Petri-Net and GA-Based Approach to Modeling, Scheduling, and Performance Evaluation for Wafer Fabrication

Jyh-Horng Chen, Li-Chen Fu, *Member, IEEE*, Ming-Hung Lin, and An-Chih Huang

Abstract—In this paper, a genetic algorithm (GA) embedded search strategy over a colored timed Petri net (CTPN) for wafer fabrication is proposed. Through the CTPN model, all possible behaviors of the wafer manufacturing systems such as WIP status and machine status can be completely tracked down by the reachability graph of the net. The chromosome representation of the search nodes in GA is constructed directly from the CTPN model, recording the information about the appropriate scheduling policy for each workstation in the fab. A better chromosome found by GA is received by the CTPN based schedule builder, and then a near-optimal schedule is generated.

Index Terms—Genetic algorithm, modeling and scheduling, Petri-net, wafer fabrication.

I. INTRODUCTION

WAFER FABRICATION is the most costly phase of semiconductor manufacturing [3], [6], [15]. A significant amount of risk is involved in the wafer fabrication because it requires huge investment costs. To survive from such competitive and risky environment, the company must not only improve quality and throughput rate but also meet customers' demand. If product delivery is frequently late, the company loses customers' goodwill as well as market, which will affect unfortunately the long-term sales opportunity. In addition, wafer fabrication involves a complex sequence of processing steps with the number of operations typically in hundreds, which result in long cycle time of the production. Long production cycle time causes wafer fabrication to be extremely expensive because of their adverse effects on the product yield, and the ability of forecasting the future market. In addition, product life cycles are short in general in the semiconductor industry, and, hence, the risk of obsolescence for finished goods inventory is ever present.

Recent papers by Uzsoy *et al.* [6], [8], Johri [7], and Duenyas *et al.* [9] highlight the difficulties in planning and scheduling of wafer fabrication facilities. These papers also survey the literature on related topics. Effective shop floor scheduling can be a major component of reduction in cycle time. The benefits of effective scheduling include higher machine utilization, shorter cycle time, higher throughput rate, and greater customer satisfaction. This is particularly true of semiconductor manufacturing, with its rapidly changing markets and complex manufacturing processes [1], [2], [10]–[13]. Yet in many wafer fabrications the product spends much more waiting time than actually being processed, so there is a large potential for reducing waiting time and a great benefit for doing so. People also considered other issues of wafer fabrication systems, such as batch processing system [4] and maintenance scheduling as well as staff policy [14].

It is well known in the scheduling literature that the general job shop problem is NP-hard, which leads to no efficient algorithm existing for solving the scheduling problems optimally in polynomial time for wafer fabrication, and therefore it is the reason why we use the genetic algorithm (GA) to solve the problem. GA is a search procedure that uses random choices as a guide tool through a coding in the parameter space [17]–[22]. While randomized, however, GAs are by no means a simple random-walk approach. They efficiently exploit historical information to speculate on new search nodes with expected improved performance. That is, GA samples large search space randomly and efficiently to find a good solution in polynomial time, which however does not require enormous memory space as other traditional search algorithms such as A*. Many researchers have used GA to deal with job shop scheduling problem in traditional industries. Lee *et al.* [21] focused on solving the scheduling problem in a flow line with variable lot sizes. Lee *et al.* [20] combined the machine learning and genetic algorithm in the job shop scheduling. Ulusoy *et al.* [22] have addressed on simultaneous scheduling of machines and automated guided vehicles (AGVs) using genetic algorithm. In addition, Cheng *et al.* [19] have surveyed relational topics on solving the job shop problem using GA. They also discussed chromosome representation in details. Unlike the previous research, we use GA methodology to solve the more complex scheduling problem in wafer fabrication. However, since wafer fabrication is a complex discrete event system, schedulers cannot be easily realized on this kind of system, and thus how to model a complex wafer fab manufacturing system is an imperative task. A good model not only helps us to employ our scheduler easily, but also helps us to track the status of lots and machines efficiently, which is useful to tune the scheduling policy at the right moment.

In the modeling field, Petri net (PN) [23], [24] has played an important role; it has been developed into a powerful tool for discrete event systems, particularly in manufacturing sys-

The authors are with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C. (e-mail: lichen@csie.ntu.edu.tw).

tems. PNs have gained more and more attentions in semiconductor manufacturing due to their graphical and mathematical advantages over traditional tools to deal with discrete event dynamics and characteristics of complex systems [25], [27]–[30]. Zhou *et al.* [27] reviewed applications of PNs in semiconductor manufacturing automation. It can also serve as a tutorial paper. Xiong *et al.* [29] proposed and evaluated two Petri-net based heuristic search strategies to semiconductor test facility scheduling. The colored timed Petri net (CTPN) is used to model the furnace in the IC wafer fabrication [30] and in the entire wafer fabrication manufacturing system [25]. Jeng *et al.* [28] applied Petri net methodologies to detailed modeling, qualitative analysis, and performance evaluation of the etching area in an IC wafer fabrication system located in the Science Based Industrial Park in Hsinchu, Taiwan.

In this paper, we propose a systematic CTPN model embedded with a GA scheduler. The CTPN [30] can be used to model the complex process flows in wafer fab efficiently and the detailed manufacturing characteristics such as lots processing, machines setup, machines failure, batch processing, and rework of defective wafers. In addition, new transitions are introduced in this paper, which significantly reduce the complexity of Petri-net model. The GA scheduler can be used to dynamically search for an appropriate dispatching rule for each workstation or processing unit family through the CTPN model. In addition, we develop a powerful simulation tool for modeling and scheduling in wafer fabrication facilities. The scheduling policies generated by the scheduler based on GA in this tool always perform much better than other conventional dispatching rules.

The organization of this paper is described as follows. In Section II, the definitions of the proposed CTPN are revealed here. Moreover, the detailed systematic method of CTPN model is discussed. In Section III, some often seen scheduling problems and lot released control policies used in wafer fabrication are discussed first. Then, a GA embedded search method over the CTPN model is employed. In addition, a schedule builder is used to transform a chromosome to a feasible schedule. In Section IV, we briefly describe the implementation of the simulation tool. In Section V, we demonstrate two example of using the proposed mechanism and analyze the performance. Finally, conclusions are provided in Section VI.

II. COLORED TIMED PETRI NET

The ordinary PN do not include any concept of time and color. With this class of nets, it is possible only to describe the logical structure and behavior of the modeled system, but not its evolution over time and color. Responding to the need to model the manufacturing system in wafer fab, time and color attributes must be introduced. In the proposed CTPN model [30], [31], we introduced two kinds of places, namely places and communication places, and five kinds of transitions, i.e., immediate transitions, mapping transitions, deterministically timed transitions, stochastic transitions, and macro transitions. The details were described as follows.

Definition 2.1: A CTPN is a bipartite graph, defined by a 11-tuple CTPN $= (P_i, P_c, T_u, T_t, T_s, T_p, T_m, C, I, O, M)$, where:

P_i set of places;
P_c set of communication places;
T_u set of immediate transitions;
T_t set of deterministically timed transitions;
T_s set of stochastic transitions;
T_p set of mapping transitions;
T_m set of macro transitions;
C color set of transition and place;
I input functions;
O output functions;
M the marking.

□ $P = \{p1, p2, \ldots, pd\}$: P is finite set of ordinary place with $d \geq 1$, including places and communication places.

■   Places: Places are the same as the ordinary places in CPNs (colored Petri nets). They can be used to model the conditions or properties of resource in the manufacturing system.

■   Communication places: Communication places are used to represent the signals or conditions that are sent to or are received from the other sub-models to signify some upcoming events. They are also used to achieve the interconnection among different modules.

□ $T = \{t_1, t_2, \ldots, t_e\}$: T is a finite set of transitions with $e \geq 1$, including immediate transitions, mapping transitions, deterministically timed transitions, stochastic transitions, and macro transitions.

■   Immediate transitions: Immediate transitions are the same as the transitions in original CPNs. They can be used to model behaviors or events of resources in manufacturing systems.

■   Mapping transitions: After firing the mapping transitions, the token with respect to a certain color that enable this type of transition is transferred to another place with predefined color. Notice that the enabling and firing rules of the mapping transitions are the same as the ordinary transitions in colored Petri nets, the mapping transitions are immediate. And the name "mapping transition" indicates the special meaning of the transition function.

■   Deterministically timed transitions: Deterministically timed transitions describe the time properties of resources. Time can be incorporated into Petri nets by introducing a time delay after a transition is enabled. This kind of transitions can be used to model the operations in a manufacturing system.

■   Stochastic transitions: The transition times may belong to one of several distributions in stochastic transitions. When a stochastic transition becomes enabled, it fires after a random interval of time. It is useful for modeling transportation time, failure, and repair times of individual machines.

■   Macro transitions: A macro transition is the combination of a series of transitions, places and arcs.

□ $P \cup T \neq \varnothing, P \cap T = \varnothing.$

□ $C(p) = \{pc_i1, \ldots, pc_{iui}, u_i = |C(p_i)|, i = 1, \ldots, n\}$ where n are nonnegative integers; the pcs are the associated colors, and $C(p)$ represents the color sets of places in P.

$\square$ $C(t) = \{tc_{j1}, \ldots, tc_{jvi} v_i = |C(t_j)|, j = 1, \ldots, m\}$ where m are nonnegative integers; the tcs are the associated colors. $C(p)$ and $C(t)$ represents the color sets of places in P and transitions in T.

$\square$ $I(p_i, t_j)(pc_{ih}, tc_{jr})$: $C(p) \times C(t) \rightarrow N$ (nonnegative integers) corresponding to the input arc from a place p_i with respect to the color pc_{ih}, to a transition t_j, with respect to the color tc_{jr}.

$\square$ $Ih(p_i, t_j)(pc_{ih}, tc_{jr})$: $C(p) \times C(t) \rightarrow \{1, 0\}$ corresponding to the inhibitor arc from a place p_i with respect to the color pc_{ih}, to a transition t_j, with respect to the color tc_{jr}.

$\square$ $O(p_i, t_j)(pc_{ih}, tc_{jr})$: $C(t) \times C(p) \rightarrow N$ (nonnegative integers) corresponding to the output arc from a transition t_j with respect to the color tc_{jr}, to a place p_i with respect to the color pc_{ih}.

$\square$ A *marking* of a CTPN is a vector function M defined on P: $M(p)$: $C(p) \rightarrow N$, for all places. Every component of the marking is a $(u_i \times 1)$ vector. The tokens in a place now become a *summation of place colors*: $M(p_i) = \sum n_{ih}pc_{ih}$, for $h = 1, \ldots, u_i$, where n_{ih} is the number of tokens of color pc_{ih} in place p_i, denoted by $n_{ih} = M(p_i)(pc_{ih})$.

$\square$ A transition t_j is said to be enabled with respect to a color tc_{jk} in a marking M iff, for $\forall p_i, p_x \in P$, we have: $M(p_i)(pc_{ih}) \geq I(p_i, t_j)(pc_{ih}, tc_{jk})$ and if $Ih(p_x, t_j)(pc_{xy}, tc_{jk}) = 1$, $M(p_x)(pc_{xy}) = 0$. An enabled transition fires, resulting in a new marking M', where $M'(p_i)(pc_{ih}) = M(p_i)(pc_{ih}) + O(p_i, t_j)(pc_{ih}, tc_{jk}) - I(p_i, t_j)(pc_{ih}, tc_{jk})$. When a stochastic or deterministically timed transition becomes enabled, it fires after a random or deterministic interval of time.

$\square$ Time Function: $f(t_j)(tc_{jk})$: $C(t) \rightarrow N$, is the time delay required by the deterministically timed transition t_j with the token with respect to the color tc_{jk}. For stochastic transitions, the output of the time function is the mean value of random variables that belongs to one of several distributions. The time delay is zero in other transitions.

The icon definition of CTPN is illustrated in Fig. 1.

Fig. 1. Icon definition of CTPN.

Fig. 2. Wafer processing model.

III. WAFER PROCESSING MODEL

Each wafer lot entering the clean room has an associated process flow that consists of precisely specified operations executed in a prescribed sequence on prescribed pieces of equipment. In this paper, we used the proposed CTPN to model the whole wafer manufacturing systems that include the deposition, photolithography, etching, and diffusion areas. The implementation of the model generator was discussed in Section IV, and the details of the proposed Wafer Processing Model including Routing Module and Elementary Module Fig. 2 are described in the following sections.

If we did not mention the color set of the transitions or places in the following wafer processing model, then the color set of these places p and transitions t is $C(t) = C(p) = \{xzws | x \in X, z \in Z, w \in W, s \in S\}$, where the color is encoded as a string $xzws$, X is the set of route ID, Z is the set of lot ID, W is the set of operation type ID, and S is the set of operation step. Similarly, if we did not mention, the input and output function
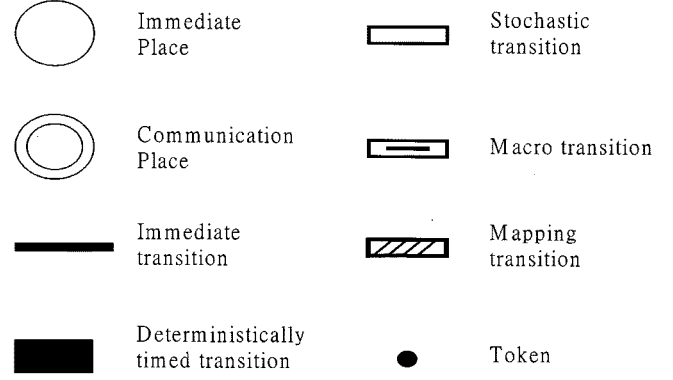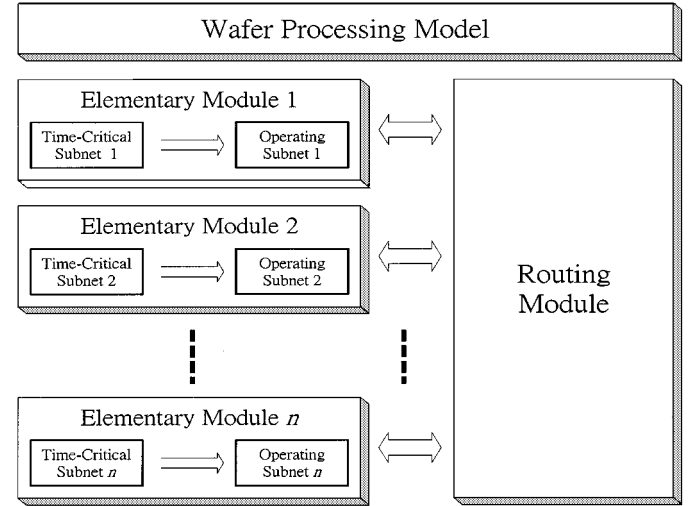
of these transitions are $\forall c \in C(p) \cap C(t)$, $I(p, t)(c, c) = O(p, t)(c, c) = 1$ and otherwise 0.

The routing module's main purpose is to model the logical process flow of the manufacturing systems. The basic concept of this module is described as follows. First, we divide all the machines in the fab into n workstations (machine groups or processing unit family), each of which contains one or more identical machines (or processing units). In the beginning, a token (lot) in the place with respect to the color $xzws_0$ enters the model, where x is the route ID, z is the lot ID, w is the operation type ID and s_0 is operation step. This_token (lot) is then checked in and assigned into the place *dispatch* with the color $xzws_1$.

Each operation has its associated workstation to be performed, thus lots travel to and take operations in the proper workstation in the fab according to the predefined process flow. After the lot finishes the current operation, the corresponding color number of the lot will be increased by one, and ready to do the next operation. Systematically, after the lot finishes all its operations, it enters the *end* place and finishes the work.

The Routing Module, which is shown in Fig. 3, implements the idea mentioned above. For clarity, we explain the notations used in Fig. 3 as follows.

$\square$ *enter:* It is a place. When a lot is released to the fab, we put a token in this place with the specific color (i.e., $xzws_0$).
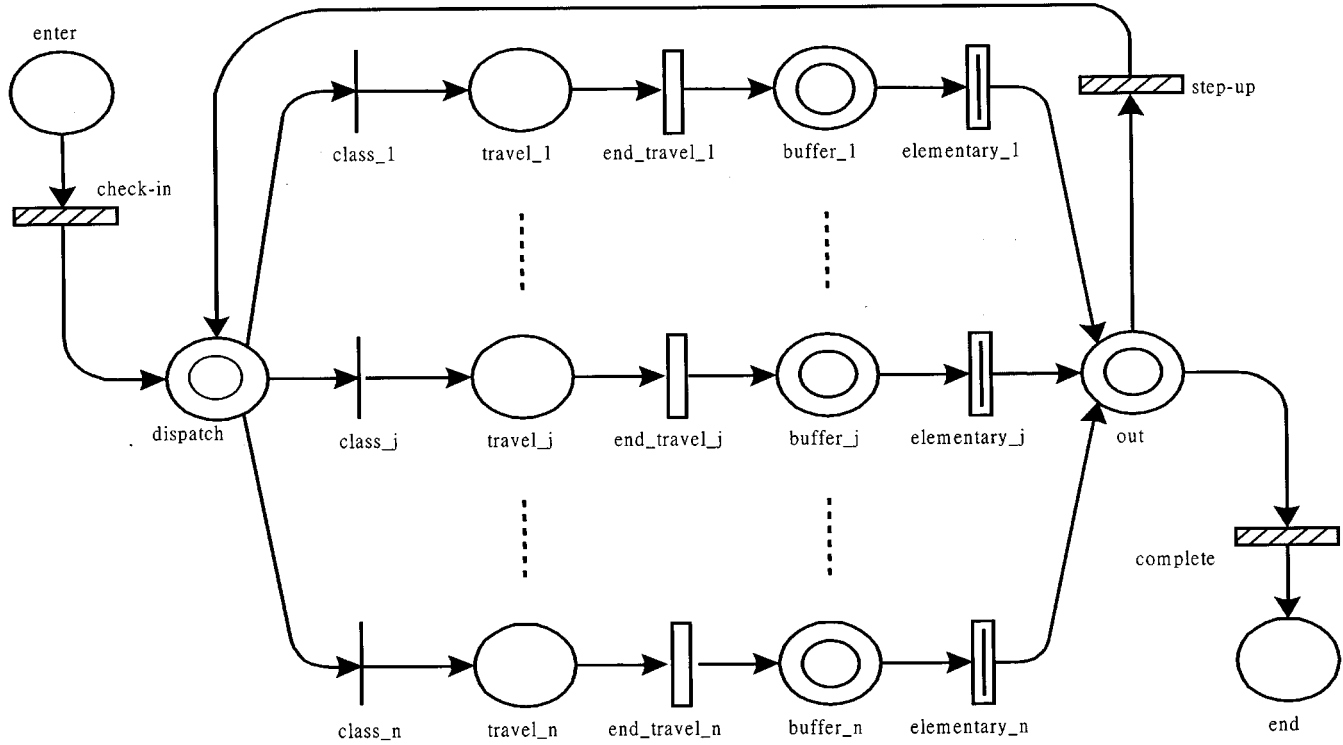
Fig. 3. Routing module.

□ *check-in*: It is a mapping transition. When *check-in* is fired, the corresponding color s of the token that fired the *check-in* will be increased by one (i.e., $xzws_1$). This event represents that a lot is released into the fab, and is being prepared for its first operation.

□ *dispatch*: The place *dispatch* is a communication place. It can be thought of as the dispatching manager. A token in the *dispatch* can enable one of its output transitions (*class_1, class_2, ..., class_j, ..., class_n*) according to its corresponding color. It represents that the lot has to be sent to the proper workstation according to its current operation step that has not been performed.

□ *class_j*: The transition *class_j* ($j = 1, 2, ..., n$) is an immediate transition. It has a color set that denotes what tokens can be fired through this transition. For example, if only the s_3 and s_5 step (the third step and the fifth step) of route x_1 can be processed on jth workstation, its color set will be $C(class\_j) = \{xzws | x \in X, z \in Z, w \in W, s \in \{s_3, s_5\}\}$. In the wafer fab, this transition controls what operation steps can be done in the jth workstation.

□ *travel_j*: The place *travel_j* is a place. A token in the place *travel_j* ($j = 1, 2, ..., n$) represents that the loop-vehicle or AGV was used to transport the corresponding lot to the jth workstation.

□ *end_travel_j*: The transition *end_travel_j* ($j = 1, 2, ..., n$) is a stochastic transition. The time delay of this transition is the mean transportation time of the lot. A token fired through the transition *end_travel_j* ($j = 1, 2, ..., n$) represents that the corresponding lot has been transported to the buffer of jth workstation.

□ *buffer_j*: The place *buffer_j* ($j = 1, 2, ..., n$) is a communication place, which is an interconnection among Routing

Module and Elementary Module. A token in the place *buffer_j* ($j = 1, 2, ..., n$) represents that a lot is in the buffer of jth workstation and prepares to be processed.

□ *elementary_j*: The transition *elementary_j* ($j = 1, 2, ..., n$) is a macro transition. It represents the jth Elementary Module that is to be discussed in the next section.

□ *out*: The place *out* is a communication place. A token in the place out represents that the corresponding lot completes the current operation. A token in the place *out* may enable one of place *out*'s output transitions. If the corresponding color of the token denotes the final operation step, then the token may enable the transition *complete* to finish the work; otherwise, the token may enable the transition *step-up* to do the next operation.

□ *step-up*: The transition *step-up* is a mapping transition. When *step-up* is fired, the corresponding color s of the token that fired the *step-up* will be increased by one (be assigned into the place with the next step's color). For example, if a lot (token) z_3 with route x_2 and operation type w_4 just finish its s_6 step, and we assume that the next operation type is w_7 and step is s_6. After transition *step-up* is fired, the token will be assigned into the place with the color $x_2 z_3 w_7 s_6$. This event represents that the corresponding lot has finished the current operation and is prepared to do the next operation. After the token is fired through the *step-up* transition, the token entered the place *dispatch*, and again the corresponding lot can perform the next operation.

□ *complete*: The transition *complete* is a mapping transition. When the transition *complete* is fired, the lot has finished all the operations and entered the place *end*.

□ *end*: The place *end* is a place. A token in the place *end* represents that the corresponding lot has finished all the operations and is ready to shift.
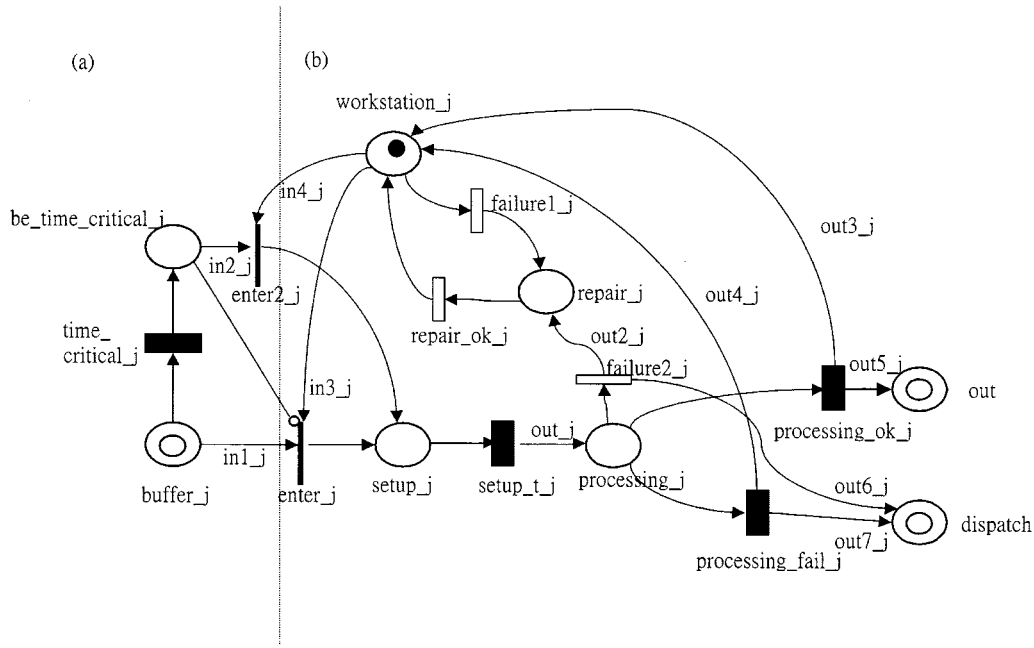
Fig. 4. (a) Time-critical subnet and (b) operating subnet.

Elementary Module

□ The purpose of Elementary Module is to model the detailed manufacturing system in a wafer fab, such as processing, setup, rework, unscheduled machine breakdown, and time-critical operation. We divide the Elementary Module into two subnets, which will be explained in details as follows.

Time-Critical Subnet

Time-Critical Subnet is shown in Fig. 4(a). The idea of the Time-Critical Subnet is that if a lot is waiting for a time-critical operation, and has waited for some specific period, the lot can get a higher priority. Thus it can be performed first.

□ *be_time_critical_j:* The place *be_time_critical_j* ($j = 1, 2, \ldots, n$) is a place. Tokens in this place represent that the corresponding lots have higher priority and can be performed first in the jth workstation. A token in this place (with any color) will block the transition *enter_j* of the operating subnet. Thus the priority of the lots (tokens) in *time_critical_j* is higher than the priority of lots (tokens) in *buffer_j*, this guarantees that the lot with time-critical operation can be done first to avoid unnecessary rework.

□ *buffer_j:* The place *buffer_j* ($j = 1, 2, \ldots, n$) is a communication place. A token in *buffer_j* represents that a lot is ready to be processed in the jth workstation.

□ *time_critical_j:* The transition *time_critical_j* ($j = 1, 2, \ldots, n$) is a deterministically timed transition. A token in *buffer_j* that enable this transition will be moved into the place *be_time_critical_j* after the delay time of this transition. It can represent the behavior of the lot getting higher priority to be processed.

□ *enter2_j:* The place *enter2_j* ($j = 1, 2, \ldots, n$) is an immediate transition. Firing this transition represent a "time-critical" lot entering the operation subnet. The color set of this transition is $C(enter2\_j) = \{bxzws | b \in W, x \in X, z \in Z, w \in$

$W, s \in S\}$, where b represents last operation type performed on the machine. The two tokens in place *be_time_critical* with the color $xzws(x \in X, z \in Z, w \in W, s \in S)$ and in place *workstation_j* with color $b \in W$ (the color of last operation perform on machine) will enables the transition *enter2_j* with color $bxzws$. After the firing of this transition, a token will be assigned into the place *setup_j* with the color $bxzws$.

The time associated with the transition *time_critical_j* depends on the corresponding color of the token. If the lot (token) requires time-critical operation, then the time is some finite delay time; but if otherwise, the time delay is infinite. Such time delay mainly depends on the machine, and should be given by the fab engineer. In our experimental example, we simply choose an arbitrary number for this time delay to demonstrate the idea. Therefore, now the transition *enter_j* will not be fired if there exist tokens in place *be_time_critical_j*. The transition *time_critical_j* will then be fired if the lot in place *buffer_j* requires time-critical operation and the time spent in that place exceeds the delay time.

Operating Subnet

Operating Subnet is shown in Fig. 4(b). The purpose of this subnet is to model processing, machine setup, machine failure, and to check whether the lot needs to be reworked. Similarly, we explained the notation used in Fig. 4(b) as follows.

□ The color set of *failure1_j*, *repair_ok_j*, *failure2_j*, *workstation_j*, and *repair_j* is $C(p) = C(t) = \{w | w \in W\}$, and the color set of the transitions *enter_j*, *setup_j* and *setup_t_j* is the same as that of transition *enter2_j*.

□ *enter_j:* The place *enter_j* ($j = 1, 2, \ldots, n$) is an immediate transition. Firing this transition represents a normal (not time-critical operation) lot entering the operation subnet. The two tokens in place *buffer_j* with the color $xzws(x \in X, z \in Z, w \in W, s \in S)$ and in place *workstation_j* with color $b \in W$ (the color of last operation) will enable the transition

enter_j with color $bxzws$. After the firing of the transition, a token will be assigned into the place *setup_j* with the color $bxzws$.

□ *in1_j (in2_j):* $in1\_j$ ($in2\_j$) ($j = 1, 2, \ldots, n$) is the input arc of the transition *enter_j* (*enter2_j*). The place (*buffer_j*, *be_time_critical_j*) with the color $xzws$ ($x \in X, z \in Z, w \in W, s \in S$) has arcs directly connected to the transition with the color $bxzws$ ($b \in W$).

□ *in3_j (in4_j):* $in3\_j$ ($in4\_j$) ($j = 1, 2, \ldots, n$) is the input arc of the transition *enter_j* (*enter2_j*). The place *workstation_j* with the color $wj \in W$ has arcs directly connected to the transition *enter_j* (*enter2_j*) with the color $bxzws$ ($b = wj, x \in X, z \in Z, w \in W, s \in S$).

□ *setup_j:* The place *setup_j* ($j = 1, 2, \ldots, n$) is a place. The token in *setup_j* represents the machine is being set.

□ *setup_t_j:* The transition *setup_t_j* ($j = 1, 2, \ldots, n$) is a deterministically timed transition. The function of *setup_t_j* is to compare the color of the last operation and current operation, i.e., to check whether the corresponding machine in the jth workstation needs setup. If the two colors are the same, the corresponding machine does not need setup, and the corresponding lot can be processed immediately. When the color of the transition is $bxzws$ ($b = w, b \in W, x \in X, z \in Z, w \in W, s \in S$), the associated time is zero, and vice versa ($b \neq w$, time ≥ 0). After the corresponding machine completes setup, the corresponding lot can be processed.

□ *out_j:* out_j is the output arc of the transition setup_t_j ($j = 1, 2, \ldots, n$). The transition *setup_t_j* with the color $bxzws$ ($b \in W, x \in X, z \in Z, w \in W, s \in S$) has an arc directly connected to the place *processing_j* with the color $xzws$.

□ *out_2 (out3_j, out4_j):* $out2\_j$ ($out3\_j$, $out4\_j$) ($j = 1, 2, \ldots, n$) is the output arc of the *transition failure2_j* (*processing_ok_j*, *processing_fail_j*). The transition with the color $xzws$ ($x \in X, z \in Z, w \in W, s \in S$) has an arc directly connected to the place *repair_j* (*workstation_j*) with the color w. The meaning of firing these transitions is that the last operation performed on this machine is memorized in place *repair_j* (*workstation_j*).

□ *processing_j:* The place *processing_j* ($j = 1, 2, \ldots, n$) is a place. A token in *processing_j* represents that the corresponding lot is processed by the machine in the jth workstation. A rework probability is set in this place. If the jth workstation contains an inspection device, the rework probability is greater than zero; otherwise, the rework probability is equal to zero. The token in *processing_j* can enable the transition *process_ok_j*, *process_fail_j*, and *failure_2_j*, and one of them can be fired according to the time delay and the rework probability of these transitions. When a token enters into this place, it first checks whether the lot needs to be reworked. If the lot must be reworked and the transition *failure_2* is not fired, the token fires the transition *process_fail_j* after the associated processing time; otherwise, it fires *process_ok_j*.

□ *process_fail_j:* The transition *process_fail_j* ($j = 1, 2, \ldots, n$) is a deterministically timed transition. The time delay (processing time) of this transition depends on the current route and step of the lot (token). When *process_fail_j* is fired, two tokens are generated. One of the generated tokens is in the place *workstation_j*, which implies that the current lot released the

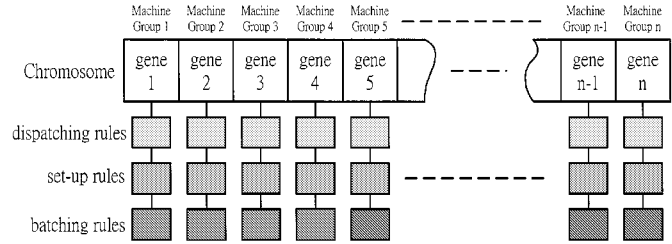| Dispatching Rules		Set-up Rules		Batching Rules	
Name	Code	Name	Code	Name	Code
FCFS	1	FCFS	1	W1T	1
MINS	2	SSU	2	W2T	2
SRPT	3			W3T	3
EDD	4			W4T	4

Fig. 5.   Gene codes.

Fig. 6.   Chromosome representation.

control of the corresponding machine. The other one of the generated tokens is in the place dispatch, which implies that the lot must be reworked from the specific operation step.

□ *process_ok_j:* The transition *process_ok_j* ($j = 1, 2, \ldots, n$) is a deterministically timed transition. Similarly, The time delay (processing time) of this transition depends on the current route and step of the lot (token). The firing of *process_ok_j* represents that the lot just finished the current operation and released the control of the corresponding machine in the jth workstation.

□ *workstation_j:* The place *workstation_j* ($j = 1, 2, \ldots, n$) is a place. The number of tokens in workstation_j denotes how many available machines are in the jth workstation. The firing of the stochastic transition *failure1_j* represents the machine failures during idleness, otherwise, the token enables the transition *enter_j* or *enter2_j* if necessary.

□ *repair_j:* The place *repair_j* ($j = 1, 2, \ldots, n$) is a place. A token in *repair_j* represents that a machine in the jth workstation is in failure. After some stochastic time period passed, the token in the *repair_j* can fire the transition *repair_ok_j*. The time delay used in *repair_ok_j* depends on the mean time to repair (MTTR).

□ *failure1_j, failure2_j and repair_ok_j* ($j = 1, 2, \ldots, n$): These are stochastic transitions. As mentioned above, the time delay used in *repair_ok_j* depends on the mean time to repair (MTTR), and the time delay used in *failure1_j* and *failure2_j* depends on the mean time to fail (MTTF). The firing of the transition *failure2_j* represents machine failures during an operation.

The place *workstation_j* is initially marked. We model the machine failure probability with the transition *failure1_j* and *failure2_j*. If the lot is in processing, the transition *failure2_j* can be fired. If the machine is in idle state, the transition *failure1_j* can be fired. Note that the machine cannot fail when being set up in this subnet. The reason for this simplification is that compared to the processing time and idle time, the setup time of machine is relatively short, so that it can be ignored. In addition, if lot is reworked or a machine is in failure while
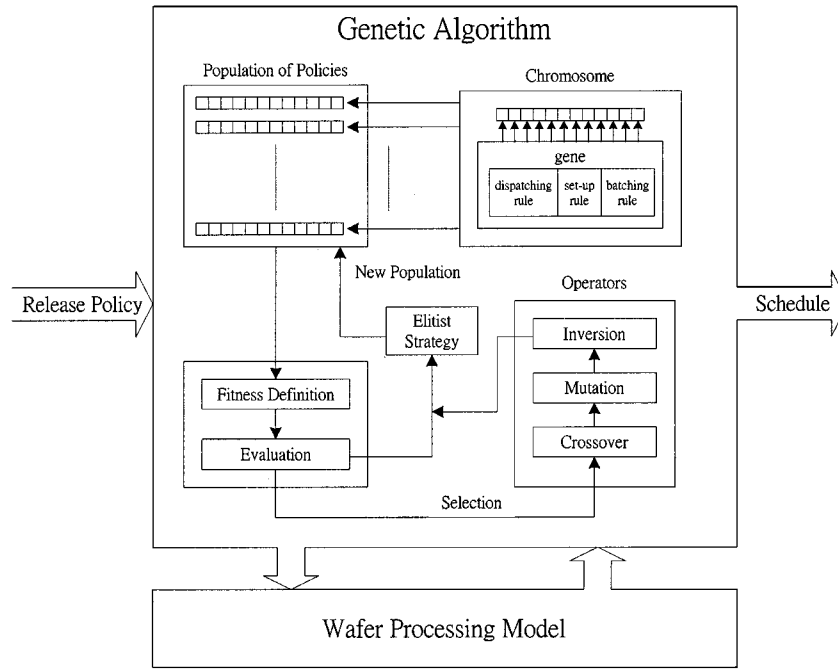
Fig. 7. The architecture of the scheduler.

processing, a token will enter the place *dispatch* of the routing module. If the lot is successfully processed, a token will enter to place *out* of Routing module to indicate that the lot is ready for next wafer operation.

The case of batch processing machine is somewhat complicated, although its figure is the same as Fig. 4. The color set of *enter_j*, *setup_j*, *enter2_j*, *setup_t_j*, *processing_j*, *failure_j*, *process_ok_j*, and *process_fail_j* depends on the batch size of the machine. If the batch size is two, the color set is $C(t) = C(p) = \{bx_1z_1w_1s_1x_2z_2w_2s_2 | b \in W, x_1, x_2 \in X, z_1, z_2 \in Z, w_1, w_2 \in W, s_1, s_2 \in S\}$; If the batch size is three, the color set is $C(t) = C(p) = \{bx_1z_1w_1s_1x_2z_2w_2s_2x_3z_3w_3s_3 | b \in W, x_1, x_2, x_3 \in X, z_1, z_2, z_3 \in Z, w_1, w_2, w_3 \in W, s_1, s_2, s_3 \in S\}$, and so on. We take the case of the batch size 2 as an example, and the differences are described as follows.

□ *enter_j (enter2_j):* The transition *enter_j* (*enter2_j*) $(j = 1, 2, \ldots, n)$ is an immediate transition. Firing this transition represents a normal (time-critical operation) batch entering the operation subnet. Two tokens in place *buffer_j* (*be_time_critical_j*) with the color $xzws(x \in X, z \in Z, w \in W, s \in S)$, $x_1z_1ws_1(x_1 \in X, z_1 \in Z, s_1 \in S)$ and one token in place *workstation_j* with color $b \in W$ (the color of last operation) will enable the transition *enter_j* (*enter2_j*) with the color $bxzwsx_1z_1ws_1$ or $bx_1z_1ws_1xzws$. After the firing of the transition, a token will be assigned into the place *setup_j* with the color $bxzwsx_1z_1ws_1$ or $bx_1z_1ws_1xzws$. Firing this transition represents that the two lots are grouped together. Since we have to memorize the corresponding color of each token (lot) in a batch, we have to expand the length of the color string of the transitions and places mentioned above.

□ *in1_j (in2_j):* *in1_j* (*in2_j*) $(j = 1, 2, \ldots, n)$ are the input arcs of the transition *enter_j* (*enter2_j*). The place *buffer_j* (*be_time_critical*) with the color

Fig. 8. UltraSim.

$xzws(x \in X, z \in Z, w \in W, s \in S)$ has arcs directly connected to the transition with the color $bxzwsx_1z_1w_1s_1(b \in W, x_1 \in X, z_1 \in Z, w_1 \in W, s_1 \in S)$ and $bx_2z_2w_2s_2xzws(b \in W, x_2 \in X, z_2 \in Z, w_2 \in W, s_2 \in S)$.

□ *in3_j (in4_j):* *in3_j* (*in4_j*) $(j = 1, 2, \ldots, n)$ are the input arcs of the transition *enter_j* (*enter2_j*). The place *workstation_j* with the color $wj \in W$ has arcs directly connected to the transition with the color $bxzwsx1z1ws1(b = wj, x \in X, z \in Z, w \in W, s \in S, x1 \in X, z1 \in Z, s1 \in S)$.

□ *setup_t_j:* The transition *setup_t_j* $(j = 1, 2, \ldots, n)$ is a deterministically timed transition. The function of *setup_t_j* is

Fig. 9. The main window of UltraSim.

to compare the color of the last operation and current operation, i.e., to check whether the corresponding machine in the jth workstation needs setup. If the two operations are the same, the corresponding machine does not need setup, and the corresponding batch can be processed immediately. When the color of the transition is $bx_1z_1ws_1x_2z_2ws_2(b = w, x_1, x_2 \in X, z_1, z_2 \in Z, w \in W, s_1, s_2 \in S)$, the associated time is zero, and vice versa ($b \neq w$, time ≥ 0). After the corresponding machine completes setup, the corresponding batch can be processed.

□ **out_j:** out_j ($j = 1, 2, \ldots, n$) is the output arc of the transition *setup_t_j*. The output function can be expressed as an identity matrix in the batch-processing machine [$\forall c \in C(t), I(p, t)(c, c) = O(p, t)(c, c) = 1$ and otherwise 0].

□ **out2_j (out3_j, out4_j):** out2_j (out3_j, out4_j) ($j = 1, 2, \ldots, n$) is the output arc of the transition *failure2_j (processing_ok_j, processing_fail_j)*. The transition with the color $bx_1z_1ws_1x_2z_2ws_2(b \in W, x_1, x_2 \in X, z_1, z_2 \in Z, w \in W, s_1, s_2 \in S)$ has an arc directly connected to the place *repair_j (workstation_j)* with the color w. The meaning of firing these transitions is that the last operation performed on this machine is memorized in place *workstation_j*.

Fig. 10. The data access architecture of UltraSim.

□ **out5_j (out6_j, out7_j):** out5_j (out6_j, out7_j) ($j = 1, 2, \ldots, n$) is the output arc of the transition *failure2_j (processing_ok_j, processing_fail_j)*. The transition with the color $bx_1z_1w_1s_1x_2z_2w_2s_2(b \in W, x_1, x_2 \in X, z_1, z_2 \in$

Fig. 11. Lot release wizard.

Z, w_1, $w_2 \in W$, s_1, $s_2 \in S$) has two (depending on the batch size) arcs directly connected to the place dispatch (out) with the color $x_1 z_1 w_1 s_1$ and $x_2 z_2 w_2 s_2$. The meaning of firing these transitions is that the batch is separated into lots and ready to do its next operation.

IV. WAFER FAB SCHEDULER

A typical wafer fabrication facility contains many different products and processes, some with small quantities, competing for resources. Each product flow can contain hundreds of processing steps demanding production time of the same resource many times during the flow. Hence, it is important to develop a production scheduling system that can help t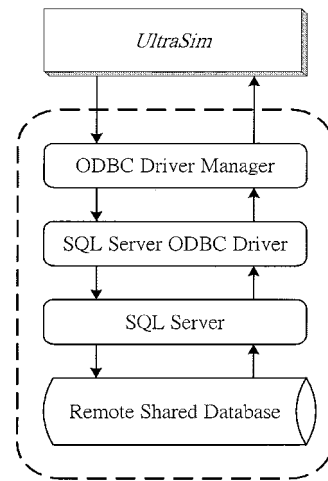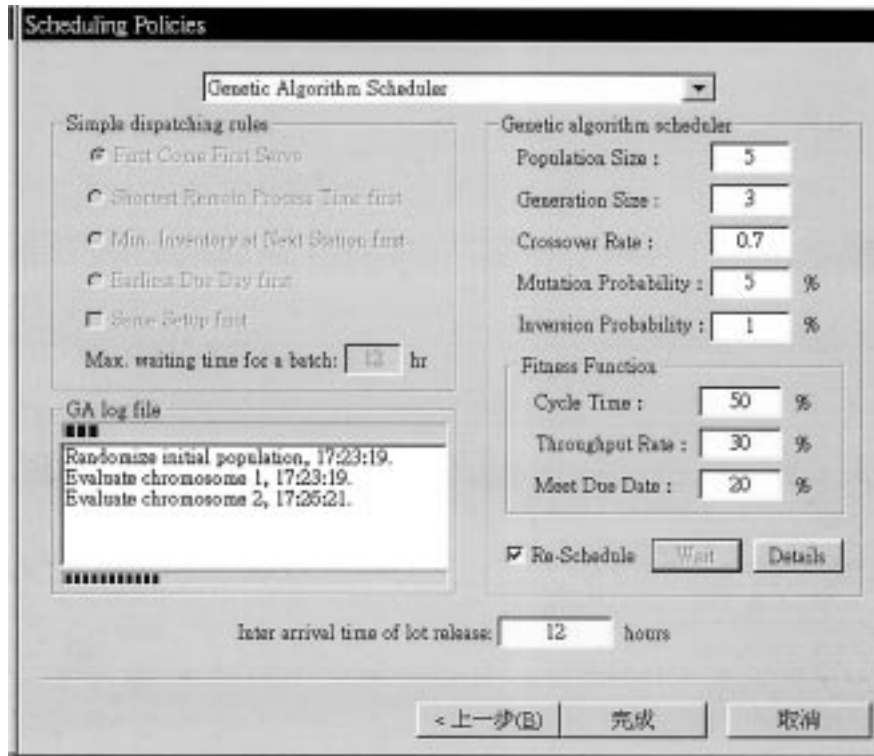o minimize cycle time as well as to maximize the throughput rate and the rate of meeting due dates. Similarly, it is very important to control the release of new lots to the system to avoid starvation of bottlenecks wherever possible.

The UNIF (uniform) rule is the most common release control used in wafer fabrication, i.e., release new lots into the fab at a constant rate equal to the desired output rate, but independent of the current inventory level or the machine status. In this paper, we also use UNIF as the lot release control policy, and tune the lot interarrival time to an appropriate value so that the fab can keep a steady WIP level.

We can address the following issues in wafer fab [9]:

- What priorities should be assigned to different jobs competing for the same resource and for late jobs?
- How to reduce time lost due to setups?
- Which batching decisions are performed well?
- How to take machine breakdown into account?

- How to avoid rework in some time-critical operations?

This is more complex than the traditional job-shop problem. The problem we considered involves reentrant flow, rework, unscheduled machine breakdown, sequence dependent setup times, batch processing machines, and due dates, but we assume the stocker capacity between machines are infinite.

GAs were proposed by Holland, whose colleagues and students at the University of Michigan in 1975 used it as a new learning paradigm to model a natural evolution mechanism [17]. Although GAs were not well known in the beginning, after the publication of Goldberg's book [18], GAs have recently attracted considerable attention in a number of fields such as methodology for optimization, and job-shop scheduling. In this paper, we allow our computation model to support the search algorithm over a CTPN model, i.e., search can be performed in both axes of multiple resources and different time segments. Here, we propose a new scheme to represent a schedule for the problem of production scheduling in wafer fab using GA embedded search over a CTPN model. Before we explain the approach to the GA, we first introduce the following notations.

- N_p: Population Size.
- N_g: Number of Generations.
- R_c: Crossover Rate.
- N_o: Number of Offsprings. $N_o = N_p \times R_c$
- P_u: Mutation Probability.
- P_i: Inversion Probability.

The algorithm starts with an initial set of random configurations called a population, which is a collection of chromosomes. The chromosome here denotes a total scheduling solution for wafer fabrication. The size of the population is always fixed N_p. Following this, a mating pool is established in which pairs

TABLE I
EQUIPMENT DESCRIPTION

Workstation		Type of Operation	Description	# of Machines	Batch Size	MST	MPT	MTBF	MTTR
NO.	Name								
1	CLAEN	Deposition	Clean wet bench for OXI/DIFF tubes	4	1	0.16	1.40	42.18	2.22
2	TMGOX	Deposition	Oxidation tube	4	1	0.50	4.48	101.11	10.00
3	TMNOX	Deposition	Oxidation tube	2	4	0.55	4.90	113.25	5.21
4	TMFOX	Deposition	Oxidation tube	2	1	0.47	4.21	103.74	12.56
5	TU11	Deposition	Metal alloy tube	2	1	0.61	5.53	100.55	6.99
6	TU43	Deposition	Annealing for silicides	2	1	0.78	6.98	113.25	5.21
7	TU72	Deposition	Low pressure CVD tube	2	1	0.62	5.61	16.78	4.38
8	TU73	Deposition	Low pressure SINI CVD tube	2	1	0.44	3.92	13.22	3.43
9	TU74	Deposition	Low pressure SI02 CVD tube	2	1	0.47	4.27	10.59	3.74
10	PLM5L	Deposition	Plasma enhanced CVD lower tube	2	1	0.41	3.65	47.53	12.71
11	PLM5U	Deposition	Plasma enhanced CVD upper tube	2	1	0.79	7.07	52.67	19.78
12	SPUT	Deposition	Perkin-Elmer 4400 sputter	2	1	0.61	5.49	72.57	9.43
13	PHPPS	Lithography	Pre-bake/positive spin resist	8	1	0.42	3.81	22.37	1.15
14	PHGCA	Lithography	GCA align/developers	8	1	0.78	7.04	21.76	4.81
15	PHHB	Lithography	Hardbake station	2	1	0.09	0.78	387.20	12.80
16	PHBI	Lithography	Bake inspect	4	1	0.30	2.66	No Failures	
17	PHFI	Lithography	Final inspect	2	1	0.16	1.40	119.20	1.57
18	PHJPS	Lithography	Positive spin resist	2	1	0.36	3.23	No Failures	
19	PLM6	Etching	Plasma etcher for aluminum	2	2	1.39	12.49	46.38	17.42
20	PLM7	Etching	Plasma etcher	2	1	0.54	4.87	36.58	9.49
21	PLM8	Etching	Oxide/nitride dry TEK etch	4	1	0.76	6.82	36.58	9.49
22	PHWET	Etching	Wet etch station	4	1	0.10	0.94	118.92	1.08
23	PHPLO	Resist Strip	Etchers and strip/clean for plasma etch	4	1	0.11	0.98	No Failures	
24	IMP	Ion Implant	Ion implanter	4	1	0.39	3.47	55.18	12.86

of individuals from the population are chosen. The probability of choosing a particular individual for mating is proportional to its fitness value. Chromosomes with higher fitness values have a greater chance of being selected for mating. Applying crossover R_c to generate new offsprings. Mutation and inversion are also applied with a low probability. Next, the offsprings generated are evaluated on the basis of fitness, and selecting some of the parents and some of the offsprings forms a new generation. The above procedure is executed N_g times, where N_g is the number of generations. After a fixed number of generations N_g, the fittest chromosome, i.e., the one with the highest fitness value is returned as the desired solution.

A. Chromosome Representation

In this paper, we use priority rule-based representation of chromosomes in the GA. This representation belongs to indirect approaches as described above. With this approach, brings us advantages such as the simplicity of the chromosome structure, simple GA operators, and shorter computation time. First, we define a gene as follows.

P_e A *gene set* is used to indicate the scheduling policy over the CTPN model.

That is, each machine group (identical machines) has a gene associated with it. A gene $g = (d, s, b)$ is a 3-tuple where

☐ *d*: one type of dispatching rules.
☐ *s*: one type of setup rules.
☐ *b*: one type of batching rules.

The rules we selected for gene codes are listed in Fig. 5. For each rule, it is described as follows:

☐ **FCFS**: First Come First Serve.
☐ **MINS**: Minimum Inventory at the Next Station first. In this rule, a lot has a higher priority if its next operation workstation has a lower inventory.
☐ **SRPT**: Shortest Remaining Processing Time first.
☐ **EDD**: Earliest Due Date first.
☐ **SSU**: Same Set-Up first.
☐ **W1T**: Waiting for the arrival lot to complete the batch within one unit of lot interarrival time. When the batch is completed within this specific period, the batch is started immediately. Otherwise, the partial batch is started right after one unit of lot interarrival time.
☐ **W2T**: The same as W1T, but the maximum waiting for completing a batch is two units of lot interarrival time.
☐ **W3T**: The same as W1T, but the maximum waiting for completing a batch is three units of lot interarrival time.
☐ **W4T**: The same as W1T, but the maximum waiting for completing a batch is four units of lot interarrival time.

The dispatching rules, setup rules and batching rules are directly related to the Elementary Module of the CTPN model.

TABLE II
PROCESS FLOW FOR ROUTE 1

Process Step	Station NO.	PT (hours)	ST (hours)	Process Step	Station NO.	PT (hours)	ST (hours)	Process Step	Station NO.	PT (hours)	ST (hours)
001	01	1.27	0.16	059	01	1.33	0.16	117	22	0.86	0.09
002	02	4.84	0.51	060	02	4.79	0.46	118	17	1.46	0.15
003	13	3.43	0.45	061	13	3.58	0.41	119	01	1.51	0.17
004	14	7.67	0.74	062	14	6.90	0.80	120	03	4.90	0.57
005	23	0.88	0.10	063	23	0.97	0.10	121	13	3.70	0.38
006	15	0.77	0.09	064	15	0.76	0.09	122	14	6.90	0.77
007	20	4.63	0.49	065	16	2.39	0.27	123	23	1.01	0.12
008	22	0.96	0.10	066	24	3.71	0.37	124	15	0.77	0.08
009	23	1.05	0.11	067	24	3.12	0.39	125	16	2.58	0.30
010	22	0.86	0.11	068	23	1.00	0.11	126	23	1.07	0.11
011	17	1.34	0.16	069	22	0.85	0.10	127	15	0.73	0.09
012	13	3.62	0.39	070	17	1.47	0.15	128	16	2.50	0.31
013	14	7.67	0.71	071	24	3.40	0.39	129	24	3.71	0.41
014	15	0.78	0.09	072	01	1.47	0.16	130	23	1.06	0.12
015	23	0.90	0.12	073	02	4.70	0.53	131	22	0.87	0.11
016	16	2.66	0.28	074	07	5.55	0.63	132	17	1.47	0.16
017	24	3.75	0.43	075	01	1.27	0.16	133	01	1.53	0.15
018	23	0.88	0.10	076	03	4.66	0.50	134	03	5.19	0.50
019	22	0.95	0.11	077	22	0.99	0.10	135	10	3.69	0.44
020	17	1.46	0.16	078	13	4.08	0.43	136	22	0.91	0.11
021	01	1.37	0.15	079	15	0.71	0.08	137	12	5.65	0.66
022	08	4.04	0.45	080	23	1.05	0.10	138	06	7.61	0.85
023	04	4.38	0.46	081	22	0.96	0.11	139	22	0.85	0.10
024	22	0.90	0.11	082	22	0.94	0.11	140	06	7.54	0.74
025	22	0.97	0.10	083	22	0.98	0.10	141	01	1.39	0.16
026	01	1.43	0.15	084	17	1.34	0.15	142	01	1.40	0.15
027	02	4.57	0.50	085	13	3.85	0.41	143	04	4.50	0.44
028	08	3.65	0.45	086	14	7.67	0.74	144	10	3.72	0.37
029	13	3.81	0.43	087	18	3.13	0.38	145	19	13.61	1.45
030	14	6.62	0.81	088	23	0.98	0.10	146	23	0.89	0.12
031	18	3.00	0.39	089	15	0.77	0.09	147	01	1.33	0.14
032	23	1.06	0.10	090	16	2.55	0.32	148	10	3.65	0.42
033	15	0.75	0.09	091	20	4.63	0.49	149	13	3.96	0.40
034	16	2.82	0.32	092	23	0.89	0.12	150	14	7.04	0.83
035	23	0.88	0.12	093	01	1.47	0.15	151	16	2.87	0.28
036	18	2.91	0.37	094	17	1.51	0.17	152	21	6.41	0.76
037	22	0.90	0.11	095	01	1.33	0.17	153	12	5.54	0.63
038	01	1.50	0.17	096	01	1.43	0.16	154	13	3.70	0.40
039	01	1.48	0.17	097	03	4.70	0.53	155	14	6.62	0.73
040	13	3.54	0.45	098	13	3.66	0.45	156	18	3.10	0.37
041	14	6.69	0.80	099	14	6.90	0.82	157	23	0.95	0.10
042	23	0.98	0.10	100	16	2.55	0.28	158	15	0.80	0.09
043	15	0.80	0.09	101	24	3.78	0.41	159	15	0.83	0.08
044	16	2.90	0.29	102	23	1.01	0.11	160	15	0.80	0.08
045	24	3.37	0.38	103	22	0.92	0.10	161	16	2.50	0.31
046	23	0.90	0.10	104	17	1.39	0.17	162	19	11.37	1.25
047	22	0.97	0.09	105	09	4.53	0.42	163	23	0.99	0.12
048	17	1.48	0.15	106	21	6.55	0.73	164	22	1.00	0.11
049	01	1.27	0.16	107	01	1.51	0.16	165	17	1.47	0.15
050	02	4.88	0.47	108	03	4.95	0.57	166	11	7.49	0.80
051	08	3.92	0.46	109	13	3.85	0.41	167	13	3.43	0.44
052	09	3.89	0.51	110	14	7.46	0.46	168	14	6.97	0.85
053	21	6.48	0.80	111	15	0.82	0.10	169	15	0.79	0.10
054	22	0.96	0.11	112	23	1.00	0.10	170	21	7.23	0.76
055	01	1.43	0.16	113	15	0.76	0.09	171	23	1.05	0.12
056	04	4.13	0.42	114	16	2.39	0.29 ·	172	05	5.09	0.56
057	22	0.97	0.10	115	24	3.26	0.37				
058	22	0.98	0.10	116	23	0.92	0.12				

The dispatching rules and set-up rules are associated with the transition *enter_j* and *enter2_j*. Each time when there are more than one token in place *buffer_j* (or *be_time_critical_j*) and the machine is idle, the scheduler must select one of the tokens (lots) to process. This selection depends on the rules it used. For example, if set-up rule "SSU" and dispatching rule "FCFS" is selected, it will first select the tokens with the same operation type of the last lot processed on machine. Then, we select among them using FCFS rule, the lot entering the queue with the earliest time will be selected first.

The batching rule that we called *dynamic batch-size* rule will automatically calculate the most suitable waiting time to complete a batch. Normally, the batch size of each machine is fixed so that, when the partial batch will be processed, the system will

TABLE III
PROCESS FLOW FOR ROUTE 2

Process Step	Station NO.	PT (hours)	ST (hours)	Process Step	Station NO.	PT (hours)	ST (hours)	Process Step	Station NO.	PT (hours)	ST (hours)
001	01	1.44	0.15	048	22	0.90	0.09	095	16	2.45	0.32
002	02	4.48	0.50	049	17	1.26	0.16	096	24	3.78	0.39
003	13	3.62	0.41	050	21	6.82	0.80	097	23	0.99	0.11
004	14	7.39	0.76	051	12	5.65	0.57	098	22	0.85	0.09
005	23	0.92	0.12	052	13	3.70	0.41	099	17	1.51	0.15
006	15	0.70	0.09	053	14	7.46	0.73	100	01	1.44	0.16
007	20	5.02	0.51	054	18	3.46	0.36	101	03	5.15	0.57
008	22	0.95	0.09	055	23	0.88	0.10	102	10	3.54	0.11
009	23	1.00	0.11	056	15	0.74	0.08	103	22	0.86	0.09
010	22	0.99	0.09	057	15	0.80	0.09	104	12	5.54	0.56
011	17	1.37	0.15	058	15	0.73	0.09	105	06	6.98	0.75
012	01	1.43	0.15	059	16	2.39	0.27	106	22	0.92	0.10
013	01	1.48	0.15	060	19	11.99	1.43	107	06	6.98	0.81
014	03	5.24	0.53	061	23	0.99	0.12	108	01	1.51	0.15
015	13	3.51	0.38	062	22	0.86	0.10	109	01	1.40	0.16
016	14	7.32	0.71	063	17	1.29	0.17	110	04	4.46	0.45
017	16	2.74	0.30	064	01	1.39	0.15	111	10	3.61	0.43
018	24	3.57	0.43	065	02	4.75	0.46	112	19	13.49	1.47
019	23	1.01	0.11	066	08	3.72	0.48	113	23	1.04	0.10
020	22	0.93	0.11	067	09	4.10	0.49	114	01	1.53	0.15
021	17	1.33	0.16	068	21	6.68	0.69	115	10	3.36	0.38
022	23	0.92	0.11	069	22	1.02	0.09	116	13	3.70	0.40
023	18	3.39	0.36	070	01	1.40	0.16	117	14	7.04	0.80
024	22	0.97	0.09	071	04	4.46	0.45	118	16	2.85	0.28
025	01	1.33	0.17	072	22	0.95	0.11	119	01	1.33	0.16
026	01	1.51	0.15	073	22	0.99	0.09	120	08	3.53	0.43
027	13	4.00	0.44	074	01	1.46	0.17	121	04	3.79	0.46
028	14	7.46	0.70	075	02	4.12	0.52	122	22	1.03	0.10
029	23	0.98	0.17	076	13	3.85	0.41	123	22	0.99	0.10
030	15	0.74	0.05	077	14	6.48	0.77	124	01	1.33	0.15
031	16	2.69	0.27	078	23	0.99	0.11	125	02	4.39	0.47
032	24	3.44	0.41	079	15	0.73	0.09	126	08	4.08	0.44
033	23	0.98	0.10	080	16	2.90	0.28	127	13	3.81	0.44
034	22	0.88	0.11	081	24	3.23	0.41	128	14	6.90	0.78
035	17	1.26	0.15	082	24	3.68	0.36	129	18	3.10	0.33
036	01	1.30	0.16	083	23	0.88	0.11	130	23	0.97	0.11
037	03	5.05	0.51	084	22	0.88	0.10	131	15	0.84	0.08
038	13	3.62	0.45	085	17	1.40	0.15	132	16	2.39	0.31
039	14	6.69	0.73	086	09	4.44	0.46	133	11	7.42	0.81
040	23	1.00	0.10	087	21	7.37	0.78	134	13	3.73	0.46
041	15	0.77	0.09	088	01	1.34	0.16	135	14	6.41	0.80
042	16	2.63	0.28	089	03	4.61	0.54	136	15	0.83	0.10
043	23	1.03	0.12	090	13	3.92	0.43	137	21	6.89	0.78
044	15	0.83	0.08	091	14	6.55	0.77	138	23	0.95	0.11
045	16	2.66	0.33	092	15	0.74	0.10	139	05	5.59	0.56
046	24	3.16	0.38	093	23	1.03	0.12				
047	23	0.96	0.11	094	15	0.73	0.09				

automatically insert dummy lots (or wafers) to the partial batch to complete a batch.

After genes are defined, the chromosome can be created. In this paper, the length of a chromosome is fixed, and is equivalent to the number of machine groups. The structure of the chromosome is depicted in Fig. 6.

B. Fitness Function

One of the simplest methods to combine multiple objective functions into a scalar fitness solution is the following weighted sum approach:

$$f(c) = w_1 \cdot f_1(c) + \cdots + w_i \cdot f_i(c) + \cdots + w_n \cdot f_n(c)$$

where

c	chromosome (i.e., solution);
$f(c)$	combined fitness function;
$f_i(c)$	ith objective function;
w_i	constant weight for $f_i(c)$;
n	number of the objective functions.

We use *two* objective functions in our implementation. The fitness function is defined as follows:

$$f(c) = w_1 \cdot f_1(c) + w_2 \cdot f_2(c)$$

where f_1 is the score for mean cycle time and f_2 is the score for number of lots which meet due date.

Before evaluating the fitness function, f_1 and f_2 must be determined first. We calculate f_1 and f_2 by the following way: For f_1, sort all the chromosomes by mean cycle time, and give each chromosome a rank such that the one with shorter mean cycle

TABLE IV
PROCESS FLOW FOR ROUTE 3

Process Step	Station NO.	PT (hours)	ST (hours)	Process Step	Station NO.	PT (hours)	ST (hours)	Process Step	Station NO.	PT (hours)	ST (hours)
001	01	1.46	0.14	038	22	0.98	0.10	075	19	12.74	1.43
002	02	4.08	0.50	039	17	1.36	0.17	076	23	1.08	0.12
003	13	3.51	0.40	040	13	3.43	0.43	077	01	1.46	0.17
004	14	7.25	0.78	041	14	7.67	0.83	078	10	3.36	0.43
005	23	0.89	0.12	042	15	0.76	0.09	079	13	3.43	0.43
006	15	0.79	0.08	043	23	1.06	0.10	080	14	6.69	0.76
007	20	4.87	0.51	044	16	2.42	0.33	081	16	2.90	0.31
008	22	0.94	0.11	045	24	3.16	0.40	082	01	1.26	0.15
009	23	1.06	0.12	046	23	0.95	0.10	083	02	4.52	0.45
010	22	0.86	0.10	047	22	1.02	0.10	084	08	3.96	0.41
011	17	1.43	0.17	048	17	1.41	0.15	085	09	4.06	0.43
012	01	1.36	0.16	049	21	7.02	0.81	086	21	7.02	0.78
013	08	3.72	0.45	050	12	5.54	0.56	087	22	0.90	0.10
014	04	4.08	0.51	051	13	4.15	0.42	088	01	1.47	0.16
015	22	0.89	0.10	052	14	7.39	0.83	089	04	3.83	0.46
016	22	0.93	0.10	053	18	3.17	0.34	090	22	1.03	0.09
017	01	1.50	0.18	054	23	0.91	0.10	091	22	0.95	0.10
018	02	4.17	0.52	055	15	0.70	0.10	092	01	1.40	0.15
019	08	3.76	0.44	056	15	0.83	0.10	093	02	4.44	0.54
020	13	3.89	0.45	057	15	0.79	0.09	094	13	3.89	0.46
021	14	7.04	0.80	058	16	2.90	0.28	095	14	7.32	0.82
022	18	3.33	0.33	059	19	13.74	1.36	096	23	0.99	0.11
023	23	1.01	0.10	060	23	0.99	0.12	097	15	0.74	0.09
024	15	0.80	0.10	061	22	0.99	0.10	098	16	2.63	0.32
025	16	2.71	0.33	062	17	1.27	0.16	099	24	3.16	0.43
026	09	4.18	0.44	063	01	1.29	0.15	100	24	3.12	0.40
027	21	7.43	0.83	064	03	4.75	0.54	101	23	0.93	0.11
028	01	1.51	0.15	065	10	3.50	0.39	102	22	0.85	0.11
029	03	4.90	0.58	066	22	0.97	0.11	103	17	1.40	0.16
030	13	4.15	0.43	067	12	5.65	0.65	104	11	7.56	0.77
031	14	7.53	0.83	068	06	7.61	0.80	105	13	3.54	0.43
032	15	0.73	0.09	069	22	0.99	0.10	106	14	7.32	0.71
033	23	0.95	0.11	070	06	6.98	0.74	107	15	0.76	0.10
034	15	0.82	0.09	071	01	1.27	0.16	108	21	7.02	0.82
035	16	2.69	0.32	072	01	1.34	0.16	109	23	0.90	0.11
036	24	3.12	0.42	073	04	4.63	0.48	110	05	5.86	0.57
037	23	0.93	0.11	074	10	3.36	0.44				

time can get a higher rank. For example, we now have 4 chromosomes c_1, c_2, c_3 and c_4 and

$$MCT(c_1) = 843, \quad MCT(c_2) = 821,$$
$$MCT(c_3) = 859, \quad MCT(c_4) = 833,$$

where MCT is the mean cycle time, then

$$f_1(c_1) = 2, \quad f_1(c_2) = 4,$$
$$f_1(c_3) = 1, \quad f_1(c_4) = 3.$$

For f_2, sort all the chromosomes by the number of lots which meet the due date, and give each chromosome a rank such that the more number of lots which meet the due date can get a higher rank.

Note that the functions $f_1(c)$ and $f_2(c)$ are transformed into functions with discrete rank values. The purpose of this transformation is to make the selection of the value w_i in the fitness function more easily. After transformation, the value of cycle time and meet-due-date rate can fall into the same range (in following example 1~5). Then if we assign w_1 with the value larger than w_2, which means the cycle time is more important than meet-due-date rate in fitness function, and vice versa.

Therefore, assume that we have fivechromosomes c_1, c_2, c_3, c_4, *and* c_5, and each chromosome leads to the following values:

$$f_1(c_1) = 2; \quad f_2(c_1) = 5$$
$$f_1(c_2) = 1; \quad f_2(c_2) = 4$$
$$f_1(c_3) = 4; \quad f_2(c_3) = 3$$
$$f_1(c_4) = 5; \quad f_2(c_4) = 1$$
$$f_1(c_5) = 3; \quad f_2(c_5) = 2$$

If $w_1 = 0.5$, $w_2 = 0.5$, then

$$f(c_1) = 3.5, \quad f(c_2) = 2.5, \quad f(c_3) = 3.5, \quad f(c_4) = 3$$
$$f(c_5) = 2.5.$$

Thus, c_1 or c_3 is the best chromosome.

C. Genetic Operators

Crossover: Crossover is the main genetic operator. It operates on two individuals and generates an offspring. It is an inheritance mechanism where the offspring inherits some of the characteristics of the parents. The operation consists of choosing a random cut point and generating the offspring by combining the

TABLE V
THREE ORDERS

Order ID	Route	Quantity (lots)	Due Date
Order 01	Route 1	30	1999/12/18 08:00
Order 02	Route 2	20	2000/01/03 08:00
Order 03	Route 3	50	2000/01/13 08:00

System date in the beginning of simulation: 1999/10/30

TABLE VI
THE CONTENTS OF THE CHROMOSOME

Gene\Rule	Dispatch rule	Setup rule	Batch rule
Gene 01	3	2	4
Gene 02	4	2	3
Gene 03	3	2	1
Gene 04	2	2	1
Gene 05	4	1	2
Gene 06	1	1	2
Gene 07	4	1	3
Gene 08	1	1	4
Gene 09	1	2	4
Gene 10	3	1	1
Gene 11	1	1	3
Gene 12	3	2	4
Gene 13	4	1	3
Gene 14	2	1	4
Gene 15	2	1	3
Gene 16	4	2	1
Gene 17	2	1	1
Gene 18	2	1	1
Gene 19	2	1	2
Gene 20	2	1	3
Gene 21	4	1	2
Gene 22	4	1	3
Gene 23	2	1	2
Gene 24	1	1	2

TABLE VII
THE COMPARED RESULT FOR CASE 1

Item	MQT (hours)		MDD (%)	
	Mean	Std-Dev	Mean	Std-Dev
FCFS	468.6	7.8	69.4	3.0
FCFS+SSU	627.2	66.8	44.4	9.3
SRPT	472.8	15.9	74.2	2.7
SRPT+SSU	551.2	31.2	56.3	8.2
MINS	462.0	11.9	71.4	3.8
MINS+SSU	628.8	65.3	43.5	9.7
EDD	500.1	18.0	64.5	6.7
EDD+SSU	628.1	68.2	38.3	11.3
GA	377.1	4.8	91.4	0.9

TABLE VIII
FOUR ORDERS TO BE RELEASED INTO THE FAB

Order ID	Route	Quantity (lots)	Due Date
Order 01	Route 2	20	1999/12/12 08:00
Order 02	Route 3	20	1999/12/17 08:00
Order 03	Route 1	10	1999/12/24 08:00
Order 04	Route 3	30	2000/01/04 08:00

System date in the beginning of simulation: 1999/10/30

TABLE IX
THE COMPARED RESULT FOR CASE 2

Item	MQT (hours)		MDD (%)	
	Mean	Std-Dev	Mean	Std-Dev
FCFS	443.3	5.8	62.2	2.9
SRPT	490.8	57.0	74.0	3.1
MINS	439.2	8.6	62.2	2.8
EDD	474.3	11.1	49.8	3.1
GA	372.7	5.1	82.5	2.5

segment of one parent to the left of the cut point with the segment of the other parent to the right of the cut.

Mutation: Mutation produces incremental random changes in the offspring generated by the crossover. The commonly used mutation mechanism is pairwise interchange. We also use this mechanism in our implementation. Fig. 8 shows an example.

Inversion: We add an additional genetic operator called *inversion* [16] in the implementation. In the *inversion* operation, two points are randomly chosen along the length of the chromosome so that the order of genes within the two end points of the chromosome reverses.

D. Schedule Builder

A schedule builder is dedicated to transforming a chromosome to a feasible schedule, such that we can evaluate the aforementioned indirect chromosome representation. Based on the CTPN model, the evolution of the system can be addressed by the change of marking in the net.

Consequently, all possible kinds of behavior of the system can be completely tracked by the reachability graph of the net. In other words, we can track the WIP status from the CTPN model while the schedule was performed. A chromosome consists of many genes, each of which consists of rules which are directly related to the Elementary Module of CTPN model. Thus, given a CTPN model and a chromosome, the schedule builder can generate a feasible schedule that consists of the firing se-

quence of the transitions, which provides the order of the initiation of operations. In the following algorithm, we describe how a schedule builder uses the chromosome to construct a feasible schedule over the CTPN model. One simulation of the whole model carried out for the policy that corresponds to the chromosome. Once the scheduling environment is changed, the scheduler builder will be executed.

The architecture of the scheduler was shown in Fig. 7, in which, we first select a lot release policy to control the timing for release of a lot (token) to the CTPN model. Second, apply the GA over the CTPN to find a good chromosome.

Algorithm 1 Schedule Builder
Begin
```
Select a chromosome c_j ∈ C;
Apply the scheduling rules to the according genes;
Set the marking m as the initial marking m_0;
Find the set of all enabled transitions in the marking m, namely T_e(m);
```
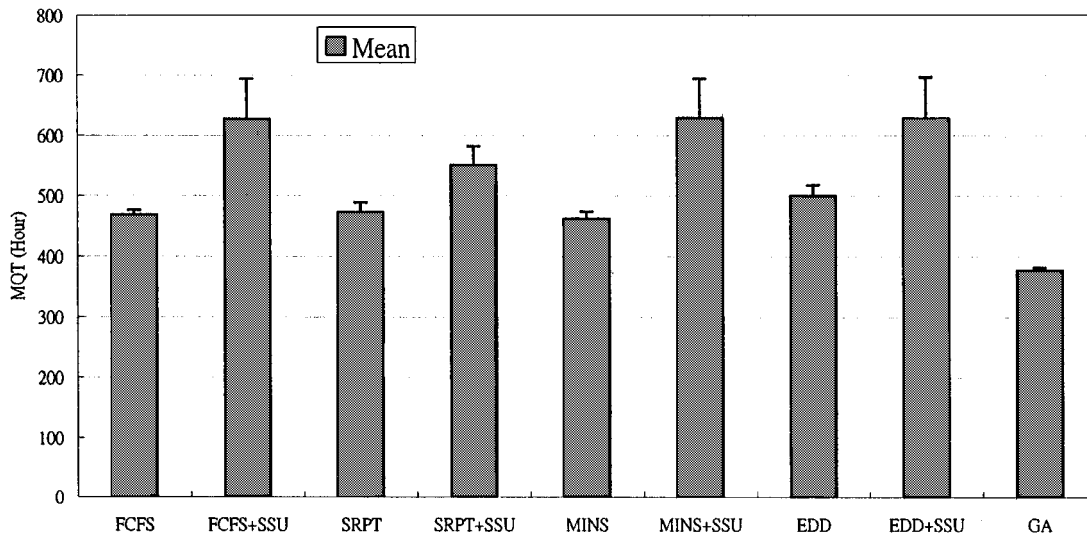
Fig. 12. Performance evaluation for mean queuing time in case 1.

```
While m is not the final marking  Do
  If T_e(m) == NULL   Then
     Update the elapsing time;
     Set the marking m as the current
     marking m;
     Update T_e(m);
  Else
     For all t ∈ T_e(m)  Do
        Fire the transition t by according
        scheduling rules if necessary;(if
        the transition t is associated with
        rule, we have to check which color
        of this transition can be fired
        first to resolve the conflict)
        Set the marking m as the current
        new marking m;
        Update T_e(m);
     End For
  End IF
End While
End.
```

V. IMPLEMENTATION

In order to implement the systematic wafer fab modeling as well as the scheduling method, we develop a simulation tool called *UltraSim* (see Fig. 8) that was designed by an object oriented method and programming with C++ in Microsoft Visual C++ 6.0. In Fig. 9, it is a main window of the *UltraSim.*

The additional function of *UltraSim* is that it can access the remote shared database through ODBC (open database connectivity) manager. The remote shared database here, is implemented by Microsoft SQL Server 6.5. Fig. 10 is the remote data access architecture between *UltraSim* and SQL Server. Through the remote database, we can easily get the WIP status for simulation. This is so convenient that we do not need to run the model from the empty fab, and, the results of simulation will be more close to a real fab. Besides the remote shared database, which can be used in *UltraSim*, we can also save/load our model and

WIP status to/from files. This characteristic makes *UltraSim* applicable almost everywhere, even if the network does not function. Moreover, *UltraSim* provides a friendly user interface to help users to create a CTPN model.

In addition, *UltraSim* has a lot release wizard to help customers to insert their orders easily. In Fig. 11, we have two choices to select scheduling policies to be applied to the production scheduling. The two kinds of scheduling policies are simple dispatching rules and genetic algorithm scheduler. If we select simple dispatching rules, we must select one of the four dispatching rules in advance. After the operation on the lot release wizard is finished, the selected scheduling policy will be shown in status bar under the main window. Moreover, if results is good enough compared to the popular schedule rules, then, the GA search is stopped.

VI. EXPERIMENT RESULTS

The simulation model we used in this paper is based on Wein's model [1]. It describes a fictitious wafer fab, but most of the parameters of the model are derived from data gathered at the Hewlett-Packard Technology Research Center Silicon fab (the TRC fab), which is a large R&D facility in Palo Alto, CA. Unlike the model studied by Wein [1], we enlarge the capacity of the fab to increase the complexity of the simulation. Moreover, we add the rework probability to the inspection machines, and define the reworked step in each route. Batch processing machines are also included in our simulation model, so that the simulation model can be close to real fab. We assume both of the workstations of TMNOX and PLM6 are batch-processing machines, and their batch sizes are 4 and 2, respectively. The whole equipment information of the simulation fab is listed in Table I, which describes the detailed parameters of the fab, such as mean processing time (MPT), mean set-up time (MST), mean time to repair (MTTR), and mean time between failure (MTBF).

In the simulation model, the machine downtime that consists of unscheduled breakdowns. Time between failures and time to repair for each workstation is randomly generated from uniform
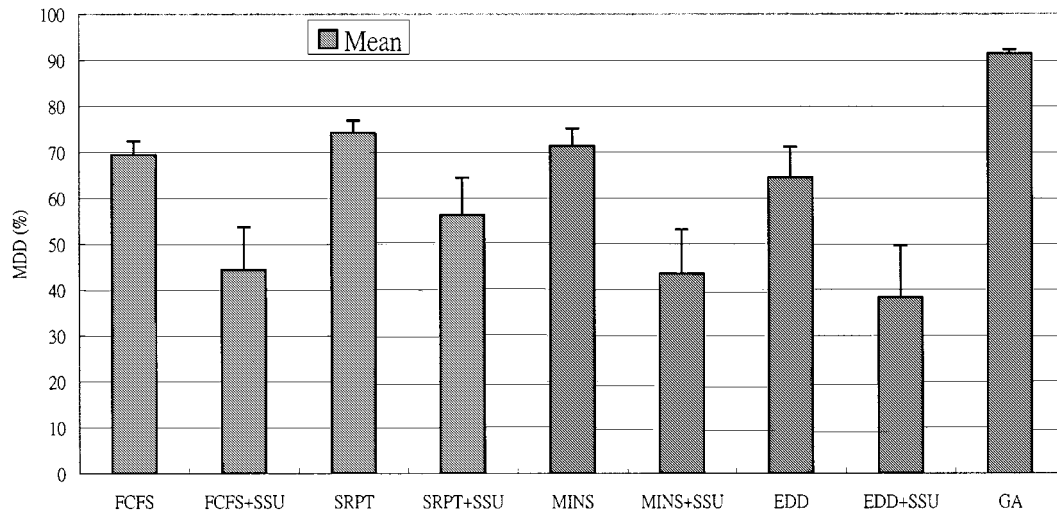
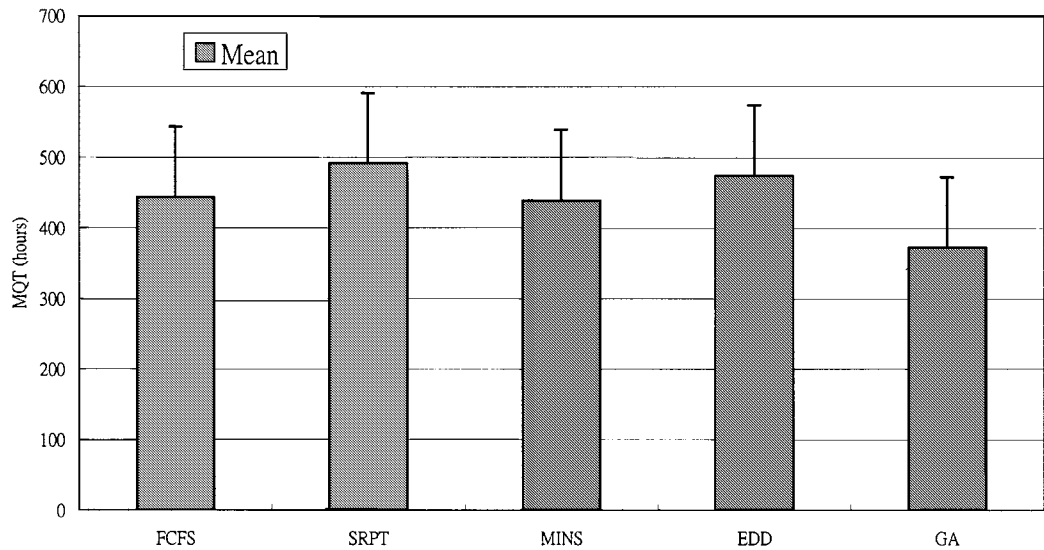Fig. 13. Performance evaluation for meeting due date in case 1.

Fig. 14. Performance evaluation for mean queuing time in case 2.

distributions with given mean values in implementation. In TRC fab model, the *MPT* contains the mean processing time and mean set-up time, but we separate them in our simulation model. The *MPT* in our simulation model is equal to $0.9 \times MPT$ in TRC fab model, and the *MST* is equal to $0.1 \times MPT$ in TRC fab model.

In our simulation model, each lot entering the fab is based on a specific process flow. There are three different process flows in our simulation model, and the sequence of stations to be visited in three process flows are listed in Tables II–IV, respectively, where the numbers refer to the station numbers in the first column of Table I. The process flow for *route 1* listed in Table II is the same as the one earlier studied by Wein [1], which requires 12 loops, i.e., a lot visits lithography exposure workstations 12 times. The process flow for *route 2* and *route 3* is generated by randomly selecting 10 loops and 8 loops from *route 1*, respectively.

We assume the re-entrance of operation on the same machine may have different processing time, since they have different recipes to be processed. In addition, if the machine takes processes with different recipes, the machine must be set-up. Here,

the processing time (PT) for a lot is randomly generated from a uniform distribution between $0.9 \times MPT$ and $1.1 \times MPT$, where MPT is given for each station in Table I. It is likewise for the set-up time (ST), which is randomly generated from a uniform distribution between $0.9 \times MST$ and $1.1 \times MST$, where MST is given for each station in Table I.

In our experiments, the average population size is 10, the average generation size is 10, the crossover rate is 0.7, the mutation probability is 5%, and the inversion probability is 1%.

Case 1: Three Orders for Total 100 Lots: The problem description is listed in Table V, and the chromosome searched by GA scheduler is listed in Table VI, where the rules listed in the chromosome can be referred to Fig. 5 in Section III. Note that if workstations do not perform batch-processing operation, their corresponding batch rules are useless. For the four simple dispatching rules, the maximum waiting time to complete a batch is 24 hours. The lot interarrival time we used in this simulation is 12 hours for all the scheduling policies, since this lot interarrival time can build a steady WIP level for our simulation model.
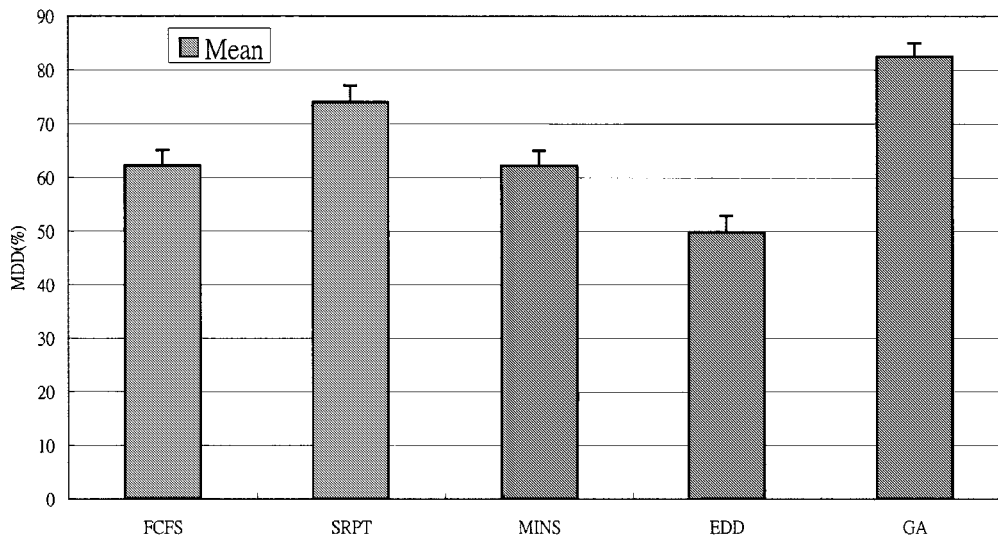
Fig. 15.   Performance evaluation for meeting due date in case 2.

We evaluated the performance of the five scheduling policies, which are FCFS, SRPT, MINS, EDD, and GA. In addition, the four simple dispatching rules plus SSU are also evaluated. For each scheduling policy, we run 10 times of simulation and calculate mean value and standard deviation. The compared results are listed in Table VII, where the two criteria are mean queuing time (MQT) and the rate of meeting due date (MDD). The total computation time of case 1 is 23 minutes (on Pentium II 450MHz with 128MB SDRAM).

Case 2: Four Orders for Total 80 Lots: Unlike the case 1, we have four orders in the case 2. In addition, the route sequence we used as the input pattern in the case 2 is different from the case 1. However, the chromosome used in the case 2 is the same as one used in the case 1, i.e., we did not re-schedule to search for the new chromosome given for the new input pattern. The purpose of using the same chromosome is to test whether the pre-searched chromosome can be applicable under different input patterns and can still perform well. As for other conditions, they are made the same as in case 1. Here we listed the four orders, and one compared result in Tables VIII and IX, respectively. The total computation time of case 2 is 20 minutes (on Pentium II 450MHz with 128MB SDRAM).

From Figs. 14 and 15, we found that the proposed GA scheduler performs much better than other conventional dispatching rules. It has a lower queuing time for lots spent in the fab, a higher rate for meeting the customers' due date. In addition, the experimental results show that the proposed GA scheduler has a lower variability on the total queuing time and the rate of meeting duedate, which increases the accuracy of the simulation based prediction. As a result, the proposed GA scheduler has a significant impact on wafer fab scheduling, by providing obvious improvements over the other conventional dispatching rules, even though the fab has a mixed production.

VII. Conclusion

In this paper, we consider the wafer fab scheduling problem. We first proposed a systematic CTPN model for a wafer fab. The entire CTPN model is composed of two modules, one is routing module, and the other is elementary module. Then, in order to make better scheduling policies on wafer fab, we proposed a genetic algorithm scheduler, which dynamically searches for the appropriate dispatching rules for each machine group or processing unit family. Through the experiments, we found that the GA scheduler provides more superior performance than the conventional dispatching rules do. In implementation, we developed a simulation tool for modeling and scheduling for wafer fab. The simulation tool, called *UltraSim*, provides a friendly user interface to help users to model their fab easily, and provides an efficient method to search for a better solution on wafer fab scheduling by using genetic algorithm.

References

[1] L. M. Wein, "Scheduling semiconductor wafer fabrication," *IEEE Trans. Semicond. Manufact.*, vol. 1, pp. 115–130, 1988.

[2] C. R. Glassey and M. C. Resende, "Closed-loop job release control for VLSI circuit manufacturing," *IEEE Trans. Semicond. Manufact.*, vol. 1, pp. 36–46, 1988.

[3] H. Chen, J. M. Harrison, A. Mandelbaum, A. V. Ackere, and L. M. Wein, "Empirical evaluation of a queuing network model for semiconductor wafer fabrication," *Operat. Res.*, vol. 36, no. 2, pp. 202–215, 1988.

[4] C. R. Glassey and W. W. Weng, "Dynamic batching heuristic for simultaneous processing," *IEEE Trans. Semicond. Manufact.*, vol. 4, pp. 77–82, 1991.

[5] R. Uzsoy, L. A. Martin-Vega, C. Y. Lee, and P. A. Leonard, "Production scheduling algorithm for a semiconductor test facility," *IEEE Trans. Semicond. Manufact.*, vol. 4, pp. 270–279, 1991.

[6] R. Uzsoy, C. Y. Lee, and L. A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry—Part I: System characteristics, performance evaluation and production planning," *IIE Trans.*, vol. 24, no. 4, pp. 47–60, 1992.

[7] P. K. Johri, "Practical issues in scheduling and dispatching in semiconductor wafer fabrication," *J. Manufact. Syst.*, vol. 12, no. 6, pp. 474–485, 1993.

[8] R. Uzsoy, C. Y. Lee, and L. A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry—Part II: Shop-floor control," *IIE Trans.*, vol. 26, no. 5, pp. 44–55, 1994.

[9] L. Duenyas, J. W. Fowler, and L. W. Schruben, "Planning and scheduling in Japanese semiconductor manufacturing," *J. Manufact. Syst.*, vol. 13, no. 5, pp. 323–332, 1994.

[10] S. Li, T. Tang, and D. W. Collins, "Minimum inventory variability schedule with applications in semiconductor fabrication," *IEEE Trans. Semicond. Manufact.*, vol. 9, pp. 145–149, 1996.

[11] Y. Narahari and L. M. Khan, "Modeling the effect of hot lots in semi-conductor manufacturing systems," *IEEE Trans. Semicond. Manufact.*, vol. 10, pp. 185–188, 1997.

[12] Y. D. Kim, J. U. Kim, S. K. Lim, and H. B. Jun, "Due-date based scheduling and control policies in a multiproduct semiconductor wafer fabrication facility," *IEEE Trans. Semicond. Manufact.*, vol. 11, no. 1, pp. 155–164, 1998.

[13] Y. D. Kim, D. H. Lee, and J. U. Kim, "A simulation study on lot release control, mask scheduling, and batch scheduling in semiconductor wafer fabrication facilities," *J. Manufact. Syst.*, vol. 17, no. 2, pp. 107–117, 1998.

[14] S. A. Mosley, T. Teyner, and R. M. Uzsoy, "Maintenance scheduling and staffing policies in a wafer fabrication facility," *IEEE Trans. Semicond. Manufact.*, vol. 11, no. 2, pp. 316–323, 1998.

[15] S. M. Sze, *VLSI Technology*. New York: McGraw-Hill, 1983.

[16] S. M. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*. New York: McGraw-Hill, 1995.

[17] J. H. Holland, *Adaptation in Natural and Artificial Systems*: Michigan Univ. Press, 1975.

[18] D. E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[19] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithm—Part I: Representation," *Comput. Indust. Eng.*, vol. 30, no. 4, pp. 983–997, 1996.

[20] C. Y. Lee, S. Piramuthu, and Y. K. Tsai, "Job shop scheduling with a genetic algorithm and machine learning," *Int. J. Prod. Res.*, vol. 35, no. 4, pp. 1171–1191, 1997.

[21] R. S. Lee and M. J. Shaw, "A genetic algorithm-based approach to flexible flow-line scheduling with variable lot sizes," *IEEE Trans. Syst., Man, Cybern. B*, vol. 27, pp. 36–54, 1997.

[22] G. Ulusoy, S. S. Funda, and U. Bilge, "A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles," *Comput. Operat. Res.*, vol. 24, no. 4, pp. 335–351, 1997.

[23] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*: McGraw-Hill, 1981.

[24] A. A. Desrochers and R. Y. Al-Jaar, *Applications of Petri Nets in Manufacturing Systems: Modeling, Control, and Performance Analysis*. New York: IEEE Press, 1994.

[25] M. H. Lin and L. C. Fu, "Modeling, analysis, simulation, and control of semiconductor manufacturing systems: A generalized stochastic colored-timed Petri-net approach," in *IEEE Int. Conf. Syst., Man, Cybern.*, 1999.

[26] R. Zurawski and M. C. Zhou, "Petri nets and industrial applications: A tutorial," *IEEE Trans. Industrial Electron.*, vol. 41, pp. 567–583, 1994.

[27] M. C. Zhou and M. D. Jeng, "Modeling, analysis, simulation, scheduling, and control of semiconductor manufacturing systems: A Petri net approach," *IEEE Trans. Semicond. Manufact.*, vol. 11, pp. 333–357, 1998.

[28] M. D. Jeng, X. Xie, and S. W. Chou, "Modeling, qualitative analysis, and performance evaluation of the etching area in an IC wafer fabrication system using Petri nets," *IEEE Trans. Semicond. Manufact.*, vol. 11, pp. 358–373, 1998.

[29] H. H. Xiong and M. C. Zhou, "Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search," *IEEE Trans. Semicond. Manufact.*, vol. 11, pp. 384–393, 1998.

[30] S. Y. Lin and H. P. Huang, "Modeling and emulation of a furnace in IC fab based on colored-timed Petri net," *IEEE Trans. Semicond. Manufact.*, vol. 11, pp. 410–420, 1998.

[31] K. Jensen, "Colored Petri nets and the invariant method," in *Theoretical Computer Science*. Amsterdam: North-Holland, 1981, vol. 14, pp. 317–336.

Li-Chen Fu was born in Taipei, Taiwan, R.O.C., in 1959. He received the B.S. degree from National Taiwan University, Taipei, Taiwan, in 1981, and the M.S. and Ph.D. degrees from the University of California, Berkeley, in 1985 and 1987, respectively.

Since 1987, he has been a professor with both the Department of Electrical Engineering and the Department of Computer Science and Information Engineering of National Taiwan University. He currently also serves as the Deputy Director of Tjing Ling Industrial Research Institute of National Taiwan University. His areas of research interest include robotics, FMS scheduling, shop floor control, home automation, visual detection and tracking, E-commerce, and control theory and applications.

Dr. Fu is a member of the IEEE Robotics and Automation Society and Automatic Control Society. He is also a member of the Board of the Chinese Automatic Control Society and the Chinese Institute of Automation Engineers. During 1996–1998 and 2000, he was appointed a member of AdCom of IEEE Robotics and Automation Society, and will serve as the Program Chair of 2003 IEEE International Conference on Robotics and Automation. He has been Editor of *Journal of Control and Systems Technology* and an Associate Editor of the prestigious control journal, *Automatica*. Since 1999, he has been Editor-in-Chief of a new control journal, the *Asian Journal of Control*.

Ming-Hung Lin (M'88) received the M.S. and Ph.D. degrees from National Taiwan University, Taipei, Taiwan, R.O.C., in 1993 and 2001, in computer science and information engineering, respectively.

Since 1995, he has been with Computer Communication Research Labs., Industrial Technology Research Institute (ITRI), where he is researcher on advanced broadband wired and wireless communication systems. He has been involved in several system prototypes such as PC-based video-on-demand, mobile Internet system, and RLC/MAC protocol for GPRS/UMTS. He joined Philips Research East Asia in December 1999, where he is working on information appliance systems. His current research interests include wireless connectivity for home and away environment (Bluetooth and UMTS), mobile multimedia systems, VLSI design for mobile display system, hybrid queuing theorem, and automatic production research.

An-Chih Huang received the B.S. and M.S. degrees from National Taiwan University, Taipei, Taiwan, R.O.C., in 1999 and 2001, respectively.

His research interests include modeling, simulation, prediction, and scheduling of manufacturing systems. He is currently fulfilling his military service.

Jyh-Horng Chen received the M.S. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1999.

His research interests include modeling, simulation, and scheduling of manufacturing systems. He joined TSMC, the largest domestic semiconductor company, as an engineer immediately following graduation.