applied sciences

MDPI

Article

Scheduling of Single-Arm Cluster Tools with Residency Time Constraints and Chamber Cleaning Operations

Jie Li [1], Yan Qiao [1,*], Siwei Zhang [1], Zhiwu Li [1], Naiqi Wu [1,2] and Tairan Song [1]

[1] Institute of Systems Engineering and Collaborative Laboratory for Intelligent Science and Systems, Macau University of Science and Technology, Macao 999078, China; li.jie@ikasinfo.com (J.L.); swzhang@must.edu.mo (S.Z.); zwli@must.edu.mo (Z.L.); nqwu@must.edu.mo (N.W.); song.tairan@ikasinfo.com (T.S.)

[2] State key Laboratory of Precise Electronic Manufacturing Technology and Equipment, Guangdong University of Technology, Guangzhou 510006, China

* Correspondence: yqiao@must.edu.mo

Abstract: To ensure wafer quality, engineers have to impose wafer residency time constraints and chamber cleaning operations on cluster tools; this has been widely used in semiconductor manufacturing. Wafer residency time constraints and chamber cleaning operations make the scheduling problem of cluster tools more challenging. This work aims to solve such a scheduling problem for single-arm cluster tools and presents a novel method based on the use of virtual wafers. Under a one-cyclic schedule obtained for single-arm cluster tools without chamber cleaning requirements, virtual wafers are loaded into the tool such that when a process module (PM) processes virtual wafers, a chamber cleaning operation is performed in practice. The key to solve this scheduling problem is to find a wafer loading sequence with the highest performance in terms of cycle time. With this idea, this work constructs a genetic algorithm to search for such a solution. Since the obtained solution is a periodical wafer loading sequence based on a one-wafer cyclic schedule, it can be easily implemented. Therefore, this work has high practical value to numerous semiconductor manufacturers. Experiments were performed to show the efficiency and effectiveness of the proposed method.

Keywords: chamber cleaning; cluster tools; scheduling; semiconductor manufacturing

Citation: Li, J.; Qiao, Y.; Zhang, S.; Li, Z.; Wu, N.; Song, T. Scheduling of Single-Arm Cluster Tools with Residency Time Constraints and Chamber Cleaning Operations. *Appl. Sci.* **2021**, *11*, 9193. https://doi.org/10.3390/app11199193

Academic Editor: Emanuele Carpanzano

Received: 14 August 2021
Accepted: 27 September 2021
Published: 2 October 2021

1. Introduction

In semiconductor manufacturing, cluster tools are important for wafer fabrication. A modern semiconductor fabrication plant (fab) is equipped with hundreds of cluster tools. A cluster tool is very expensive, and the cost for equipment accounts for more than a half of the total investment for a fab. Therefore, maximizing the productivity of cluster tools is an important means for manufacturers to ensure the return on their investment. A cluster tool consists of a wafer handling robot, an aligner (AL), several process modules (PMs), and two loadlocks (LLs) for raw wafer lot loading/unloading. Figure 1a,b shows two kinds of cluster tools equipped with a one-arm or two-arm robot, called a single-arm cluster tool (SACT) and dual-arm cluster tool (DACT), respectively. The single-wafer processing technology is adopted for processing large-size wafers in cluster tools. By such a technology, a PM in a tool can process only one wafer at a time. In this way, precision processing requirements can be satisfied such that the quality of processed wafers can be guaranteed.

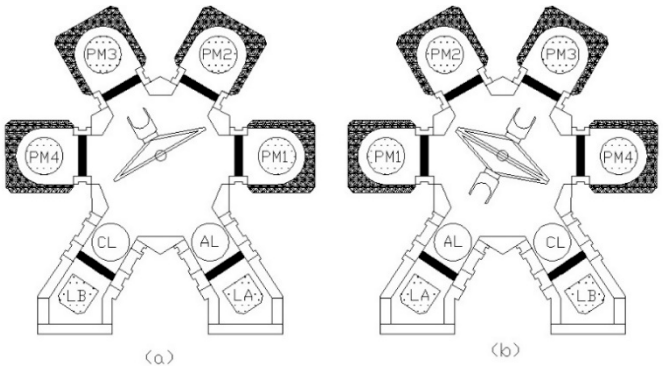

Figure 1. Cluster tools: (**a**) single-arm robot and (**b**) dual-arm robot.

For a cluster tool, the robot moving time between two PMs or a PM and an LL is much shorter than the wafer processing time at a PM in practice. Thus, the time taken for completing a wafer at the bottleneck step determines the system cycle time. With such a property, a tool is said to be process-bound. A backward strategy is optimal for process-bound SACTs [1], while a swap strategy is efficient for process-bound DACTs [2].

With the increasing of wafer diameter and the shrinkage of circuit width, in order to guarantee wafer quality, strict wafer residency time constraints (WRTCs) are imposed on many wafer fabrication processes. They require that a wafer should be removed out of a PM within a given time window after it is processed, otherwise the wafer may be damaged by the residual chemicals, particles, and high temperature in the PM. Low pressure chemical-vapor deposition is such a typical process. Without intermediate buffers between PMs, WRTCs definitely make the scheduling problem of cluster tools complicated.

Important efforts have been made in scheduling cluster tools with WRTCs by reducing the wafer delay time in PMs, i.e., the time a wafer is detained in a PM after it is processed. Rostami et al. [3] and Lee and Park [4] present methods for DACTs with WRTCs to find an optimal and feasible periodic schedule. To reduce the wafer delay time for DACTs, a modified swap strategy is proposed in [5] such that a wafer waits on the robot for some time during a swap operation. For SACTs, the wafer delay time at a step can be reduced by delaying the time point of unloading a wafer at the previous step, so that the time point of loading the wafer at the step is delayed [1,6]. Feedback control methods are proposed in [7,8] to control wafer delays such that a feasible schedule can be found for both SACTs and DACTs. If a conventional swap strategy cannot achieve a feasible schedule for DACTs with WRCTs, a new class of robot task sequences is proposed in [9] such that WRCTs can be met. Roh et al. [10] developed closed-form formulas for most frequently used wafer flow patterns and an optimization model that computes the worst-case wafer delay of dual-armed cluster tools. They also analyze the factors that affect the worst-case wafer delay and their influences by experiments. By considering processing time variation, an adaptive scheduling method is presented for cluster tools with wafer delay constraints in [11]. With WRTCs and multiple wafer types being processed, efficient scheduling approaches are proposed for SACTs in [12] and DACTs in [13,14].

For process-bound cluster tools, robot tasks take much less time than processing a wafer in a PM such that the system cycle time is determined by the time taken for completing a wafer at the bottleneck step. Hence, in a cycle, the robot has idle time that can be treated as robot waiting time. By regulating robot waiting time at the process steps, schedulability conditions are established for both SACTs and DACTs with WRTCs in [15,16]. According to these conditions, if a tool is schedulable, a feasible and optimal schedule can be found by parameterizing robot waiting time. Then, a robot idle time regulating method can be extended to find a feasible schedule for both SACTs and DACTs with WRTCs and bounded activity time variations as in [17–21].

In practice, after a PM completes a number of wafers, the processing environment in the chamber at the PM is contaminated to some extent. Hence, for high quality wafer

fabrication, it should be cleaned at this time. Chamber cleaning is necessary for wafer quality, but this makes the scheduling problem more challenging. Yu et al. [22,23] present an interesting idea of partially loading wafers into parallel PMs at a step for SACTs and DACTs with chamber cleaning requirements. By doing so, parallel PMs can be alternately cleaned. Their studies were conducted based on the assumption that a chamber cleaning operation is performed for a PM after every wafer is processed in the PM, which is also called a purge operation. However, it is also important to consider a more general case in which a chamber in a PM is required to be cleaned after a specified quantity of wafers is consecutively processed, which arises from practical wafer fabrication cases. For such a case with WRTCs being considered at the same time, the existing method in [22,23] is no longer applicable. Thus, this work aims to solve such a new and challenging problem for SACTs with WRTCs and chamber cleaning operations by proposing a virtual wafer strategy.

The rest of the paper is organized as follows. The virtual wafer-based method is introduced in Section 2. Then, based on the virtual wafer method, a genetic algorithm is constructed to find an approximate solution in Section 4. Finally, extensive experiments are described to verify the proposed method.

2. Virtual Wafer-Based Method

2.1. Robot Tasks

Assume that there are n Steps in an SACT. Each Step may have parallel PMs that are configured to perform the same operation. Raw wafers in cassettes are loaded into the tool through LLs one by one. Then, each wafer in a cassette should sequentially visit Steps 1–n for being processed. After all operations are completed, they are returned to the cassette where they come from. Let $\mathrm{N_n} = \{0, 1, 2, \ldots, n\}$ and $\mathrm{N^+_n} = \mathrm{N_n} \backslash \{0\}$. The wafer flow pattern in the tool can be defined as $(n_1, n_2, \ldots, n_n)$, where n_i denotes the number of parallel PMs at Step i, $i \in \mathrm{N^+_n}$.

The wafer sojourn time in a PM at a step is defined as the time duration that starts from the time point when the robot completes loading a wafer into the PM and ends at the time point when the robot starts unloading the wafer from the PM. The wafer sojourn time consists of the wafer processing time and the time for staying there after being processed, named the wafer delay time. Let τ_i and ρ_i, $i \in \mathrm{N^+_n}$, denote the wafer sojourn time and the processing time in a PM at Step i, respectively. WRTCs require that the wafer delay time should be within a permitted time window. Let δ_i, $i \in \mathrm{N^+_n}$, denote the upper bound of the wafer delay time in a PM at Step i. Thus, it is important to schedule the robot tasks such that WRTCs are met for each step, i.e., $0 \leq \tau_i - \rho_i, \leq \delta_i$, $i \in \mathrm{N^+_n}$.

The robot tasks include wafer unloading and loading, moving, and waiting. In this work, U_i and L_i are used to denote the robot tasks of unloading and loading a wafer from and into a PM at Step i, $i \in \mathrm{N_n}$, respectively. Note that we use Step 0 to represent LLs. M_{ij} is used to denote the robot task of moving from a PM/LL at Step i to a PM/LL at Step j. W_i, $i \in \mathrm{N_n}$, is used to denote the robot waiting before unloading a wafer from a PM at Step i. Robot tasks take time, and the time needed for robot unloading/loading at all steps is same. Then, α and β are used to denote the time needed for U_i and L_i, $i \in \mathrm{N_n}$, respectively. The robot moving time between any two steps can be treated as the same, and μ is used to denote the time needed for M_{ij} with or without carrying a wafer. ω_i is used to denote the time needed for W_i, $i \in \mathrm{N_n}$.

For process-bound SACTs, the backward strategy is proved to be optimal in terms of cycle time. Thus, in this work, it is supposed that the backward strategy is adopted for operating an SACT. Assume that in each PM in an SACT there is a wafer being processed. Then, by the backward strategy, a robot task sequence named σ is performed as follows: $U_n \rightarrow M_{n0} \rightarrow L_0 \rightarrow M_{0(n-1)} \rightarrow U_{n-1} \rightarrow M_{(n-1)n} \rightarrow L_n \rightarrow M_{n(n-2)} \rightarrow \ldots \ldots \rightarrow U_1 \rightarrow M_{12} \rightarrow L_2 \rightarrow M_{20} \rightarrow U_0 \rightarrow M_{01} \rightarrow L_1 \rightarrow M_{1n}$. In fact, σ forms a robot task cycle and it is repeatedly performed in the following operations.

2.2. One-Wafer Cyclic Schedule

By the backward strategy, after a wafer is loaded into a PM at Step i by the robot, the wafer stays in the PM for being processed. When the robot comes to the PM again, it unloads the wafer and moves to a PM at Step $i + 1$ for loading the wafer. Then, the robot moves to a PM at Step $i-1$ for unloading a wafer. At this time, it may need to wait there since the wafer may be not processed. After the wafer is processed, the robot unloads the wafer and moves to the empty PM at Step i. When it arrives at the PM, it loads a wafer into the PM again. This implies that a wafer is completed at the PM at Step i. Thus, the time taken for completing a wafer at the PM is $\tau_i + 2\alpha + 2\beta + 3\mu + \omega_{i-1}$. Since there are n_i parallel PMs serving for Step i, $i \in \mathrm{N}^+_{\mathrm{n}}$, the time taken for completing a wafer at Step i is

$$\theta_i = (\tau_i + 2\alpha + 2\beta + 3\mu + \omega_{i-1})/n_i, \; i \in \mathrm{N}^+_{\mathrm{n}} \tag{1}$$

With WRTCs, $\rho_i \leq \tau_i \leq \rho_i + \delta_i$, $i \in \mathrm{N}^+_{\mathrm{n}}$, should hold. By the backward strategy, the robot task sequence σ is repeatedly performed, implying that the robot cycle time is the time taken for performing σ. Let ψ denote the robot cycle time. Then, we have

$$\psi = (n+1)(\alpha + \beta) + 2(n+1)\mu + \sum_{d=0}^{n} \omega_d = \psi_1 + \psi_2 \tag{2}$$

In Equation (2), $\psi_1 = (n + 1)(\alpha + \beta) + 2(n + 1)\mu$ is known in advance and $\psi_2 = \sum_{d=0}^{n} \omega_d$ is determined by a schedule. This work focuses on the scheduling problem of process-bound SACTs with WRTCs and chamber cleaning requirements. For a process-bound SACT, the robot is not busy all the time, and has time to wait at some PMs. Thus, the key to schedule an SACT is to regulate the robot waiting time in a cycle such that a one-wafer cyclic schedule is obtained.

Definition 1. *An SACT is operated under a one-wafer cyclic schedule if the robot waiting times ω_i, $i \in N_n$, in each cycle keep unchanged.*

In Definition 1, if robot waiting time ω_i, $i \in \mathrm{N}_{\mathrm{n}}$, and each cycle is unchanged, the robot cycle time ψ must be fixed. Further, by Equations (1) and (2), if an SACT is operated by a one-wafer cyclic schedule, we have

$$\theta_i = \psi, \; i \in \mathrm{N}^+_{\mathrm{n}} \tag{3}$$

Wu et al. [15] present an efficient method to obtain a feasible and optimal one-wafer cyclic schedule by properly regulating the robot waiting time. Readers can refer to [15] for more details. Based on a one-wafer cyclic schedule, this work introduces a virtual wafer-based method to deal with the chamber cleaning requirements next.

2.3. Approach to Deal with Chamber Cleaning Requirements

In order to ensure wafer quality, a chamber in a PM needs to be cleaned after a specified quantity of wafers are consecutively processed. In this way, residual chemicals and air particles can be removed from the chamber. Let m_i, $i \in \mathrm{N}^+_{\mathrm{n}}$, denote the maximal number of wafers that a chamber in a PM at Step i consecutively process before the PM is required to clean, and C_i, $i \in \mathrm{N}^+_{\mathrm{n}}$, the required that. Note that $m_i \geq 1$ and C_i, $i \in \mathrm{N}^+_{\mathrm{n}}$, are known in advance and can be obtained by statistical experiments performed by process engineers and data engineers.

With chamber cleaning requirements, once the robot completes unloading a wafer from a PM at Step i, the PM can start to clean its chamber. Then, by the backward strategy, the robot performs a sequence of robot tasks (i.e., $M_{i(i+1)} \to L_{i+1} \to M_{(i+1)(i-1)} \to W_{i-1} \to U_{i-1} \to M_{(i-1)i}$) such that it moves to the PM again for being ready to load a wafer. Note that the robot waiting time taken by W_{i-1} is determined by a one-wafer cyclic schedule in advance. If the time needed for a chamber cleaning is too long, leading to $max(\alpha + \beta + 3\mu + \omega_{i-1}, C_i) = C_i$, it would postpone the starting time of the subsequent robot tasks. In other

words, the clamber cleaning operation at Step i means that the robot cannot arrive at a PM at Step j, $1 \leq j \leq i-2$, for unloading a processed wafer in time, leading to a long wafer delay time, violating WRTCs. Such a long wafer delay time may cause a quality problem to the wafer. Therefore, it is very important to develop a precise scheduling and control method for an SACT with WRTCs and chamber cleaning operations. This work aims to do so.

In fact, by ignoring chamber cleaning requirements, a one-wafer cyclic schedule can be found by properly regulating the robot waiting time [15]. With chamber cleaning requirements, virtual wafers are introduced into SACTs based on a one-wafer cyclic schedule such that the chambers in the tools can be cleaned in time and WRTCs are not violated. Specifically, by a one-wafer cyclic schedule, sometimes when the robot comes to LLs for unloading a raw wafer, it unloads a virtual wafer. Thus, after a real wafer is processed in a PM, if the next wafer to be processed at the PM is a virtual one, it implies that the chamber in the PM can be cleaned. Thus, when the robot task of unloading the real wafer from the PM is performed, it triggers a cleaning task for the chamber in the PM. When a virtual is loaded into the PM, its processing is not performed since it is a virtual one. Instead, the chamber performs the cleaning operation. Then, by the backward strategy and the one-wafer cyclic schedule, the robot can move to any step according to the schedule without any delay. In this way, enough time for chamber cleaning at the PM can be ensured and, at the same time, the starting time of any robot task in a cycle is not postponed such that the WRTCs can be met.

However, if the required chamber cleaning time is too long, it may need to consecutively load more virtual wafers to provide enough time for chamber cleaning. In order to determine how many virtual wafers should be consecutively loaded into a PM it is necessary to know how long can be scheduled for the chamber cleaning operation if a specified number of virtual wafers are consecutively loaded into a PM. Let ${O^d}_i$ denote the scheduled time for the chamber cleaning operation of a PM at Step i, $i \in \mathrm{N}^+_{\mathrm{n}}$, if there are d virtual wafers being consecutively loaded into the PM.

Assume that there is no virtual wafer being processed in a PM at Step i, $i \in \mathrm{N}^+_{\mathrm{n}}$. For such a case, with chamber cleaning requirements, once the robot completes unloading a wafer from a PM at Step i, the PM can start to clean its chamber. Then, by the backward strategy, the robot performs a sequence of robot tasks (i.e., $M_{i(i+1)} \to L_{i+1} \to M_{(i+1)(i-1)} \to W_{i-1} \to U_{i-1} \to M_{(i-1)i}$) such that it moves to the PM again for loading a wafer. When the robot starts to perform a loading task, the chamber cleaning operation ends. Thus, for this case, the time can be scheduled for cleaning the chamber in the PM at Step i is ${O^0}_i = \alpha + \beta + 3\mu + \omega_{i-1}$.

Assume that after a real wafer is unloaded from a PM at Step i, there is a single virtual wafer being loaded into the PM only to deal with chamber cleaning before another real wafer is loaded into the PM for processing. For this case, when a real wafer is unloaded from the PM, the chamber in the PM starts to clean its internal environment. Then, the robot sequentially performs the following robot task sequence: $M_{i(i+1)} \to L_{i+1} \to M_{(i+1)(i-1)} \to W_{i-1} \to U_{i-1} \to M_{(i-1)i} \to L_i$ such that a virtual wafer is loaded into the PM. Note that the time for the virtual wafer staying in the PM is τ_i, which is determined by the one-wafer cyclic schedule. By the backward strategy, when the robot comes to the PM again, the robot performs the following robot tasks: $U_i \to M_{i(i+1)} \to L_{i+1} \to M_{(i+1)(i-1)} \to W_{i-1} \to U_{i-1} \to M_{(i-1)i}$ such that the robot moves to the PM again carrying a real wafer. At this time, the cleaning operation ends, and the real wafer is loaded into the PM by performing L_i. In this case, the time that can be scheduled for the chamber cleaning at a PM at Step i equals the time needed for the following consecutive activities: $M_{i(i+1)} \to L_{i+1} \to M_{(i+1)(i-1)} \to W_{i-1} \to U_{i-1} \to M_{(i-1)i} \to L_i \to$ the virtual wafer staying in the PM $\to U_i \to M_{i(i+1)} \to L_{i+1} \to M_{(i+1)(i-1)} \to W_{i-1} \to U_{(i-1)} \to M_{(i-1)i}$. Therefore, with Equation (1), we have ${O^1}_i = \tau_i + 3\alpha + 3\beta + 6\mu + 2\omega_{i-1} = n_i \times \theta_i + \alpha + \beta + 3\mu + \omega_{i-1}$. Similarly, if there are d virtual

wafers being consecutively loaded into a PM at Step i, $i \in N^+_n$, the scheduled time for the chamber cleaning in the PM is

$$O^d{}_i = d \times n_i \times \theta_i + \alpha + \beta + 3\mu + \omega_{i-1}, d \geq 0 \quad (4)$$

Expression (4) gives the time that can be scheduled for the chamber cleaning at a PM at Step i, $i \in N^+_n$, if there are d virtual wafers being consecutively loaded into the PM. Let d_i denote the minimal number of virtual wafers to be consecutively loaded into the PM such that there is enough time scheduled for chamber cleaning. Assume that the chamber cleaning time at parallel PMs at Step i is same. If the exact time required for the chamber cleaning at a PM at Step i, $i \in N^+_n$, is given, by Equation (4) we have

$$d_i = \arg\min_d (h = d \times n_i \times \theta_i + \alpha + \beta + 3\mu + \omega_{i-1} - C_i | h \geq 0),\ i \in N^+_n, \quad (5)$$

In Equation (5), θ_i and ω_{i-1} are determined by the one-wafer cyclic schedule. Now, the question is how to obtain a schedule by introducing virtual wafers such that chamber cleaning requirements are ensured while the throughput is maximized. In real semiconductor fabs, a periodic schedule for cluster tools is preferred since it is easy to be implemented and WRTCs are easily met in order to avoid the quality problem. It is necessary to note that, in this work, wafers (including both real and virtual wafers) are loaded into cluster tools based on the one-wafer cyclic schedule defined by Definition 1. In addition, there is a sequence of real and virtual wafers loaded into a cluster tool periodically. Thus, we have the following definition.

Definition 2. *A schedule is called a Periodical Virtual-wafer Schedule (PVS) if real and virtual wafers are loaded into a cluster tool in a periodically repeated sequence based on a one-wafer cyclic schedule.*

This work aims to find a feasible and optimal PVS, i.e., a PVS that can achieve the maximal throughput while meeting chamber cleaning requirements.

3. Approximation Solution Algorithm

Let $x_j \in \{0, 1\}$ represent the j-th wafer (also called Wafer-j) loaded into a cluster tool in a cylce. If $x_j = 1$, it represents a virtual wafer, otherwise it is a real one. Let $\pi = (x'_1, x'_2, \ldots, x'_q)$ denote a PVS for a cluster tool, $x'_j \in \{0, 1\}$. Thus, $\pi = (x'_1, x'_2, \ldots, x'_q)$ indicates a repeated sequence of real and virtual wafers loaded into the tool, where q is the number of wafers in a cycle and x'_j represents a wafer at the j-th position in the wafer loading sequence, $j \in N^+_q$. Further, if $x'_j = 1$, it represents a virtual wafer, and otherwise it is a real one. $\pi = (x'_1, x'_2, \ldots, x'_q)$ is called the wafer loading sequence for a cluster tool under a PVS.

In a cluster tool, there may be several parallel PMs serving for a step. Since π is a repeated sequence of wafers loaded into a tool, there must exist a sequence of wafers being repeatedly processed by a PM at a Step that is different from π. Let k_i, $i \in N^+_n$, denote the number of wafers in a repeated sequence being processed by a PM at Step i of a cluster tool. Then, we have

$$k_i = \begin{cases} \frac{q}{n_i}, & if\ \frac{q}{n_i}\ is\ an\ integer \\ q, & otherwise \end{cases},\ i \in N^+_n \quad (6)$$

We explain the meaning of Expression (6) as follows. Assume that a cluster tool is operated under a PVS $\pi = (x'_1, x'_2, \ldots, x'_q)$ and q/n_i is an integer. After q wafers are sequentially loaded into Step i for being processed according to the PVS, this implies that each PM at Step i has just processed q/n_i wafers since there are n_i parallel PMs at the step. Note that q is the number of wafers in a repeated wafer loading sequence under the PVS. This means that the sequence of q/n_i wafers (including virtual and real ones) being processed at a PM forms a wafer processing sequence that is continuously repeated as the tool operates. Therefore, $k_i = q/n_i$ if q/n_i is an integer for this case.

Assume that an SACT is operated under a PVS $\pi = (x'_1, x'_2, \ldots, x'_q)$ and q/n_i is not an integer. For this case, after $q \times n_i$ wafers are sequentially loaded into Step i for being processed according to the PVS, each PM at Step i has just processed q wafers since there are n_i parallel PMs at the step. Since q is the number of wafers in a repeated wafer loading sequence under the PVS, the sequence of q wafers (including virtual and real ones) being processed at a PM forms a wafer processing sequence that is continuously repeated as the tool operates. Therefore, $k_i = q$ if q/n_i is not an integer for this case.

A solution π for the addressed problem can be treated as a bit-string. Thus, a genetic algorithm (GA) with a binary encoding is applied to find an optimal or near optimal solution for the addressed problem. GA is a meta-heuristic and inspired by Charles Darwin's theory of natural evolution. Many studies have shown that GA can achieve a promising solution for optimization problems [24–26].

3.1. Solution Encoding and Modification

GA relies on a population of individuals to explore a search space. Each individual is a set of chromosomes representing a candidate solution. In this work, a solution is denoted by π. Thus, each individual (i.e., a candidate solution) in GA is encoded as a string of 1 (if the corresponding position is a virtual wafer) and 0 (if the corresponding position is a real wafer).

For Step i, $i \in N^+_n$, if $d_i = 1$ obtained by Equation (5), it means that at least one virtual wafer is required to be loaded into a PM at the step to ensure the chamber cleaning requirement after m_i real wafers are continuously processed in the PM, at most. In order to check if the chamber cleaning requirement at Step i, $i \in N^+_n$, with $d_i = 1$ is met, it needs to know the type (i.e., virtual or real) of the j-th wafer processed in a PM at the step. Thus, Algorithm 1 is developed to update x_j, $j > 0$, based on $\pi = (x'_1, x'_2, \ldots, x'_q)$.

Algorithm 1. Updating x_j based on π for the case with $d_i = 1$

```
Input:  π;
Output: x_j, 1 ≤ j ≤ (m_i + k_i) × n_i;
(1)  For j = 1 to q
(2)      x_j = x'_j;
(3)  If k_i = q
(4)      For j = q + 1 to n_i × q
(5)          For g = 1 to q
(6)              If (j − g)/q is an integer
(7)                  x_j = x_g;
(8)  If m_i + 1 < k_i
(9)      For g = 1 to n_i
(10)         For h = 0 to m_i − 1
(11)             x_{g+h×n_i+k_i×n_i} = x_{g+h×n_i};
```

In Algorithm 1, Statements (1) and (2) determine the type of the j-th wafer loaded into Step i for being processed by setting $x_j = x'_j$, $1 \le j \le q$. Note that, if $k_i = q$, when each PM at Step i has just processed k_i wafers, this implies that $n_i \times k_i$ wafers have been loaded into Step i for being processed. Due to $\pi = (x'_1, x'_2, \ldots, x'_q)$ and $x_j = x'_j$, $1 \le j \le q$, Statements (3)–(7) obtain the type of the j-th wafer, $q + 1 \le j \le n_i \times k_i$. In this way, the types of the $n_i \times k_i$ wafers are determined.

If $m_i + 1 \ge k_i$, there is at least one virtual wafer in the k_i wafers being consecutively processed at each PM at Step i, $i \in N^+_n$. If there is no virtual wafer in the k_i wafers, the chamber cleaning requirement at the step cannot be met since k_i is the number of wafers in a repeated wafer processing sequence under a PVS. For the case with $m_i + 1 < k_i$, this requires that there is at least one virtual wafer among any $m_i + 1$ wafers being consecutively processed by each PM at Step i, $i \in N^+_n$. In this way, the chamber cleaning requirement at the step can be guaranteed. For this case, it is necessary to know the type of the j-th wafer, $1 + n_i \times k_i \le j \le (m_i + k_i) \times n_i$, for being processed at Step i as well.

Notice that after $n_i \times k_i$ wafers are processed at Step i, k_i wafers are processed by each PM at the Step. Further, the next k_i wafers (including virtual and real ones) to be processed by a PM has the same wafer loading sequence as the processed k_i wafers. Thus, Statements (8)–(11) obtain the type of the j-th wafer, $1 + n_i \times k_i \leq j \leq (m_i + k_i) \times n_i$, loaded into the step.

For Step i, $i \in N^+_n$, if $d_i = 2$ obtained by Equation (5), it means that at least two virtual wafers are required to be consecutively loaded into a PM at the step to ensure the chamber cleaning requirement after m_i real wafers are continuously processed in the PM, at most. Then, Algorithm 2 is developed to obtain the type of the j-th wafer processed at the step, i.e., update x_j, $j > 0$, based on π.

Algorithm 2: Updating x_j based on π for the case with $d_i = 2$

Input: π;
Output: x_j, $1 \leq j \leq (2k_i + 1) \times n_i$;
(1)–(7) Same as the ones in Algorithm 1, respectively
(8) For $g = 1$ to n_i
(9) For $h = 0$ to $k_i - 1$
(10) $x_{g+h\times n_i+k_i\times n_i} = x_{g+h\times n_i}$;
(11) $x_{g+2k_i\times n_i} = x_g$;

In Algorithm 2, Statements (1)–(7) are same as those in Algorithm 1, and are used for setting x_j, $1 \leq j \leq n_i \times k_i$.

Notice that, under a PVS, the wafer type sequence to be processed by a PM at Step i is periodically repeated, and the number of wafers in such a sequence is k_i obtained by Equation (6). Then, three consecutive wafer processing sequence are named as WPS-1, WPS-2, and WPS-3, respectively. Further, let x_j, $j = g + h\times n_i$, $g \in \{1, 2, \ldots, n_i\}$, $0 \leq h \leq k_i - 1$, denote a wafer in WPS-1; x_j, $j = g + h\times n_i$, $g \in \{1, 2, \ldots, n_i\}$, $k_i \leq h \leq 2k_i - 1$, a wafer in WPS-2; and x_j, $j = g + h\times n_i$, $g \in \{1, 2, \ldots, n_i\}$, $2k_i \leq h \leq 3k_i - 1$, and a wafer in WPS-3. There may exist a case when only two virtual wafers are consecutively processed by a PM in a processing sequence consisting of WPS-1 and WPS-2. In such a case, one virtual wafer is at the k_i-th position in WPS-1 (i.e., x_j, $j = g + h\times n_i$, $h = k_i - 1$), while the other virtual wafer is at the first position in WPS-2 (i.e., x_j, $j = g + h\times n_i$, $h = k_i$). These two consecutive wafers represent a chamber cleaning operation. In order to check if the chamber cleaning requirement can be met, how many real wafers are processed between this chamber cleaning operation and the next one needs to be known. In this case, two consecutive virtual wafers at the k_i-th position in WPS-2 (i.e., x_j, $j = g + h\times n_i$, $h = 2k_i - 1$) and the first position in WPS-3 (i.e., x_j, $j = g + h\times n_i$, $h = 2k_i$) represent the next one. Thus, if the number of real wafers being processed by the PM between these two chamber cleaning operations is no more than m_i, it means that the chamber cleaning requirement is met for the PM all the time, since the wafer loading sequence, being same as WPS-1, WPS-2, and WPS-3, is periodically repeated. Then, it is necessary to know the type of the j-th wafer, $n_i \times k_i + 1 \leq j \leq (2k_i + 1) \times n_i$, to check if the chamber cleaning requirement at the step can be met for this case. In fact, for other cases, if the type of the j-th wafer, $n_i \times k_i + 1 \leq j \leq (2k_i + 1) \times n_i$, is known, we can check if the chamber cleaning requirement at the step can be met under a PVS. Then, Statements (8)–(11) in Algorithm 2 are used to determine the type of the j-th wafer, $n_i \times k_i + 1 \leq j \leq (2k_i + 1) \times n_i$.

For Step i, $i \in N^+_n$, one of Algorithms 1 and 2 can be applied to update x_j, $j > 0$. For different steps, the ranges of index j of x_j obtained by an applied algorithm are different. Let $s = argmax_i\{\{k_i \times n_i \mid i \in N^+_n, d_i = 1, \text{and } m_i + 1 \geq k_i\}, \{(m_i + k_i) \times n_i \mid i \in N^+_n, d_i = 1, \text{and } m_i + 1 < k_i\}, \{(2k_i + 1) \times n_i \mid i \in N^+_n, d_i = 2\}\}$. Then, for Step s, the largest value j, the index of x_j obtained by an applied algorithm, is also the largest one among those for all steps. Thus, we just need to update x_j based on π for Step s by one of Algorithms 1 and 2.

Notice that an individual (solution) may result in the chamber cleaning requirement at a step not being satisfied. Thus, three algorithms are developed to modify the individual for different cases such that the chamber cleaning requirements at all steps are satisfied.

For Step i with $d_i = 1$ and $m_i + 1 \geq k_i$, $i \in N^+_n$, Algorithm 3 is developed to modify π such that there is at least one virtual wafer in the k_i wafers that are consecutively processed by each PM. In this way, the chamber cleaning requirement at the step can be met.

Algorithm 3: Individual modification for the case with $d_i = 1$ and $m_i + 1 \geq k_i$

```
(1)  For g = 1 to n_i
(2)      R = 0;
(3)      For h = 0 to k_i − 1
(4)          If x_{g+h×n_i} = 0
(5)              R = R + 1;
(6)          Else If x_{g+h×n_i} = 1
(7)              R = 0;
(8)          If R = k_i
(9)              j = random [0, k_i − 1];
(10)             x_{g+j×n_i} = 1;
(11)             For f = 1 to q
(12)                 If (g + j×n_i − f)/q is a non-negative integer
(13)                     x_f = 1;
(14)                     x_f' = x_f;
(15)                     Perform one of Algorithms 1 and 2 for Step s;
```

In the algorithm, R is used to record the number of real wafers to be consecutively processed by a PM. By performing Statements (4)–(7), the value of R is updated. When the value of R equals k_i, one real wafer of the k_i wafers is selected by Statement (9) and replaced by a virtual wafer by Statement (10) to meet the chamber cleaning requirement. Note that random$[0, k_i - 1]$ in Statement (9) is used to generate an integer randomly in the range $[0, k_i - 1]$. Furthermore, π is modified according to Statements (11)–(15). Then, based on the modified π by Statement (14), x_j should be updated by one of Algorithms 1 and 2 for Step s as well by Statement (15). Thus, by Algorithm 3, an individual is modified such that for Step i with $d_i = 1$ and $m_i + 1 \geq k_i$, $i \in N^+_n$, the chamber cleaning requirement is met.

For Step i with $d_i = 1$ and $m_i + 1 < k_i$, $i \in N^+_n$, Algorithm 4 is developed to modify π so there is at least one virtual wafer among any $m_i + 1$ wafers being consecutively processed by each parallel PM. In this way, the chamber cleaning requirement at the step can be met. In the algorithm, Statements (4)–(7) are the same as those in Algorithm 3. For the $(g + h \times n_i)$-th wafer, if it is a real one (i.e., $x_{g+h\times n_i} = 0$) and $R = m_i + 1$ obtained by performing Statements (4)–(7), the wafer is replaced by a virtual one and at the same time R is updated to zero by performing Statements (9) and (10). In this way, it prevents $m_i + 1$ wafers from being consecutively processed by a PM. Then,π is modified according to Statements (11)–(15). Based on the modified π, x_j is updated by one of Algorithms 1 and 2 for Step s as well by performing Statement (15). Thus, by Algorithm 4, an individual is modified such that for Step i with $d_i = 1$ and $m_i + 1 < k_i$, $i \in N^+_n$, the chamber cleaning requirement is met.

Algorithm 4: Individual modification for the case with $d_i = 1$ and $m_i + 1 < k_i$

```
(1)  For g = 1 to n_i
(2)      R = 0;
(3)      For h = 0 to k_i − 1 + m_i
(4)          If x_{g+h×n_i} = 0
(5)              R = R + 1;
(6)          Else If x_{g+h×n_i} = 1
(7)              R = 0;
(8)          If R = m_i + 1
(9)              x_{g+h×n_i} = 1;
(10)             R = 0;
(11)             For f = 1 to q
(12)                 If (g + h×n_i − f)/q is a non-negative integer
(13)                     x_f = 1;
(14)                     x_f' = x_f;
(15)                     Perform one of Algorithms 1 and 2 for Step s;
```

For Step i with $d_i = 2$, Algorithm 5 is developed to modify π so that at least two virtual wafers are consecutively loaded into a PM for being processed after no more than m_i real wafers are continuously processed by the PM. In this way, the chamber cleaning requirement can be met.

Note that in the algorithm symbol V is used to record the number of virtual wafers to be consecutively processed by a PM. Notice that there must exist at least two virtual wafers to be consecutively processed by a PM during two adjacent wafer loading sequences to meet the chamber cleaning requirement for the PM. Thus, V_0 is used to record if there exists two such virtual wafers being consecutively processed. By performing the algorithm, the type of wafers processed by each PM at the step is checked. Then, there are two cases: (1) a wafer represented by $x_{g+h\times n_i}$ is a real one, i.e., $x_{g+h\times n_i} = 0$; and (2) a wafer represented by $x_{g+h\times n_i}$ is a virtual one, i.e., $x_{g+h\times n_i} = 1$.

In Case (1), R and V are updated by Statements (7) and (8), respectively. At this time, if the number of real wafers to be consecutively processed by a PM is greater than m_i (i.e., $R > m_i$), the wafer represented by $x_{g+h\times n_i}$ should be replaced by a virtual wafer such that $x_{g+h\times n_i} = 1$. Statement (10) is performed to do so. In addition, R is updated by performing Statement (11). Similar to Statements (11)–(15) in Algorithms 3 and 4, Statements (12)–(16) in Algorithm 5 are performed to modify π and x_j.

In Case (2), V is updated by Statement (18). At this time, if $V = 2$, it means that two virtual wafers are consecutively loaded into a PM such that the chamber cleaning requirement is met. Next, if a real wafer is loaded into the PM, R and V should be updated to be one and zero, respectively. Thus, if $V = 2$ holds after Statement (18) is performed, R and V are updated by Statements (20) and (21), respectively. In addition, V_0 is updated to be one by Statement (22). By Statement (23), if $V_0 = 0$, it means that there does not exist two virtual wafers to be consecutively processed by a PM during two adjacent wafer loading sequences so as to meet the chamber cleaning requirement for the PM. Thus, two adjacent wafers (represented by $x_{g+(k_i-1)\times n_i}$ and $x_{g+k_i\times n_i}$) are replaced by virtual wafers if they are real wafers. Then, similar to Statements (12)–(16), Statements (25)–(29) and (32)–(36) are performed to modify π and x_j for the cases if $x_{g+(k_i-1)\times n_i} = 0$ and $x_{g+k_i\times n_i} = 0$ checked by Statements (23) and (30), respectively. In this way, an individual is modified such that for Step i with $d_i = 2$ and $i \in \mathrm{N}^+_{\mathrm{n}}$, the chamber cleaning requirement is met.

Algorithm 5: Individual modification for the case with $d_i = 2$

```
(1)  For g = 1 to n_i
(2)      R = 0;
(3)      V = 0;
(4)      V_0 = 0;
(5)      For h = 0 to 2k_i
(6)          If x_{g+h×n_i} = 0
(7)              R = R + 1;
(8)              V = max(V−1, 0);
(9)              If R > m_i
(10)                 x_{g+h×n_i} = 1;
(11)                 R = max(R−1, 0);
(12)                 For f = 1 to q
(13)                     If (g + h×n_i − f)/q is a non-negative integer
(14)                     x_f = 1;
(15)                     x_f' = x_f;
(16)                     Perform one of Algorithms 1 and 2 for Step s;
(17)         If x_{g+h×n_i} = 1
(18)             V = V + 1;
(19)             If V = 2
(20)                 R = 0;
(21)                 V = 0;
(22)                 V_0 = 1;
(23)     If V_0 = 0 and x_{g+(k_i−1)×n_i} = 0;
(24)         x_{g+(k_i−1)×n_i} = 1;
(25)         For f = 1 to q
(26)             If (g + (k_i−1)×n_i − f)/q is a non-negative integer
(27)             x_f = 1;
(28)             x_f' = x_f;
(29)             Perform one of Algorithms 1 and 2 for Step s;
(30)     If V_0 = 0 and x_{g+k_i×n_i} = 0;
(31)         x_{g+k_i×n_i} = 1;
(32)         For f = 1 to q
(33)             If (g + k_i×n_i − f)/q is a non-negative integer
(34)             x_f = 1;
(35)             x_f' = x_f;
(36)             Perform one of Algorithms 1 and 2 for Step s;
```

3.2. Selection, Crossover, and Mutation Mechanism

For the selection mechanism, a much simpler and faster selection scheme, called n-tournament, is adopted in this work. For n-tournament selection, n individuals are randomly selected from the population where n is determined in advance. Then, an individual among the selected ones wins the tournament according to their fitness values and is finally selected. Note that this work aims at maximizing the number of real wafers in π. Thus, the fitness value of an individual can be obtained by the following expression.

$$F_\pi = \left(\sum_{j=1}^{q}(1 - x_j)\right)/q \tag{7}$$

In Algorithm 2, F_π is the fitness value of an individual represented by π. Thus, for n-tournament selection, an individual with the highest fitness value in the randomly selected ones wins the tournament in this work.

For the selected individuals (parents) by the n-tournament selection mechanism, crossover operations are applied to generate new offsprings. After crossover operation between two individuals, their chromosomes are combined to create a new offspring. Since a single-point crossover operator is simple and can be easily implemented, it is adopted in this work and illustrated in Figure 2. For two individuals, a crossover point in the array of

bits is randomly selected, and exchanging then takes place between them such that their two offsprings are generated. Further, in order to avoid the premature convergence in the searching process, single-point mutation operations are adopted on the offsprings obtained by crossover operations. Specifically, a position of an offspring individual is randomly selected. If the chromosome at the position is one/zero, then it is changed to be zero/one. Single-point mutation operations are illustrated in Figure 2.

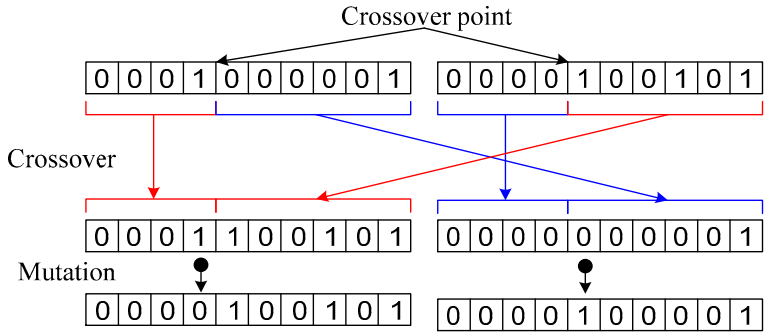

Figure 2. Crossover and mutation operations.

3.3. Procedure of Designed GA

In this work, GA is applied to solve the addressed problem. The population consists of γ individuals, and they are randomly generated for the initial population. Each individual represents a candidate solution denoted by $\pi = (x'_1, x'_2, \ldots, x'_q)$, i.e., a periodically repeated sequence of real and virtual wafers loaded into the tool. In order to check if the chamber cleaning requirement at Step i, $i \in N^+_n$, is met, the type (i.e., virtual or real) of the j-th wafer processed by a PM at the step needs to be known. Thus, for each individual, one of Algorithms 1 and 2 is applied to update x_j, $j > 0$. Then, one of Algorithms 3–5 is applied to modify the individual such that it is a feasible candidate solution. When all individuals are modified to be feasible ones, explorations are performed by selection, crossover, and mutation operations such that a new generation is obtained. Each individual in the new generation is modified to be feasible by Algorithms 1–5 accordingly. After that, selection, crossover, and mutation operations are performed to generate new offspring individuals again. Such a searching process is repeatedly performed until the termination condition is met. At this time, an optimal or near optimal solution can be obtained.

Let P_c and P_m be the probability of crossover and mutation, respectively, and rand [0,1] denote a randomly generated real number in [0,1]. Further, assume that $\gamma = 2\gamma_1$ with γ_1 being a positive integer. Then, the proposed GA is given in Procedure 1.

Procedure 1: The proposed genetic algorithm is summarized as follows.

Step 1 *Initialization: Randomly generate a population with γ individuals;*

Step 2 *Individual Modification:*

(1) Update x_j, $j > 0$, for each individual by one of Algorithms 1 and 2; and

(2) Modify each individual by one of Algorithms 3–5.

Step 3 *Fitness value calculation*: For each individual, its fitness value is obtained by (7).

Step 4 *Selection:* The n-tournament selection is performed for γ times such that γ selected individuals form a new population.

Step 5 *Crossover (Obtain a new generation):*

(1) γ individuals of the new population obtained by Step 4 are divided into $\gamma/2$ groups, and

(2) For each group, if $P_c >$ rand[0,1], a single-point crossover operation is performed to generate two new individuals which are put in the new generation, and otherwise the two individuals in the group are directly put into the new generation.

Step 6 *Mutation:* For each individual, if P_m > rand[0,1], a single-point mutation operation is performed.
Step 7 Same as Step 2.
Step 8 Same as Step 3.
Step 9 If the termination condition is met, then output an individual with highest fitness value in the current population, else go to Step 4.

4. Experiments

4.1. Parameter Setting

To test the performance of the proposed method in this work, numerical experiments are carried out. The GA presented in Procedure 1 is coded in MATLAB and runs on a laptop with eight Intel(R) Core(TM) i7-10750H CPU @2.60 GHz. In order to select a better group of parameters (including γ, n-tournament, P_c, and P_m) for the GA, experiments were performed with different parameter settings for 20 cases. Let $\Gamma_1 = (n_1, n_2, \ldots, n_n)$, $\Gamma_2 = (m_1, m_2, \ldots, m_n)$, and $\Gamma_3 = (d_1, d_2, \ldots, d_n)$. Note that n is the number of steps in a cluster tool. These 20 cases are shown in Table 1.

Table 1. The 20 cases used for experiments.

1:	n = 2; Γ_1 = (1, 1); Γ_2 = (3, 2); Γ_3 = (1, 1);	2:	n = 2; Γ_1 = (1, 2); Γ_2 = (6, 4); Γ_3 = (1, 1);	3:	n = 2; Γ_1 = (2, 2); Γ_2 = (8, 8); Γ_3 = (1, 1);	4:	n = 2; Γ_1 = (3, 2); Γ_2 = (8, 10); Γ_3 = (1, 1);
5:	n = 2; Γ_1 = (1, 3); Γ_2 = (7, 8); Γ_3 = (1, 1);	6:	n = 2; Γ_1 = (1, 4); Γ_2 = (5, 10); Γ_3 = (1, 1);	7:	n = 3; Γ_1 = (1, 2, 2); Γ_2 = (5, 5, 7); Γ_3 = (1, 1, 1);	8:	n = 3; Γ_1 = (1, 1, 2); Γ_2 = (7, 6, 9); Γ_3 = (1, 1, 1);
9:	n = 3; Γ_1 = (1, 2, 3); Γ_2 = (7, 8, 7); Γ_3 = (1, 1, 1);	10:	n = 4; Γ_1 = (1, 2, 2, 1); Γ_2 = (6, 6, 8, 8); Γ_3 = (1, 1, 1, 1);	11:	n = 2; Γ_1 = (1, 1); Γ_2 = (6, 8); Γ_3 = (1, 2);	12:	n = 2; Γ_1 = (2, 1); Γ_2 = (5, 5); Γ_3 = (2, 1);
13:	n = 2; Γ_1 = (2, 2); Γ_2 = (8, 9); Γ_3 = (2, 2);	14:	n = 2; Γ_1 = (3, 2); Γ_2 = (8, 10); Γ_3 = (2, 2);	15:	n = 2; Γ_1 = (1, 3); Γ_2 = (7, 10); Γ_3 = (1, 2);	16:	n = 2; Γ_1 = (1, 4); Γ_2 = (5, 10); Γ_3 = (1, 2);
17:	n = 3; Γ_1 = (1, 2, 2); Γ_2 = (5, 7, 8); Γ_3 = (1, 2, 2);	18:	n = 3; Γ_1 = (3, 1, 2); Γ_2 = (10, 8, 6); Γ_3 = (2, 1, 2);	19:	n = 3; Γ_1 = (1, 2, 3); Γ_2 = (7, 8, 7); Γ_3 = (1, 2, 1);	20:	n = 4; Γ_1 = (1, 3, 2, 2); Γ_2 = (4, 7, 5, 5); Γ_3 = (1, 1, 1, 2);

In this work, $\pi = (x'_1, x'_2, \ldots, x'_q)$ denotes a PVS for a cluster tool. Furthermore, each individual in the GA represents a candidate solution denoted by π. From the perspective of scheduling and control, it is not desired to operate a cluster tool under a PVS with a large value of q since a cluster tool may frequently switch from processing one wafer type to another in a modern semiconductor fab. Thus, in this work, the upper bound of q is set to be 100. Note that, if q equals one, for the 20 cases with the chamber cleaning requirement, π can be obtained by setting $x'_1 = 1$. This implies that the cluster tool always processes virtual wafers and such a PVS is meaningless. Thus, the lower bound of q is set to be two. Then, for each case, GA with a group of parameters as shown in Table 2 is performed for 99 times to find a better π. Note that the termination condition of GA is that when the iteration time reaches 200. Therefore, we have totally performed $20 \times 99 \times 45 = 89{,}100$ experiments in order to select a group of parameters for GA with good performance.

Table 2. Experimental results of the presented GA with different parameter settings.

Population Size	n-Tournament	P_c	P_m	No.	TI_i, $i \in \{1, 2, \ldots, 45\}$	φ_i, $i \in \{1, 2, \ldots, 45\}$
10	5	0.1	0.05	1	441	44.35
			0.15	2	583	30.35
			0.25	3	734	21.50
		0.3	0.05	4	452	43.30
			0.15	5	604	30.20
			0.25	6	732	19.85
		0.5	0.05	7	447	42.55
			0.15	8	627	29.10
			0.25	9	741	18.80
20	5	0.1	0.05	10	513	39.60
			0.15	11	682	24.70
			0.25	12	834	12.40
		0.3	0.05	13	531	36.55
			0.15	14	721	21.60
			0.25	15	833	12.25
		0.5	0.05	16	534	35.75
			0.15	17	721	20.40
			0.25	18	872	9.25
	10	0.1	0.05	19	525	39.55
			0.15	20	667	22.15
			0.25	21	815	12.40
		0.3	0.05	22	513	38.85
			0.15	23	719	24.95
			0.25	24	850	10.10
		0.5	0.05	25	548	36.40
			0.15	26	713	22.40
			0.25	27	890	8.30
30	5	0.1	0.05	28	558	34.30
			0.15	29	772	19.90
			0.25	30	909	6.85
		0.3	0.05	31	576	32.40
			0.15	32	770	14.65
			0.25	33	931	4.95
		0.5	0.05	34	596	28.85
			0.15	35	763	13.70
			0.25	36	*970*	*6*
	10	0.1	0.05	37	552	32.90
			0.15	38	759	19.55
			0.25	39	906	7.45

Table 2. *Cont.*

Population Size	*n*-Tournament	P_c	P_m	No.	TI_i, $i \in \{1, 2, \ldots, 45\}$	φ_i, $i \in \{1, 2, \ldots, 45\}$
		0.3	0.05	40	587	32
			0.15	41	746	17.45
			0.25	42	936	6.3
		0.5	0.05	43	581	31.70
			0.15	44	816	14.65
			0.25	45	963	3.8

One way to measure a group of parameter settings of GA is introduced as follows. Let $F_\pi(i, j, q)$ denote the fitness value obtained by GA with the i-th group of parameters for Case j under a PVS in which the number of wafers in a repeated wafer loading sequence is q, $i \in \{1, 2, \ldots, 45\}$, $j \in \{1, 2, \ldots, 20\}$, and $q \in \{2, 3, \ldots, 100\}$. Let $I(u_1, u_2) = 1$ if $u_1 = u_2$, and otherwise $I(u_1, u_2) = 0$. Then, $I_i(max(F_\pi(i, j, q) \mid i \in \{1, 2, \ldots, 45\}), F_\pi(i, j, q))$ is used to record if GA with the i-th group of parameters achieves the best solution among the 45 groups of parameter settings under the given j and q. Let TI_i denote the total number of best solutions that GA with the i-th group of parameters achieves among the 45 group of parameter settings under different j and q. Thus, we have

$$TI_i = \sum_{j=1}^{20} \sum_{q=2}^{100} I_i(max(F_\pi(i,j,q)|i\{1,\ 2,\ \ldots,\ 45\}), F_\pi(i,j,q)),\ i\ \in\ \{1,\ 2,\ \ldots,\ 45\} \quad (8)$$

TI_i reflects the capability of GA with the i-th group of parameters to search a better solution among the 45 different parameter settings to some extent. For a given group of parameters and a case, the capability of GA to search for a good solution may decline as the value of q increases since increasing the value of q enlarges the solution space. In fact, as the value of q increases, the value of F_π of a local optimal solution (i.e., an optimal solution under a given q) obtained by (7) is much closer to that of a global optimal solution. This implies that the trend of F_π's value becomes stable as the q's value increases. Thus, the stability of GA to search a good solution is used to measure a group of parameter settings as well, and introduced next.

Let $\Psi_1(i, j)$ represent the stability of GA with the i-th group of parameters to search for a solution if q varies from two to 100 for Case j, $i \in \{1, 2, \ldots, 45\}$ and $j \in \{1, 2, \ldots, 20\}$. Then, $\Psi_1(i, j)$ is obtained as

$$\Psi_1(i,\ j) = \left(\sum_{q=2}^{100} [\max(F_\pi(i,j,q)|q \in \{2,3,\ldots,100\}) - F_\pi(i,j,q)]^2\right)/99, \quad i\ \in\ \{1,\ 2,\ \ldots,\ 45\}\text{and}j\ \in\ \{1,\ 2,\ \ldots,\ 20\} \quad (9)$$

Further, $\Psi_2(i, j)$ records the ranking value of the i-th group of parameters among the 45 groups of parameter settings according to $\Psi_1(i, j)$ for a given Case j. Specifically, for Case j (i.e., the value of j is determined), if $\Psi_1(i, j)$ is the f-th smallest one of all values of $\Psi_1(i, j)$ for all $i \in \{1, 2, \ldots, 45\}$, value f is given to $\Psi_2(i, j)$. In this way, $\Psi_2(i, j)$ can be obtained, $i \in \{1, 2, \ldots, 45\}$ and $j \in \{1, 2, \ldots, 20\}$. Further, φ_i is used to record the average ranking value of the j-th group of parameter settings for the 20 cases. Then, we have

$$\varphi_i = \left(\sum_{j=1_2}^{20}(i,j)\right)/20,\ i\ \in\ \{1,\ 2,\ \ldots,\ 45\} \quad (10)$$

Note that for the i-th group of parameters, the smaller the value of φ_i is, the more stable GA is to explore a good solution even if the solution space is enlarged, i.e., the value of q increases. Then, TI_i and φ_i are used to measure a group of parameter settings. After 89,100 experiments with different parameter settings and the data analysis according

to Expressions (8)–(10), experimental results are summarized in Table 2. It follows from Table 2 that the parameter setting of the 36th group has the largest value of TI_i for all $i \in \{1, 2, \ldots, 45\}$ and at the same time the value of φ_i is quite small. Thus, the parameter setting of the 36th group is selected for the presented GA in this work.

4.2. Performance Evaluation

To test the performance of the presented GA, for the 20 cases in Table 3, comparisons are made between GA and the upper bound of the average number of wafers being produced in each robot cycle. Let UB denote such an upper bound. For Step i with $d_i = 1$, $i \in N^+_n$, a PM at the step has to process one virtual wafer after no more than m_i real wafers being processed. This means that in every $m_i + 1$ robot cycles, m_i real wafers are processed at most since each robot cycle completes one wafer only. Thus, the upper bound of the average number of real wafers being produced at the step is $m_i/(m_i + 1)$ in each robot cycle. Similarly, at Step i with $d_i = 2$, $i \in N^+_n$, the upper bound of the average number of real wafers being produced is $m_i/(m_i + 2)$ in each robot cycle. Then, we have

$$\text{UB} = min(\{m_i/(m_i + 1) \mid d_i = 1, i \in N^+_n\}, \{m_i/(m_i + 2) \mid d_i = 2, i \in N^+_n\}) \quad (11)$$

Table 3. Comparison results.

Case No.	UB	GA		
		Results	Running Time	GAP-G-U
1	0.6667	0.6667	17.71 s	0
2	0.8000	0.8000	28.30 s	0
3	0.8889	0.8889	27.68 s	0
4	0.8889	0.8889	54.97 s	0
5	0.8750	0.8750	51.38 s	0
6	0.8333	0.8276	78.58 s	0.68%
7	0.8333	0.8000	28.69 s	4%
8	0.8571	0.8571	28.50 s	0
9	0.8750	0.8571	56.43 s	2.05%
10	0.8571	0.8571	28.53 s	0
11	0.8000	0.7500	18.13 s	6.25%
12	0.7143	0.7143	31.28 s	0
13	0.8000	0.8000	31.84 s	0
14	0.8000	0.7500	65.52 s	6.25%
15	0.8333	0.8182	59.47 s	1.81%
16	0.8333	0.7955	90.39 s	4.54%
17	0.7778	0.7241	32.13 s	6.91%
18	0.7500	0.7188	66.84 s	4.16%
19	0.8000	0.7500	51.95 s	6.25%
20	0.7143	0.7143	53.46 s	0
	Average:		45.09 s	2.15%

In Table 3, GAP-G-U is the gap between the results obtained by GA and UB. It follows from the comparison results in the table that for Cases 1–5, 8, 10, 12, 13, and 20, the values of GAP-G-U equal zero, implying that GA finds the optimal solutions for these cases. For Cases 6, 7, 9, 15, 16 and 18, the values of GAP-G-U are less than 5%; while for Cases 11, 14,

17, and 19, all values of GAP-G-U are between 6% and 7%. Furthermore, the average GAP-G-U of the 20 cases is 2.15%, that is acceptable from the perspective of real applications. Moreover, the average time to find a solution by the constructed GA is just 45.09 s which is quite short. Therefore, the proposed method in this work can be put into practice.

5. Conclusions

This work deals with a scheduling problem for SACTs with WRTCs and chamber cleaning operations which are commonly seen requirements in real semiconductor manufacturing. Different from the studies on scheduling SACTs with purge operations in [22], this work focuses on a more general case in which a chamber at a PM may process more than one wafer before a cleaning operation is required. To do so, this work presents a virtual wafer-based method such that a PM processes either a real wafer or a virtual wafer at a time. Then, based on an optimal one-wafer cyclic schedule that can be obtained by the method in [15], by the proposed virtual wafer-based method, a genetic algorithm is developed to find an optimal or near optimal solution. Extensive experiments are conducted to verify the proposed method. Based on the experimental results, GA can be used to find a high-quality solution within reasonable time. Furthermore, the obtained solution is a periodical wafer loading sequence based on a one-wafer cyclic schedule that can be easily implemented. Therefore, it has a high practical value in semiconductor manufacturing.

In practice, the PMs in cluster tools are prone to failure [27,28]. Such a PM failure may result in a deadlock such that the WRTCs in PMs are violated. It is necessary to establish real-time failure response policies to deal with PM failures in different situations. Thus, our future work should deal with PM failures for cluster tools with WRTCs and chamber cleaning operations.

Author Contributions: Conceptualization, J.L. and Y.Q.; methodology, J.L., Y.Q. and S.Z.; validation, J.L. and S.Z.; investigation, T.S.; writing—original draft preparation, J.L.; writing—review and editing, Y.Q., Z.L. and N.W.; supervision, Y.Q., Z.L. and N.W.; funding acquisition, N.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by National Natural Science Foundation of China (NSFC), grant number 61803397 and in part by Science and Technology development fund (FDCT), Macau SAR (File Nos: 0017/2019/A1, 0018/2021/A1, and 0083/2021/A2).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lee, T.-E.; Lee, H.-Y.; Shin, Y.-H. Workload balancing and scheduling of a single-armed cluster tool. In Proceedings of the 5th APIEMS Conference, Gold Coast, Australia, 12–15 December 2004; pp. 1–15.
2. Venkatesh, S.; Davenport, R.; Foxhoven, P.; Nulman, J. A steady state throughput analysis of cluster tools: Dual-blade versus single-blade robots. *IEEE Trans. Semicond. Manuf.* **1997**, *10*, 418–424. [CrossRef]
3. Rostami, S.; Hamidzadeh, B.; Camporese, D. An optimal periodic scheduler for dual-arm robots in cluster tools with residency constraints. *IEEE Trans. Robot. Autom.* **2001**, *17*, 609–618. [CrossRef]
4. Lee, T.-E.; Park, S.-H. An extended event graph with negative places and tokens for timed window constraints. *IEEE Trans. Autom. Sci. Eng.* **2005**, *2*, 319–332. [CrossRef]
5. Kim, J.-H.; Lee, T.-E.; Lee, H.-Y.; Park, D.-B. Scheduling analysis of timed-constrained dual-armed cluster tools. *IEEE Trans. Semicond. Manuf.* **2003**, *16*, 521–534. [CrossRef]
6. Xiong, W.Q.; Pan, C.R.; Qiao, Y.; Wu, N.Q.; Chen, M.X.; Hsieh, P.H. Reducing wafer delay time by robot idle time regulation for single-arm cluster tools. *IEEE Trans. Autom. Sci. Eng.* **2020**. [CrossRef]
7. Jacob, R.; Amari, S. Output feedback control of discrete processes under time constraint: Application to cluster tools. *Int. J. Comput. Integr. Manuf.* **2017**, *30*, 880–894. [CrossRef]
8. Kim, C.; Lee, T.-E. Feedback Control of Cluster Tools for Regulating Wafer Delays. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 1189–1199. [CrossRef]

9. Lim, Y.; Yu, T.S.; Lee, T.E. A new class of sequences without interferences for cluster tools with tight wafer delay constraints. *IEEE Trans. Autom. Sci. Eng.* **2019**, *16*, 392–405. [CrossRef]
10. Roh, D.H.; Lee, T.G.; Lee, T.E. K-cyclic schedules and the worst-case wafer delay in a dual-armed cluster tool. *IEEE Trans. Semicond. Manuf.* **2019**, *32*, 236–249. [CrossRef]
11. Lim, Y.; Yu, T.S.; Lee, T.E. Adaptive scheduling of cluster tools with wafer delay constraints and process time variation. *IEEE Trans. Autom. Sci. Eng.* **2020**, *17*, 375–388. [CrossRef]
12. Wang, J.; Pan, C.; Hu, H.; Li, L.; Zhou, Y. A cyclic scheduling approach to single-arm cluster tools with multiple wafer types and residency time constraints. *IEEE Trans. Autom. Sci. Eng.* **2019**, *16*, 1373–1386. [CrossRef]
13. Ko, S.G.; Yu, T.S.; Lee, T.E. Wafer delay analysis and workload balancing of parallel chambers for dual-armed cluster tools with multiple wafer types. *IEEE Trans. Autom. Sci. Eng.* **2021**, *18*, 1516–1526. [CrossRef]
14. Wang, J.; Hu, H.; Pan, C.; Li, L.; Zhou, Y.; Li, L. Scheduling dual-arm cluster tools with multiple wafer types and residency time constraints. *IEEE/CAA J. Autom. Sin.* **2020**, *7*, 776–789. [CrossRef]
15. Wu, N.Q.; Chu, C.B.; Chu, F.; Zhou, M.C. A Petri net method for schedulability and scheduling problems in single-arm cluster tools with wafer residency time constraints. *IEEE Trans. Semicond. Manuf.* **2008**, *21*, 224–237. [CrossRef]
16. Wu, N.Q.; Zhou, M.C. A closed-form solution for schedulability and optimal scheduling of dual-arm cluster tools with wafer residency time constraint based on steady schedule analysis. *IEEE Trans. Autom. Sci. Eng.* **2010**, *7*, 303–315.
17. Pan, C.R.; Qiao, Y.; Wu, N.Q.; Zhou, M.C. A novel algorithm for wafer sojourn time analysis of single-arm cluster tools with wafer residency time constraints and activity time variation. *IEEE Trans. Syst. Man Cybern. Syst.* **2015**, *45*, 805–818.
18. Qiao, Y.; Wu, N.Q.; Zhou, M.C. Petri net modeling and wafer sojourn time analysis of single-arm cluster tools with residency time constraint and activity time variation. *IEEE Trans. Semicond. Manuf.* **2012**, *25*, 432–446. [CrossRef]
19. Qiao, Y.; Wu, N.Q.; Zhou, M.C. Real-time scheduling of single-arm cluster tools subject to residency time constraints and bounded activity time variation. *IEEE Trans. Autom. Sci. Eng.* **2012**, *9*, 564–577. [CrossRef]
20. Wu, N.Q.; Zhou, M.C. Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets. *IEEE Trans. Autom. Sci. Eng.* **2012**, *9*, 446–454.
21. Wu, N.Q.; Zhou, M.C. Schedulability analysis and optimal scheduling of dual-arm cluster tools with residency time constraint and activity time variation. *IEEE Trans. Autom. Sci. Eng.* **2012**, *9*, 203–209.
22. Yu, T.S.; Kim, H.J.; Lee, T.E. Scheduling single-armed cluster tools with chamber cleaning operations. *IEEE Trans. Autom. Sci. Eng.* **2018**, *15*, 705–716. [CrossRef]
23. Yu, T.S.; Lee, T.E. Scheduling dual-armed cluster tools with chamber cleaning operations. *IEEE Trans. Autom. Sci. Eng.* **2019**, *16*, 218–228. [CrossRef]
24. Abderrabi, F.; Godichaud, M.; Yalaoui, A.; Yalaoui, F.; Amodeo, L.; Qerimi, A.; Thivet, E. Flexible job shop scheduling problem with sequence dependent setup time and job splitting: Hospital catering case study. *Appl. Sci.* **2021**, *11*, 1504. [CrossRef]
25. Park, J.-S.; Ng, H.-Y.; Chua, T.-J.; Ng, Y.-T.; Kim, J.-W. Unified genetic algorithm approach for solving flexible job-shop scheduling problem. *Appl. Sci.* **2021**, *11*, 6454. [CrossRef]
26. Zou, P.; Rajora, M.; Liang, S.Y. Multimodal optimization of permutation flow-shop scheduling problems using a clustering-genetic-algorithm-based approach. *Appl. Sci.* **2021**, *11*, 3388. [CrossRef]
27. Qiao, Y.; Wu, N.Q.; Pan, C.R.; Zhou, M.C. How to respond to process module failure in residency time-constrained single-arm cluster tools. *IEEE Trans. Semicond. Manuf.* **2014**, *27*, 462–474. [CrossRef]
28. Qiao, Y.; Pan, C.R.; Wu, N.Q.; Zhou, M.C. Response policies to process module failure in single-arm cluster tools subject to wafer residency time constraints. *IEEE Trans. Autom. Sci. Eng.* **2015**, *12*, 1125–1139. [CrossRef]