperform the other ruled-based heuristics with respect to the objective of minimizing makespan. Additionally, GASFTSP was shown to consistently obtain a near-optimal schedule within a reasonable computing time and with low costs to maintain. A decision support system (DSS) utilizing the proposed GASFTSP with a graphical user interface has been developed to assist the engineers in testing scheduling in an industrial-collaborative semiconductor final testing facility.

Future research can be done to improve efficiency of the proposed GASFTSP. One possible direction would be incorporating several single dispatching rules as chromosomes into the initial population of GA instead of all randomly generating combinations of dispatching rules as chromosomes. On the other hand, investigating novel dispatching rules could also contribute to ease of computation. Moreover, the computing effort and solution quality of the genetic algorithm are greatly influenced by the factors including encoding and decoding methods, selection strategies, crossover and mutation operators, and crossover and mutation rates. With efficiency improvement, the proposed GASFTSP can possibly deal with dynamic job arrival problems. Consequently, schedulers can continuously run daily schedules and reschedule whenever a disruption occurs (Perry & Uzsoy, 1993). Continuously well-performed scheduling will increase long-term testing profit. In addition, further research should be done to develop effective indices to evaluate the performance of machine configurations (Chien, Chen, Wu, & Hu, 2007). Furthermore, it would be beneficial to examine the sensitivity of solution quality with respect to the availability of different accessories to see possible trade-offs between accessory purchase and productivity performance, e.g., makespan or total weighted tardiness.

Acknowledgements

This research is partially supported by National Science Council (NSC95-2221-E-007-125; NSC 96-2213-E-007-002) and Macronix International Co., Ltd. The authors appreciate generous assistance from the Macronix Backend Operation Center. Special thanks go to Allen S.C. Dai and Chung-Jen Kuo for their assistance in empirical studies and Professor Mitsuo Gen for his inspirational advice in his visit to Taiwan.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job-shop scheduling. *Management Science, 34*(3), 391–401.

Beasley, J. E., & Chu, P. C. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research, 94*, 392–404.

Chen, T. R., Chang, T. S., Chen, C. W., & Kao, J. (1995). Scheduling for IC sort and test with preemptiveness via Lagrangian relaxation. *IEEE Transactions on System, Man, and Cybernetics, 25*(8), 1249–1256.

Chen, T. R., & Hsia, T. C. (1994). Job shop scheduling with multiple resources and an application to a semiconductor testing facility. In *IEEE Proceedings of the 33rd conference on decision and control* (pp. 1564–1570).

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms: I. Representation. *Computers and Industrial Engineering, 30*, 983–997.

Cheng, R., Gen, M., & Tsujimura, Y. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms: II. Hybrid genetic search strategies. *Computers and Industrial Engineering, 36*(2), 343–364.

Chien, C.-F., & Wu, J.-Z. (2003). Analyzing repair decisions in the site imbalance problem of semiconductor test machines. *IEEE Transactions on Semiconductor Manufacturing, 16*(4), 704–711.

Chien, C.-F., Wang, W.-C., & Cheng, J.-C. (2007). Data mining for yield enhancement in semiconductor manufacturing and an empirical study. *Expert Systems with Applications, 33*(1), 192–198.

Chien, C.-F., Chen, H.-K., Wu, J.-Z., & Hu, C.-H. (2007). Construct the OGE for promoting tool group productivity in semiconductor manufacturing. *International Journal of Production Research, 45*(3), 509–524.

Chien, C.-F., & Chen, C. (2007a). Using GA and CTPN for Modeling the optimization-based schedule generator of a generic production scheduling system. *International Journal of Production Research, 45*(8), 1763–1789.

Chien, C.-F., & Chen, C. (2007b). A novel timetabling algorithm for a furnace process for semiconductor fabrication with constrained waiting and frequency-based setups. *OR Spectrum, 29*(3), 391–419.

Chikamura, A., Nakamae, K., & Fujioka, H. (1998). Influence of lot arrival distribution on production dispatching rule scheduling and cost in the final test process of LSI manufacturing system. *IEE Proceedings-Science Measurement and Technology, 145*(1), 26–30.

Dagli, C. H., & Sittisathanchai, S. (1995). Genetic neuro-scheduler: A new approach for job shop scheduling. *International Journal of Production Economics, 41*, 135–145.

Dengiz, B., Altiparmak, F., & Smith, A. E. (1997). Efficient optimization of all-terminal reliable netoperations using evolutionary approach. *IEEE Transactions on Reliability, 46*(1), 18–26.

Dorndorf, U., & Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers and Operations Research, 22*, 25–40.

Freed, T., & Leachman, R. C. (1999). Scheduling semiconductor device test operations on multihead testers. *IEEE Transactions on Semiconductor Manufacturing, 12*(4), 523–530.

Gen, M., & Cheng, R. (1996). Optimal design of system reliability using interval programming and genetic algorithms. *Computers and Industrial Engineering, 31*(1-2), 237–240.

Gen, M., & Cheng, R. (1997). *Genetic algorithms and engineering design*. New York, NY: John Wiley and Sons.

Hinterding, R. (1998). Mapping, order-independent genes and the knapsack problem. In *Proceedings of the first IEEE conference on evolutionary computation*, pp. 13–17.

ILOG, Inc. (2003), ILOG CPLEX 9.0, user manual.

Kobayashi, S., Ono, I., & Yamamura, M., 1995. An efficient genetic algorithm for job shop scheduling problems. In *Proceedings of the sixth international conference on genetic algorithms*, pp. 506–511.

Kubota, N., & Fukuda, T. (1996). Schema representation in virus-evolutionary genetic algorithm for knapsack problem. In *Proceedings of the IEEE international conference on evolutionary computation*, pp. 834–838.

Leachman, R. C. (1993). *Modeling techniques for automated production planning in the semiconductor industry*. New York, NY: John Wiley and Sons.

Lee, C. Y., Uzsoy, R., & Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research, 40*(4), 764–775.

Lin, J., Wang, F., & Lee, W. (2004). Capacity-constrained scheduling for a logic IC final test facility. *International Journal of Production Research, 42*(1), 79–99.

Moon, C., Kim, C. K., & Gen, M. (1999). Genetic algorithm for maximizing the parts flow within cells in manufacturing cell design. *Computers and Industrial Engineering, 2*, 1730–1733.

Ovacik, I. M., & Uzsoy, R. (1996). Decomposition methods for scheduling semiconductor testing facilities. *International Journal of Flexible Manufacturing Systems, 8*(4), 357–388.

Perry, C. N., & Uzsoy, R. (1993). Reactive scheduling of a semiconductor testing facility. *IEEE/CHMT International Electronics Manufacturing Technology Symposium*, 191–194.

Reeves, C. (1995). A genetic algorithm for flow shop scheduling. *Computers and Operations Research, 22*, 5–13.

Storer, R., Wu, S., & Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science, 38*(10), 1495–1510.

Uzsoy, R., Church, L. K., & Ovacik, I. M. (1992). Dispatching rules for semiconductor testing operations: a computational study. *IEEE/CHMT International Electronics Manufacturing Technology Symposium*, 272–276.

Uzsoy, R., Lee, C. Y., & Martin-Vega, L. A. (1992). Scheduling semiconductor test operations: minimizing maximum lateness and number of tardy jobs on a single machine. *Naval Research Logistics, 39*, 369–388.

Uzsoy, R., Martin-Vega, L. A., Lee, C. Y., & Leonard, P. A. (1991). Production scheduling algorithms for a semiconductor test facility. *IEEE Transactions on Semiconductor Manufacturing, 4*(4), 270–280.