https://doi.org/10.20965/jaciii.2015.p0058

Paper:

Comparison of Knowledge Acquisition Methods for Dynamic Scheduling of Wafer Test Processes with Unpredictable Testing Errors

Tsubasa Matsuo*, Masahiro Inuiguchi*, and Kenichiro Masunaga**

*Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
E-mail: {matsuo@inulab., inuiguti@}sys.es.osaka-u.ac.jp
**Renesas Electronics Co.
2-6-2 Otemachi, Chiyoda-ku, Tokyo 100-0004, Japan
E-mail: kenichiro.masunaga.te@renesas.com
[Received April 20, 2014; accepted August 25, 2014]

The scheduling of semiconductor wafer testing processes may be seen as a resource constraint project scheduling problem (RCPSP), but it includes uncertainties caused by wafer error, human factors, etc. Because uncertainties are not simply quantitative, estimating the range of the parameters is not useful. Considering such uncertainties, finding a good situation-dependent dispatching rule is more suitable than solving an RCPSP under uncertainties. In this paper we apply machine learning approaches to acquiring situation-dependent dispatching rule. We compare obtained rules and examine their effectiveness and usefulness in problems with unpredictable wafer testing errors.

Keywords: semiconductor manufacturing, simulation, genetic-based machine learning

1. Introduction

The semiconductor manufacturing process is divided into front-end and back-end. Front-end manufacturing is the process of making integrated circuits from blank wafers. Back-end manufacturing involves tests, assembly and packaging, in which the completed wafer is split into individual chips. Semiconductor manufacturing is known as high-mix low-volume manufacturing. Much time is needed to complete a product, which is why just-in-time inventory systems cannot be applies to the semiconductor industry. Semiconductor production lead time is about three months and the product life cycle is about six months. This means that the product process is redesigned at short intervals and not stable. The key to semiconductor manufacturing is effective, efficient process management.

Three major scheduling approaches are taken to general flexible manufacturing systems with uncertainty [1]. The first method is completely reactive scheduling such as using dispatching rules, which set the priority for jobs waiting for machine processing. No firm schedule is generated in advance and decisions are made locally in real time. The second method is predictive-reactive scheduling, which makes a schedule in advance and revises it in response to real-time events. The third method is robust proactive scheduling, which sets predictive schedules predictably satisfying performance requirements in a dynamic environment. Several methods have been proposed to deal with semiconductor manufacturing scheduling in flexible manufacturing systems. Ovacik and Uzsoy dealt with the final-test process using multiple test equipment [2]. They modeled the process using a disjunctive graph and decomposed it. They solved a constituent problem using Rolling Horizon, showing that using decomposition and Rolling Horizon is more suitable than using simple dispatching rules. Shen and Leachman discussed the scheduling problem in the wafer fabrication process [3], assuming that uncertainties arise in demand forecast, yield and equipment capacity. They proposed robust solution methodology based on stochastic linear quadratic (SLQ) optimal control theory and used simulation to compare it to LP Rolling Horizon and simple dispatching rules. SLQ theory was best for minimizing standard variation in yield. These two studies deal with small-scale problems, so treat large-scale problems, Wu and Chien proposed using a greedy algorithm, i.e., a sequence of dispatching rules and simulations [4]. They used a genetic algorithm (GA) [5] to obtain a sequence of dispatching rules and compared it to simple dispatching rules, showing that the proposed approach is the best among them.

In this paper, we examine a scheduling problem in the wafer test process, which is the final process of front-end manufacturing. The more desired function and performance are enhanced, the higher the cost of the wafer test process. This is because test time expands and test equipment becomes more expensive with increasing requested integration degree and memory size of the chipset. We must conduct several tests to check wafer properties in the

wafer test process. The test equipment is not identical. In other words, some jobs (products) cannot be tested with the same equipment used to test others. We thus must ensure both the suitability and status of test equipment. The two jigs required for testing depend on how equipment and jobs are paired, in other words, jigs must be selected to match the equipment and job. Setup time is necessary to regulate temperature and to replace jigs. The scheduling problem in the wafer test process is eventually considered a resource constraint product scheduling problem (RCPSP). Wu and Chien [4] proposed approaching this problem heuristically. As stated, their method is designed to handle large-scale problems, and they propose using a greedy algorithm to select test equipment and jigs, along with a sequence of dispatching rules to assign a job. The sequence for dispatching rules corresponds to the chromosome in GA. A chromosome is evaluated by simulation. By using numerical experiments in a deterministic environment, Wu and Chien proved that their proposed method performed better than simple dispatching rules. Real-world problems include the uncertainty of testing errors, however, the advantage in a deterministic environment will not necessarily be preserved in a nondeterministic environment. Matsuo et al. [6] used the Pitts approach [7], i.e., a genetic-based machine learning (GBML) approach, to obtain a situation-dependent dispatching (SDD) rule [8], which is the approach used in the real-world scheduling problem in wafer testing. Unlike the rescheduling approach to uncertainty, the SDD rule approach may be seen as a trouble-resolution-free approach. That is the rescheduling approach must resolve the scheduling problem if an accident happens. In contrast, the SDD approach applies dispatching rules based on the situation. The SDD rule acquired by the Pitts approach performed better than simple dispatching rules if a testing error occurred.

In the sections that follow, we apply three machine learning methods to find an appropriate SDD rule, executing the machine learning method in a deterministic environment. This execution is done for problems formulated by using real-world data. We show that the SDD rule obtained works well even in nondeterministic environments.

This paper is organized as follows: in Section 2, we present the problem setting. In Section 3, we explain job allocation based on a given SDD rule and the dispatching rules considered in this study. In Section 4, we introduce an application of C4.5, the algorithm used for generating a decision tree. In Section 5, we present an improved way of applying the Pitts approach. In Section 6, we apply genetic programming (GP). In Section 7, the performances of approaches are examined and analyzed in numerical experiments using real-world data. In Section 8, we present concluding remarks.

2. Problem Setting

Figure 1 shows the wafer test process, in which products must be tested several times, and some kinds of products are trimmed in the mid-flow. In the wafer test process, we must decide how and when to allocate operations and jigs to test equipment. A batch of wafers is processed in an operation. The jigs we used in the wafer test process are probe cards and performance boards. The jigs used depend on the combination of test equipment and operations. Operations include information on the product class. Equipment processes only single batch at a time. Replacing jigs takes about 10 minutes. If required process temperatures differ between consecutive operations, a break of a few hours is needed to adjust the temperature. If testing errors occur in a wafer batch, subsequent operations must be canceled for wafer investigation and another wafer batch is reloaded to the test equipment. Given the unpredictability of errors, their nonnegligible frequency and their significant influence, it is not a good idea to find an optimal schedule a priori assuming that no error will occur. It is preferable to take a reactive scheduling approach that allocates a job and corresponding jigs to equipment as requested. Dispatching rules are often used for reactive scheduling. To adapt to many possible situations, simple dispatching rules are not enough, meaning that some mechanism for adapting a dispatching rule to the situation is desirable. Finding such a mechanism is difficult so, as a simple alternative, we consider a situation-dependent dispatching (SDD) rule that changes the applied dispatching rule based on the situation.

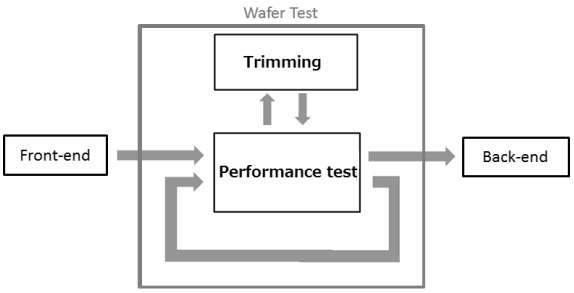

Fig. 1. Wafer test process in semiconductor manufacturing.

3. SDD Rule and Allocation Algorithm

The job allocation algorithm based on a given SDD rule allocates a job to test equipment. The allocation algorithm is executed whenever jobs are waiting and equipment is idle. An SDD rule is a tree whose leaves are associated with dispatching rules. At every non-leaf node, the tree has a branch with a condition about the production situation. Given an SDD, we check conditions from the root to the leaf using branches whose conditions are satisfied. When we reach a leaf. the associated dispatching rule is applied, so the dispatching rule applied depends on the production situation under the SDD rule.

Criteria used to evaluate the production situation. are as follows:

Launched operation rate: The number of launched op-

erations processed per minute
Job completion rate: The number of completed jobs per minute. A job consists of operations necessary for the test process, so this rate differs from the launched operation rate. A single job may include many wafers.
Wafer completion rate: The number of wafers completed per minute. Completion implies that all operations necessary for the test process are finished, so the wafer completion rate differs from the job completion rate. The number of wafers in a lot varies so no one-to-one correspondence exists between the wafer completion rate and the job completion rate.
Delayed and uncompleted job: The number of uncompleted jobs that are behind in their due dates
Delayed and uncompleted wafer: The number of uncompleted wafers that are behind in their due dates

No one-to-one correspondence exists between delayed and uncompleted jobs and delayed and uncompleted wafers.
Least slack time: The minimum slack time among jobs. Slack time is the difference between the remaining time before a due date and the sum of process times of remaining operations.
Wasted time: The sum of setup time, temperature adjustment time, and idling time over all test equipment.

We use the following six dispatching rules as candidates:
FIFO: First in, first out rule, i.e., the job which comes first in the queue
SPT: The shortest processing time rule, i.e., the job with the shortest processing time in the queue
EDD: The earliest due date rule, i.e., the job with the earliest due date in the queue
SLACK: The least slack time rule, i.e., the job with the least slack time in the queue
LWKR: The least work remaining rule, i.e., the job with the smallest sum of processing time for operations remaining before completion is selected from the queue
MP: The most profit rule. The most profitable job is selected from the queue.

SDD rule representation differs with the method. Individual representation is described in Sections 4 to 6.

Before describing the approaches proposed to acquire an SDD rule, we explain the algorithm for job allocation to test equipment under the constraints in the previous section. Once the algorithm is applied, it determines the schedule, i.e., the value of the criterion for the schedule is obtained. Our algorithm is a modified version of the algorithm proposed by Wu and Chien. [4]. Although the algorithm considers the constrained allocation of jigs to equipment and the setup time needed for jig changes, it does not consider wafer testing error or the waiting time for adjusting equipment temperature. We introduce these parameters into our algorithm.

In the first step of our algorithm, we choose test equipment with the longest waiting time. In the second step, we allocate a job chosen by the SDD rule. In the third step, we allocate jigs with the longest waiting time. In the fourth step, we update parameters showing the production situation, e.g., start time and completion time. We repeat these four steps within the planning period. If an error occurs during a wafer test, we remove the job from the list of those to be processed and put it in the waiting list for error investigation.

Applying this algorithm to an SDD rule, we measure its scheduling performance. In other words, an SDD rule is evaluated using numerical simulation based on the algorithm. It is assumed, in numerical simulation that an error occurs randomly.

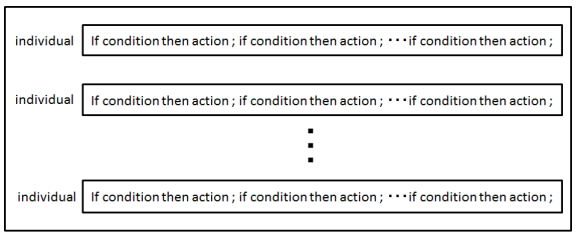

Fig. 2. Individuals and population in the Pitts approach.

4. SDD Rule Acquisition by C4.5

Among the ways used to find and represent a suitable SDD rule, C4.5 acquires a decision tree based on information entropy [9]. We call the decision tree as an SDD rule. C4.5 needs a decision table to acquire a decision tree. We set the dispatching rule as the decision class, rather than the lot, so that we can get the decision table from the scheduling result produced by the GA method [4]. Testing errors are unpredictable even though the probability of occurrence is estimated. Nevertheless, we explore the suitable SDD rule in an environment where no testing error occurs. Then we examine whether the SDD rule obtained in the environment of no testing error works well in environments where testing errors occur.

5. SDD Rule Acquisition by the Pitts Approach

The Pitts approach [6] is appropriate for offline learning and regards a rule group as an individual, evaluating a group of rules. The individuals and population of the Pitts approach are shown in **Fig. 2**, together with the sequence of if-then rules as an individual seen as a linear tree. That is, the premise of an if-then rule corresponds to a non-leaf node and the conclusion to a leaf.

The approach proposed in [6] only sorts prepared if-then rules without considering thresholds and attributes. The decision-maker must make the if-then rules. We propose a new representation of the SDD rule for searching for an appropriate permutation of if-then rules, meaning that thresholds and attributes are selected automatically. The SDD rule is shown in **Fig. 3**. Because each if-then rule has the thresholds of the antecedent and the dispatching rule as a conclusion, it consists of real value codes

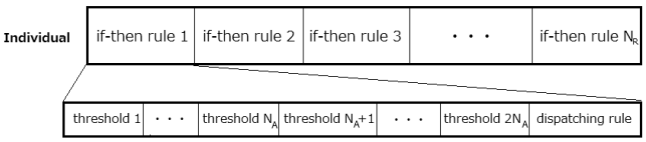

Fig. 3. The representation of SDD rule in the Pitts approach.

and an integer value code. The real value code takes a real value from 0 to 1, or 2 – the value from 0 to 1 represents the threshold and value 2 represents "don't care." The integer value code takes an integer value from 0 to 5, representing the dispatching rule. Here, we apply genetic operators such as selection, crossover, and mutation. The genetic operators used for GBML in this study are as follows:

Crossover: One-point crossover. We select one point as a part of each individual in the current population, with a crossover rate probability. If no individual is selected, we return to the first individual and continue probabilistic selection until two individuals are selected. If the number of selected individuals is odd, we return to the first individual and continue probabilistic selection until one additional individual is selected. This is the way we build a parent set. Adjacent parents in the parent set are paired and for each pair of parents, we choose a single crossover point. The codes before the crossover point of Parent 1 are copied to Child 1. The rest of the chromosome for Child 1 is filled with codes behind the crossover point of Parent 2. Child 2 is built in the same way by switching Parents 1 and 2.

This is an example of a crossover operator when the chromosome consists of 9 genes:

Parent 1	0.1	2	0.3	2	2	2	0.7	0.8	3
Parent 2	2	2	0.5	0.3	2	0.2	2	2	1
↓									
Child 1	0.1	2	0.3	0.3	2	0.2	2	2	1
Child 2	2	2	0.5	2	2	2	0.7	0.8	3

Mutation: We decide whether we generate a child using the mutation operator for each individual or not by the probability of the mutation rate. To generate a child, we select one gene randomly in the chromosome and replace it with a new random value.

This is an example of a mutation operator when the chromosome consists of 9 genes:

Parent	0.1	2	0.3	2	[2]	2	0.7	0.8	3
↓									
Child	0.1	2	0.3	2	[0.4]	2	0.7	0.8	3

Selection: Let N be the population size. Collecting children generated by the crossover operator, children generated by the mutation operator and individuals in the current generation, we build an enlarged population. We choose N individuals as the population for the next generation from the enlarged population by using ranking selection and elite preservation:

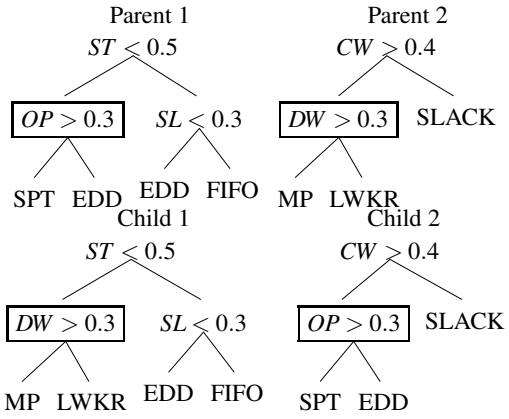

Fig. 4. Example of crossover in GP.

Ranking selection: We rank individuals based on their fitness function values, then assign probability $p(n)$ to the individual with the n-th rank. In this study, the probability $p(n)$ is defined as

$$p(n) = \frac{2(N-n)}{N(N-1)}.$$

We apply a roulette selection, so that the individual with the n-th rank has selection probability $p(n)$. We repeat individual selection by using roulette $(N-l)$ times, where l is the number of elite individuals.

Elite Preservation: We select the top l individuals in the current population and keep them in the next generation.

6. SDD Rule Acquisition by GP

We consider the SDD rule acquired by the Pitts approach as a linear decision tree, and try to acquire a more complex decision tree as an SDD rule by applying GP. GP is an extended GA for representing trees (data structure). A tree is generally translated omto an S-expression used in LISP. We also use the S-expression as a genotype. Genetic operators used for GP are as follows:

Crossover: Parents are chosen the same as used in GBML. We select one node as a crossover point for each parent. We swap subtrees below the crossover point between parents to become children. **Fig. 4** is an example of a crossover operator when the chromosome is the tree.

Mutation: Parents are chosen the same as used in GBML. We select a node and swap the subtree for a new subtree produced randomly. **Fig. 5** is an example of a mutation operator when the chromosome is the tree.

Selection: This is the same as used in GBML.

We set the max depth of the tree to keep the tree from becoming too big. If a tree is too big to be a child, we retry the operator.

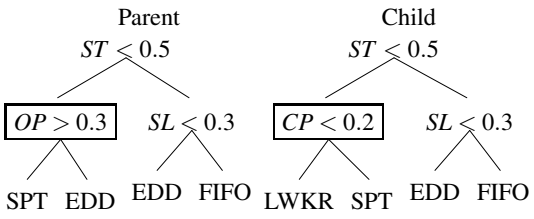

Fig. 5. Example of mutation in GP.

7. Experiment Environment

An individual corresponds to an SDD rule, so we obtain scheduling performance by using simulation as described in the previous section. Based on scheduling performance, we define a fitness value. We use penalties that consider improvement of the operation rate, tardiness, and throughput. Three kinds of penalties are considered: *SPenalty* for idling time, such as setup time and waiting time; *DPenalty* for the number of delayed wafers and delay time; and *CPenalty*, the number of completed wafers. These are modeled by

$SPenalty =$ (Sum of tester idling time),
$DPenalty =$ (Sum of the product of the number of delayed wafers and delay time),
$CPenalty =$ (Sum of the number of complete wafers).

Delay time is measured by the minute. Penalties are normalized by the difference in the two extreme values in order to make scales uniform. The two extreme values are set by a preliminary experiment and may be exceeded. Fitness is defined by aggregating penalties described above, i.e.,

$$Fitness = w_1 \times SPenalty + w_2 \times DPenalty, + w_3 \times (1 - CPenalty),$$

where w_1, w_2, $w_3 \geq 0$ are weights. Because fitness is a penalty function value, the smaller the fitness value, the better the individual.

The genetic algorithm procedure we used is shown in **Fig. 6**.

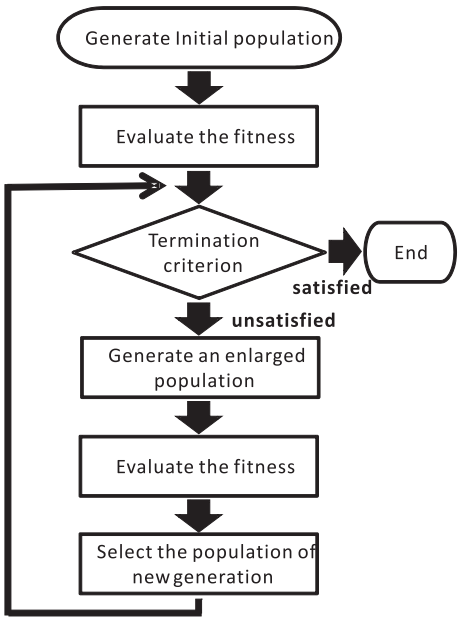

Fig. 6. GA algorithm.

8. Numerical Experiments

To confirm the usefulness of the proposed methods, we conducted numerical experiments based on data obtained from a real-world wafer factory. By the proposed method, we generate SDD rules suitable for the given production environment with discarding testing error. In this environment, we compare the efficiency of SDD rules generated by different acquisition techniques. Then we confirm the robustness of the obtained SDD rules by applying them to the production environment with testing error. The common reactive scheduling, Left-Shift (LS) and simple dispatching rules are also applied for comparison. The LS method means left shifting all the remaining jobs together in the time horizon so that the disruption length is accommodated but the job sequence remains unchanged. The job sequence is acquired in advance by the GA approach.

We now describe real-world data in our scheduling problem and the parameters of our proposed method. The scheduling period is 1 day. There are jobs delayed from the beginning. Several jobs have due dates on this day. There are many test equipments and jobs to be processed. The data scale is big. Test error probability and setup time for replacing jigs and adjusting temperature are assumed to be constant reflecting the company opinion. Temperature adjustment takes much longer than jig replacement. Operation time varies widely with the type of operation. The due date is regarded as a soft constraint rather than a hard constraint. Therefore, even if the due date is not met, the schedule remains feasible by paying the *DPenalty* in the previous section.

Weights of *SPenalty*, *DPenalty*, and *CPenalty* for *Fitness* are determined as $(w_1, w_2, w_3) = (1,1,1)$, $(1,1,100)$, $(1,100,1)$, $(1,100,100)$, $(100,1,1)$, $(100,1,100)$, $(100,100,1)$. GA parameters are selected based on preliminary experiments. In the Pitts approach, a chromosome consists of 750 genes. Population size N is 200. The algorithm terminates after the 50th generation run. The crossover rate is set to 0.5 and the mutation rate is set to 0.5. The top 2 individuals ($l = 2$) survive in the next generation for elite preservation. In GP, the max tree depth is 7. Other GA parameters are the same as in the Pitts approach. We execute the four approaches in an environment where no testing error occurs.

Comparing these SDD rules and the sequence of dispatching rule (SDR) obtained by GA where no testing error occurs, we obtained the results in **Table 1**. The best value in each column is underlined, the SDD rule acquired by Pitts approach is represented by SDD_{Pitts}, and other SDD rules are represented in similar form. $SDD_{[6]}$ is the

Table 1. Evaluation of SDD rule acquisition.

rule	(1,1,1)	(1,1,100)	(1,100,1)
SDD_{Pitts}	1.016	23.034	2.317
SDD_{GP}	1.016	22.758	2.323
$SDD_{C4.5}$	1.419	23.022	35.101
$SDD_{[6]}$	1.195	23.034	2.639
SDR/LS	0.953	22.282	2.230
(1,100,100)	(100,1,1)	(100,1,100)	(100,100,1)
41.639	27.928	76.507	32.502
41.724	24.333	54.884	29.354
47.889	54.674	79.522	82.423
76.415	35.563	79.177	37.938
41.566	21.204	51.570	25.883

SDD rule proposed in [6]. From the results in **Table 1**, we found that SDR is the best. This is a natural result because SDR fits the problem most flexibly. SDR, however considers only the sequence, not the situation, so it may not adapt to changes.

To examine the robustness of the obtained SDD rule, we compared the obtained SDD rules to LS through 100 simulations in an environment where testing errors occur. The job sequence used in LS was acquired by SDR in an environment where no testing errors occur. LS and SDR thus have the same result, which is better than SDD rules in the case of no testing errors.

Results when $(w_1,w_2,w_3) = (1,1,1)$, $(1,100,100)$, $(100,1,100)$ are shown in **Table 2**. When $(w_1,w_2,w_3) = (1,1,1)$, LS is the best rule, but when $(w_1,w_2,w_3) = (1,100,100)$, SDD_{Pitts} and SDD_{GP} are better than LS. When $(w_1,w_2,w_3) = (100,1,100)$, SDD_{GP} is better than LS, and SDD_{Pitts} is better than LS only for a high error rate. We confirmed that SDD_{Pitts} is superior to $SDD_{[6]}$.

In the case of other weights, such as when $(w_1,w_2,w_3) = (1,100,1)$, $(1,1,100)$, we obtained results similar to when $(w_1,w_2,w_3) = (1,100,100)$. When $(w_1,w_2,w_3) = (100,1,1)$, $(100,100,1)$, the result was similar to the case when $(w_1,w_2,w_3) = (1,1,1)$. These results show that the SDD rule acquired by evolutionary computing performs well when the weight of *DPenalty* or that of *CPenalty* is big. This is because we do not use the dispatching rule considering *SPenalty*, so when the weight of *SPenalty* is big, SDD rule performance becomes worse.

We checked how many times dispatching rules are used in each SDD rule where no test error occurs.

As shown in the results in **Table 3**, all dispatching rules are used a dozen times in SDR when $(w_1,w_2,w_3) = (1,1,1)$, namely, when LS is the best. When $(w_1,w_2,w_3) = (1,100,100)$, however, namely, when SDD_{Pitts} and SDD_{GP} are better than LS, only SPT and EDD are used in almost all cases in SDR. In the case of other weights, when $(w_1,w_2,w_3) = (1,100,1)$, $(100,1,1)$, $(100,100,1)$, all dispatching rules are used a dozen times in SDR. When $(w_1,w_2,w_3) = (1,1,100)$, $(100,1,100)$, only one or two dispatching rules are almost always used in SDR.

Table 2. Evaluation of the robustness of obtained SDD rules.

rule	error rate	mean	min	max	var
$(w_1,w_2,w_3) = (1,1,1)$					
SDD_{Pitts}	low	1.037	0.891	1.251	0.00322
	mid	1.057	0.794	1.412	0.00759
	high	1.058	0.844	1.412	0.00598
SDD_{GP}	low	1.046	0.891	1.703	0.00884
	mid	1.076	0.794	1.843	0.0231
	high	1.082	0.844	1.53	0.0144
$SDD_{C4.5}$	low	1.483	1.346	1.789	0.0136
	mid	1.497	1.296	1.955	0.0165
	high	1.546	1.296	1.926	0.0167
$SDD_{[6]}$	low	1.289	0.921	1.688	0.0223
	mid	1.391	0.944	1.766	0.0303
	high	1.419	0.919	1.787	0.0298
SDR	low	1.112	0.951	1.630	0.0341
	mid	1.197	0.917	1.609	0.0368
	high	1.240	0.947	1.616	0.0311
LS	low	1.007	0.953	1.449	0.0104
	mid	1.046	0.953	1.449	0.0166
	high	1.071	0.953	1.633	0.019
$(w_1,w_2,w_3) = (1,100,100)$					
SDD_{Pitts}	low	43.923	25.129	65.007	26.124
	mid	45.847	25.049	71.270	61.710
	high	46.385	22.029	74.603	61.160
SDD_{GP}	low	43.878	25.194	65.091	26.633
	mid	46.184	25.114	71.335	64.653
	high	46.418	22.094	74.668	62.617
$SDD_{C4.5}$	low	46.794	25.127	71.257	37.215
	mid	47.013	25.047	72.689	67.486
	high	46.763	22.029	76.699	67.057
$SDD_{[6]}$	low	78.798	69.434	103.874	56.206
	mid	81.882	58.849	108.077	126.176
	high	84.509	59.079	104.571	157.464
SDR	low	50.201	41.440	70.593	43.880
	mid	53.902	38.934	83.561	78.473
	high	53.564	42.514	84.222	47.829
LS	low	45.230	41.566	65.093	31.119
	mid	49.074	41.566	73.340	71.515
	high	49.722	41.566	83.860	66.889
$(w_1,w_2,w_3) = (100,1,100)$					
SDD_{Pitts}	low	80.122	76.504	98.107	17.913
	mid	83.149	76.307	97.022	32.221
	high	86.288	74.507	107.275	50.676
SDD_{GP}	low	56.982	41.171	73.215	24.015
	mid	58.625	18.184	82.047	76.164
	high	56.300	24.124	88.773	135.524
$SDD_{C4.5}$	low	82.621	76.507	101.122	16.709
	mid	84.875	76.507	97.022	27.352
	high	87.397	74.507	107.272	43.250
$SDD_{[6]}$	low	81.946	65.940	106.603	32.942
	mid	84.281	52.030	117.292	87.675
	high	82.255	42.144	113.099	177.601
SDR	low	67.669	51.570	105.877	235.388
	mid	77.928	51.570	106.667	244.170
	high	82.543	51.585	110.127	210.175
LS	low	65.683	51.570	124.645	358.093
	mid	74.959	51.570	157.840	523.889
	high	90.578	51.570	193.250	840.950

Thus, if there are only a few appropriate dispatching rules for a problem, the SDD rule obtained by evolutionary computing may work well. If the problem requires a variety of dispatching rules, however, it is hard to acquire a good SDD rule.

We check the number of occurrences of condition at-

Table 3. Number of applications of dispatching rules.

rule	FIFO	SPT	LWKR	MP	EDD	SLACK
$(w_1,w_2,w_3)=(1,1,1)$						
SDD_{Pitts}	0	0	48	0	55	0
SDD_{GP}	0	0	46	0	57	0
SDD_{C45}	12	14	48	42	6	0
SDR	13	14	17	18	15	13
$(w_1,w_2,w_3)=(1,100,100)$						
SDD_{Pitts}	0	106	0	0	36	0
SDD_{GP}	0	109	0	0	32	0
SDD_{C45}	0	104	0	0	37	0
SDR	1	89	2	0	33	6

Table 4. Number of occurrences of condition attributes.

rule	OP	CP	CW	DP	DW	SL	ST
$(w_1,w_2,w_3)=(1,1,1)$							
SDD_{Pitts}	1	1	0	0	1	0	1
SDD_{GP}	0	1	1	1	1	0	2
$SDD_{C4.5}$	10	8	0	0	0	4	6
$(w_1,w_2,w_3)=(1,100,100)$							
SDD_{Pitts}	0	0	0	0	0	0	1
SDD_{GP}	1	1	0	1	0	0	2
$SDD_{C4.5}$	0	0	0	0	0	0	2

Table 5. Evaluation of robustness of modified SDD_{Pitts}.

rule	error rate	mean	min	max	var
SDD'_{Pitts}	low	1.037	0.891	1.251	0.00322
	mid	1.057	0.794	1.412	0.00759
	high	1.058	0.844	1.412	0.00598

tributes.

As **Table 4** shows, $SDD_{C4.5}$, methods acquiring the SDD rule from the decision table, do not use DP and DW as a condition attribute. From the results, we say that DP and DW are redundant, so we then check SDD_{Pitts} performance without DP and DW as attributes, when $(w_1,w_2,w_3)=(1,1,1)$.

As shown in **Table 5**, the result is the same as before, that is DP and DW are redundant for this problem. Finally, we try to obtain new knowledge from SDD rules whose performance is good, i.e., we analyze SDD_{Pitts} and SDD_{GP} with $(w_1,w_2,w_3)=(1,100,100)$. We delete DP and DW from the SDD rule before analysis. **Fig. 7** shows acquired SDD rules.

When the operation processing rate is large, the testing flow goes smoothly. In other words, when the operation processing rate is large, time loss may be small, so SDD_{GP} is the same as SDD_{Pitts}. Both rules mean "If ST is low, i.e., if the sum of the setup times is low, EDD is chosen; otherwise, SPT is chosen." This gives us the new knowledge that if ST is low, we should choose EDD; otherwise, we should choose SPT. Other good SDD rules also have a simple structure. Our method of acquiring the SDD rule works well when the problem does not require a complex rule.

In executing the same numerical experiments with other data, we get data from the same factory on a different day and focus on the three best rules, i.e., SDD_{Pitts}, SDD_{GP}, and LS (see **Tables 6** and **7**).

We find that they yield similar results about rule performance. Comparing **Tables 3** and **8**, we find that the

SDD_{GP}, (w_1,w_2,w_3)=(1,100,100)

```
( if ( >= OP 0.48296 )
    ( apply SPT )
    ( if ( >= ST 0.50551 )   ( apply EDD )
        ( apply SPT ) ) )
```

SDD_{Pitts}, (w_1,w_2,w_3)=(1,100,100)

```
1. if ST >= 0.48198 then apply EDD
2. otherwise then apply SPT
```

Fig. 7. SDD rules acquired.

Table 6. Evaluation of SDD rule without testing errors in other data.

rule	(1,1,1)	(1,1,100)	(1,100,1)
SDD_{Pitts}	1.082	7.10	50.288
SDD_{GP}	1.082	6.750	46.132
SDR/LS	0.650	6.300	46.930
(1,100,100)	(100,1,1)	(100,1,100)	(100,100,1)
77.343	20.748	35.964	95.624
73.251	21.357	31.194	86.298
58.413	6.838	1.267	63.397

Table 7. Evaluation of robustness of obtained SDD rules in other data.

rule	error rate	mean	min	max	var
$(w_1,w_2,w_3)=(1,1,1)$					
SDD_{Pitts}	low	1.196	1.080	1.453	0.00589
	mid	1.339	1.116	1.693	0.0145
	high	1.457	1.146	1.966	0.0212
SDD_{GP}	low	1.196	1.080	1.453	0.00589
	mid	1.339	1.116	1.693	0.0145
	high	1.457	1.146	1.966	0.0214
LS	low	1.191	0.650	2.190	0.139
	mid	1.655	0.694	2.812	0.211
	high	1.979	0.770	2.798	0.243
$(w_1,w_2,w_3)=(1,100,100)$					
SDD_{Pitts}	low	86.186	69.940	163.311	227.153
	mid	95.474	69.811	163.519	281.236
	high	105.132	78.455	172.726	249.283
SDD_{GP}	low	84.948	71.408	117.703	77.976
	mid	96.777	77.190	122.342	111.177
	high	107.403	77.719	137.668	174.545
LS	low	91.505	58.413	187.067	999.957
	mid	120.781	63.965	204.540	1503.232
	high	147.051	71.270	230.847	1644.947
$(w_1,w_2,w_3)=(100,1,100)$					
SDD_{Pitts}	low	39.898	21.056	62.940	36.508
	mid	46.156	26.335	148.217	191.554
	high	48.924	31.610	75.848	93.193
SDD_{GP}	low	35.744	29.482	56.867	28.056
	mid	42.919	25.048	69.952	77.680
	high	48.295	31.072	80.533	105.118
LS	low	43.741	1.267	197.774	1583.854
	mid	77.903	1.336	192.512	2194.052
	high	106.041	1.401	236.665	2943.872

Table 8. Number of applications of dispatching rules in other data.

rule	FIFO	SPT	LWKR	MP	EDD	SLACK
$(w_1, w_2, w_3) = (1, 1, 1)$						
SDD_{Pitts}	47	116	0	0	0	0
SDD_{GP}	45	117	0	0	1	0
SDR	6	101	8	7	13	14
$(w_1, w_2, w_3) = (1, 100, 100)$						
SDD_{Pitts}	0	0	72	0	56	0
SDD_{GP}	0	128	0	0	38	0
SDR	7	7	87	4	15	5

Table 9. Evaluation of SDD rule with SST added.

rule	(100,1,1)	(100,1,100)	(100,100,1)
SDD_{Pitts}	24.928	54.703	28.785
SDD_{GP}	18.321	51.702	28.918
SDR/LS	18.179	51.226	25.735

numbers of applications of dispatching rules are different. We then say that the method for acquiring the SDD rule works well in the wafer test, but we must not reuse the same SDD rule on a different initial state. We must obtain another SDD rule for each problem, and must choose between SDD or LS based on the weight in the wafer test scheduling problem. In the real factory, we get the initial state of the next day by simulation – that is to say, we get the SDD rule overnight and apply it the day, meaning that this method is practical and effective in the wafer testing process.

From the previous analysis, we know that the candidate dispatching rule of the SDD rule is not sufficient, so we add another dispatching rule to the candidate and check whether the SDD rule is improved. There is no dispatching rule relevant to *SPenalty*, so we add the shortest setup time (SST) rule. The SST rule selects the job with the shortest setup time in the queue. We evaluate the SDD rule for a case in which the weight of *SPenlaty* is large. The result where no testing error occurs is shown in **Table 9**. Note that all approaches are improved by adding SST. Next, comparing the rules in an environment where testing errors occur, we get the results for a case in which the SDD rule does not include SST, as shown in **Table 10**. Results for the SDD rule including SST are shown in **Table 11**. When $(w_1, w_2, w_3) = (100, 1, 1)$, SDD_{GP} is better than LS afterwards, SDD_{GP} was worse than LS before adding SST, When $(w_1, w_2, w_3) = (100, 1, 100)$, $(100, 100, 1)$, SDD_{Pitts} gets the same result, so in conclusion, we say that we have succeeded in adding a good dispatching rule to the candidate based on analysis. As shown in **Table 12**, when the numbers of applications of rules in SDR are not distributed evenly, SDD rules are better than LS. As stated, we select a more appropriate approach from SDD or LS by checking the numbers of applications of rules in SDR. Of course, we also must set the appropriate candidate rules in advance.

Table 10. Evaluation of SDD rule without SST added.

rule	error rate	mean	min	max	var
$(w_1, w_2, w_3) = (100, 1, 1)$					
SDD_{Pitts}	low	28.481	26.888	33.918	2.468
	mid	28.887	25.848	43.183	6.976
	high	29.079	24.688	42.519	9.405
SDD_{GP}	low	25.215	23.185	35.949	5.942
	mid	26.825	23.212	41.235	18.913
	high	27.339	23.293	40.011	16.953
LS	low	23.544	21.204	44.964	30.204
	mid	25.229	21.204	44.964	30.204
	high	26.761	21.204	56.678	83.302
$(w_1, w_2, w_3) = (100, 1, 100)$					
SDD_{Pitts}	low	80.122	76.504	98.107	17.913
	mid	83.149	76.307	97.022	32.221
	high	86.288	74.507	107.275	50.676
SDD_{GP}	low	56.982	41.171	73.215	24.015
	mid	58.625	18.184	82.047	76.164
	high	56.300	24.124	88.773	135.524
LS	low	65.683	51.570	124.645	358.093
	mid	74.959	51.570	157.840	523.889
	high	90.578	51.570	193.250	840.950
$(w_1, w_2, w_3) = (100, 100, 1)$					
SDD_{Pitts}	low	37.788	28.783	76.761	110.344
	mid	40.336	22.121	82.501	188.967
	high	41.180	21.589	78.229	141.724
SDD_{GP}	low	40.295	25.115	60.063	164.374
	mid	48.571	25.115	76.921	152.357
	high	49.224	25.115	66.886	161.505
LS	low	33.657	28.558	78.145	104.988
	mid	37.537	28.558	78.145	166.883
	high	39.461	28.558	96.566	190.569

9. Concluding Remarks

We have considered a scheduling problem for the semiconductor wafer test process. Because of the unpredictability of errors, non-negligible frequency, and significant influence, we have proposed three methods: C4.5, the Pitts approach, and GP to acquire an SDD rule. Using real data from a factory, we examined the performance of the proposed methods. Comparing them to LS, we confirmed that the obtained SDD rule is the most robust against unpredictable testing errors in the case that the error rate is a value that reflects a real factory situation when the weights of *DPenalty* or *CPenalty* are big. We have examined the performances of SDD rules using different data and found it yields the same result. We have also added SST as a candidate and confirmed its effectivity. In future work, we will look for an algorithm for finding a more complex SDD rule. We will also consider a method for acquiring appropriate candidate dispatching rules automatically, and improve the evolutionary computing convergence speed. We will apply also the proposed approach to more general scheduling problems in wafer testing and other processes.

References:

[1] D. Ouelhadj and S. Petrovic, "A Survey of Dynamic Scheduling in Manufacturing Systems," J. Sched., Vo.12, pp. 417-431, 2009.

[2] I. M. Ovacik and R. Uzsoy," "Decomposition Methods for Scheduling Semiconductor Testing Facilities," The Int. J. of Flexible Manufacturing Systems, Vol.8, pp. 357-388, 1996.

[3] Y. Shen and R. C. Leachman, "Stochastic Wafer Fabrication

Table 11. Evaluation of SDD rule with SST added.

rule	error rate	mean	min	max	var
$(w_1, w_2, w_3) = (100, 1, 1)$					
SDD_{Pitts}	low	25.242	22.928	34.168	2.302
	mid	25.239	18.728	29.949	2.607
	high	25.752	22.928	29.949	2.236
SDD_{GP}	low	18.428	17.281	22.342	0.481
	mid	18.141	8.401	21.463	4.041
	high	17.632	8.081	21.354	9.717
LS	low	19.885	18.179	42.380	24.272
	mid	21.144	18.179	42.180	39.627
	high	22.856	18.179	54.093	80.520
$(w_1, w_2, w_3) = (100, 1, 100)$					
SDD_{Pitts}	low	64.317	54.443	85.393	38.678
	mid	57.286	38.183	74.193	64.873
	high	61.201	24.483	79.543	96.943
SDD_{GP}	low	55.297	51.702	76.452	20.097
	mid	57.711	50.662	69.792	29.203
	high	60.239	50.662	78.947	41.215
LS	low	69.506	51.226	150.111	499.982
	mid	86.255	51.226	170.606	827.800
	high	97.796	51.226	192.756	1183.631
$(w_1, w_2, w_3) = (100, 100, 1)$					
SDD_{Pitts}	low	30.300	25.312	56.644	20.821
	mid	32.191	21.438	52.103	44.539
	high	31.642	21.438	57.137	30.292
SDD_{GP}	low	35.744	25.444	71.056	106.769
	mid	38.462	18.998	71.256	174.244
	high	41.700	18.998	75.097	146.398
LS	low	31.344	25.735	78.923	130.778
	mid	36.048	25.735	78.923	226.152
	high	38.223	25.735	104.104	260.831

Table 12. Number of applications of dispatching rules with SST added.

rule	FIFO	SPT	LWKR	MP	EDD	SLACK	SST
$(w_1, w_2, w_3) = (100, 1, 1)$							
SDD_{Pitts}	0	0	0	81	7	0	0
SDD_{GP}	0	8	0	16	1	0	66
SDR	11	6	10	18	7	17	16
$(w_1, w_2, w_3) = (100, 1, 100)$							
SDD_{Pitts}	0	51	0	0	0	0	70
SDD_{GP}	0	0	74	0	0	29	14
SDR	1	64	51	5	4	11	3
$(w_1, w_2, w_3) = (100, 100, 1)$							
SDD_{Pitts}	0	0	0	0	50	0	44
SDD_{GP}	24	0	0	0	18	7	35
SDR	10	9	19	12	11	12	10

Scheduling," IEEE Tran. Semiconductor Manufacturing, Vol.16, No.1, pp. 2-14, 2003.

[4] J.-Z. Wu and C.-F. Chien, "Modeling Semiconductor Testing Job Scheduling and Dynamic Testing Machine Configuration," Expert Systems with Applications, Vol.35, pp. 485-496, 2008.

[5] Z. Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs," 3rd, Revised and Extended Ed., Springer, Berlin, 1998.

[6] T. Matsuo, M. Inuiguchi, K. Masunaga, and D. Hirota, "Dynamic Scheduling Approaches to Wafer Test Scheduling with Unpredictable Error," J. of Advanced Computational Intelligence and Intelligent Informatics, Vol.17, pp. 526-534, 2013.

[7] S. F. Smith, "Flexible learning of problem solving heuristics through adaptive search," Proc. 8th Int. Joint Conf. on Artificial Intelligence, Vol.1, pp. 422-425, 1983.

[8] K. Sakakibara, "Research about Scheduling Rule Acquisition Based on Genetic Based Machine Learning," Graduate School of Science and Technology, Doctor Thesis, Kobe University, 2004.

[9] J. R. Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufmann Publishers, 1993.

Name:
Tsubasa Matsuo

Affiliation:
Graduate School of Engineering Science, Osaka University

Address:
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

Brief Biographical History:
2012 Received B.E. degree in engineering science, Osaka University
2014 Received M.E. degree in engineering science, Osaka University

Main Works:
• He is interested in scheduling, machine learning, and rough sets.

Name:
Masahiro Inuiguchi

Affiliation:
Graduate School of Engineering Science, Osaka University

Address:
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

Brief Biographical History:
1985, 1987, 1991 Received B.E., M.E., and D.E. degrees, respectively, in industrial engineering at Osaka Prefecture University
1987-1992 Research Associate at Osaka Prefecture University
1992-1997 Associate Professor at Hiroshima University
1997-2003 Associate Professor at Osaka University
2003- Full Professor at Osaka University

Main Works:
• His interests include possibility theory, fuzzy programming, rough sets, and approximate reasoning. He works as area editors of Fuzzy Sets and Systems, Fuzzy Optimization and Decision Making, and Journal of Multi-Criteria Decision Analysis and members of editorial boards of several other journals.

Name:
Kenichiro Masunaga

Affiliation:
Renesas Electronics Co.

Address:
2-6-2 Otemachi, Chiyoda-ku, Tokyo 100-0004, Japan

Brief Biographical History:
1993,1995 Received B.E. and M.E. degrees, respectively, in engineering at Osaka University
1995-2002 Worked at Semiconductor Group, Hitachi, Ltd.
2002- Works at Manufacturing System Technology Department, Process Technology Division, Renesas Electronics Co.

Main Works:
• He develops and applies the dispatching and scheduling system for improving the efficiency of the semiconductor manufacturing line.