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-
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 demonstrate two examples of using the proposed CTPN model. In Section V, we briefly describe the implementation of the simulation tool. In Section VI, we demonstrate two examples 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 $\text{CTPN} = (P_\tau, P_c, T_u, T_i, T_s, T_p, T_m, C, I, O, M)$, where:

- $P_\tau$ set of places;
- $P_c$ set of communication places;
- $T_u$ set of immediate transitions;
- $T_i$ 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.

$\square P = \{p_1, p_2, \ldots, p_l\}; P$ 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.

$\square T = \{t_1, t_2, \ldots, t_c\}; T$ is a finite set of transitions with $c \geq 1$, 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.

$\square P \cup T \neq \emptyset, P \cap T = \emptyset$.

$\square C(p) = \{p_i, \ldots, p_c\}, u_i = |C(p_i)|, i = 1, \ldots, n\}$. Where $n$ are nonnegative integers; the $p_c$s are the associated colors, and $C(p)$ represents the color sets of places in $P$. 

where $m$ are nonnegative integers; the $c_{ij}$ are the associated colors. $C(p)$ and $C(t)$ represents the color sets of places in $P$ and transitions in $T$.

$\delta(I(p_i, t_j)(c_{i,j}, t_{c,j})): C(p) \times C(t) \rightarrow N$ (nonnegative integers) corresponding to the input arc from a place $p_i$ with respect to the color $c_{i,j}$, to a transition $t_j$, with respect to the color $t_{c,j}$.

$\delta(H(p_i, t_j)(c_{i,j}, t_{c,j})): C(p) \times C(t) \rightarrow \{1, 0\}$ corresponding to the inhibitor arc from a place $p_i$ with respect to the color $c_{i,j}$, to a transition $t_j$, with respect to the color $t_{c,j}$.

$\delta(O(p_i, t_j)(c_{i,j}, t_{c,j})): C(t) \times C(p) \rightarrow N$ (nonnegative integers) corresponding to the output arc from a transition $t_j$ with respect to the color $t_{c,j}$, to a place $p_i$ with respect to the color $c_{i,j}$.

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 $(u \times 1)$ vector. The tokens in a place now become a summation of place colors: $M(p_i) = \sum n_{ih} c_{ih}$, for $h = 1, \ldots, u$, where $n_{ih}$ is the number of tokens of color $c_{ih}$ in place $p_i$, denoted by $n_{ih} = M(p_i)(c_{ih})$.

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

Time Function: $f(t_j)(t_{c,j}): C(t) \rightarrow N$, is the time delay required by the deterministically timed transition $t_j$ with the token with respect to the color $t_{c,j}$. 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.

### 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 $C(t) = C(p) = \{x2wus\mid x \in X, z \in Z, w \in W, s \in S\}$, where the color is encoded as a string $x2wus$, $X$ 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 $x2wus0$ enters the model, where $x$ is the route ID, $z$ is the lot ID, $w$ is the operation ID and $s0$ is operation step. This token (lot) is then checked in and assigned into the place dispatch with the color $x2wus1$.

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.

- **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., $x2wus0$).
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., $x_2w_{y_{j1}}$). 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 ($\text{class}_1$, $\text{class}_2$, $\ldots$, $\text{class}_i$, $\ldots$, $\text{class}_n$) 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**: The transition $\text{class}_j$ ($j = 1, 2, \ldots, n$) 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 $x_4$ can be processed on $j$th workstation, its color set will be $C(\text{class}_j) = \{xzws|x \in X, z \in Z, w \in W, s \in \{s_3, s_5\}\}$. In the wafer fab, this transition controls what operation steps can be done in the $j$th workstation.

- **travel**: The place $\text{travel}_j$ is a place. A token in the place $\text{travel}_j$ ($j = 1, 2, \ldots, n$) represents that the loop-vehicle or AGV was used to transport the corresponding lot to the $j$th workstation.

- **end_travel**: The transition $\text{end_travel}_j$ ($j = 1, 2, \ldots, n$) is a stochastic transition. The time delay of this transition is the mean transportation time of the lot. A token fired through the transition $\text{end_travel}_j$ ($j = 1, 2, \ldots, n$) represents that the corresponding lot has been transported to the buffer of $j$th workstation.

- **buffer**: The place $\text{buffer}_j$ ($j = 1, 2, \ldots, n$) is a communication place, which is an interconnection among Routing Module and Elementary Module. A token in the place $\text{buffer}_j$ ($j = 1, 2, \ldots, n$) represents that a lot is in the buffer of $j$th workstation and prepares to be processed.

- **elementary**: The transition $\text{elementary}_j$ ($j = 1, 2, \ldots, n$) is a macro transition. It represents the $j$th 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 $s_5$ step, and we assume that the next operation type is $w_7$ and step is $s_0$. After transition step-up is fired, the token will be assigned into the place with the color $x_2z_3w_7s_0$. 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.
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.

\[ \text{be}_{\text{time-critical}} \]

The place \( \text{be}_{\text{time-critical}} \) is a place. Tokens in this place represent that the corresponding lots have higher priority and can be performed first in the \( j \)-th workstation. A token in this place (with any color) will block the transition \( \text{enter}_{j} \) of the operating subnet. Thus the priority of the lots (tokens) in \( \text{time-critical} \) is higher than the priority of lots (tokens) in \( \text{buffer} \), this guarantees that the lot with time-critical operation can be done first to avoid unnecessary rework.

\[ \text{buffer}_{j} \]

The place \( \text{buffer}_{j} \) is a communication place. A token in \( \text{buffer}_{j} \) represents that a lot is ready to be processed in the \( j \)-th workstation. The color set of \( \text{buffer}_{j} \) is \( \{x \in X, \ z \in Z, \ w \in W, \ s \in S\} \), where \( b \) represents last operation type performed on the machine. The two tokens in place \( \text{be}_{\text{time-critical}} \) with the color \( xzws(x \in X, \ z \in Z, \ w \in W, \ s \in S) \) and in place \( \text{workstation}_{j} \) with color \( b \in W \) (the color of last operation perform on machine) will enable the transition \( \text{enter}_{2,j} \) with color \( bxzws \). After the firing of this transition, a token will be assigned into the place \( \text{setup}_{j} \) with the color \( bxzws \).

The time associated with the transition \( \text{time-critical} \) 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 \( \text{enter}_{j} \) will not be fired if there exist tokens in place \( \text{be}_{\text{time-critical}} \).

The transition \( \text{time-critical} \) will then be fired if the lot in place \( \text{buffer}_{j} \) 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.

\[ \text{enter}_{j} \]

The place \( \text{enter}_{j} \) is an immediate transition. Firing this transition represents a normal (not time-critical operation) lot entering the operation subnet. The two tokens in place \( \text{buffer}_{j} \) with the color \( xzws(x \in X, \ z \in Z, \ w \in W, \ s \in S) \) and in place \( \text{workstation}_{j} \) with color \( b \in W \) (the color of last operation) will enable the transition...
enter_j with color \textcolor{blue}{bezus}. After the firing of the transition, a token will be assigned into the place setup_j with the color \textcolor{blue}{bezus}.

\textbf{in1_j (in2_j)}: \textcolor{blue}{in1_j} (in2_j) \ (j = 1, 2, \ldots, n) is the input arc of the transition enter_j (enter2_j). The place \textcolor{blue}{buffer_j} (be_time_critical_j) with the color \textcolor{blue}{xuzus} \ (x \in X, z \in Z, w \in W, s \in S) has arcs directly connected to the transition with the color \textcolor{blue}{bezus} (b \in W).

\textbf{in3_j (in4_j)}: \textcolor{blue}{in3_j} (in4_j) \ (j = 1, 2, \ldots, n) is the input arc of the transition enter_j (enter2_j). The place \textcolor{blue}{workstation_j} with the color \textcolor{blue}{w_j} \in W has arcs directly connected to the transition enter_j (enter2_j) with the color \textcolor{blue}{bezus} (b = w_j, x \in X, z \in Z, w \in W, s \in S).

\textbf{setup_j}: The place setup_j \ (j = 1, 2, \ldots, n) is a place. The token in \textcolor{blue}{setup_j} represents the machine is being set.

\textbf{setup_t_j}: The transition setup_t_j \ (j = 1, 2, \ldots, n) is a deterministically timed transition. The function of \textcolor{blue}{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 \textcolor{blue}{j}th 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 \textcolor{blue}{bezus} \ (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, \text{time} \geq 0). After the corresponding machine completes setup, the corresponding lot can be processed.

\textbf{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 \textcolor{blue}{bezus} \ (b \in W, x \in X, z \in Z, w \in W, s \in S) has an arc directly connected to the place \textcolor{blue}{processing_j} with the color \textcolor{blue}{xuzus}.

\textbf{out2 (out3_j, out4_j)}: out2_j (out3_j, out4_j) \ (j = 1, 2, \ldots, n) is the output arc of the transition \textcolor{blue}{failure2_j (processing_ok_j, processing_fail_j)}. The transition with the color \textcolor{blue}{xuzus} \ (x \in X, z \in Z, w \in W, s \in S) has an arc directly connected to the place \textcolor{blue}{repair_j (workstation_j)} with the color \textcolor{blue}{w}. The meaning of firing these transitions is that the last operation performed on this machine is memorized in place \textcolor{blue}{repair_j (workstation_j)}.

\textbf{processing_j}: The place \textcolor{blue}{processing_j} \ (j = 1, 2, \ldots, n) is a place. A token in \textcolor{blue}{processing_j} represents that the corresponding lot is processed by the machine in the \textcolor{blue}{j}th workstation. A rework probability is set in this place. If the \textcolor{blue}{j}th workstation contains an inspection device, the rework probability is greater than zero; otherwise, the rework probability is equal to zero. The token in \textcolor{blue}{processing_j} can enable the transition \textcolor{blue}{processing_ok_j, process_fail_j, and failure2_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 \textcolor{blue}{failure2} is not fired, the token fires the transition \textcolor{blue}{process_fail_j} after the associated processing time; otherwise, it fires \textcolor{blue}{process_ok_j}.

\textbf{process_fail_j}: The transition \textcolor{blue}{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 \textcolor{blue}{process_fail_j} is fired, two tokens are generated. One of the generated tokens is in the place \textcolor{blue}{workstation_j}, which implies that the current lot released the 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.

\textbf{process_ok_j}: The transition \textcolor{blue}{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 \textcolor{blue}{process_ok_j} represents that the lot just finished the current operation and released the control of the corresponding machine in the \textcolor{blue}{j}th workstation.

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

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

\textbf{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 \textcolor{blue}{repair_ok_j} depends on the mean time to repair (MTTR), and the time delay used in \textcolor{blue}{failure1_j} and \textcolor{blue}{failure2_j} depends on the mean time to fail (MTTF). The firing of the transition \textcolor{blue}{failure2_j} represents machine failures during an operation.

The place \textcolor{blue}{workstation_j} is initially marked. We model the machine failure probability with the transition \textcolor{blue}{failure1_j} and \textcolor{blue}{failure2_j}. If the lot is in processing, the transition \textcolor{blue}{failure2_j} can be fired. If the machine is in idle state, the transition \textcolor{blue}{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
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 $C(t) = C(y) = \{b_{x_1z_1w_1s_1x_2z_2w_2s_2} | b \in W, x_1, x_2 \in X, z_1, z_2 \in Z, w_1, w_2 \in W, s_1, s_2 \in S\}$; If the batch size is three, the color set is $C(t) = C(y) = \{b_{x_1z_1w_1s_1x_2z_2w_2s_2x_3z_3w_3s_3} | b \in W, x_1, x_2, x_3 \in X, z_1, z_2, z_3 \in Z, w_1, w_2, w_3 \in W, s_1, s_2, s_3 \in S\}$, 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 $xzw(x \in X, z \in Z, w \in W, s \in S), x_1zw_1(z_1 \in Z, s_1 \in S)$ 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 $b_{xzw}x_1z_1w_1s_1$ or $b_{xzw}x_1z_1w_1s_2$. After the firing of the transition, a token will be assigned into the place setup_j with the color $b_{xzw}x_1z_1w_1s_1$ or $b_{xzw}x_1z_1w_1s_2$. 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_j) with the color $xzw(x \in X, z \in Z, w \in W, s \in S)$ has arcs directly connected to the transition with the color $b_{xzw}x_1z_1w_1s_1(b \in W, x_1 \in X, z_1 \in Z, w_1 \in W, s_1 \in S)$ and $b_{xzw}x_2z_2w_2s_2xzw(b \in W, x_2 \in X, z_2 \in Z, w_2 \in W, s_2 \in S)$.

**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 $w_j \in W$ has arcs directly connected to the transition with the color $b_{xzw}x_1z_1w_1s_1(b = w_j, x \in X, z \in Z, w \in W, s \in S, x_1 \in X, z_1 \in Z, s_1 \in S)$.

**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
to compare the color of the last operation and current operation, i.e., to check whether the corresponding machine in the $j$th 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 $\text{setup}(\text{out}_j)$, the associated time is zero, and vice versa ($\text{out}_j = 1$, time $= 0$). After the corresponding machine completes setup, the corresponding batch can be processed.

- **out\_j**: $\text{out}_j$ ($j = 1, 2, \ldots, n$) is the output arc of the transition $\text{setup}_j$. The output function can be expressed as an identity matrix in the batch-processing machine $\forall c \in C(t), \{p(t), t(c, c) = O(p(t), t(c, c)) = 1 \text{ and otherwise } 0\}$.

- **out2\_j (out3\_j, out4\_j)**: $\text{out}_2 \text{ (out}_3 \text{, out}_4 \text{)}$ ($j = 1, 2, \ldots, n$) is the output arc of the transition $\text{failure2}_j$ ($\text{processing}_\text{ok}_j$). The transition with the color $\text{lw}_2(x_1, x_2)\text{setup}_2(\text{out}_2 \text{, out}_3 \text{, out}_4 \text{)} (b \in W, x_1, x_2 \in X, z_1, z_2 \in Z, w \in W, s_1, s_2 \in S)$ has an arc directly connected to the place $\text{repair}_j(\text{workstation}_j)$ with the color $w$. The meaning of firing these transitions is that the last operation performed on this machine is memorized in place $\text{workstation}_j$.
has two (depending on the batch size) arcs directly connected to the place dispatch (out) with the color $x_1 x_2 y_1 s_1$ and $x_2 x_2 y_2 s_2$. 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 to 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.

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

- 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.

- $N_g$: Number of Generations.
- $R_c$: Crossover Rate.
- $N_o$: Number of Offsprings. $N_o = N_p \times R_c$
- $P_m$: 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
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 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 \( R_c \) times, where \( R_c \) 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.
The dispatching rules and set-up rules are associated with the transition \( \text{enter}_j \) and \( \text{enter2}_j \). Each time when there are more than one token in place \( \text{buffer}_j \) (or \( \text{be_time_critical}_j \)) 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...
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;
- \( f_i(c) \) \( i \)-th 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 each chromosome, sort all the chromosomes by mean cycle time, and give each chromosome a rank such that the one with shorter mean cycle...
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

\[
f_1(c_1) = 2, \quad f_2(c_2) = 4, \quad f_3(c_3) = 1, \quad f_4(c_4) = 3.
\]

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 $w_{jk}$ 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 five chromosomes $c_1$, $c_2$, $c_3$, $c_4$, and $c_5$, and each chromosome leads to the following values:

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

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

\[
f(c_1) = 3.5; \quad f(c_2) = 2.5; \quad f(c_3) = 3.5; \quad f(c_4) = 3; \quad f(c_5) = 2.5.
\]

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...
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 sequence 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 \( c_j \in C \);
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 \( T_e(m) \);
```

TABLE V

<table>
<thead>
<tr>
<th>Order ID</th>
<th>Route</th>
<th>Quantity (lots)</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order 01</td>
<td>Route 1</td>
<td>30</td>
<td>1999/12/18 08:00</td>
</tr>
<tr>
<td>Order 02</td>
<td>Route 2</td>
<td>20</td>
<td>2000/01/03 08:00</td>
</tr>
<tr>
<td>Order 03</td>
<td>Route 3</td>
<td>50</td>
<td>2000/01/13 08:00</td>
</tr>
</tbody>
</table>

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

TABLE VI

<table>
<thead>
<tr>
<th>Gene/Rule</th>
<th>Dispatch rule</th>
<th>Setup rule</th>
<th>Batch rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene 01</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Gene 02</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Gene 03</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Gene 04</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Gene 05</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Gene 06</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Gene 07</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Gene 08</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Gene 09</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Gene 10</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Gene 11</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Gene 12</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Gene 13</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Gene 14</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Gene 15</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Gene 16</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Gene 17</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Gene 18</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Gene 19</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Gene 20</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Gene 21</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Gene 22</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Gene 23</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Gene 24</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

TABLE VII

<table>
<thead>
<tr>
<th>Item</th>
<th>MQT (hours)</th>
<th>MDD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std-Dev</td>
</tr>
<tr>
<td>FCFS</td>
<td>468.6</td>
<td>7.8</td>
</tr>
<tr>
<td>FCFS+SU</td>
<td>627.2</td>
<td>66.8</td>
</tr>
<tr>
<td>SRPT</td>
<td>472.8</td>
<td>15.9</td>
</tr>
<tr>
<td>SRPT+SU</td>
<td>551.2</td>
<td>31.2</td>
</tr>
<tr>
<td>MINS</td>
<td>462.0</td>
<td>11.9</td>
</tr>
<tr>
<td>MINS+SU</td>
<td>628.8</td>
<td>65.3</td>
</tr>
<tr>
<td>EDD</td>
<td>500.1</td>
<td>18.0</td>
</tr>
<tr>
<td>EDD+SU</td>
<td>628.1</td>
<td>68.2</td>
</tr>
<tr>
<td>GA</td>
<td>377.1</td>
<td>4.8</td>
</tr>
</tbody>
</table>

TABLE VIII

<table>
<thead>
<tr>
<th>Order ID</th>
<th>Route</th>
<th>Quantity (lots)</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order 01</td>
<td>Route 2</td>
<td>20</td>
<td>1999/12/12 08:00</td>
</tr>
<tr>
<td>Order 02</td>
<td>Route 3</td>
<td>20</td>
<td>1999/12/17 08:00</td>
</tr>
<tr>
<td>Order 03</td>
<td>Route 1</td>
<td>10</td>
<td>1999/12/24 08:00</td>
</tr>
<tr>
<td>Order 04</td>
<td>Route 3</td>
<td>30</td>
<td>2000/01/04 08:00</td>
</tr>
</tbody>
</table>

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

TABLE IX

<table>
<thead>
<tr>
<th>Item</th>
<th>MQT (hours)</th>
<th>MDD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std-Dev</td>
</tr>
<tr>
<td>FCFS</td>
<td>443.3</td>
<td>5.8</td>
</tr>
<tr>
<td>SRPT</td>
<td>490.8</td>
<td>57.0</td>
</tr>
<tr>
<td>MINS</td>
<td>439.2</td>
<td>8.6</td>
</tr>
<tr>
<td>EDD</td>
<td>474.3</td>
<td>11.1</td>
</tr>
<tr>
<td>GA</td>
<td>372.7</td>
<td>5.1</td>
</tr>
</tbody>
</table>
While \( m \) is not the final marking Do
  If \( T_e(m) \equiv \text{NULL} \) Then
    Update the elapsing time;
    Set the marking \( m \) as the current marking \( m_i \);
    Update \( T_e(m) \);
  Else
    For all \( t \in T_e(m) \) Do
      Fire the transition \( t \) by according to 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 \( m_i \);
      Update \( T_e(m) \);
    End For
  End If
End While

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
distributions with given mean values in implementation. In TRC fab model, the $\text{MPT}$ contains the mean processing time and mean set-up time, but we separate them in our simulation model. The $\text{MPT}$ in our simulation model is equal to $0.9 \times \text{MPT}$ in TRC fab model, and the $\text{MST}$ is equal to $0.1 \times \text{MPT}$ 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 $0.9 \times \text{MPT}$ and $1.1 \times \text{MPT}$, 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 $0.9 \times \text{MST}$ and $1.1 \times \text{MST}$, 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 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.
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. 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 due date, 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


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.

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 Tjeng 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.