> SMCA-25-01-0239 <

Workload Balancing for Photolithography Machines in Semiconductor Manufacturing via Estimation of Distribution Algorithm Integrating Kmeans Clustering

LiangChao Chen, Yan Qiao, *Senior Member*, *IEEE*, NaiQi Wu, *Fellow*, *IEEE*, Mohammadhossein Ghahramani, YongHua Shao, and SiJun Zhan

***Abstract*—This work focuses on the scheduling of a photolithography area with multiple machine groups and each one consists of a predetermined number of *photolithography machines* (PMs). PMs belonging to the same machine group should have identical processing capacities. Additionally, all PMs are designated with downward processing compatibility. This means that the wafers requiring relatively low pattern precision can be processed by the PMs used to deal with high pattern precision. After executing a photolithography process, a circuit pattern is transferred from an auxiliary resource called a reticle onto the wafer surface. Moreover, when processing wafers with different reticle and processing environment requirements, the machine setup is necessary. With those complex processing requirements, the objective is to minimize the difference between the longest and shortest working time of PMs so as to balance the workloads among all PMs. To do so, a mixed-integer linear programming model is built and then solved by using CPLEX for the small-sized problem. For medium- and large-sized problems, a designed estimation of distribution algorithm integrating a Kmeans clustering is constructed to improve the productivity of the photolithography area. Comparison results show that the proposed method outperforms the compared algorithms regardless of problem sizes.**

***Index Terms*—Estimation of distribution algorithm, photolithography, scheduling, semiconductor manufacturing**

I. Introduction

In semiconductor manufacturing, the photolithography process is widely acknowledged as a bottleneck step due to its reliance on highly expensive machines (*i.e.*, steppers) for production [1]. Moreover, the photolithography area serves as the dispatching center of wafer fabs and controls the performance of the whole manufacturing process [2–4]. Under this situation, an efficient production schedule for the photolithography area becomes necessary, making scheduling analysis in photolithography areas important in modern semiconductor fabs.

In a photolithography area, raw wafers typically made of silicon are placed into the *photolithography machines* (PMs) where geometric circuit patterns are then printed on their photoresist layers. To achieve this, auxiliary resources known as reticles (or masks) are necessary which contain the required circuit patterns that will be transferred onto wafer surfaces by the ultraviolet (also known as UV in short) light provided by PMs. Due to the considerable expense of reticles, for example, a single reticle may exceed $100K in cost [5], it is a common practice in semiconductor fabs where all reticles used are different. The reticles required by wafers depend on their wafer fabrication process requirements. Moreover, it should be noticed that each reticle contains only one circuit pattern. Therefore, if two raw wafers require identical circuit patterns, they should share the same reticle.

The circuit pattern created on a wafer surface has a corresponding pattern precision. To make the pattern precision satisfied, only relying on reticles is not enough. PMs in use should provide appropriate processing environments as well. Notice that semiconductor fabs normally have different types of PMs that can provide different processing environments to meet the different pattern precision demands. To guarantee the processing environment and reticle required by a raw wafer are matched, before a PM starts to run, the machine setup is necessary.

In this work, there exist three different situations for machine setup. The first situation occurs for the reticle switching. There are two different scenarios for performing reticle switching since reticles are auxiliary resources. The first scenario is that the required reticle is already stored in the internal buffer (also known as the reticle pod) of a PM, while the second scenario is that the required reticle should be manually delivered from elsewhere [6] and then installed in the internal buffer. Typically, such a manual delivery process takes around 15 minutes. Once a reticle is placed in the internal buffer, the reticle switching process is automatically performed by the PM

This work was supported in part by the Science and Technology development fund (FDCT), Macau SAR, under Grant 0018/2021/A1, Grant 0011/2023/RIA1, and 0120/2024/RIA2. (*Corresponding author: YanQiao*.)

LiangChao Chen, Yan Qiao, and NaiQi Wu are with the Macao Institute of Systems Engineering and Collaborative Laboratory for Intelligent Science and Systems, Macau University of Science and Technology, Macau, China (e-mail: liangchao_chen@163.com; yqiao@must.edu.mo ; nqwu@must.edu.mo).

Mohammadhossein Ghahramani is with the Birmingham City University, Birmingham, UK (e-mail: mohammadhossein.ghahramani@bcu.ac.uk).

YongHua Shao and SiJun Zhan are with the AscenPower Semiconductors Co., Ltd., Guangzhou 510000, China (e-mail: yonghua.shao@ascenpower.com; james.zhan@ascenpower.com).

itself. This process is extremely fast, typically taking only a few seconds.

The second situation of machine setup is recipe switching. The recipe of a wafer at a PM mainly contains the processing parameters, such as the required temperature and pressure. Therefore, the recipe of a wafer could specify the processing environment at a PM. The time used for recipe switching is spent on adjusting the internal processing environment of a PM to match the required one for the recipe of the upcoming wafers. Given that each circuit pattern is associated with one specific pattern precision that is related to the processing environment in a PM, when there is a need to perform recipe switching in the PM, this means that circuit patterns printed on two consecutively processed wafer surfaces are different, and at the same time, reticle switching is necessary. According to real-world applications, the time required for recipe switching is around 20 to 30 minutes. Importantly, when the recipe switching is in progress, the reticle switching can be performed simultaneously. Therefore, if there is a need to perform both recipe and reticle switching, then the total time needed is determined by the time spent on recipe switching.

The last situation of machine setup is machine initialization preparation. The investigated photolithography area operates in shifts. The duration of a shift depends on the number of raw wafers needed to be processed. Before starting a new shift, PMs are required to perform a machine initialization preparation if the initial required recipe differs from the last one of the previous shift. Notice that the time required for machine initialization preparation is machine-dependent. Typically, the machine initialization preparation takes around 60 minutes, making it the most time-consuming process among the three different situations of machine setup. Also, during the machine initialization preparation period, reticle and recipe switching can be performed simultaneously. Therefore, the time spent on machine initialization preparation is sufficient to cover both the recipe and reticle switching processes.

In general, the processing environment required for achieving a high pattern precision is much stricter than those for lower pattern precisions. Additionally, to improve machine utilization, PMs are all designed with downward processing compatibility. To describe such a downward processing capability, let $\boldsymbol{A}$ be a set of integers representing pattern precisions. Further, if Pattern Precision-a is higher than Pattern Precision-l, a, $l \in \boldsymbol{A}$, then $l > a$ should hold. In the meantime, based on the downward processing compatibility of each PM, if a PM can serve raw wafers to achieve Pattern Precision-a, then it can also serve raw wafers to achieve Pattern Precision-l. Under such an operational mode, the processing capability of a PM is defined as the highest pattern precision that it can handle. This allows us to create multiple unrelated parallel machine groups where PMs belonging to the same machine group should have identical processing capabilities.

As reticles are auxiliary resources, their usage is flexible, *i.e.*, reticles can be shared among different PMs. Hence, a raw wafer can be processed at any PM capable of providing the suitable processing environment to meet its pattern precision requirement. Thus, there exists a scheduling problem of assigning raw wafers to PMs while satisfying production requirements. In fact, the lower the pattern precision required by a raw wafer, the more PMs it can be assigned to. This is due to the downward processing compatibility of PMs and the flexible usage of reticles. Notice that the processing time of a raw wafer processed at different PMs with different processing capacities may be different.

Given that each PM comes at a significant cost, the number of PMs is limited in a photolithography area. Besides, in real-world applications, the photolithography area is arranged a large number of raw wafers to process, resulting in the workloads of PMs being extremely heavy. An uneven distribution of workloads can lead to overloading certain PMs, resulting in significant performance discrepancies (also known as machine variance), which directly affects the quality consistency of wafers produced by PMs. Therefore, the distribution of workloads among PMs has a significant impact not only on the productivity of the system but also on the equipment longevity. Consequently, it is essential to find an efficient schedule to balance machine workloads while satisfying production requirements. Motivated by this real application requirement, this work conducts a scheduling analysis of a photolithography area with the objective of minimizing the difference between the longest and shortest working time of PMs, *i.e.*, balancing the workloads among the PMs.

For this complex manufacturing system, we first formulate it as a *mixed integer linear programming* (MILP) model. Due to the complexity of the system, optimal solutions are difficult to obtain as the problem size grows. To address this challenge, we propose an *Estimation of Distribution Algorithm* (EDA) integrated with *K*means clustering. The efficiency and effectiveness of the proposed methods are evaluated through extensive experiments on three problem sizes (small, medium, and large) based on practical applications. For the small-sized problem, the MILP model is solved by using CPLEX to obtain the optimal solution, which serves as a benchmark for evaluating the performance of proposed methods. For medium- and large-sized practical problems, experimental results demonstrate that our method balances computational efficiency and solution quality, outperforming other popular metaheuristics.

The rest of the work is arranged as follows. Section II reviews the related studies. In Section III, after describing the problem, we build a MILP model. Due to the complexity of the addressed problem, Section IV constructs a metaheuristic method to find high-quality solutions. Then, the performance of the proposed method is analyzed and compared with some existing methods by computational experiments in Section V. Finally, conclusions are given in Section VI.

II. Literature Review

A. Photolithography Scheduling Problem

Scheduling challenges in photolithography areas are extensively discussed in the literature under various processing

constraints [5]. In early studies, although reticles are necessary for printing circuit patterns on wafer surfaces, the reticle requirements in semiconductor manufacturing are not given much attention until the work in [7] is conducted. In [7], a discrete-event simulation model coupled with a network flow is employed to illustrate that managing reticles effectively in semiconductor fabs can greatly improve manufacturing system performance. After that, more and more studies focus on the scheduling problems of photolithography areas by considering reticle requirements as auxiliary resource constraints.

In general, scheduling problems in photolithography areas can be formulated as integer programming or MILP models. It is well known that exact algorithms (such as the branch-and-bound algorithm) can be used to obtain the optimal solution. However, Cakici and Mason [8] state that the scheduling problem with the consideration of reticle requirements is NP-hard, supported by its reduction to a well-established NP-hard problem. Once a problem is examined as NP-hard, exact algorithms would fail to solve it efficiently, especially when dealing with large-scale instances [9]. Doleschal *et al.* [10] also state that employing discrete event simulation and mathematical programming approaches are not feasible for solving complex scheduling problems in photolithography areas. They arrive at this conclusion through an analysis of the limitations of these methods and demonstrate the need for more efficient approaches through simulation experiments and practical application. Under these situations, other more efficient and effective approaches are necessary.

It is worth mentioning that heuristic algorithms are widely used in photolithography scheduling problems. Heuristic algorithms often generate solutions quickly, making them well-suited for handling large-scale problems. In [8], two different heuristics are proposed to minimize the total weighted completion time of PMs with the consideration of reticle requirements. In [11], a time extended objective-oriented Petri nets (EOPNs) is constructed based on a multiple-objective scheduling and real-time dispatching approach. In addition to the proposed EOPNs, a priority-ranking algorithm with four performance objectives is introduced to give a high-quality initial scheduling guidance. In [10], a heuristic scheduling rule called the solver-based reticle allocation approach is proposed. The effectiveness of the solver-based reticle allocation approach is verified by comparing it with classical rule-based dispatching approaches on both the representative test data and real-world cases. Based on the studies reviewed above, it can be concluded that heuristic algorithms have problem-specific natures that result in a better performance for addressing particular types of problems.

With the continuous advancement in semiconductor manufacturing, the industry is witnessing a progressive sophistication in its manufacturing processes. To address the growing complexity of manufacturing processes and improve the productivity, metaheuristics are increasingly employed to pursue high-quality solutions. In [6], a scheduling problem with dual resource constraints which are machine resource and reticle resource constraints is addressed. To do so, two mixed-integer programming models and an improved naked mole-rat algorithm are introduced. Experiments show that the improved naked mole-rat algorithm outperforms the *genetic algorithm* (GA) and the variable neighborhood search algorithm. In [12], a metaheuristic called memetic algorithm is applied to solve the scheduling problem in a photolithography workshop. The effectiveness of the memetic algorithm is evaluated by two criteria which are weighted flow time and the number of processed products. In [13], a two-phase decoding GA is proposed to minimize the idle time for all machines with complex constraints, such as photo mask availability, available machines for jobs, and limited waiting time. In [14], an imperialist competitive algorithm with the connection of a predictive neural network is proposed for addressing the scheduling problem in a dynamic environment under the consideration of wafer arrival constraints, dedicated machines constraints, and auxiliary resources constraints.

B. Unrelated Parallel Machine Scheduling Problem

The scheduling problems in photolithography areas can be treated as unrelated parallel machine scheduling problems (UPMSPs) mentioned in [6, 12]. Thus, studies about UPMSPs in other manufacturing systems can also provide a rich source of inspiration and potential solution approaches.

In [15], an UPMSP is examined in bar-turning manufacturing. In their work, an MILP model is proposed for solving small-sized instances and introduces a two-step approach that employs a relaxed version of the proposed MILP model followed by a heuristic algorithm for large-sized instances. Li *et al.* [16] propose several heuristics that rely on the best fit longest processing time rule to address a scheduling problem of the unrelated parallel batch processing machines with the objective of minimizing the completion time of the last job. Further, many studies consider machine or job setup time before a processing process starts which is similar to the problem addressed in this work. In [17], a non-preemptive UPMSP with job sequence and machine-dependent setup time is examined by a constraint programming model with two customized branching strategies. Afzalirad and Rezaeian [18] propose a GA along with an artificial immune system to solve an UPMSP incorporating sequence-dependent setup time. Also, Arroyo and Leung [19] address an UPMSP where each job has an arbitrary job size and non-zero ready time. They provide the lower bound of the problem and formulate it as an MILP model. Then, they propose an iterated greedy algorithm to tackle the problem and show its effectiveness with comparisons to GA, artificial bee optimizer, and simulated annealing algorithm.

C. Summary

Based on the no free lunch theorem [20], no optimization technique can claim to be the best for all situations, whether for general or specific ones. Thus, to achieve superior optimization results, most studies reviewed above employ an approach that combines different solution methodologies. By considering the practical demands from the investigated wafer fab, the addressed scheduling problem of the photolithography area in this work differs from the abovementioned studies in terms of

the following two aspects: 1) there are three different situations of machine setup, *i.e.*, reticle switching, recipe switching, and machine initialization preparation; and 2) PMs are all designed with downward processing compatibility. To the best knowledge of authors, there is no research report with the consideration of the two aspects. This motivates us to conduct this work.

III. Problem Formulation

A. Problem Description

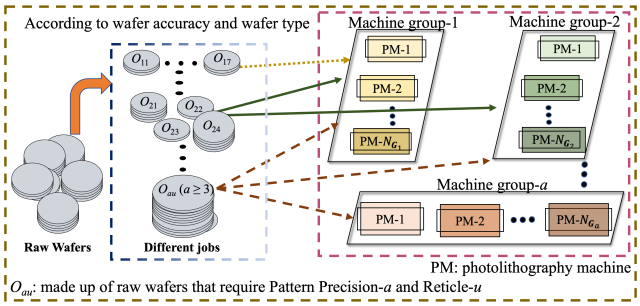


Fig. 1. Layout of a photolithography area.

To well describe the addressed scheduling problem, some notations are introduced as follows. Additionally, all notations used are summarized in Table A in Supplementary File. The layout of the investigated photolithography area is shown in Fig. 1.

The total number of pattern precisions is represented by N_A and $\boldsymbol{A}$ is a set of pattern precisions, where $\boldsymbol{A} = \{1, 2, \ldots, a, \ldots, N_1\}$. Notice that a smaller value of pattern precision indicates a higher precision required by the pattern. For Pattern Precision-a, it has a corresponding set of reticles represented by $\boldsymbol{\beta}_a$. Similarly, N_2 represents the number of elements in $\boldsymbol{\beta}_a$, where $\boldsymbol{\beta}_a = \{1, 2, \ldots, u, \ldots, N_2\}$. This means that the reticle in $\boldsymbol{\beta}_a$ should be processed at a PM that can handle Pattern Precision-a. Note that each reticle is associated with a unique pattern precision. Besides, the total number of machine groups is denoted by N_3 and $\boldsymbol{G} = \{1, 2, \ldots, g, \ldots, N_3\}$ represents the set of machine groups. In this work, the number of machine groups matches the number of pattern precisions, resulting in $N_3 = N_1$. Moreover, if the highest pattern precision that Machine Group-g can handle is Pattern Precision-a, then $a = g$ holds. In other words, the highest pattern precision that Machine Group-g can handle is Pattern Precision-g. In this way, each pattern precision can be dealt with by at least one machine group. The set of PMs belonging to Machine Group-g is denoted as $\boldsymbol{G}_g$ and the number of PMs in $\boldsymbol{G}_g$ is represented by N_4, where $\boldsymbol{G}_g = \{1, 2, \ldots, n, \ldots, N_4\}$.

In semiconductor fabs, each reticle is unique since it is quite expensive. Therefore, the reticles are tight resources in fabs. Thus, to improve the productivity of PMs, raw wafers requiring the same reticle (implying that they should be processed under the same precision requirements) are desired to be continuously processed in a PM so as to reduce the significant setup time. To do so, all raw wafers requiring Pattern Precision-a and Reticle-u can be treated as a job denoted as O_{au}, where $a \in \boldsymbol{A}$, $u \in \boldsymbol{\beta}_a$. As mentioned, jobs with lower pattern precision requirements can be processed at more PMs. As shown in Fig. 1, O_{11} can only be assigned to a PM of Machine Group-1 for processing, while O_{21} can be assigned to a PM of both Machine Group-1 and Machine Group-2 for processing. Accordingly, the key of the addressed scheduling problem is to assign jobs to PMs with the objective of minimizing the difference between the longest and shortest working time of PMs.

In practice, a PM is allowed to equip several reticles in its reticle pod with the same pattern precision. In other words, the reticles simultaneously equipped in a PM should have the same pattern precision or require the same processing environment. When those reticles need to switch, the time is quite short. Also, when a PM continuously processes raw wafers with the same reticle, there is no need to switch reticles or recipes. Under such a situation, the processing time of a job is determined by the number of such raw wafers and the processing time required for each wafer. Additionally, the processing time of a wafer is determined by both the assigned PM and the pattern precision it requires. Let NW_{au} represent the number of raw wafers made up of O_{au} and P_{ga} indicate the processing time of a raw wafer processed at a PM of Machine Group-g with required Pattern Precision-a. To calculate the processing time of each job at each PM, we have

$$T_{gnau} = X_{gnau} \times NW_{au} \times P_{ga}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, a \in \boldsymbol{A}, u \in \boldsymbol{\beta}_a, \quad (1)$$

where X_{gnau} is a binary variable. If O_{au} is assigned to be processed at PM-n of Machine Group-g, then $X_{gnau} = 1$; otherwise, $X_{gnau} = 0$. With Equation (1), the time needed for processing O_{au} at PM-m of Machine Group-g (*i.e.*, T_{gnau}) can be obtained.

There is a fact that wafer products with a higher pattern precision are more valuable. According to real production demands, the PM starts to process raw wafers with the highest pattern precision. Therefore, the processing sequence of raw wafers at each PM follows a descending order with respect to the pattern precision, *i.e.*, such a sequence starts from a wafer with the highest pattern precision and ends to the one with the lowest pattern precision. Thus, at a PM, when the processing for all jobs requiring Pattern Precision-a is completed, the processing for jobs requiring Pattern Precision-l can start, where $a, l \in \boldsymbol{A}$, $a < l$. Notice that the reticles simultaneously equipped in a PM should have the same pattern precision. When the PM should handle some jobs with different pattern precision, the PM should perform a recipe switching to prepare the processing environment, and at the same time the required reticles by these jobs should be equipped into the PM. The recipe switching time is quite long such that it is sufficient to cover the time required for reticle switching, even if the required reticle should be manually delivered from elsewhere. Besides, when the PM handles the jobs requiring the reticles that have been equipped in the PM, the reticle switching processes can be automatically performed accordingly. Note that PMs belonging to the same machine group have identical reticle switching times. Let ST_g represents the time needed for

reticle switching at any PM of Machine Group-g. Then, with the consideration of reticle switching, the total time for processing jobs requiring Pattern Precision-a at PM-n of Machine Group-g is denoted as T_{gna}. To obtain T_{gna}, we have

$$T_{gna} = \sum_{u\in\boldsymbol{\beta}_a} T_{gnau} + (H_{gna} + (\sum_{u\in\boldsymbol{\beta}_a} T_{gnau}) - 1)\times ST_g,\ \forall\ g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g,\ a \in \boldsymbol{A}, \quad (2)$$

where $\sum_{u\in\boldsymbol{\beta}_a} T_{gnau}$ is the sum of total processing time for all jobs requiring Pattern Precision-a at PM-n of Machine Group-g. The rest part $(H_{gna} + (\sum_{u\in\boldsymbol{\beta}_a} T_{gnau} - 1) \times ST_g$ is the total time needed for reticle switching processes. H_{gna} is a binary variable. If there is no job requiring Pattern Precision-a processed at PM-n of Machine Group-g (*i.e.*, $\sum_{u\in\boldsymbol{\beta}_a} T_{gnau} = 0$), then $H_{gna} = 1$; otherwise, $H_{gna} = 0$. As a result, $H_{gna} + (\sum_{u\in\boldsymbol{\beta}_a} T_{gnau}) - 1$ gives the reticle switching times. Let T_{gn} denote the total working time of PM-n belonging to Machine Group-g. Then, T_{gn} can be calculated by adding the consideration of recipe switching and machine initialization preparation. To do so, we introduce the following equation (3).

$$T_{gn} = Q_{gn} \times SC_{gn} + \sum_{a=g}^{N_1} T_{gna} + \sum_{a=g}^{N_1-1} \sum_{l=a+1}^{N_1} (Z_{gnal} \times S_{gal}),\ \forall\ g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g, \quad (3)$$

In Equation (3), Q_{gn} is a binary variable. If the machine initialization preparation is necessary for PM-n of Machine Group-g, then $Q_{gn} = 1$; otherwise, $Q_{gn} = 0$. Accordingly, SC_{gn} denotes the time required for machine initialization preparation of PM-n belonging to Machine Group-g. Besides, $\sum_{a=g}^{N_1} T_{gna}$ is the sum of total working time for PM-n of Machine Group-g to process jobs with different pattern precision requirements. The rest part $\sum_{a=g}^{N_1-1} \sum_{l=a+1}^{N_1} (Z_{gnal} \times S_{gal})$ gives the time spent on switching recipes. Among them, Z_{gnal} is a binary variable and if there is a need to switch the recipe from achieving Pattern Precision-a to Pattern Precision-l at PM-n of Machine Group-g, then $Z_{gnal} = 1$; otherwise, $Z_{gnal} = 0$. Additionally, S_{gal} represents the time required for recipe switching from achieving Pattern Precision-a to Pattern Precision-l at any PM of Machine Group-g.

Before presenting the mathematical programming model, we give the following assumptions: 1) the processing capability of each PM is known in advance; 2) both the reticle and recipe required by each job are known in advance; 3) the number of raw wafers to be processed in a shift is deterministic in advance; 4) there is no cancellation of jobs; 5) the processing activity cannot be interrupted once it has started; and 6) once a PM starts to process a job, it cannot process others until this job is completed.

B. Mathematical Programming Model

The established MILP model contains three types of constraints: 1) job assignment constraints (to determine the values of X_{gnau} and H_{gna}); 2) recipe switching constraints (to determine the value of Z_{gnal}); and 3) machine initialization preparation constraints (to determine the value of Q_{gn}). Constraints are shown below.

1) Job assignment constraints

Each job should be assigned to one PM only, which can be ensured by Constraint (4) shown as follows. Additionally, to guarantee that each PM can handle the jobs assigned to it, Constraint (5) is necessary. Constraint (5) makes sure that if a job requires Pattern Precision-a, it cannot be assigned to any PM belonging to Machine Group-g, where $g > a$. Then, the value of H_{gna} can be determined by Constraints (6) and (7). Notice that, B is a large enough integer.

$$\sum_{g=1}^{N_3} \sum_{n=1}^{N_4} X_{gnau} = 1,\ \forall\ a \in \boldsymbol{A},\ u \in \boldsymbol{\beta}_a, \quad (4)$$

$$X_{gnau} = 0,\ \forall\ g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g,\ a \in \boldsymbol{A},\ u \in \boldsymbol{\beta}_a,\ g > a, \quad (5)$$

$$\sum_{u\in\boldsymbol{\beta}_a} X_{gnau} \geq 1 - H_{gna},\ \forall\ g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g,\ a \in \boldsymbol{A}, \quad (6)$$

$$\sum_{u\in\boldsymbol{\beta}_a} X_{gnau} \leq B \times (1 - H_{gna}),\ \forall\ g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g,\ a \in \boldsymbol{A}, \quad (7)$$

2) Recipe switching constraints

Constraints (8) – (18) are added into the established MILP by considering the recipe switching situations at each PM.

$$Z_{gnal} \leq \sum_{u=1}^{N_2} X_{gnlu},\ \forall\ g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g,\ l \in \boldsymbol{A}, \quad (8)$$

$$Z_{gnal} \leq \sum_{u=1}^{N_2} X_{gnau},\ \forall\ g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g,\ a \in \boldsymbol{A}, \quad (9)$$

$$\sum_{l=1}^{a} Z_{gnal} = 0,\ \forall g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g,\ a \in \boldsymbol{A}, \quad (10)$$

$$\sum_{l=a+1}^{N_1} Z_{gnal} \leq 1,\ \forall\ g \in \boldsymbol{G},\ n \in \boldsymbol{G}_g,\ a \in \{1, \ldots, N_1 - 1\}, \quad (11)$$

$$\sum_{a=g}^{N_1-1} \sum_{l=a+1}^{N_1} Z_{gnal} \leq N_1 - g,\ \forall\ g \in \{1, \ldots, N_3 - 1\},\ n \in \boldsymbol{G}_g, \quad (12)$$

By Constraints (8) and (9), the prerequisite of a recipe switching from achieving Precision-a to Precision-l is that both Precision-a and Precision-l are required by jobs, *i.e.*, if $Z_{gnal} = 1$, then $X_{gnau} = 1$ and $X_{gnlu} = 1$ hold. Moreover, Constraint (10) ensures that the recipe switching is performed from achieving a higher pattern precision to a lower one, thereby guaranteeing that the processing sequence of jobs follows a descend order with respect to the pattern precision.

Constraints (11) and (12) ensure that PMs can start to process a job requiring a different pattern precision after they have finished all jobs requiring the current pattern precision. To do so, Constraint (11) limits the number of times for the recipe switching from achieving one pattern precision to a lower pattern precision being one at most. Additionally, Constraint (12) ensures that the total times of recipe switching processes at each PM of Machine Group-g should not exceed $N_1 - g$. To illustrate the purpose of Constraint (12), an example is adopted in which $N_1 = 5$ and $g = 3$. In this example, the pattern precisions handled by PMs of Machine Group-3 are: Pattern Precision-3, Pattern Precision-4, and Pattern Precision-5. According to Constraint (12), the maximum number of times for recipe switching processes should not exceed $N_1 - g = 2$. Given that the pattern precision handled by a PM starts from the highest one (*i.e.*, Pattern Precision-3), the recipe switching process with the maximum number of times should be: Pattern Precision-3 $\rightarrow$ Pattern Precision-4 $\rightarrow$ Pattern Precision-5. It can be observed that the times for recipe switching equals two. Therefore, it can be concluded that the recipe switching times for other recipe switching sequences in this example should be smaller than two. However, if the recipe switching is required from achieving Pattern Precision-3 to Pattern Precision-5 directly, Constraints (8) – (12) are not enough to guarantee this. Thus, the following constraints are added into the established

MILP.

$$Y_{gna} \geq X_{gnau}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, a \in \boldsymbol{A}, u \in \boldsymbol{\beta}_a, \quad (13)$$

$$Y_{gna} \leq \sum_{u \in \boldsymbol{\beta}_a} X_{gnau}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, a \in \boldsymbol{A}, \quad (14)$$

$$1 - ((\sum_{q=a}^{l} Y_{gnq}) - Y_{gnl} - Y_{gna}) \times B \leq V_{gnal}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, a, l \in \boldsymbol{A}, \quad (15)$$

$$V_{gnal} \leq (1 - Y_{gnq}) \times B, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, a, q, l \in \boldsymbol{A}, a < q < l, \quad (16)$$

$$Z_{gnal} \geq 1 - ((\sum_{q=a}^{l} Y_{gnq})) - Y_{gnl} - Y_{gna}) \times B - (1 - Y_{gnl}) \times B - (1 - Y_{gna}) \times B, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, a, l \in \boldsymbol{A}, a < l, \quad (17)$$

$$Z_{gnal} \leq ((\sum_{q=a}^{l} Y_{gnq}) - Y_{gnl} - Y_{gna}) \times S + (1 - Y_{gnl}) \times B + (1 - Y_{gna}) \times B + V_{gnal} \times B, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, a, l \in \boldsymbol{A}, a < l, \quad (18)$$

Y_{gna} is a binary variable. Further, Y_{gna} = 1 if there exists at least one job requiring Pattern Precision-a processed at PM-n of Machine Group-g; otherwise, Y_{gna} = 0. The value of Y_{gna} is also determined according to the value of X_{gna} by Constraints (13) and (14). Notice that, the value of Y_{gna} is opposite to the value of H_{gna}. Also, a binary variable V_{gnal} is introduced. It indicates that if the recipe switching from achieving Pattern Precision-a to Pattern Precision-l directly, then V_{gnal} = 1; otherwise, V_{gnal} = 0. Then, by Constraints (17) and (18), the value of Z_{gnal} can be determined.

3) Machine initialization preparation constraints

As described, if the first recipe of the current shift differs from the last one of the previous shift, a machine initialization preparation is needed. Since recipes are employed to achieve different pattern precisions, the need for machine initialization preparation can be determined by comparing the pattern precisions required by the related jobs.

For PM-n of Machine Group-g, D_{gn} denotes the pattern precision required by the first job to be processed in the current shift, while d_{gn} denotes the pattern precision required by the last processed job of the previous shift. Notice that, d_{gn} is known in advance. Then, to determine the value of D_{gn}, Constraints (19) – (23) are proposed.

$$W_{gnl} \leq \sum_{a=1}^{l} Y_{gna} \times B, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, l \in \boldsymbol{A}, \quad (19)$$

$$W_{gnl} \geq \sum_{a=1}^{l} Y_{gna} \times S, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, l \in \boldsymbol{A}, \quad (20)$$

$$\sum_{l=1}^{N_1} W_{gnl} \geq 1 - F_{gn}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (21)$$

$$\sum_{l=1}^{N_1} W_{gnl} \leq B \times (1 - F_{gn}), \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (22)$$

$$D_{gn} = (N_1 - \sum_{l=1}^{N_1} W_{gnl} + 1) - (N_1 + 1 - d_{gn}) \times F_{gn}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (23)$$

Among these constraints, $\sum_{a=1}^{l} Y_{gna}$ calculates the total number of pattern precisions, ranging from Pattern Precision-1 to Pattern Precision-l, required by jobs at PM-n of Machine Group-g. Furthermore, a binary variable W_{gnl} is introduced. If $\sum_{a=1}^{l} Y_{gna} \geq 1$, then W_{gnl} = 1; otherwise, W_{gnl} = 0. Moreover, if W_{gnl} = 1, then W_{gnq} = 1, where l, $q \in A$ and $l \leq q$. Therefore, if $\sum_{l=1}^{N_1} W_{gnl} \geq 1$, it implies that there is at least one job assigned to be processed at PM-n of Machine Group-g. Conversely, if $\sum_{l=1}^{N_1} W_{gnl} = 0$, indicating that there is no job assigned to be processed at PM-n of Machine Group-g, then D_{gn} should be equal to d_{gn}. To ensure this, an additional binary variable F_{gn} is adopted such that if $\sum_{l=1}^{N_1} W_{gnl} \geq 1$, then F_{gn} = 0; otherwise, F_{gn} = 1. Finally, the pattern precision required by the first job to be processed in the current shift at each PM can be determined by Constraint (23).

Finally, the value of Q_{gn} is determined based on the values of D_{gn} and d_{gn} as shown in Constraints (24) – (28). Notice that, a and b are two binary variables, and their sum should be equal to one as shown in Constraint (24).

$$A + b = 1, \quad (24)$$

$$a \times B + (d_{gn} - D_{gn}) \geq Q_{gn}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (25)$$

$$b \times B + (D_{gn} - d_{gn}) \geq Q_{gn}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (26)$$

$$Q_{gn} \geq (d_{gn} - D_{gn}) \times S, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (27)$$

$$Q_{gn} \geq (D_{gn} - d_{gn}) \times S, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (28)$$

Further, the maximum and minimum working time of PMs are denoted as T_{MAX} and T_{MIN}, respectively.

$$T_{gn} \geq 0, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (29)$$

$$T_{MAX} \geq T_{gn}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (30)$$

$$T_{MIN} \leq T_{gn}, \forall\, g \in \boldsymbol{G}, n \in \boldsymbol{G}_g, \quad (31)$$

Constraint (29) ensures that the total working time of each PM should be greater than or equal zero. Besides, Constraints (30) and (31) illustrate that the total working time of each PM should be within the range [T_{MIN}, T_{MAX}]. The objective of the addressed problem is to minimize the difference between T_{MAX} and T_{MIN} such that the workloads at PMs can be balanced. Then, an MILP model can be established as follows:

$$\textbf{MILP}: \text{Minimize } (T_{MAX} - T_{MIN}) \quad (32)$$

Subject to: (4) – (31)

IV. Procedure of Proposed Approach

As mentioned above, the addressed problem can be formulated as an MILP. However, the computational time required for solving an MILP increases exponentially as the problem size grows [21]. Thus, exact solutions are suitable for small-sized problems. For larger-sized problems, it is advisable to apply metaheuristic algorithms, which are problem-scale independent and use intelligent ways to guide the search for high-quality solutions [9, 22–24].

To solve the addressed scheduling problem, the EDA introduced in [25] is adopted. EDAs are a class of stochastic optimization techniques. Different from traditional evolutionary algorithms (such as GA) that rely on crossover and mutation to explore the solution space, EDAs take a sampling approach based on a probability model to generate high-qualities solutions. This unique searching strategy enables EDA to capture the relationship between different variables by updating the probability model at each iteration. In this work, a *designed EDA* (DEDA) is proposed, and its main procedures are: 1) individual encoding and population initialization; 2) fitness calculation; 3) probability model initialization, update, and creation of new population; and 4) greedy local improvement.

A. Individual Encoding and Population Initialization

This work applies integer coding for individual representation. An example is used to illustrate the applied coding approach. Assume that there are three machine groups and each one contains three PMs. Each PM is assigned a specific Arabic number, starting

with a PM in Machine Group-1 as one and ending with a PM in Machine Group-3 as nine. By doing so, PMs are numbered with different Arabic numbers from one to nine. Similarly, this numbering approach is also applied to jobs. Each job is assigned a specific Arabic number, starting with the job requiring Pattern Precision-1 and using Reticle-1 as one. Subsequently, the job requiring Pattern Precision-1 and using Reticle-2 is numbered as two. Jobs continue to be numbered in this way until all jobs requiring Pattern Precision-1 are numbered. Then, jobs requiring Pattern Precision-2 can start to be numbered.

EDAs rely on a population of γ individuals to search for high-quality solutions. In this work, each individual is denoted as Π_x, $x \in \mathbb{N}_\gamma = \{1, 2, \ldots, \gamma\}$. Let $|\bullet|$ denote the number of elements in a set or the size of a list. $\Pi_x = [\pi_{x\_1}, \pi_{x\_2}, \ldots, \pi_{x\_|\boldsymbol{J}|}]$, where $\boldsymbol{J}$ is the set of jobs. Here, π_{x_j} represents the PM that Job-j is assigned to, $j \in \boldsymbol{J}$. Then, the population of γ individuals is denoted by $\Theta_\gamma = \{\Pi_x | x \in \mathbb{N}_\gamma\}$. The schematic diagram of a population is shown in Fig. 2. It can be found that the size of an individual is determined by the number of jobs.

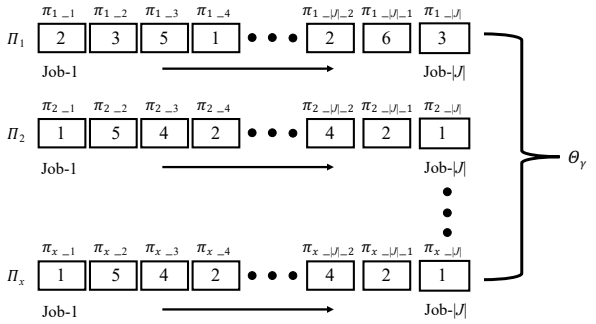


Fig. 2. Schematic diagram of population.

Undoubtedly, the number of jobs far exceeds the number of PMs in real-world production. In this work, individuals (*i.e.*, job assignments) are randomly generated initially. This may result in some PMs having no jobs to process. When such a situation arises, the generated individual is not a high-quality one due to the unbalanced workloads among PMs. Moreover, it is well known that having a diverse and high-quality initial population is beneficial for evolutionary search [26]. In many existing studies, the initial population for solving scheduling problems is improved by initialization strategies or using simple heuristics. Thus, in this work, Algorithm 1 is proposed to improve the quality of a randomly generated initial population denoted as Θ_γ.

The main idea of Algorithm 1 is to assign a job to a PM if it currently has no jobs to process. This idea is applied to each individual of the initial population, as shown in Statement 1). By Statement 2), two sets $\boldsymbol{\vartheta}$ and $\boldsymbol{\delta}$ are used to record the numbers representing PMs and jobs, respectively. Initially, both $\boldsymbol{\vartheta}$ and $\boldsymbol{\delta}$ are all empty. By Statements 3) – 5), the number representing each PM that has at least one job to process is recorded in $\boldsymbol{\vartheta}$. Notice that, $\Pi_{x,j} = \pi_{x\_j}$, $j \in \boldsymbol{J}$. Also, elements in a set should be unique. Therefore, if the number of Arabic numbers recorded in $\boldsymbol{\vartheta}$ is less than the total number of all PMs, then at least one PM has no jobs to process. Notice that, the numbers representing PMs are all summarized in the set $\boldsymbol{M}$, where $m \in \boldsymbol{M}$.

If Statement 6) holds, Statements 7) and 8) are used to identify the PM that has no jobs to process. In Statement 8), $\boldsymbol{JM}_m$ is a set made up of the numbers representing the jobs assigned to be processed at the PM-m. If the number of elements in $\boldsymbol{JM}_m$ equals zero, then PM-m is examined as the one with no jobs to process. When such a situation occurs, all numbers representing the jobs that can be moved to be processed at the PM-m will be recorded in $\boldsymbol{\delta}$. This is achieved by Statements 9) – 12). For Job-j, the numbers representing the PMs that can process it are summarized in the set $\boldsymbol{MJ}_j$, $\boldsymbol{MJ}_j \subseteq \boldsymbol{M}$. However, before recording the number representing a job in $\boldsymbol{\delta}$, it should ensure that this job is not the only one at its current PM. To do so, Statement 11) is necessary. If there are numbers in $\boldsymbol{\delta}$ (*i.e.*, if Statement 13) holds), a number is randomly selected from $\boldsymbol{\delta}$ and then the job represented by this number is assigned to PM-m by modifying the individual coding. This is achieved by Statements 14) and 15). Notice that, Randomint(1, $|\boldsymbol{\delta}|$)) is a function used to randomly select an integer within the range [1, $|\boldsymbol{\delta}|$].

Algorithm 1: Initial population improvement

Input: Θ_γ,
Output: Θ_γ
1) For $x \leftarrow 1$ to γ
2) $\quad \boldsymbol{\vartheta} = \varnothing$, $\boldsymbol{\delta} = \varnothing$;
3) $\quad$ For $j \leftarrow 1$ to $|\boldsymbol{J}|$
4) $\quad\quad$ If $\Pi_{x,j}$ not in $\boldsymbol{\vartheta}$
5) $\quad\quad\quad \boldsymbol{\vartheta} \cup \{\Pi_{x,j}\}$;
6) $\quad\quad$ If $|\boldsymbol{\vartheta}| < |\boldsymbol{M}|$
7) $\quad\quad\quad$ For $m \leftarrow 1$ to $|\boldsymbol{M}|$
8) $\quad\quad\quad\quad$ If $|\boldsymbol{JM}_m| = 0$
9) $\quad\quad\quad\quad\quad$ For $j \leftarrow 1$ to $|\boldsymbol{J}|$
10) $\quad\quad\quad\quad\quad\quad$ If m in $\boldsymbol{MJ}_j$
11) $\quad\quad\quad\quad\quad\quad\quad$ If $|\boldsymbol{JM}_{\Pi_{x,j}}| > 1$
12) $\quad\quad\quad\quad\quad\quad\quad\quad \boldsymbol{\delta} = \boldsymbol{\delta} \cup \{j\}$;
13) $\quad\quad\quad\quad\quad$ If $|\boldsymbol{\delta}| > 0$
14) $\quad\quad\quad\quad\quad\quad z =$ Randomint(1, $|\boldsymbol{\delta}|$);
15) $\quad\quad\quad\quad\quad\quad$ *New*-$\Pi_{x,\boldsymbol{\delta}_z} = m$;

B. Fitness Calculation

To evaluate the quality of an individual, Algorithm 2 is proposed to calculate its fitness value. Statements 1) – 2) aim to identify the numbers representing the jobs assigned to be processed at each PM. For Job-j of Individual-x, the number representing its assigned PM is $\Pi_{x,j}$. Next, the number representing Job-j is recorded in the set $\boldsymbol{JM}_{\Pi_{x,j}}$. Subsequently, the processing sequence of jobs at each PM should be determined. By Statements 1) – 2), the initial processing sequence of jobs at PM-m is represented by $\boldsymbol{E}_m$. Notice that, $\boldsymbol{E}_m$ is a sorting list made up of job numbers from small to large. Besides, $\Lambda_{\boldsymbol{E}_{m,j}}$ and $\Delta_{\boldsymbol{E}_{m,j}}$ respectively denote the required pattern precision and reticles of the j-th Job at PM-m, $j \in \{1, \ldots, |\boldsymbol{E}_m|\}$. Statements 6) – 7) are proposed to ensure that jobs requiring a higher pattern precision are given higher processing priority, thereby ensuring the processing sequence of wafers is a descending order with respect

to the pattern precision. Notice that, Equations 8) – 18) of the MILP model are addressed by Statements 3) – 7).

After determining the processing sequences of jobs at all PMs, the total working time of PM-m denoted as T_m can be calculated by Statements 8) – 17). Statements 9) – 10) are used to calculate the total processing time of all jobs processed at PM-m, where $pc(\boldsymbol{E}_{m,j})$ denotes the processing time of the j-th processed Job at PM-m. Statements 11) – 12) are used to calculate the time spent on machine initialization preparation, where lw_m represents the pattern precision required by the last processed job of the previous shift and $\Lambda_{\boldsymbol{E}_{m,1}}$ denotes the pattern precision required by the first job of the current shift. If lw_m and $\Lambda_{\boldsymbol{E}_{m,1}}$ are different, then there exists a machine initialization preparation at PM-m and the time needed for such a preparation is denoted as MS_m. Statements 9), 13), and 14) – 17) are used to calculate the total time spent on recipe switching or reticle switching. When a PM starts to process a different job, reticle switching is always necessary. At this time, whether there is a recipe switching requirement should be further determined by Statement 14). If Statement 14) holds, it indicates that both reticle switching and recipe switching are required and the total time needed for them is governed by the time taken for the recipe switching represented by $SW_{\Lambda_{\boldsymbol{E}_{m,j}},\Lambda_{\boldsymbol{E}_{m,z}}}$. However, If Statement 14) is not true, then there exists the reticle switching only and the time needed for the reticle switching is represented by $ST_{\Delta_{\boldsymbol{E}_{m,j}},\Delta_{\boldsymbol{E}_{m,z}}}$. 2). Equations 19) – 28) of the proposed MILP model are addressed by Statements 8) – 17).

After Statements 8) – 17) are done, Statements 18) and 19) can identify the PMs with the longest and shortest working time, respectively, which corresponds to Equations 29) – 31) of the proposed MILP model. Finally, the fitness value of Individual-x denoted as F_x is given by Statement 20).

Algorithm 2: Fitness calculation

Input: Π_x
Output: F_x

1) For $j \leftarrow 1$ to $|\boldsymbol{J}|$
2) $\quad \boldsymbol{JM}_{\boldsymbol{\Pi}_{x,j}} = \boldsymbol{JM}_{\boldsymbol{\Pi}_{x,j}} \cup \{i\}$;
3) For $m \leftarrow 1$ to $|\boldsymbol{M}|$
4) $\quad$ For $j \leftarrow 1$ to $|\boldsymbol{E}_m|$
5) $\quad\quad$ For $z \leftarrow j+1$ to $|\boldsymbol{E}_m|$
6) $\quad\quad\quad$ If $\Lambda_{\boldsymbol{E}_{m,j}} > \Lambda_{\boldsymbol{E}_{m,z}}$
7) $\quad\quad\quad\quad$ $\boldsymbol{E}_{m,j}, \boldsymbol{E}_{m,z} = \boldsymbol{E}_{m,z}, \boldsymbol{E}_{m,j}$;
8) For $m \leftarrow 1$ to $|\boldsymbol{M}|$
9) $\quad$ For $j \leftarrow 1$ to $|\boldsymbol{E}_m|$
10) $\quad\quad$ $T_m = T_m + pc(\boldsymbol{E}_{m,j})$;
11) $\quad\quad$ If $lw_m \neq \Lambda_{\boldsymbol{E}_{m,1}}$
12) $\quad\quad\quad$ $T_m = T_m + MS_m$;
13) $\quad\quad$ For $z \leftarrow j+1$ to $|\boldsymbol{E}_m|$
14) $\quad\quad\quad$ If $\Lambda_{\boldsymbol{E}_{m,j}} \neq \Lambda_{\boldsymbol{E}_{m,z}}$
15) $\quad\quad\quad\quad$ $T_m = T_m + SW_{\Lambda_{\boldsymbol{E}_{m,j}},\Lambda_{\boldsymbol{E}_{m,z}}}$;
16) $\quad\quad\quad$ Else
17) $\quad\quad\quad\quad$ $T_m = T_m + ST_{\Delta_{\boldsymbol{E}_{m,j}},\Delta_{\boldsymbol{E}_{m,z}}}$;
18) $T_{max} = \max(T_m, m \in \boldsymbol{M})$;
19) $T_{min} = \min(T_m, m \in \boldsymbol{M})$;
20) $F_x = T_{max} - T_{min}$;

C. Probability Model Initialization, Update, and Creation of New Population

EDA describes population evolution trends by a probabilistic model. The representation of a probability model is described as a $|\boldsymbol{J}| \times |\boldsymbol{M}|$ matrix denoted by

$$p(X) = \begin{bmatrix} p(X_{1,1}) & p(X_{1,2}) & \cdots & p(X_{1,|\boldsymbol{M}|}) \\ p(X_{2,1}) & p(X_{2,2}) & \cdots & p(X_{2,|\boldsymbol{M}|}) \\ \vdots & \vdots & \vdots & \vdots \\ p(X_{|\boldsymbol{J}|,1}) & p(X_{|\boldsymbol{J}|,2}) & \cdots & p(X_{|\boldsymbol{J}|,|\boldsymbol{M}|}) \end{bmatrix},$$

where $\boldsymbol{J}$ is the set of jobs and $\boldsymbol{M}$ is the set of PMs. Then, $p(X_{j,m})$ indicates the probability of Job-j assigned to be processed at PM-m, $j \in \boldsymbol{J}$ and $m \in \boldsymbol{M}$. Notice that, if $m \in \boldsymbol{M} - \boldsymbol{MJ}_j$, then $p(X_{j,m})$ is marked as "None". Here, "None" is used to indicate that such a probability does not exist. Initially, the probability model is set to a uniform distribution state. To achieve this, $p(X_{j,m}) = p(X_{j,m'}) = \frac{1}{|\boldsymbol{MJ}_j|}$, where $j \in \boldsymbol{J}$, $m, m' \in \boldsymbol{MJ}_j$.

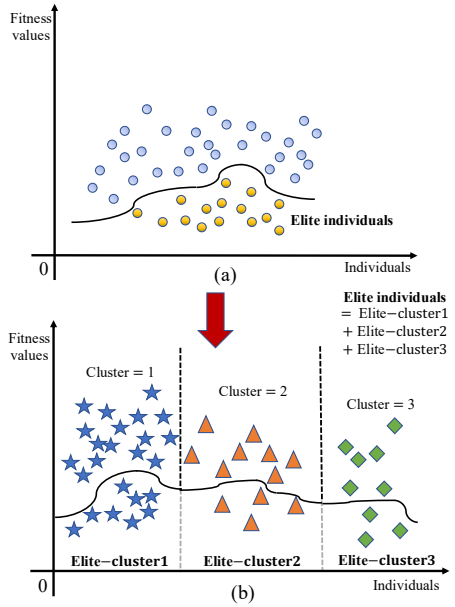


Fig. 3. Two approaches to select elite individuals.

After the initialization of the probability model, the next is to select a subset of candidate individuals (also known as elite individuals) to update the probability model. Here, we examine two different selection approaches, illustrated by using a same case shown in Fig. 3 (a) and (b). In general, elite individuals are selected based on their fitness values in most of the existing studies. To exemplify this method, Fig. 3 (a) shows that the top 1/3 individuals with the lowest (best) fitness values are chosen to be elite individuals. Although elite individuals have high qualities, there is a possibility of them being excessively similar. Here, "similar" refers to that two different elite individuals have a significant portion of identical elements. There is no doubt that such a similarity can result in a rapid increase in the

probability of certain elements within the probability model, potentially causing the EDA to be easily trapped in local optima. To avoid this, the second approach is considered by dividing the population into multiple clusters. Furthermore, a proportional number of individuals from each cluster are selected as elite individuals. As shown in Fig. 3 (b), the number of clusters is three. For each cluster, a proportion of 1/3 individuals with the best fitness values are selected such that the total number of elite individuals matches the one in Fig. 3 (a). By doing so, the diversity of the elite individuals in Fig. 3 (b) can be higher than that in Fig. 3 (a).

To divide a population into multiple clusters, a *Manhattan distance-based Kmeans* (MD-Kmeans) clustering approach is adopted as shown in Algorithm 3. The effectiveness of Kmeans clustering has been verified in [27]. Let N_5 denote the number of clusters and $\boldsymbol{C}_k$, $k \in \{1, \ldots, N_5\}$, denote Cluster-k. Initially, all clusters are set as empty sets by Statement 1). Then, there are three steps to group γ individuals into N_5 clusters.

1) Step 1: Manhattan distance calculation between any two different individuals

The *Manhattan distance* (MD) calculation is achieved by Statements 2) − 6). As shown in Statement 5), the following equation is necessary.

$$md_{xx'} = \sum_{j=1}^{|\boldsymbol{J}|} \left| \Pi_{x,j} - \Pi_{x',j} \right|, \forall\, x, x' \in \mathbb{N}_\gamma = \{1, 2, \ldots, \gamma\}. \quad (33)$$

In equation (33), md is a $\gamma \times \gamma$ zero matrix used to record the MD between any two individuals for a population of γ individuals. Accordingly, $md_{xx'}$ is used to represent the MD of Individuals x and x'. Notice that, $md_{xx'} = md_{x'x}$.

2) Step 2: The initial centroid selection for each cluster from the dataset

By Step 2, individuals from the population are selected. Let μ_k, $k \in \{1, \ldots, N_5\}$, be the centroid of Cluster-k. The initial centroids are given by selecting random individuals from the population as shown in Statement 7).

3) Step 3: The centroids updating

Step 3 is done by Statements 9) – 20). Notice that, the third step is repeatedly performed by a loop procedure. If the flag is less than N_5, then the loop continues. By Statements 11) – 14), each individual is included in a cluster (set) whose centroid has the minimum MD to it. Subsequently, Statements 15), 17), and 18) are used to obtain a new centroid for each cluster. To do so, the following equation is needed.

$$\mu_{k,f} = \sum_{x=1}^{|\boldsymbol{C}_k|} (\Pi_{x,f}) / |\boldsymbol{C}_k|, f \in \{1, \ldots, |\mu_k|\}, k \in \{1, \ldots, N_5\}. \quad (34)$$

In equation (34), the f-th element of μ_k is obtained by the mean of all corresponding elements of individuals assigned to Cluster-k. If a centroid does not move (*i.e.*, the new centroid is equal to the current one), then the flag is added by one. This is achieved by Statements 16) and 19). If all centroids do not move (*i.e.*, when the flag is equal N_5), then the loop ends.

Algorithm 3: MD-Kmeans clustering approach

Input: N_5, γ
Output: $\boldsymbol{C}_k$, $k \in \{1, \ldots, N_5\}$

```
1)  Set C_k = ∅, k ∈ {1, …, N5};
2)  md = 0_{γ×γ};
3)  For i ← 1 to γ
4)      For j ← i + 1 to γ
5)          md_{i,j} = Σ_{z=1}^{|J|} | Π_{i,z} − Π_{j,z} |;
6)          md_{j,i} = Md_{i,j};
7)  Randomly select N5 individuals as μ_k, k ∈ {1, …, N5};
8)  flag = 0;
9)  While flag < N5
10)     flag = 0;
11)     For z ← 1 to γ
12)         For h ← 1 to N5
13)             If md_{z,h} = min(md_{z,k}, k ∈ {1, …, N5})
14)                 C_h = C_h ∪ {Π_z};
15)     For k ← 1 to N5
16)         ρ_k = μ_k;
17)         For f ← 1 to |μ_k|
18)             μ_{k,f} = Σ_{x=1}^{|C_k|} (Π_{x,f}) / |C_k|;
19)         If μ_k = ρ_k
20)             flag = flag + 1;
```

After the selection of elite individuals, Algorithm 4 is employed to update the probability model. Let $\boldsymbol{ep}$ denote the set made up of elite individuals and the average fitness value of elite individuals is denoted as $Ave\_f = \sum_{i=1}^{|\boldsymbol{ep}|} F_{\boldsymbol{ep}_i} / |\boldsymbol{ep}|$, where $F_{\boldsymbol{ep}_i}$ represents the fitness value of Individual-i of $\boldsymbol{ep}$ obtained by Algorithm 2. Besides, two $|\boldsymbol{J}| \times |\boldsymbol{M}|$ zero matrixes B and W are used to record the times of jobs assigned to PMs and the sum of fitness values of individuals, respectively. Notice that, B and W can guide the update of the probability model.

Statements 4) – 8) are used to update the elements in both B and W. As shown in Statement 6), z is the number representing the PM that Job-j of Elite Individual-i is assigned to. Then, the times for Job-j assigned to be processed at PM-m denoted by $B_{j,z}$ should be added by one. Additionally, the fitness value of Elite Individual-i is added to $W_{j,z}$, where $W_{j,z}$ represents the sum of fitness values of individuals in which Job-j is assigned to PM-z. Subsequently, the probability model is updated by B and W as shown in Statements 9) – 18). Notice that, the updated probability model keeps part of the historical experience from the last probability model by a learning rate denoted as $\alpha = iter / Maxiter$. As *iter* and *Maxiter* respectively represent the current iteration number and the maximum number of iterations, the updates to the probability model rely more on historical experience with iteration proceeds. If $W_{i,j} < Ave\_f \times B_{i,j}$, which means that Job-$j$ assigned to PM-m has a positive impact on the fitness values, then $P(X_{i,j})$ can be increased by λ, where λ is a value randomly generated within the range $[0, P(X_{i,j}))$ by the function Random$(0, P(X_{i,j}))$. Conversely, if $W_{i,j} > Ave\_f \times B_{i,j}$, then $P(X_{i,j})$ should be decreased by λ.

Algorithm 4: Probability model update

Input: $\boldsymbol{ep}$, $p(X)$, α
Output: $p(X)$

1) $Ave\_f = \sum_{i=1}^{|\boldsymbol{ep}|} F_{\boldsymbol{ep}_i} / |\boldsymbol{ep}|$;
2) $B = 0_{|\boldsymbol{J}|\times|\boldsymbol{M}|}$;
3) $W = 0_{|\boldsymbol{J}|\times|\boldsymbol{M}|}$;
4) For $i \leftarrow 1$ to $|\boldsymbol{ep}|$
5) For $j \leftarrow 1$ to $|\boldsymbol{J}|$
6) $z = ep_{i,j}$;
7) $B_{j,z} = B_{j,z} + 1$;
8) $W_{j,z} = W_{j,z} + F_{\boldsymbol{ep}_i}$;
9) For $i \leftarrow 1$ to $|\boldsymbol{J}|$
10) For $j \leftarrow 1$ to $|\boldsymbol{M}|$
11) If $B_{i,j} > 0$
12) $\lambda = \text{Random}(0, P(X_{i,j}))$;
13) If $W_{i,j} < Ave\_f \times B_{i,j}$
14) $P(X_{i,j}) = \alpha \times P(X_{i,j}) + (1-\alpha) \times (B_{i,j} / |\boldsymbol{ep}|) + \lambda$;
15) Elif $W_{i,j} > Ave\_f \times B_{i,j}$
16) $P(X_{i,j}) = \alpha \times P(X_{i,j}) + (1-\alpha) \times (B_{i,j} / |\boldsymbol{ep}|) - \lambda$;
17) Else
18) $P(X_{i,j}) = \alpha \times P(X_{i,j}) + (1-\alpha) \times (B_{i,j} / |\boldsymbol{ep}|)$;

After updating the probability model, a new population is created by applying the *roulette wheel selection* (RWS). In this work, RWS aims to select a PM for each job according to the probability model. To ensure a proper application of RWS, the total probability of all selectable parts equals one. However, it may occur that $\sum_{m=1}^{|\boldsymbol{M}|} p(X_{j,m}), j \in \boldsymbol{J}$ does not equal one after the probability model updates. To address this situation, the normalization processing is performed. By doing so, the probability of PM-m that will be selected for processing Job-j is equal to $p(X_{j,m}) / \sum_{m=1}^{|\boldsymbol{M}|} p(X_{j,m})$.

D. Greedy Local Improvement

The objective of the addressed problem is to minimize the difference between the longest and shortest working time of PMs. With such an objective, the improvement for an individual can be quickly achieved by adjusting job assignments for the PMs with the longest and shortest working time. To do so, reassigning jobs from the PM with the longest working time to others or reassigning jobs from other PMs to the one with the shortest working time may work. Inspired by this thought, Algorithm 5 introduces a greedy local improvement approach. Different from the initial population improvement (Algorithm 1) that aims to make each PM just have jobs to process, the greedy local improvement employs a greedy strategy. This greedy strategy tries every feasible job reassignment for the PMs with the longest and shortest working time to find the one that provides the highest fitness value improvement until no further improvement can be achieved.

Notice that, the greedy local improvement is repeatedly performed by a loop procedure to improve the quality of the given individual Π_x, $x \in \gamma$. The termination condition for this loop is that the flag should be greater than or equal one. At first, the flag is set as zero by Statement 1). Then, there are three main steps to perform greedy local improvement shown as follows.

1) Step 1: Identify the longest and shortest working time PMs

The first step is to identify the PMs with the longest and shortest working time and their representing numbers are denoted as *Max* and *Min*, respectively. This step is accomplished by Statements 3) − 7), where T_m represents the total working time of PM-m, $m \in \boldsymbol{M}$. Moreover, $\mathcal{Gi}$ and $\mathcal{Gf}$ are two empty sets used to record individuals and fitness values, respectively. Then, by Statement 9), the given individual Π_x and its corresponding fitness value F_x are stored in $\mathcal{Gi}$ and $\mathcal{Gf}$, respectively.

2) Step 2: Reassigning jobs to longest working time PM

The second step is to generate new individuals by reassigning a job at the PM with the longest working time to other PMs. This is achieved by Statements 5) − 9). The numbers representing jobs assigned to be processed at the PM with the longest working time are summarized in the set $\boldsymbol{JM}_{Max}$. Then, the numbers representing PMs that Job-j, $j \in \boldsymbol{JM}_{Max}$, can be moved to is summarized in the set $\boldsymbol{MJ}_j - \{Max\}$. The greedy nature is demonstrated by reassigning Job-j to each PM whose number is in $\boldsymbol{MJ}_j - \{Max\}$. Then, $|\boldsymbol{JM}_{Max}| \times |\boldsymbol{MJ}_j - \{Max\}|$ new individuals and their corresponding fitness values are stored in $\mathcal{Gi}$ and $\mathcal{Gf}$, respectively.

3) Step 3: Reassigning jobs to shortest working time PM

Similarly, the third step is to generate new individuals by reassigning a job to the PM with the shortest working time from other PMs. This is achieved by Statements 15) − 19). Notice that, $\boldsymbol{J} - \boldsymbol{JM}_{Min}$ is a set made up of all jobs except the ones that are already at the PM with the shortest working time. By Statements 15) − 16), whether a job can be reassigned to the PM with the shortest working time is determined by checking if the number representing this PM is in the set of numbers representing PMs capable of processing this job. Similarly, if Statement 16) holds, a newly generated individual and its corresponding fitness value are then stored in $\mathcal{Gi}$ and $\mathcal{Gf}$, respectively.

After performing the above-mentioned three steps, if the individual with the best fitness values recorded in $\mathcal{Gi}$ is different from Π_x, it then replaces Π_x and the loop continues; otherwise, the flag is set as one such that the loop ends. Notice that, all new individuals stored in $\mathcal{Gi}$ have only one element different from Π_x. Thus, in each loop iteration, at most one job reassignment is executed.

Algorithm 5: Greedy local improvement

Input: Π_x,
Output: Π_x
1) *flag* = 0;
2) While *flag* < 1
3) For $m \leftarrow 1$ to $|\boldsymbol{M}|$
4) If $T_m = \max(T_k, k \in \{1, \ldots, |\boldsymbol{M}|\})$
5) $Max = m$;
6) If $T_m = \min(T_k, k \in \{1, \ldots, |\boldsymbol{M}|\})$
7) $Min = m$;
8) $\mathcal{Gi} = \varnothing$, $\mathcal{Gf} = \varnothing$;

```
9)      𝒴𝒾 = 𝒴𝒾 ∪ {Π_x}, 𝒴𝒻 = 𝒴𝒻 ∪ {F_x};
10)     For j ← 1 to |JM_Max|
11)       For m ← 1 to |MJ_j − {Max}|
12)         New-Π_x = Π_x;
13)         New-Π_{x,JM_Max_j} = MJ_{j_m};
14)         𝒴𝒾 = 𝒴𝒾 ∪ {New-Π_x}, 𝒴𝒻 = 𝒴𝒻 ∪ {New-F_x};
15)     For j ← 1 to |J − JM_Min|
16)       If Min in MJ_j
17)         New-Π_x = Π_x;
18)         New-Π_{x,j} = Min;
19)         𝒴𝒾 = 𝒴𝒾 ∪ {New-Π_x}, 𝒴𝒻 = 𝒴𝒻 ∪ {New-F_x};
20)     For z ← 1 to |𝒴𝒾|
21)       If F_z = min(F_k, k ∈ {1, …, |𝒴𝒾|})
22)         New-Π_x = Π_z;
23)         If New-Π_x = Π_x
24)           flag = 1;
25)         Else
26)           Π_x = New-Π_x;
```

E. Proposed DEDA

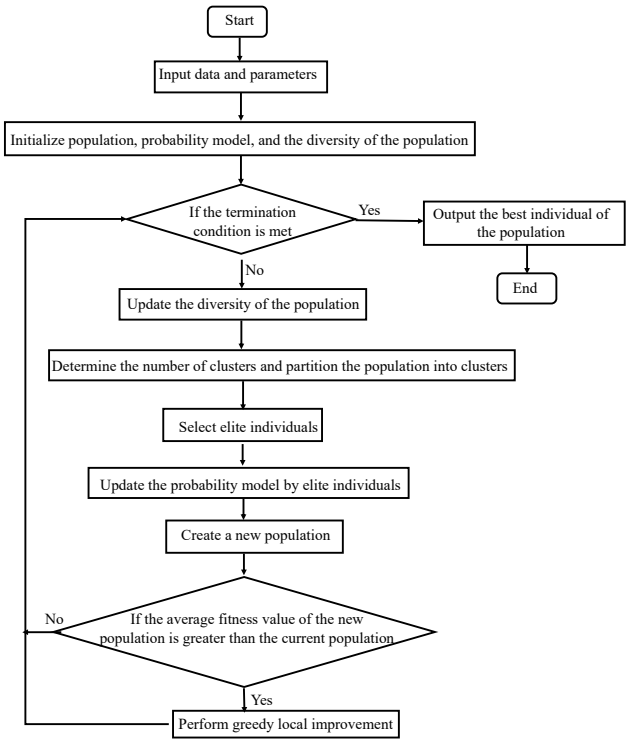


Fig. 4. Flowchart of DEDA

The flowchart of DEDA is shown in Fig. 4. In DEDA, if the population diversity at the current iteration decreases by comparing with the last iteration, the number of clusters is added by one. Conversely, if the diversity increases, the number of clusters is reduced by one. To prevent the number of clusters from becoming excessively large or too small, the lower and upper bounds of the number of clusters are set as two and five, respectively. Moreover, the initial number of clusters is an integer randomly generated within the range [2, 5].

Due to the greedy nature, Algorithm 5 undoubtedly requires high computational costs. As a result, the greedy local improvement is only performed if the average fitness value of the newly generated population becomes worse. When such a situation arises, the number of individuals randomly selected for the greedy local improvement is determined by the following equation.

$$Ng_i = \lceil (\text{ave}\_f_i - \text{ave}\_f_{(i-1)}) / \text{ave}\_f_i) \times \gamma \rceil,\ i \in \{2, \ldots, Max\_iter\}, \quad (35)$$

where ave_f_i denotes the average fitness value of the i-th Iteration and γ is the population size. With the ceiling function $\lceil \bullet \rceil$, the number of selected individuals of the i-th Iteration represented by Ng_i is ensured to be an integer. Moreover, selected individuals should be different.

V. Experiments

A. Data set

The proposed approach is tested on instances from the investigated wafer fab. For the small-sized problem, there are 42 jobs made up of 5230 raw wafers for processing at three different machine groups that have three, two, and three PMs, respectively. For the medium-sized problem, there are 72 jobs made up of 12775 raw wafers for processing at five different machine groups that have three, two, three, two, and four PMs, respectively. For the large-sized problem, there are 120 jobs made up of 20060 raw wafers for processing at ten different machine groups that have three, two, three, two, four, five, three, two, three, and three PMs, respectively. The details of these three problems can be found in Tables B – D in Supplementary File. In Table B, explanations for each data are provided, and these explanations are applicable to Tables C and D as well.

B. Parameter Setting and Experiment Design

All experiments are conducted on a personal computer equipped with 8GB of RAM and an Apple M2 processor. All algorithms are implemented by using Python 3.8 and the MILP model is solved by CPLEX 12.9.

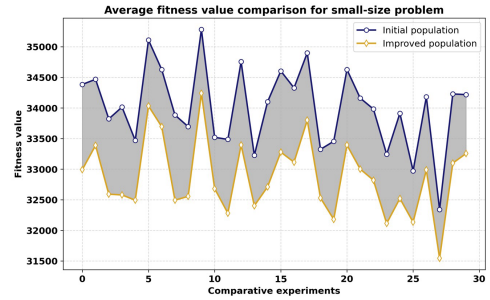
It is well known that suitable parameter settings can result in a good performance of metaheuristic algorithms. To find the most suitable combination of parameters, a statistical method called the Taguchi method proposed by Genichi Taguchi is applied in this work and the process is given in Supplementary File. As a result, the best parameter settings for DEDA are: population size = 100, iterations for DEDA = 200, and percentage of population selected as elite individuals = 25%.

After completing the parameter settings of DEDA, its effectiveness and efficiency in solving the addressed problem should be verified. To do so, a series of comparison experiments are conducted which can be divided into three parts: 1) demonstrate the effectiveness of Algorithm 1 (initial population improvement); 2) examine the efficiency of Algorithm 5 (greedy local improvement); and 3) compare the best fitness values achieved by DEDA with those obtained by other popular metaheuristics. To obtain statistically significant results, each comparison experiment is repeated for 30 times.

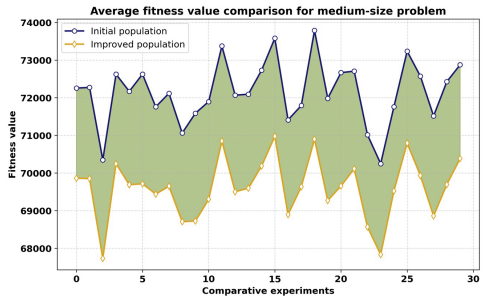
C. Experimental Results and Analysis

1) Initial population improvement evaluation

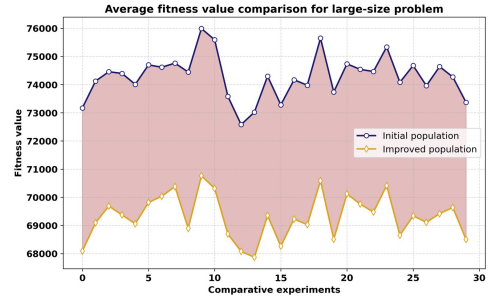
To evaluate the effectiveness of Algorithm 1, comparison experiments are conducted by comparing the average fitness of a randomly generated population with the average fitness of its improved version after applying Algorithm 1. Then, a total of 180 experiments are done. The experimental results for three different cases are summarized in Fig. 5. As shown in Fig. 5 (a), (b), and (c), Algorithm 1 is highly effective for all problem sizes. In each experiment, Algorithm 1 consistently improves the average quality of the initial population. Additionally, it is noteworthy that as the problem size increases, the improvement effect of Algorithm 1 on the initial population becomes more significant.



(a) Small-size case



(b) Medium-size case

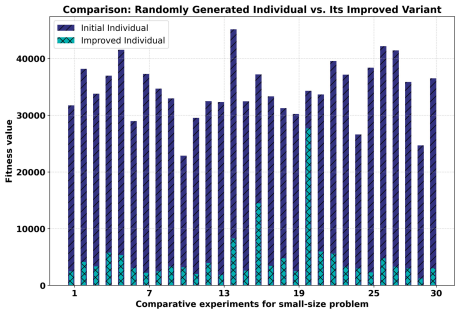


(c) Large-size case

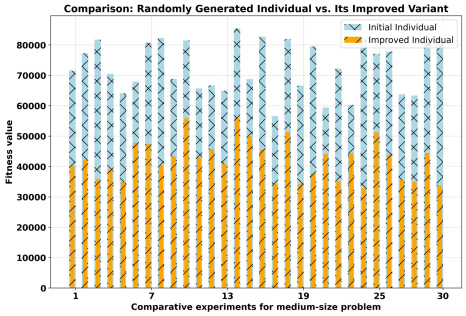
Fig. 5. Comparative experiment results for three cases.

2) Greedy local improvement evaluation

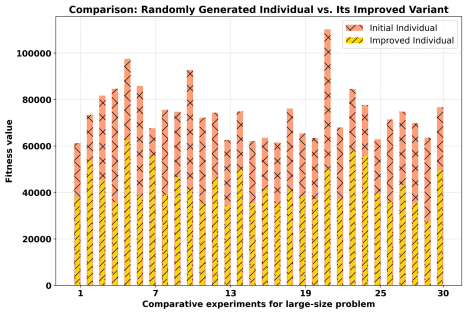
To evaluate the effectiveness of Algorithm 5, comparison experiments are conducted by comparing the fitness value of a randomly generated individual with the fitness value of its improved version by applying Algorithm 5. Also, 180 experiments are needed in total. Experimental results for three different cases are summarized in Fig. 6 (a), (b), and (c), showing that local improvement performs well across all problem sizes. In this work, the greedy local improvement is considered as a single-solution metaheuristic with a termination condition if it falls into a local optimum.



(a) Small-size case



(b) Medium-size case



(c) Large-size case

Fig. 6. Comparative experiment results for three cases.

3) Performance evaluation of DEDA

For the last part of the comparison experiments, we conduct two types of comparisons: 1) comparisons between CPLEX and DEDA using the small-sized case and 2) comparisons between the DEDA and three other metaheuristics for medium-sized and large-sized cases. Totally, 300 experiments are done.

For the small-sized case, an optimal solution can be obtained by solving the established MILP through CPLEX. Results collected from 30 times of independent experiments are summarized in Table I. The average running time (147.891 seconds) of DEDA is three times more than the average running time (33.706 seconds) of CPLEX. Notice that, the minimum fitness value obtained by DEDA is the same as the one given by CPLEX. Although DEDA achieves the optimal value (545) only once, the average fitness value of 30 times experiments provided by it is 566.53 which is quite acceptable in practical applications. To address practical production problems efficiently, it is often advisable to obtain a good solution quickly rather than pursuing the optimal solution with a long computation time [28]. Typically, CPLEX is programmed to stop running if the elapsed time exceeds 3600 seconds [9].

Under this situation, if CPLEX cannot output the optimal solution within 3600 seconds, then the addressed problem is treated as unsolvable by CPLEX in this work. Unfortunately, CPLEX cannot solve medium- or large-sized problems.

TABLE I
EXPERIMENTAL RESULTS FOR SMALL-SIZE CASE

	CPLEX	DEDA
Maximum fitness value (s)		736
Minimum fitness value (s)		545
Average fitness value (s)	545	566.53
Maximum running time (s)	36.608	164.377
Minimum running time (s)	31.250	129.168
Average running time (s)	33.706	147.891

To evaluate the efficiency of DEDA for solving medium- and large-sized problems, the comparison experiments are carried out between DEDA and other three metaheuristic algorithms, including GA, *Grey Wolf Optimizer* (GWO), and MixPso. As demonstrated by the outcomes of the second part of the comparison experiments, the greedy local improvement has a significantly positive impact on the search for high-quality solutions. Thus, the greedy local improvement is also adopted in these three compared algorithms for fair comparisons. Besides, it is essential to guarantee that the running time of each metaheuristic algorithm should be approximately equal. The running time for each case is determined by DEDA which serves as the termination criterion for the other three algorithms. For medium- and large-sized problems, the running times for all these three compared metaheuristics are set as 199.345 and 2685.679 seconds, respectively.

The experimental results for medium- and large-sized collected from 30 times of independent experiments are summarized in Table H in Supplementary File, where k denotes the k-th experiment. It can be found from Table H that DEDA outperforms the other three metaheuristics for both medium- and large-sized problems. Especially, for the large-sized problem, the worst fitness value obtained by DEDA is superior to the best fitness values provided by the other three compared metaheuristics. Furthermore, the box plots corresponding to Table H are drawn in Fig. 7 (a) and (b). Boxplot is a visual graph that is efficient in evaluating the performance of algorithms. As shown in Fig. 7, each box in a box plot represents the data distribution of one specific group within the dataset. *i.e.*, the performance of one algorithm. For each box, there are three lines which are upper, median, and lower lines used to represent the maximum, median, and minimum values of a group, respectively. Moreover, the height of a box shows the data stability of a group. As shown in both Fig. 7 (a) and (b), it is obvious that DEDA outperforms the other three compared metaheuristics in terms of both the best fitness values obtained and the algorithmic stability. The boxplots in Fig. 7 demonstrates DEDA's stability, with smaller and more compact boxes (the yellow and purple boxplots) compared to other algorithms, indicating a concentrated distribution of results. DEDA also exhibits very few outliers, confirming its consistency and minimal deviation from the main distribution. Furthermore, the median (the thick horizontal line) is near the center of the DEDA's box, suggesting that the results are symmetrically distributed around the median. Among the compared metaheuristics, GA exhibits better performance than MixPSO and GWO, especially for the large-sized problem.

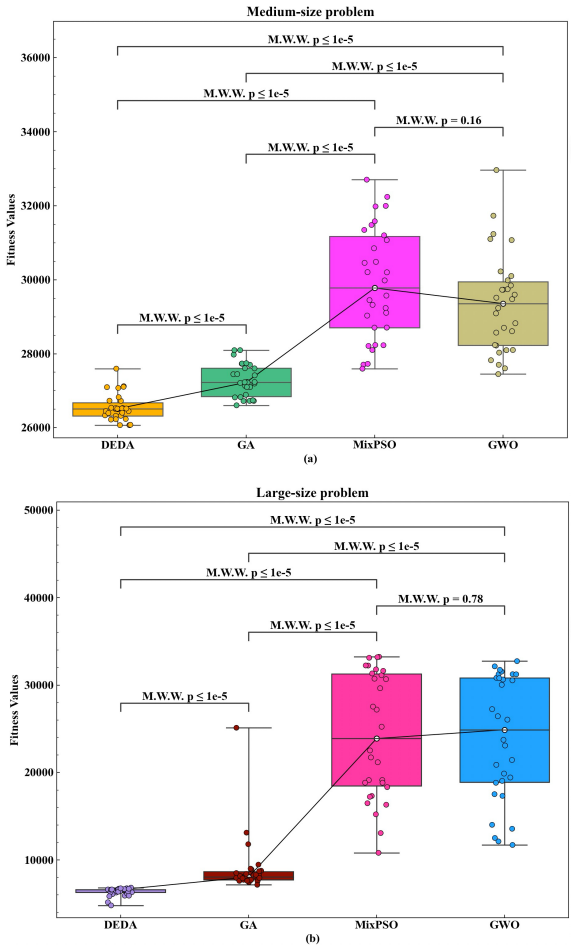


Fig. 7. Box plots for medium and large sizes problems.

To enhance the reliability of the observed performance differences among these algorithms, significance tests are employed. The method applied to perform significance tests is the Mann-Whitney U-test, a non-parametric statistical method. It compares the medians of two independent samples, making it suitable for situations under which the data does not follow a normal distribution. In statistical analysis, a smaller P-value (P-value $\leq 1e^{-5}$) usually indicates greater statistical significance, making the observed differences more reliable. The outcomes of Mann-Whitney U-test are also shown in Fig. 7. The findings consistently reveal a substantial difference (P-value $\leq 1e^{-5}$) between DEDA and each of the other three compared algorithms regarding their average fitness values. These outcomes strongly indicate the superior effectiveness of DEDA. By executing the computational experiments, it is suggested that DEDA is the preferred solution for addressing the scheduling problem in this work.

VI. Conclusion

This work aims at solving the scheduling problem of the photolithography area in semiconductor manufacturing. The objective of the addressed problem is to minimize the difference between the longest and shortest working time of PMs. We first develop an MILP model that involves job assignment and machine setup constraints. Such an MILP model is used to obtain the optimal solution for the small-sized problem. To overcome the computational complexity of addressing medium- and large-sized problems, a designed DEDA integrated with Kmeans clustering is constructed. The key of DEDA to generate high-quality solutions lies in a probability model, which is updated by elite individuals at each iteration. To select these elite individuals, an MD-Kmeans clustering approach is applied. Additionally, efficient approaches are proposed to improve the quality of the initial population and a given individual.

To verify the effectiveness of the proposed approaches, comparison experiments are divided into three parts and carried out for different size problems. The first and second parts of comparison experiments demonstrate the effectiveness of the proposed algorithms in improving the quality of the initial population and a given individual. The third part of comparison experiments shows that the solution quality obtained by DEDA is better than other metaheuristic algorithms (*i.e.*, GA, MixPSO, and GWO) regardless of problem sizes. Besides, the algorithmic stability of DEDA is the best among these four metaheuristic algorithms.

This work is conducted based on the fact that there is no tool failure or cancellation of jobs. Our future work can take uncertain arrival time of jobs and abnormal events into account for the scheduling problems of photolithography areas in semiconductor manufacturing.

References

[1] E. Akcalt, K. Nemoto, and R. Uzsoy, "Cycle-time improvements for photolithography process in semiconductor manufacturing," *IEEE Trans. Semicond. Manuf.*, vol. 14, no. 1, pp. 48-56, 2001.

[2] P. Zhang, Y. Lv, and J. Zhang, "An improved imperialist competitive algorithm based rolling horizon strategy for photolithography machines scheduling," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1295-1300, 2016.

[3] D. Y. Sha, S. Y. Hsu, Z. H. Che, and C. H. Chen, "A dispatching rule for photolithography scheduling with an on-line rework strategy," *Comput. Ind. Eng.*, vol. 50, no. 3, pp. 233-247, 2006.

[4] B. H. Zhou, X. Li, and R. Y. Fung, "Dynamic scheduling of photolithography process based on Kohonen neural network," *J. Intell. Manuf.*, vol. 26, pp. 73-85, 2015.

[5] L. Mönch, J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose, "A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations," *J. Sched*., vol. 14, pp. 583-599, 2011.

[6] H. Chen, P. Guo, J. Jimenez, Z. S. Dong, and W. Cheng, "Unrelated parallel machine photolithography scheduling problem with dual resource constraints," *IEEE Trans. Semicond. Manuf.*, vol. 36, no. 1, pp. 100-112, 2022.

[7] S. L. M. De Díaz, J. W. Fowler, M. E. Pfund, G. T. Mackulak, and M. Hickie, "Evaluating the impacts of reticle requirements in semiconductor wafer fabrication," *IEEE Trans. Semicond. Manuf*., vol. 18, no. 4, pp. 622-632, 2005.

[8] E. Cakici and S. J. Mason, "Parallel machine scheduling subject to auxiliary resource constraints," *Prod. Plann. Control*, vol. 18, no. 3, pp. 217-225, 2007.

[9] L. Chen, S. Zhang, N. Wu, Y. Qiao, Z. Zhong, and T. Chen, "Optimization of inventory space in smart factory for integrated periodic production and delivery scheduling," *IEEE Trans. Comput. Soc. Syst*., 2022.

[10] D. Doleschal, G. Weigert, A. Klemmt, and F. Lehmann, "Advanced secondary resource control in semiconductor lithography areas: From theory to practice," *in Proc. Winter Simul. Conf.* (*WSC*), 2013, pp. 3879-3890.

[11] Y. F. Lee, Z. B. Jiang, and H. R. Liu, "Multiple-objective scheduling and real-time dispatching for the semiconductor manufacturing system," *Comput. Oper. Res*., vol. 36, no. 3, pp. 866-884, 2009.

[12] A. Bitar, S. Dauzère-Pérès, C. Yugma, and R. Roussel, "A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing," *J. Sched.*, vol. 19, no. 4, pp. 367-376, 2016.

[13] T. Y. Hong, C. F. Chien, H. K. Wang, and H. Z. Guo, "A two-phase decoding genetic algorithm for TFT-LCD array photolithography stage scheduling problem with constrained waiting time," *Comput. Ind. Eng.*, vol. 125, pp. 200-211, 2018.

[14] P. Zhang, X. Zhao, X. Sheng, and J. Zhang, "An imperialist competitive algorithm incorporating remaining cycle time prediction for photolithography machines scheduling," *IEEE Access*, vol. 6, pp. 66787-66797, 2018.

[15] C. E. N. Bastos and L. C. Resendo, "Two-step approach for scheduling jobs to non-related parallel machines with sequence dependent setup times applying job splitting," *Comput. Ind. Eng.*, vol. 145, 106500, 2020.

[16] X. Li, Y. Huang, Q. Tan, and H. Chen, "Scheduling unrelated parallel batch processing machines with non-identical job sizes," *Comput. Oper. Res.*, vol. 40, no. 12, pp. 2983-2990, 2013.

[17] R. Gedik, D. Kalathia, G. Egilmez, and E. Kirac, "A constraint programming approach for solving unrelated parallel machine scheduling problem," *Comput. Ind. Eng.*, vol. 121, pp. 139-149, 2018.

[18] M. Afzalirad and J. Rezaeian, "Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions," *Comput. Ind. Eng.*, vol. 98, pp. 40-52, 2016.

[19] J. E. C. Arroyo and J. Y. T. Leung, "An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times," *Comput. Ind. Eng.*, vol. 105, pp. 84-100, 2017.

[20] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67-82, 1997.

[21] M. A. Bragin, "Survey on Lagrangian relaxation for MILP: importance, challenges, historical review, recent advancements, and opportunities," *Ann. Oper. Res.*, vol. 333, no. 1, pp. 29-45, 2024.

[22] Y. Qiao, S. Zhang, N. Wu, X. Wang, Z. Li, M. Zhou, and T. Qu, "Data-driven approach to optimal control of ACC systems and layout design in large rooms with thermal comfort consideration by using PSO," *J. Clean. Prod*., vol. 236, 117578, 2019.

[23] Y. Qiao, N. Wu, Y. He, Z. Li, and T. Chen, "Adaptive genetic algorithm for two-stage hybrid flow-shop scheduling with sequence-independent setup time and no-interruption requirement," *Expert Syst. Appl.*, vol. 208, 118068, 2022.

[24] Y. I. Kim and H. J. Kim, "Rescheduling of unrelated parallel machines with job-dependent setup times under forecasted machine breakdown," *Int. J. Prod. Res*., vol. 59, no. 17, pp. 5236-5258, 2020

[25] H. Mühlenbein and G. Paass, "From recombination of genes to the estimation of distributions I. Binary parameters," *in Parallel Problem Solving from Nature*, 1996, pp. 178-187.

[26] Y. Pan, K. Gao, Z. Li and N. Wu, "A Novel Evolutionary Algorithm for Scheduling Distributed No-Wait Flow Shop Problems," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 54, no. 6, pp. 3694-3704, June 2024

[27] W. Xiong, Y. Qiao, L. Bai, M. Ghahramani, N. Wu, P. Hsieh, and B. Liu, "Wafer reflectance prediction for complex etching process based on K-means clustering and neural network," *IEEE Trans. Semicond. Manuf.*, vol. 34, no. 2, pp. 207-216, 2021.

[28] H. Zhang, S. Yao, S. Zhang, J. Leng, L. Wei, and Q. Liu, "A block-based heuristic search algorithm for the two-dimensional guillotine strip packing problem," *Eng. Appl. Artif. Intell.*, vol. 134, no. 108624, 2024.

LiangChao Chen received two B.S. degrees in Mechanical Design Manufacture and Automation, and English from East China Jiaotong University, Nanchang, China, in 2019, respectively. In 2022, he received the M.S. degree in Intelligent Technology from Macau University of Science and Technology (MUST), Macao. He is currently pursuing the Ph.D degree in Intelligent Science and Systems with the Institute of Systems Engineering and Collaborative Laboratory for Intelligent Science and Systems, Macau University of Science and Technology, Macao. His current interests include scheduling and optimization, reinforcement learning, and smart manufacturing systems.

Yan Qiao (Senior Member, IEEE) received the B.S. and Ph.D. degrees in industrial engineering and mechanical engineering from Guangdong University of Technology, Guangzhou, China, in 2009 and 2015, respectively. From September 2014 to September 2015, he was a Visiting Student with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. From January 2016 to December 2017, he was a Post-Doctoral Research Associate with the Institute of Systems Engineering, Macau University of Science and Technology, Macau. He is currently an Associate Professor with the Institute of Systems Engineering and the Department of Engineering Science, Faculty of Innovation Engineering, Macau University of Science and Technology. He has over 100 publications, including one book chapter and more than 50 regular articles in IEEE Transactions. Besides, he was a recipient of the QSI Best Application Paper Award Finalist of the 2011 IEEE International Conference on Automation Science and Engineering, the Best Student Paper Award from the 2012 IEEE International Conference on Networking, Sensing and Control, the Best Conference Paper Award Finalist of the 2016 IEEE International Conference on Automation Science and Engineering, the Best Student Paper Award Finalist of the 2020 IEEE International Conference on Automation Science and Engineering, and the 2021 Hsue-Shen Tsien Paper Award from IEEE/CAA Journal of Automatica Sinica. He is an Associate Editor of IEEE Robotics and Automation Magazine.

NaiQi Wu (Fellow, IEEE) received the B.S. degree in electrical engineering from Anhui University of Science and Technology, Huainan, China, in 1982, and the M.S. and Ph.D. degrees in systems engineering from Xi'an Jiaotong University, Xi'an, China, in 1985 and 1988, respectively. From 1988 to 1995, he was with Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, China. From 1995 to 1998, he was with Shantou University, Shantou, China. He moved to Guangdong University of Technology, Guangzhou, China, in 1998. He joined Macau University of Science and Technology, Taipa, Macau, in 2013. He was a Visiting Professor with Arizona State University, Tempe, AZ, USA, in 1999; New Jersey Institute of Technology, Newark, NJ, USA, in 2004; the University of Technology of Troyes, Troyes, France, from 2007 to 2009; and Evry University, Evry, France, from 2010 to 2011. He is currently a Chair Professor with the Institute of Systems Engineering and the Department of Engineering Science, Faculty of Innovation Engineering, Macau University of Science and Technology. He is the author or co-author of one book, five book chapters, and more than 250 journal articles. His research interests include production planning and scheduling, manufacturing system modeling and control, discrete event systems, Petri net theory and applications, intelligent transportation systems, and energy systems. He was an Associate Editor of IEEE Transactions on Systems, Man, and Cybernetics, Part C; IEEE Transactions on Automation Science and Engineering; and IEEE Transactions on Systems, Man, and Cybernetics: Systems. He was the Editor-in-Chief of Industrial Engineering Journal. He is an Associate Editor of Information Sciences.

Mohammadhossein Ghahramani obtained his B.S. and M.S. degrees in Information Technology Engineering from Amirkabir University of Technology-Tehran Polytechnic. He earned his Ph.D. in Computer Technology and Application from Macau University of Science and Technology in 2018. He was a member of the Insight Centre for Data Analytics and a Research Fellow at University College Dublin, Ireland. Currently, he is an Assistant Professor of Data Science at Birmingham City University, UK. His research interests include smart systems, artificial intelligence, optimization, smart cities, and IoT. Dr Ghahramani has published over ten peer-reviewed journal papers as the first author in reputable journals and has received several awards, including the Best Automation Paper in Technology by the IEEE Robotics and Automation Society. He serves as a co-chair of the IEEE SMCS Technical Committee on AI-based Smart Manufacturing Systems and as an Associate Editor of IEEE Internet of Things Journal.

YongHua Shao received the M.S. degree in Chemical Engineering from East China University of Science and Technology, Shanghai, China. He is currently the General Manager of the Wafer Fab in AscenPower Semiconductors. Previously, he served as a process engineer in the Thin Films Department at TSMC (Shanghai), where he was responsible for the acceptance of new equipment, the introduction of new processes, and production ramp-up. He then held positions as the Process

Section Head in both the Thin Films and Diffusion Departments at Advanced Semiconductor Manufacturing in Shanghai, where he focused on capacity expansion, yield improvement, and cost control. He led the development of key processes including the COOLMOS superjunction process and backside hydrogen implantation process for IGBTs. Later, he served as a senior engineer and head of the Thin Films Department at Tampines Factory of GlobalFoundries (Singapore), where he focused on establishing thin film process systems and product introduction. During his tenure at GTA Semiconductor, he was a key figure in establishing the first silicon carbide production line in China, successfully developing critical SiC processes such as high-temperature gate oxide, high-temperature implantation, and high-temperature activation.

Sijun Zhan received his B.S. degree in Process Equipment and Control Engineering from Yangtze University, Jingzhou, China. He is currently the Head of the Manufacturing Department at AscenPower Semiconductors. Previously, he worked as a Manufacturing Engineer at Semiconductor Manufacturing International Corporation, where he was responsible for the construction of automation systems, equipment layout planning, and production capacity evaluation and ramp-up. He then served as a Senior Production Control Engineer at FAB7 of GlobalFoundries (Singapore), focusing on capacity expansion, production planning, and cycle time improvement projects. He led the development and optimization of ThinFilm/Etch automatic setups, RTD automatic testing, and dispatch priority settings, as well as the implementation of the FCOD VDUMMY automatic gatepass system. He also worked as a Principal Engineer at Yangtze Memory Technologies, where he was in charge of new factory layout planning, capacity planning, equipment qualification schedules, and full automation system releases. During his tenure at IKAS Industries, he served as the key CIM lead for MES, EAP, FDC, and RTD software modules, acting as a consultant and project manager (PMP certified) during system deployment and implementation.