Computer Modeling in Engineering & Sciences

Tech Science Press

https://doi.org/10.32604/cmes.2026.076245

ARTICLE

Modeling and Optimization of Diffusion Process Scheduling under Strict Queue Time Constraints in Semiconductor Manufacturing

Liangchao Chen[1], Yan Qiao[1,*], Siwei Zhang[1,*], Bin Liu[2], Yonghua Shao[3] and Sijun Zhan[3]

[1]Institute of Systems Engineering and Collaborative Laboratory for Intelligent Science and Systems, Macau University of Science and Technology, Taipa, Macao, China
[2]IKAS Holdings (Beijing) Co., Ltd., Beijing, China
[3]AscenPower Semiconductors Co., Ltd., Guangzhou, China

*Corresponding Authors: Yan Qiao. Email: yqiao@must.edu.mo; Siwei Zhang. Email: swzhang@must.edu.mo

Received: 17 November 2025; Accepted: 06 March 2026; Published: 27 April 2026

ABSTRACT: This article examines wafer lots scheduling in the diffusion area in semiconductor manufacturing. The diffusion area comprises multiple tool groups. Each of them contains non-identical semiconductor tools. All tools can process multiple wafer lots simultaneously, and wafer lots processed together in a tool are called a wafer batch. Besides, each wafer lot has specific *queue time limits* (QTLs) between consecutive processing operations, making the scheduling problem more complicated. To solve it, a *discrete backtracking search optimization algorithm* (DBSA) is designed for optimizing both wafer lot assignments and wafer batch processing sequences. Once the processing sequence of wafer batches at each tool is determined, a *linear program* (LP) is built to obtain optimal starting and completion time points of wafer batches while satisfying QTLs. If a schedule is examined to have no feasible solution by the LP, a proposed approach is used to regroup wafer lots to form wafer batches and adjust their processing sequences to potentially make it feasible. Extensive experiments show that DBSA reliably produces feasible schedules and outperforms GA, MixPSO, and GWO, with up to 17.75%, 19.19%, and 9.21% reductions in average cycle time, respectively, demonstrating its superiority in both solution quality and practical applicability.

KEYWORDS: Metaheuristic algorithms; queue time limits; scheduling; semiconductor manufacturing

1 Introduction

In semiconductor manufacturing, the diffusion process consists of two processing steps: 1) at the first step, wet bench cleaning tools (also called wet benches in short) are used to clean the wafer surfaces with chemical liquids; 2) at the second step, furnaces serve as the heat treatment systems for wafer fabrication, commonly applied for annealing, oxidation, and deposition processes. Both wet benches and furnaces are divided into several tool groups, with each group containing tools that may differ in equipment type. However, each tool group includes at least one recipe that allows wafers to be processed by any tool within that group. In this work, a *wet bench group* (WBG) is defined as a tool group consisting of wet benches, while a *furnace group* (FG) is defined as a tool group consisting of furnaces. Besides, wet benches and furnaces are *batch-processing tools* (BPTs).

This work is motivated by real production demands from semiconductor fabs. In real semiconductor manufacturing scenarios, wafers are processed and delivered in lots. Each lot typically contains 25 wafers. For the diffusion process, A and B are used to represent processing steps 1 and 2, respectively. In the investigated

Copyright © 2026 The Authors. Published by Tech Science Press. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

fab, there are two types of processing flows: 1) A → B; and 2) A → B → B. For the former, wafer lots are processed in sequence through a WBG and an FG. For the latter, wafer lots are processed in sequence through a WBG and two different FGs, i.e., visiting Step 2 twice for processing. Notice that all wafer lots being processed in an FG should come from the same WBG. As a result, the diffusion process can be divided into several subsystems, with each comprising a WBG and several FGs and being called a *one-WBG and multiple-FG* (OWB-MF) system. This work focuses on the scheduling problem of such a sub-system.

In an OWB-MF system, multiple wafer lots processed together in a tool are referred to as a wafer batch. Moreover, there is no wafer separation for all wafer lots in the investigated system. Hence, a wafer lot and a wafer batch can be treated as a job and a job batch, respectively. Although the jobs in a job batch can be processed in a tool simultaneously, they may not be concurrently processed in a tool at the downstream processing steps due to different recipes at the downstream steps. Furthermore, during a switching process of a tool between two job batches with different recipes, there should be a setup time that is independent of the processing sequence of job batches. Besides, there is a strict time window known as the *queue time limit* (QTL) between two consecutive processing operations for each job batch in an OWB-MF system. For a single job batch, different jobs may have different QTLs. Failure to meet a QTL can result in defects on the wafer surfaces (e.g., oxidation or contamination), which require re-cleaning operations that significantly reduce the system's productivity. In severe cases, wafers effectively become unusable and must be scrapped, incurring substantial economic losses. Such setup time and QTLs make the scheduling problem of an OWB-MF system complicated. Therefore, it is vital to find a schedule for all jobs to be processed in an OWB-MF system while meeting their QTLs.

In the context of Industry 4.0, our work aligns with the quality planning phase of the *zero defect manufacturing* (ZDM) framework. As emphasized by Psarommatis and Azamfirei [1], ZDM requires a dedicated plan stage to specify operational processes that meet quality objectives. Recent studies have attempted to address strict time constraints using advanced intelligent algorithms. For instance, May et al. [2] employ recurrent neural networks to predict time constraint violations for production control in semiconductor manufacturing, while Park and Huh [3] develop a GA with a three-phase decoding heuristic to minimize QTL violations. However, a critical gap remains regarding the strict feasibility guarantee required for high-precision baseline planning. While prediction models like [2] provide risk assessments, they do not generate deterministic schedules. Similarly, optimization methods (such as [3]) often treat strict time windows as optimization objectives (soft constraints) rather than hard constraints, which means the resulting schedules may still contain violations.

While the aforementioned studies focus on handling time constraints, the extensive body of literature on furnace scheduling has primarily emphasized operational efficiency. This is because the processing time of a furnace normally needs 6 h at least and 14 h at most. It is significantly longer than that of a wet bench, which ranges from 20 to 60 min. It is essential to make a proper and reasonable schedule for furnaces. In [4,5], the scheduling problems of a single furnace are addressed. Specifically, a greedy heuristic method based on some dispatching rules is proposed. Also, six variants of the simulated annealing algorithm and six variants of the tabu search algorithm are presented. Additionally, Guo et al. [6] propose a novel method by combining a metaheuristic (ant colony algorithm) and a heuristic (next arrival control method) to schedule a BPT for diffusion operations. As multiple furnaces are used in the diffusion areas in practice, furnaces need to be jointly scheduled. Great efforts have been made to address scheduling problems of multiple furnaces in [7–10]. With the assumption that the multiple furnaces are identical, metaheuristic algorithms are designed to tackle their scheduling problems in [7,8]. For the scheduling problems of multiple furnaces with the same or similar processing functionalities, a mathematical model and a dynamic programming model are proposed in [9]. Furthermore, a *genetic algorithm* (GA) is constructed to give a general solution framework

based on practical applications. Also, in [10], a due-date-based heuristic algorithm is presented to address the scheduling problems of nonidentical furnaces.

To improve the productivity of the diffusion areas, it is necessary to find efficient and effective scheduling methods considering both wet benches and furnaces. To do so, Ham et al. [11] propose a wet scheduler and a furnace scheduler. The furnace scheduler is used to generate a wish-list of jobs processed in furnaces, while the wet scheduler is applied for arranging the processing sequences of jobs in wet benches according to the wish-list. Yugma et al. [12] build a disjunctive graph to represent the batching and scheduling problem of diffusion areas and propose a constructive algorithm to solve it. In [13], a diffusion area is treated as a two-stage flowshop system. To solve its scheduling problem, a batch-oriented and furnace-driven algorithm is developed. Jung et al. [14] establish a mixed-integer linear program and propose a decomposition method to solve the scheduling problem of diffusion areas. In [15], a simulation-based method is presented such that the total cycle time of an investigated factory can be reduced by two days.

Batch scheduling is a common demand in semiconductor fabs [9]. How to properly form batches plays an important role in maximizing the productivity of the system. Yugma et al. [12] partition jobs (wafer lots) into batches according to the batching coefficients that can be calculated by the presented formulas. Rajasekaran et al. [16] propose a method based on the relationship between the batch factor and cycle time to set the optimal batch size for furnaces. Also, Schmidt and Rose [17] carry out a simulation analysis to find out whether the small batch sizes can decrease the cycle time or not. Beyond classical rule-based scheduling heuristics, recent studies have introduced metaheuristic and learning-based approaches for batch-processing environments. For instance, Li and Lei [18] develop a competitive shuffled frog-leaping algorithm that addresses machine eligibility, incompatible job families, and sequence-dependent setups in parallel batch processing machines, showing strong performance in large-scale industrial dyeing applications. Zhang et al. [19] further advance parallel batch scheduling by proposing a clustering-aided multi-agent deep reinforcement learning framework, in which Pareto-optimal solutions obtained offline guide the design of reward functions for online batch forming and scheduling. In contrast, Kong et al. [20] investigate a single batch-processing machine setting and design a deep Q-learning model that learns rule-selection policies to improve energy efficiency. Although these approaches differ in methodology, they share a fundamental structural simplification: batch scheduling is modeled as a single-stage decision problem. None of these works incorporates multi-processing flows, which are inherent in semiconductor diffusion processes.

By considering the practical demands from semiconductor fabs, the addressed scheduling problem in this work differs from the abovementioned studies in terms of three aspects: 1) at each processing step, manufacturing tools are all grouped and the tools within a group may not be identical; 2) there are two types of processing flows, i.e., A $\rightarrow$ B and A $\rightarrow$ B $\rightarrow$ B; and 3) one job batch can be processed in a specific tool group only. Further, once such a job batch is completed in a tool group, it should be dispersed into several jobs again. Then, they are delivered to a downstream step or moved out of the OWB-MF system. If they are delivered to a downstream step, they may be assigned to different tool groups at this step.

However, current approaches remain insufficient for such OWB-MF systems due to several fundamental limitations. First, most existing approaches treat QTLs as soft constraints or prioritize efficiency over strict feasibility, failing to guarantee the defect-free schedules required for high-precision diffusion processes. Second, existing batching strategies are mainly utilization-driven and may lead to infeasible schedules under tight time-window constraints. Third, if a meta-heuristic algorithm is directly applied to solve the scheduling problem, randomly generated solutions during algorithm iteration may be infeasible due to the difficulty of simultaneously satisfying all job QTLs, significantly increasing the time required to search for feasible solutions and making it difficult to ensure solution quality.

In recent years, increasing attention has been devoted to feasibility-oriented and proactive scheduling paradigms in manufacturing systems governed by strong process constraints. For example, Li et al. [21] propose a feasibility-aware adaptive large neighborhood search framework for resource-constrained unrelated parallel machine scheduling, in which infeasible solutions are proactively repaired before performance optimization. In another strongly constrained batch manufacturing domain, Kim et al. [22] investigate the steelmaking–continuous casting scheduling problem with strict waiting-time constraints and batch continuity requirements, and develop a model-driven iterated greedy framework combined with constraint programming to generate strictly feasible baseline schedules. From an industrial scheduling perspective, Geibinger et al. [23] study a real-world industrial test laboratory scheduling problem with heterogeneous resources and tight deadlines, and demonstrate that exact constraint programming combined with very large neighborhood search can efficiently produce strictly feasible and high-quality schedules. Nevertheless, such feasibility-first planning has not yet been systematically investigated for OWB-MF systems with strict QTL constraints in semiconductor diffusion processes. To the best of our knowledge, this study represents a systematic attempt to embed feasibility-oriented planning into the grouping and scheduling stages of OWB-MF systems under strict QTL constraints. The proposed framework can therefore be viewed not merely as a performance-driven metaheuristic, but as a feasibility-oriented proactive scheduling paradigm in which strict QTL compliance fundamentally shapes both modeling and algorithmic design. This positioning distinguishes the present work from purely objective-focused approaches and situates it within the broader context of feasibility-first manufacturing scheduling.

To address these challenges, this article extends our previous work in [24], in which a metaheuristic algorithm is constructed to solve the scheduling problem of the OWB-MF system without considering QTLs of jobs. To address the additional challenges introduced by QTL constraints, this work designs a *Backtracking Search Optimization Algorithm* (BSA), a metaheuristic algorithm, to seek high-quality solutions. To obtain such a solution, we employ the following steps:

1) An encoding is generated to represent the job assignments to manufacturing tools;
2) Jobs assigned to each tool are grouped into job batches by a proposed rule-based strategy since all tools in the OWB-MF system are BPTs;
3) A heuristic local search is constructed to sequence the job batches to be processed in each tool;
4) A *linear program* (LP) is proposed to check if the processing sequences of job batches are feasible, i.e., to see if QTLs are satisfied. If so, the LP determines the best starting time points of each job for each operation from the perspective of total cycle time minimization. If not, a simple modification method is presented to regroup jobs to form job batches and adjust the processing sequences of job batches such that QTLs can be met and the proposed LP is applied again to determine the best starting time points of each job for each operation.

Note that Step 4) serves as a proactive prevention strategy based on the ZDM framework [1]. It ensures the generation of a defect-free production plan before production begins, minimizing the risk of QTL violations and avoiding defects during the planning stage, rather than relying on post-process detection or corrections. It is necessary to distinguish the proposed regrouping strategy from existing scheduling methods for batch processing tools. Extensive studies, such as Guo et al. [6] and Rocholl et al. [7], primarily focus on optimizing efficiency-related objectives, including makespan and tardiness. Similarly, Scholl and Domaschke [15] aim to reduce cycle times. These approaches generally adopt efficiency-driven batching strategies that aim to maximize machine utilization, for example, by forming full batches. In an OWB-MF system with strict QTLs, utilization-oriented batching strategies may not always be appropriate, as they can result in schedules that are difficult to satisfy all QTL requirements. In contrast, the proposed approach places greater emphasis on planning-level feasibility. By giving priority to QTL considerations during the grouping

stage, the proposed method facilitates the construction of feasible schedules and serves as a complementary feasibility-oriented layer to existing efficiency-driven scheduling algorithms.

By Steps 1)–4), a feasible solution can be obtained with a high probability. Since BSA is a population-based algorithm, many solutions can be obtained by Steps 1)–4) in the same way. Then, the encodings are updated based on the operational mechanisms of BSA by introducing several popular operators. The superior performance of the designed BSA is demonstrated by a large number of comparison experiments.

2 Problem Description

Fig. 1 illustrates the layout of an OWB-MF system, where jobs from upstream wafer fabrication processes enter the system and completed jobs are delivered as outputs.

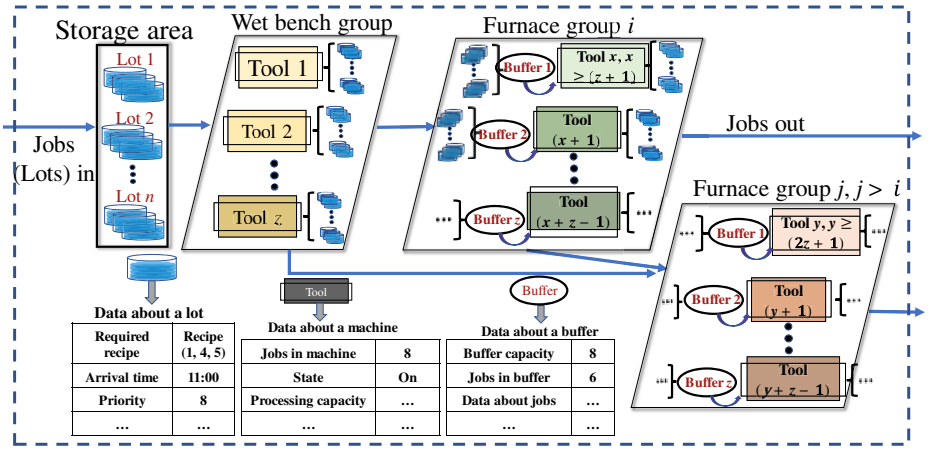


Figure 1: Layout of an OWB-MF system.

To well describe the addressed scheduling problem, some notations are introduced accordingly. Let $|\bullet|$ denote the number of elements in a set. In an OWB-MF system, manufacturing tools are represented by a set $\boldsymbol{M} = \{1, 2, \ldots, |\boldsymbol{M}|\}$, and each tool is associated with a unique number belonging to $\boldsymbol{M}$. As shown in Fig. 1, each tool has a unique number. Suppose that there are n, $n \leq |\boldsymbol{M}|$, tool groups in the OWB-MF system, and each tool should come from a tool group. Let $\boldsymbol{G}_g$, $g \in \{1, \ldots, n\}$, be a set of manufacturing tools representing the g-th tool group. Also, $\boldsymbol{G}_1 \cup \boldsymbol{G}_2, \ldots, \cup \boldsymbol{G}_n = \boldsymbol{M}$ and $\boldsymbol{G}_i \cap \boldsymbol{G}_j = \varnothing$ should hold, $i, j \in \{1, \ldots, n\}$ and $i \neq j$. Note that $\boldsymbol{G}_1$ indicates WBG and $\boldsymbol{G}_n$, $n > 1$, should be an FG. In the OWB-MF system, each operation is served by one or more tool groups. Fig. 1 provides a straightforward example, where WBG serves for Operation 1, FG i serves for Operation 2, and FG j serves for both Operations 2 and 3 of different jobs.

At the current time point t, jobs to be processed in the system are collected in a set $\boldsymbol{J}_t = \{1, 2, \ldots, |\boldsymbol{J}_t|\}$. Notice that some jobs may be processed for two operations, while others should be processed for three operations. For all jobs, the first and second operations should be completed in a wet bench and a furnace group, respectively. Further, for jobs that need to be processed for three operations, the last two operations should be completed in two different furnace groups. NO_j is used to denote the number of operations needed to be performed for Job j, $j \in \boldsymbol{J}_t$, where NO_j can only take the value of two or three. O_{ji} is used to represent Operation i of Job j, and it is associated with a starting time point t_{sji}, a completion (ending) time point t_{eji}, a processing time p_{ji}, and a QTL q_{ji}. Note that $t_{eji} = t_{sji} + p_{ji}$ and q_{ji} denotes a time duration within which the next operation should start after the completion of the current operation. Thus, we have $t_{eji} = t_{sji} + p_{ji}$ and $t_{eji} \leq t_{sj(i+1)} \leq t_{eji} + q_{ji}$ should be met so as to satisfy QTLs. This constraint is critical for quality assurance: by enforcing the upper bound $t_{eji} + q_{ji}$, the model explicitly prevents uncontrolled native oxide regrowth and

surface recontamination, thereby ensuring the physical integrity of the wafers without requiring additional penalty terms in the objective function.

Additionally, jobs arrive at the OWB-MF system according to a scheduled time. The information about jobs, such as prespecified processing recipes and arrival time, can be known and collected in advance. In practice, Job j, $j \in \boldsymbol{J}_t$, has a priority and an arrival time point to the OWB-MF system represented by ρ_j and ART_j, respectively. As mentioned above, we have two types of processing flows. For a job to be processed in the OWB-MF system, an *exact processing flow* (EPF) should be known in advance. This implies that with the EPF being given, the tool groups visited by the job for completing each operation are determined. As a result, there are a number of EPFs (each job has an EPF) in such a system. An EPF is made up of a specific WBG and one or more specific FGs. Note that jobs with the same EPF are defined as a job family. Let $\boldsymbol{K} = \{1, 2, \ldots, |\boldsymbol{K}|\}$ denote a set of job families such that each job family is associated with a unique number. Besides, the recipes handled by the system are collected in a set $\boldsymbol{R}$ in which each unique number represents a recipe. Further, $C\,rr'$ is used to denote the required setup time for tools switching from processing recipe r to recipe r', $r, r' \in \boldsymbol{R}$. Note that $C_{rr'} = C\,r'r$. Also, for each operation, the recipe of a job is known in advance. Let r_{ji} denote the recipe of Job j for Operation i, $r_{ji} \in \boldsymbol{R}$.

For each manufacturing tool, job batch creation is independent. There are three conditions for jobs to be grouped into a batch when an operation is dealt with: 1) be members of the same job family k, $k \in \boldsymbol{K}$; 2) require the same processing recipe r, $r \in \boldsymbol{R}$; and 3) be assigned to the same manufacturing tool m, $m \in \boldsymbol{M}$. This implies that the number of times for a job to be grouped into a job batch equals the number of its operations needed to be performed at the diffusion area. Further, after jobs are assigned to a manufacturing tool, they are classified into several job batches such that the jobs in such a batch belong to the same job family and have the same recipe. Notice that each tool has the maximal number of jobs that it can process at a time, and CM_m is used to denote the maximal number of jobs handled by Tool m at a time. At Tool m, if the number of jobs that belong to the same job family and have the same recipe is greater than CM_m, it implies that there should be multiple job batches in which the jobs belong to the same job family and have the same recipe. Such job batches are considered to have the same "batch type". Here, a batch type is defined as a classification based on job characteristics, including the job family and recipe. As a result, any two job batches at Tool m have the same batch type when their jobs have identical job families and recipes. It is worth noting that these batch-formation rules directly influence the inter-operation waiting times of jobs. When multiple jobs with different upstream completion times are merged into the same batch, some of them may experience extended waiting before processing starts. Likewise, unfavorable sequencing of batch operations on furnace tools may further prolong waiting due to setup or idle periods. Since QTL violations are associated with surface recontamination and unwanted native-oxide growth in diffusion processes, the scheduling decisions for batch formation and batch sequencing play an important role in preserving wafer quality. Hence, any schedule that causes at least one batch to exceed its QTL window is classified as infeasible, thereby enforcing QTL compliance as a hard quality-assurance requirement.

Let $\boldsymbol{B}_m$ be a set of job batch types to be processed at Tool m, where $\boldsymbol{B}_m = \{1, 2, \ldots, |\boldsymbol{B}_m|\}$. Moreover, $\boldsymbol{J}_{mb}$ is a set of jobs belonging to the batches with type b and to be processed at tool m, where $m \in \boldsymbol{M}$, $b \in \boldsymbol{B}_m$. Let N_{mb} be the number of job batches of type b to be processed at tool m. Then, $N_{mb} = \lceil |\boldsymbol{J}_{mb}|/CM_m \rceil$ holds. Thus, there should be $N_{mb} \times |\boldsymbol{B}_m|$ job batches to be processed at Tool m in total. Now, we will introduce how to form batches for jobs with the same batch type. For jobs with batch type b, ρ_j, for all $j \in \boldsymbol{J}_{mb}$, are sequenced in descending order. Then, the CM_m jobs with the highest priorities are picked out to form a batch and such a selecting process is repeated for $(N_{mb} - 1)$ times. Finally, the rest of the jobs with batch type b form a batch. In this way, the jobs with the same batch type are classified into multiple batches. An example of how jobs

are grouped into job batches is provided in Appendix A, where the job families, recipes, and priorities are summarized in Table A1.

When a job arrives at a manufacturing tool, the tool might be busy processing wafers, so the job has to wait at the tool. Once the tool completes the processing of wafers, the wafers can be removed soon such that the empty tool is available. Thus, each tool has an available time point at which it is just empty. Besides, at this available time point, each tool is associated with a recipe of jobs that is the last processed one on it. Let AVT_m and r_m denote the available time point and the last processed recipe of Tool m. When a job arrives at a tool, it may need to wait to form a job batch with other arriving jobs, even if the tool is available, to improve the productivity of the whole OWB-MF system. Thus, each furnace is equipped with a finite buffer to hold jobs to be processed, as shown in Fig. 1. Besides, a designated storage area is used to store jobs before they enter the wet benches for processing. The capacity of such a storage area is infinite since there is no time window in which a job should be loaded into a wet bench for processing.

Finally, T_j is used to denote the cycle time of Job j, where T_j can be obtained once a schedule is found. The objective of the addressed problem is to minimize the sum of cycle times of all jobs. Then, we have

$$\text{Minimize}\Gamma = \sum_{j=1}^{|J|} T_j. \tag{1}$$

It is hard to find an exact solution for the addressed scheduling problem. Thus, an approximate solution method is applied in this work. Before introducing the approximate solution method, some assumptions are presented based on the practical scenarios: 1) there is no cancellation of jobs. This implies that once a job enters the system, it cannot be removed out of the system before it is completed; 2) each recipe has a prespecified processing time; 3) the key parameters of each tool are known, including the recipes that they can handle and the maximum number of jobs that they can deal with at a time; 4) the processing activities of jobs cannot be interrupted once they are processed in a tool; and 5) a tool can deal with one job batch only at a time.

3 Approximate Solution

Heuristics or metaheuristics can be applied to find approximate solutions for the addressed problem. Although heuristic algorithms can quickly provide solutions, it is hard for them to stably find good solutions. For such complex scheduling problems, the use of metaheuristics remains the most popular choice [25]. Extensive studies have shown that metaheuristic algorithms are effective for solving complex scheduling problems. For example, Jia et al. [8] developed an ant colony optimization algorithm for parallel batch machine scheduling with incompatible job families, while Mönch and Roob [26] proposed a matheuristic framework for batch machine scheduling with regular sum objectives. In addition, genetic-algorithm-based approaches have been successfully applied to flexible and hybrid flow-shop scheduling problems [27,28]. Nevertheless, it should spend more time and computational resources on solution searching [6] by comparing with heuristic algorithms. However, it is worth using a metaheuristic algorithm to find high-quality solutions due to the complexity of the addressed problem.

This work constructs a BSA to solve the addressed scheduling problem. BSA is a population-based metaheuristic proposed in [29]. It comprises a current population and a historical population represented by P_{cur} and P_{old}, respectively. The standard BSA has five steps, which are initialization, selection I, mutation, crossover, and selection II. Since the addressed problem in this article is a discrete optimization problem, the standard BSA should be modified. Thus, this work constructs a *discrete Backtracking Search Optimization Algorithm* (DBSA) as follows.

A. Individual and Population Representation

This work adopts integer coding for individual representation in DBSA. Each individual contains two parts. The first part encodes the job assignments to manufacturing tools, while the second part encodes the processing sequence of job batch types at each tool. To do so, each tool is assigned a specific Arabic number, starting from a tool in WBG and ending with a tool in FG 2. Initially, the set of Arabic numbers of available manufacturing tools that can deal with O_{ji}, $j \in \boldsymbol{J}_t$, $i \in \{1, 2, 3\}$, is presented as $\boldsymbol{V}_{ji}$, $\boldsymbol{V}_{ji} \subset \boldsymbol{M}$. Then, let $\mathcal{M}_j = (\boldsymbol{V}_{j1}, \boldsymbol{V}_{j2}, \boldsymbol{V}_{j3})$ being used to collect the tools that can process Job j, $j \in J_t$, for each operation. Notice that if Job j requires only two operations, then we have $\mathcal{M}_j = (\boldsymbol{V}_{j1}, \boldsymbol{V}_{j2}, 0)$ in which "0" represents the non-existent operation.

With $\mathcal{M}_j$ being given, it has to choose a tool for Job j to perform the corresponding operation. Initially, the designed DBSA randomly selects a manufacturing tool for a job to complete the corresponding operation. By doing so, the first part of an individual in the designed DBSA is determined. Besides, for the second part, each job batch type at a tool is also named by using a specific Arabic number starting from one, while if a tool does not have any job batch to be processed, its coding is defined as 0.

It is well known that swarm intelligence algorithms use a population of individuals to find feasible solutions. Assume that there are γ individuals in a population. In this work, each individual in the population is denoted by $\Pi_x = [\pi_{1x}, \pi_{2x}]$, $x \in \mathbb{N}_\gamma = \{1, 2, \ldots, \gamma\}$, where π_{1x} and π_{2x} are two row vectors referring to Part-I (job assignments to manufacturing tools) and Part-II (job batch type processing sequences) of Individual-x, respectively. Then, the initial population is represented by $\Theta_\gamma = \{\Pi_x | \, x \in \mathbb{N}_\gamma\}$.

Specifically, in π_{1x}, the first three elements from the left-hand side represent the three tools selected from $\mathcal{M}_1$ to deal with three corresponding operations of Job 1, the second three elements indicate the three tools selected from $\mathcal{M}_2$ to deal with three corresponding operations of Job 2, . . . , the last three elements indicate the three tools selected from $\mathcal{M}_{|J_t|}$ deal with three corresponding operations of Job $|\boldsymbol{J}_t|$. Thus, the size of π_{1x} should be $|\boldsymbol{J}_t| \times 3$.

Additionally, $\pi_{2x} = [\pi_{2x\_1}, \pi_{2x\_2}, \ldots, \pi_{2x\_|M|}]$ is a vector composed of $|\boldsymbol{M}|$ sub-vectors, where sub-vector π_{2x_m}, $m \in \boldsymbol{M}$, represents the processing sequence of all types of job batches to be processed at Tool m. Then, the size of π_{2x} is determined by the number of batch types at each tool, and it is $\sum_{m=1}^{|M|} |\boldsymbol{B}_m|$. In π_{2x}, each element in each sub-vector represents a unique type number. Notice that for each type of job to be processed at a tool, several job batches may be formed. Based on the practical demands from semiconductor fabs, such job batches should be consecutively processed at a tool. In the meantime, the job batch with a higher priority (the sum of priorities of jobs in the batch) should be processed before the one with a lower priority. In this way, the processing environments can be kept stable to the greatest extent, well ensuring the quality consistency of the same type of jobs. Also, the setup time from processing one type of job to another is significantly reduced, so as to improve the productivity of the whole system. Thus, once the value of each element in π_{2x} is determined, it means that the processing sequences of jobs assigned to each tool are determined. The schematic diagram of an individual coding example is shown in Fig. 2.

B. Fitness Calculation

The objective of the addressed problem is to minimize the sum of cycle times of all jobs, which serves as the fitness value for a given individual. After batch creation, the total workload of each tool is determined. However, calculating the cycle time of each job requires precise knowledge of its starting and completion time points. Determining the starting time point for each job batch requires a global perspective, as it should simultaneously satisfy the QTLs for all jobs within the batch. Traditional methods, such as simulations or straightforward calculations, frequently fall short in practical applications. This is because they are inherently limited in their ability to holistically integrate and satisfy all QTLs throughout the entire scheduling process

simultaneously. For example, since the goal is to minimize the total cycle time, simulation-based approaches may schedule jobs to start as early as possible to reduce cycle time. However, this may make it impossible for subsequent batches to meet their QTLs. Consequently, we develop an LP model to determine the starting time points for a given schedule. Further, the LP model can be used to check the feasibility of the given schedule. Once it is examined as feasible, it ensures both QTL feasibility and minimal total cycle time. The detailed LP formulation, including the objective function and all constraints, as well as its theoretical justification, is provided in Appendix A.1. The interaction between batch formation and QTL feasibility is explicitly characterized by the joint effect of Constraints (A4) and (A7). Constraint (A7) enforces batch synchronization, requiring all jobs within the same batch to start processing simultaneously. As a result, the common batch start time is determined by the latest upstream completion time among the jobs in that batch, and jobs that complete earlier should wait until this synchronized start time. Constraint (A4) then restricts this waiting by imposing an upper bound on the allowable inter-operation interval. Therefore, if the dispersion of upstream completion times within a batch is sufficiently large, the induced waiting may violate the QTL constraint, rendering the LP infeasible. This structural coupling further implies that larger batch sizes or more heterogeneous job compositions increase the likelihood of QTL violations, since synchronization enlarges the dispersion term while the QTL bound remains fixed.

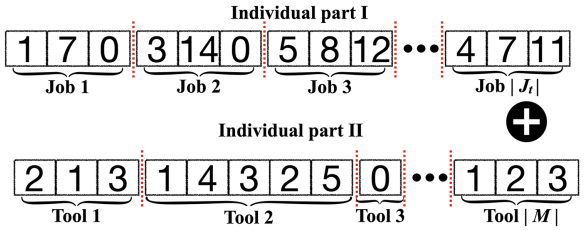


Figure 2: Schematic diagram of an individual coding example.

Notice that it is extremely challenging to satisfy QTLs in semiconductor manufacturing systems [30]. In some cases, by just adjusting the starting time points of job batches, QTLs cannot be met for some jobs, also. In these cases, it is necessary to regroup jobs to form job batches and adjust their processing sequences. Algorithm 1 is developed to do so.

Algorithm 1: Fitness evaluation with LP-guided feasibility restoration

Input: Individual-x

Apply the LP model to check the feasibility of Individual-x;
If Individual-x is examined to be feasible
  Output: Γ
Else
  For $m \leftarrow 1$ to $|\boldsymbol{M}|$
    If $|\boldsymbol{B}_m| > 0$
      $\vartheta \leftarrow \varnothing$;
      For $b \leftarrow 1$ to π_{2x_m}
        For $l \leftarrow 1$ to N_{mb}
          For $u \leftarrow 1$ to $|\boldsymbol{OMB}_{mbl}|$
            Insert $\boldsymbol{OMB}_{mblu}$ into ϑ;
      For $h \leftarrow 1$ to $\sum_{b}^{|\boldsymbol{B}_m|} \sum_{l}^{N_{mb}} |\boldsymbol{OMB}_{mbl}|$
        For $f \leftarrow 1$ to $|\vartheta|$

(Continued)

Algorithm 1 (continued)

14) $\boldsymbol{\varepsilon}_{mh} \leftarrow \{\vartheta_f\}$;
15) Solve the LP restricted to Operation 1 and obtain $\{t_{ej1} \mid j \in \boldsymbol{J}_t,\}$;
16) For $m \leftarrow 1$ to $|\boldsymbol{M}|$
17) If Tool $m \in \boldsymbol{G}_g$ that serves for Operation 2 and $|\boldsymbol{B}_m| > 0$:
18) Sort $\{\boldsymbol{\varepsilon}_{mh}\}$ by $\max\{t_{ej,1} \mid O_{j1} \in \boldsymbol{\varepsilon}_{mh}\}$ (ascending), breaking ties by job priority (descending);
19) Solve the LP restricted to Operations 1 and 2 and obtain $\{t_{ej1}, t_{ej2} \mid j \in \boldsymbol{J}_t,\}$;
20) For $m \leftarrow 1$ to $|\boldsymbol{M}|$
21) If Tool $m \in \boldsymbol{G}_g$ that serves for both Operations 2 and 3 and $|\boldsymbol{B}_m| > 0$:
22) Sort $\{\boldsymbol{\varepsilon}_{mh}\}$ by $\max\{t_{ej,i} \mid O_{ji} \in \boldsymbol{\varepsilon}_{mh}\}$ (ascending), breaking ties by job priority (descending);
23) For $m' \leftarrow 1$ to $|\boldsymbol{M}|$
24) If Tool $m' \in \boldsymbol{G}_g$ that serves for Operation 1 and $|\boldsymbol{B}_{m'}| > 0$:
25) Adjust $\{\boldsymbol{\varepsilon}_{mh}\}$ to remove cross-stage inversions with respect to $\{\boldsymbol{\varepsilon}_{m'h'}\}$, i.e., enforce a consistent job order across consecutive operations whenever applicable;
26) For $j \leftarrow 1$ to $((|\boldsymbol{J}_t| - 1)$
27) If $NO_j = NO_{(j+1)} = 3$
28) If (O_{j1} and $O_{(j+1)1}$), (O_{j2} and $O_{(j+1)2}$), and (O_{j3} and $O_{(j+1)3}$) are consecutively processed at the same tools:
29) Merge $O_{(j+1)1}$ into the batch containing O_{j1}, merge $O_{(j+1)2}$ into the batch containing O_{j2}, and merge $O_{(j+1)3}$ into the batch containing O_{j3}, respectively;
30) If $NO_j = NO_{(j+1)} = 2$:
31) If (O_{j1} and $O_{(j+1)1}$), (O_{j2} and $O_{(j+1)2}$) are consecutively processed on the same tools:
32) Merge $O_{(j+1)1}$ into the batch containing O_{j1} and merge $O_{(j+1)2}$ into the batch containing O_{j2};
33) Update Individual-x as Individual-x';
34) Apply the proposed LP model to obtain the optimal schedule of all jobs for all operations;
35) If Individual-x' is examined as feasible
36) **Output**: Γ, Individual-x'
37) Else
38) **Output**: $\Gamma \leftarrow$ a big enough number, Individual-x'

In Algorithm 1, Statement 1) examines whether a given Individual-x has feasible solutions or not by employing the established LP model. If so, as indicated by Statement 2), the proposed LP model can successfully find an optimal schedule for the whole OWB-MF system under a given Individual-x. If there is no feasible schedule checked by the LP under the given Individual-x, then three steps are used to find a feasible schedule: 1) treat each job as an individual job batch at any tool; 2) adjust the processing sequences of job batches (i.e., the processing sequence of jobs); and 3) combine jobs to form job batches.

Statements 5)–14) are used for Step 1). Among them, Statement 6) aims to identify if there are jobs assigned to be processed at Tool m. If so, Statements 7)–11) are used to store all jobs processed at Tool m in the set ϑ, where π_{2x_m} is the processing sequence of batch types at Tool m. Subsequently, by Statements 12)–14), each job in ϑ forms an individual job batch, which results in the total number of job batches at Tool

m being changed accordingly. Notice that the number of job batches at Tool m is equal to the number of jobs that is calculated by $\sum_{b}^{|\boldsymbol{B}_m|}\sum_{l}^{N_{mb}}|\boldsymbol{OMB}_{mbl}|$, as shown in Statement 12).

After Step 1), the processing sequence of job batches should be adjusted, achieved by Statements 15)–25). It should be noticed that the processing sequence of job batches at tools serving for Operation 1 is preserved to guide the sequencing of job batches of subsequent operations. Even by doing so, there is still a possibility of having no feasible solutions due to the unreasonable processing sequence of job batches at the tools serving for Operation 1. For Step 2), the focus is on determining the processing sequence of job batches at a tool serving for Operation 2 only or both Operations 2 and 3. To do so, the schedule of tools serving for Operation 1 should be determined first. This is accomplished by applying the established LP, as shown in Statement 15). Since we only require the completion time points of the first operations of jobs, the LP should be subject to Constraints (A2), (A5), and (A6), along with tools serving Operation 1. Once the completion time point of the first operation of each job is obtained, the processing sequence of its second operation at the corresponding tool should be adjusted. This is achieved by giving high priority to jobs that complete the first operation early.

To further illustrate the decision logic of Step 2), two representative cases are presented in Fig. 3. These cases correspond to situations in which two jobs share two consecutive operations processed on the same tools but cannot satisfy their QTLs simultaneously under a given processing sequence. In Fig. 3a, the processing sequence is inconsistent across operations, i.e., a job that is processed later in the first operation is processed earlier in the second operation. Under this sequence inconsistency, and with both tool assignments and processing orders fixed, adjusting the starting time of one job to satisfy its QTL may propagate delays to the other job on the shared tool. Although small adjustments may not always trigger infeasibility, there exist configurations in which no feasible schedule can simultaneously satisfy both QTLs by time shifting alone. By contrast, Fig. 3b exhibits a consistent processing sequence across operations, where the relative order of the two jobs is preserved. In this case, potential QTL violations can be resolved by appropriately delaying the starting time, without altering tool assignments or processing sequences. This comparison indicates that QTL infeasibility primarily originates from sequence inconsistency across operations rather than from insufficient waiting time.

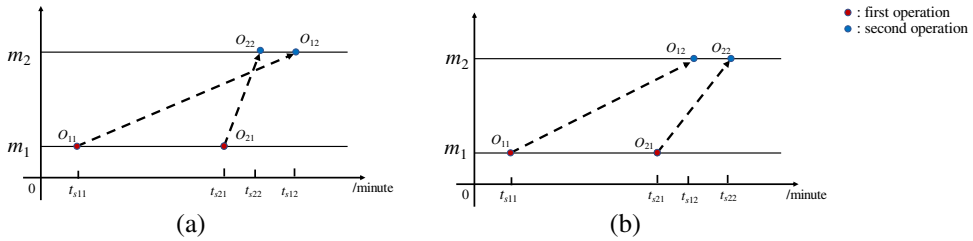


Figure 3: Illustrative examples of QTL infeasibility caused by sequence inconsistency across consecutive operations. (**a**) Inconsistent processing sequence across operations. (**b**) Consistent processing sequence across operations.

Notice that a furnace (tool) may serve for Operation 2 only or both Operations 2 and 3. Additionally, in this work, Operations 2 and 3 of a job cannot be completed with the same tool. Therefore, the schedule of a furnace can only be determined when the completion time points of the preceding operation of all jobs assigned to this furnace are known. Moreover, for a flow shop system with QTLs between two consecutive operations, assume that all jobs have the same processing routes. If the processing sequence of all jobs at each tool is kept the same, one can satisfy the QTLs by adjusting the starting time points of the jobs for a given schedule, i.e., a given processing sequence of jobs. Inspired by this property, in this work, if two tools are respectively applied to complete two operations of two jobs such that each tool is used for completing one operation of the two jobs, then the processing sequences of the two jobs at each tool are kept the same.

Otherwise, the processing sequence of any two jobs at a tool is determined according to the completion time of their previous operations, such that the earlier the previous operation is completed, the earlier the job should be started at the tool. Further details to achieve this in Algorithm 1 are given as follows.

One can find a schedule for the tools serving for Operation 2 only by applying the LP, as achieved by Statements 16)–19). The LP here is subject to Constraints (A2)–(A6), along with tools serving Operation 1 and Operation 2. Notice that this may result in the update of the schedule at the tools serving Operation 1. Similarly, we move on to adjust the processing sequences of jobs at tools serving for both Operations 2 and 3, as shown by Statements 20)–22). After this, Statements 23)–25) are used to guarantee the processing sequences of job batches should remain the same as their previous operations if they have two or more operations assigned to be processed on the same tools. Thus, Step 2) is completed.

Then, Statements 26)–32) implement Step 3) of the feasibility restoration procedure, whose purpose is to recover batch structures while preserving the processing sequences fixed in the previous steps. After Step 1), each job is treated as an individual job batch; therefore, Step 3) performs batch combination by merging adjacent entities along the established processing order, i.e., either two consecutive jobs or an existing batch and its immediately following job. This adjacency restriction is essential. Combining non-adjacent entities would require bypassing or reordering at least one intermediate job on some tool, which would implicitly modify the processing sequences determined in Statements 16)–25) and may introduce new QTL violations. To prevent such implicit resequencing, Statements 28) and 31) explicitly require that the corresponding operations of two entities be processed consecutively on the same tools for all required operations. Under this condition, batch combination preserves both tool assignments and intra-tool processing orders, ensuring that feasibility restoration is achieved without disrupting the established schedule structure. This restriction is a deliberate design choice, which prioritizes schedule stability and QTL feasibility over more aggressive batch reconstruction.

By Statements 5)–32), the given Individual-x is updated as Individual-x'. Likewise, the established LP model is applied to examine the feasibility of Individual-x' by Statement 34). If it is unfortunate that Individual-x' still has no feasible solutions, then Γ is given a big enough number such that Individual-x' will be abandoned by the iteration of DBSA.

C. Local Search Operation

Once the job assignments to manufacturing tools have been completed, the type and number of job batches tackled by a tool are also determined. In DBSA, for an individual, Part-II is constructed based on Part-I to a certain extent. In fact, after Part-I of an individual is given, the quality of the individual is significantly affected by the quality of its Part-II.

To find a high-quality Part-II, we introduce a local search procedure. Local search is highly effective when integrated with metaheuristics for solving scheduling and optimization problems [31]. Specifically, we use three operators to adjust the processing sequence of different batch types at a tool, which are swap, reverse, and insert. The swap operator generates a new sequence by exchanging two elements at different positions. The reverse operator creates a new sequence by selecting a subsequence and reversing its order. The insert operator works by removing an element from its current position and inserting it into another position. The proposed local search procedure is applied to each newly generated individual for a fixed number of iterations. In each iteration, a neighborhood operator (swap, reverse, or insert) is randomly selected to modify the processing sequence. The new solution is evaluated, and if it yields a better fitness value, it replaces the current individual.

D. Mutation Operation

The mutation operator aims to provide novel traits that are beneficial to the population. Mutation of DBSA is associated with both P_{old} and P_{cur}, resulting in a trial population called T-P. The mutation is designed based on the one in [32] realized by Algorithm 2. In Algorithm 2, γ is the population size. F and f are both the search amplitude factors used to control the exploration and represented by two binary vectors, respectively. The sizes of both F and f are equal to that of Part-I of an individual. Initially, all elements in both F and f are zero. Then, two different methods are used to adjust the elements of F and f from zero to one, respectively. Specifically, the adjustment for F requires an additional parameter called *alpha*, which determines the number of elements to be adjusted. Parameter *alpha* can be calculated by Eq. (2).

$$alpha = \lceil (1 - (iter/Maxiter)) \times |F| \rceil, \tag{2}$$

where *iter* and *Maxiter* represent the current iteration number and the maximum number of iterations, respectively. Then, F can be adjusted by Statements 6)–8). The method applied to adjust f is much simpler and shown in Statements 9)–11). Specifically, comparisons are made to check if each element in both P_{old} and P_{cur} is identical. If so, no adjustment is made, and otherwise, the value at the corresponding position of f should be adjusted to be one. Then, a new vector FF can be obtained by element-wise multiplication, denoted by $FF = F \odot f$. FF is used to guide the generation of a new individual based on the current individual. Statements 13)–17) are used to do so. If an element in FF equals one, it indicates that the job assignment of the corresponding operation needs to be rearranged. This can be achieved by randomly selecting a tool among the tools that can deal with the operation. After a new individual is generated, we keep the excellent one achieved by Statement 18).

Algorithm 2: Mutation operation

Input: P_{old}, P_{cur}, *alpha*

Output: T-P

1) T-P $\leftarrow$ P_{cur};
2) For x $\leftarrow$ 1 to γ
3) ϑ $\leftarrow$ an empty set;
4) F $\leftarrow$ $\vec{0} \times |\pi_{1x}|$;
5) f $\leftarrow$ $\vec{0} \times |\pi_{1x}|$;
6) Randomly select *alpha* indices from F and store them in φ;
7) For i $\leftarrow$ 1 to $|\phi|$
8) F_{φ_i} $\leftarrow$ 1;
9) For i $\leftarrow$ 1 to $|f|$
10) If $P_{old,i} \neq P_{cur,i}$
11) f_i $\leftarrow$ 1;
12) FF $\leftarrow$ $F \odot f$;
13) *New*-π_{1x} $\leftarrow$ π_{1x};
14) For i $\leftarrow$ 1 to $|\pi_{1x}|$
15) If $FF_i = 1$
16) Randomly select a capable tool for *New*-π_{1xi};
17) Obtain *New*-π_{2x} with local search procedure;
18) T-P_i $\leftarrow$ individual with the minimum fitness value among $P_{cur,i}$, $P_{old,i}$, and *New*-Π_x;

E. Double Crossover Operation

To enhance the quality of T-P, a double crossover operator is introduced in Algorithm 3, combining standard BSA crossover with a segment crossover. The proposed hybrid strategy is designed as a two-level recombination mechanism. The standard BSA crossover preserves global structural information inherited from parent individuals, thereby promoting convergence stability. In contrast, the segment crossover introduces localized subsequence perturbations, enabling flexible structural reshaping and enhancing diversity. By integrating both mechanisms, the algorithm achieves complementary search behaviors: global inheritance supports exploitation of high-quality schemata, while localized segment exchange enhances exploration and prevents premature structural stagnation. This hybrid strategy thus balances diversity and convergence in permutation-based scheduling.

Statements 1)–17) in Algorithm 3 are applied to realize the crossover operation of the standard BSA. The standard BSA involves computing a binary integer-valued matrix, referred to as the "Map". Note that the Map is a γ-by-L matrix, where L is the size of Part-I of an individual. Similar to the mutation operation, the element values of such a Map will be changed from 1 to 0 for updating T-P. To obtain such a Map, there are two different strategies applied, shown as Statements 4)–13). Note that Random(0, 1) indicates a value randomly generated in the range of (0, 1). Statements 4) and 9) are used to randomly select one strategy based on the comparison of a and b; their values are generated by using Random(0, 1). The difference between these two strategies lies in the application of the crossover rate p_c and whether the order of D is shuffled, where $D = (1, \ldots, L)$. Shuffling the order of D is achieved by a permuting function, denoted as $permuting(D)$, which randomly generates a new sequence of D. Note that p_c is determined in a similar way as the parameter *alpha* in the mutation operation, and it is obtained by the following equation.

$$p_c = \lceil 1 - (iter/Maxiter) \rceil . \tag{3}$$

Besides, u is the number of elements to be changed from one to zero of the Map. Once the element values of the Map are finally determined, it can guide the updating of T-P by Statements 14)–17).

Algorithm 3: Double crossover operation

Input: T-P, P_{cur}, γ, p_c
Output: T-P
1) $L \leftarrow$ size of Part-I of an individual.
2) Map = 1 × $[\gamma, L]$ // Map is a matrix of ones.
3) $D = (1, \ldots, L)$
4) If a < b // a, b ← Random(0, 1)
5) For $i \leftarrow 1$ to γ
6) $u = \lceil p_c \times |D| \times c \rceil$ // $D \leftarrow permuting(D)$, $c \leftarrow$ Random(0, 1);
7) For $j \leftarrow 1$ to u
8) $Map_{i,D_j} \leftarrow 0$;
9) Else
10) For $i \leftarrow 1$ to γ
11) $u \leftarrow \lceil |D| \times c \rceil$ // $c \leftarrow$ Random(0, 1);
12) For $j \leftarrow 1$ to u
13) $Map_{i,D_j} \leftarrow 0$;
14) For $i \leftarrow 1$ to γ
15) For $j \leftarrow 1$ to L
16) If $Map_{i,j} = 1$

(Continued)

Algorithm 3 (continued)

17) T-P_i [Part-I] [j] ← $P_{cur,i}$ [Part-I] [j];
18) For i ← 1 to γ
19) α ← $\lceil c \times \gamma \rceil$, β = $\lceil c \times \gamma \rceil$ // c ← Random(0, 1);
20) Segment crossover between Individual-α and Individual-β;
21) T-P_i ← min_fitness(parents È offsprings);

Moreover, a segment crossover operation is performed by Statements 18)–21). Fig. 4 gives an example to illustrate how the segment crossover operation works. Given two parents named as Individual-α and Individual-β, two different crossover points are randomly selected such that the cross-segments are generated. The cross-segment of Individual-α is then replaced by the cross-segment of Individual-β and *vice versa*. By doing so, we can generate two offsprings and subsequently calculate their fitness values. Note that among these four individuals, the one with the minimum fitness value is selected as a member of a new population. Such a crossover operation is performed for γ times to generate γ individuals, forming the final T-P. Fig. 5 provides the flowchart of DBSA.

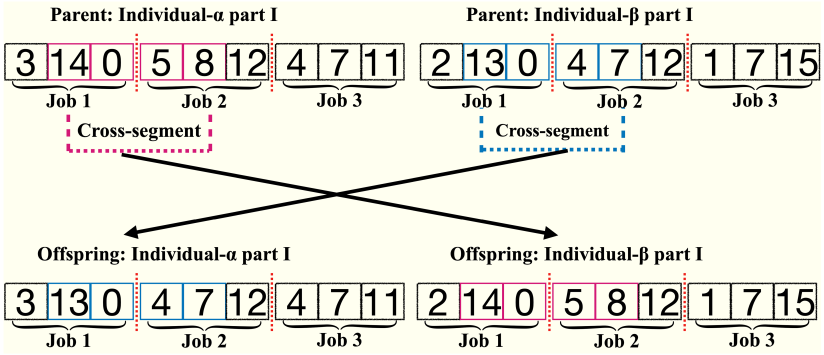


Figure 4: Segment crossover operation.

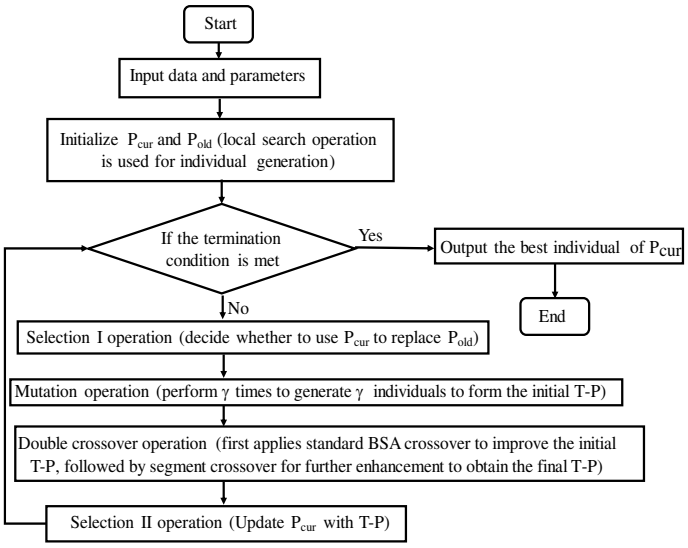


Figure 5: Flowchart of DBSA.

4 Experimental Results

A. Data Set

Since this study is application-driven, the test instances were synthetically generated based on operational parameters collected from a real-world OWB-MF system. This design ensures experimental reproducibility while maintaining the representativeness of practical semiconductor manufacturing environments. To preserve the physical interpretability of the scheduling problem, preprocessing was explicitly integrated into the instance generation process rather than applied as post hoc adjustments. The generated data were directly used without additional manipulation, and no job filtering was performed after generation; all jobs were retained to preserve the original workload characteristics. Regarding data validity, preventive constraints were enforced during the generation phase to avoid unrealistic outliers, including non-negativity constraints and upper bounds consistent with tool capacity limitations. As a result, no post hoc outlier handling was required. Since the datasets contain complete information by design, no interpolation techniques were applied. In addition, no numerical normalization (e.g., min–max scaling) was used. All time-related parameters retain their original physical units (minutes), which was empirically verified to ensure numerical stability during optimization. Finally, batch formation is handled endogenously within the scheduling process. Edge cases, such as remaining jobs fewer than the maximum batch size, are processed as valid batches to ensure complete schedule coverage. The key system parameters and instance configurations corresponding to different OWB-MF settings are summarized in Tables 1 and 2. In each table, two cases with different numbers of jobs are considered. All experiments are conducted on a laptop equipped with an Apple M1 processor and 8 GB RAM. All algorithms are implemented in Python 3.8. The LP subproblems are solved using IBM ILOG CPLEX 12.9 via the DOcplex Python API. Unless otherwise stated, default CPLEX parameter settings are adopted.

Table 1: A summary of experimental design for small-sized problems.

Parameters	Values
Number of WBG	1
Number of wet benches	5
Number of FG	3
Number of furnaces	[3, 3, 3] for FG1 to FG3, respectively
EPFs	1) WBG→FG 1, 2) WBG→FG 2, 3) WBG→FG 3, and 4) WBG→FG 2→FG 3
Number of job family	4
Number of recipes	9
Number of jobs	[30, 50]
Processing time/QTL of recipe	[30, 20, 40, 300, 400, 350, 480, 490, 500] for recipe 1 to recipe 9, respectively
Tool capacity	[2, 2, 2, 2, 2, 4, 6, 6, 4, 4, 4, 6, 6, 8] for the first tool in WBG to the last tool in FG 5, respectively
Tool available time point	[10, 5, 0, 7, 8, 300, 50, 0, 150, 0, 20, 100, 30, 80] for the first tool in WBG to the last tool in FG 3, respectively
Recipe tool can process	[[1, 3], [1, 2], [1, 3], [1, 2, 3], [1, 3], [4, 6], [4, 6], [4, 6], [5, 7], [5, 7], [8, 9], [8, 9], [8, 9]] for the first tool in WBG to the last tool in FG 3, respectively

Table 2: A summary of experimental design for large-sized problems.

Parameters	Values
Number of WBG	1
Number of wet benches	10
Number of FG	5
Number of furnaces	[4, 5, 3, 4, 5] for FG 1 to FG 5, respectively
EPFs	1) WBG→FG 1, 2) WBG→FG 2, 3) WBG→FG 3, 4) WBG→FG 4, 5) WBG→FG 5, 6) WBG→FG 2→FG 3, and 7) WBG→FG 5→FG 4
Number of job family	7
Number of recipes	20
Number of jobs	[75, 100]
Processing time/QTL of recipe	[10, 15, 20, 25, 30, 300, 400, 350, 480, 490, 500, 550, 600, 540, 480, 420, 360, 450, 440, 520] for recipe 1 to recipe 18, respectively
Tool capacity	[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 10, 8, 8, 8, 10, 8, 8, 8, 8, 10, 8, 8, 6, 8, 8, 8, 8, 6, 8, 6, 8] for the first tool in WBG to the last tool in FG 5, respectively
Tool available time point	[10, 0, 20, 50, 8, 0, 30, 10, 0, 6, 300, 500, 1000, 200, 150, 0, 300, 240, 180, 100, 300, 800, 1000, 350, 80, 800, 100, 0, 80, 180, 50] for the first tool in WBG to the last tool in FG 5, respectively
Recipe tool can process	[[1, 3, 4], [1, 2, 4], [1,3, 5], [1, 2, 3], [1, 3, 5], [1, 2, 5], [1, 2], [1, 2, 4], [1, 4, 5], [1, 3, 5], [6, 7], [6, 8], [6, 7, 8], [6, 7, 8], [9, 10], [9, 11], [9, 10, 11], [9, 10, 11], [9, 10, 11], [12, 13], [12, 14], [12, 13, 14], [15, 16], [15, 16, 17], [15, 17], [15, 16, 17], [18, 19, 20], [18, 19], [18, 20], [18, 19], [18, 20]] for the first tool in WBG to the last tool in FG 5, respectively

B. Analysis of Unadjustable Individuals

An individual should be abandoned if it is determined that there is no feasible solution by applying the established LP model after the adjustments made to the processing sequences of job batches. To demonstrate the probability of such a situation occurring, we conducted the following computational experiments. For each computational experiment, we gather 1000 randomly generated individuals that are examined and have no feasible solutions initially, i.e., satisfying Statement 4) of Algorithm 1. Then, the processing sequences of job batches at tools serving for Operation 2 only or both Operations 2 and 3 of each of the 1000 individuals are adjusted by Algorithm 1. If the fitness value provided by Algorithm 1 is a big enough number (i.e., satisfying Statement 38), then the corresponding individual is labeled as an unadjustable individual. Then, the probability of encountering such a situation can be calculated as (Number of unadjustable individuals/1000) × 100%. For each case, such a computational experiment is executed 30 times independently. Moreover, it should be noticed that setting the queue time to zero results in the most challenging condition, as it implies no waiting time between any two consecutive operations of a job. Under this situation, 30 additional computational experiments are conducted for each case independently, with all queue times set as zero. In total, there are 240,000 individuals to be examined. Results are summarized in Table 3, where k denotes the k-th experiment.

Table 3: A summary of probabilities of generating unadjustable individuals.

k	Small-Sized Problems				Large-Sized Problems			
	30 Jobs	30 Jobs-No Waiting	50 Jobs	50 Jobs-No Waiting	75 Jobs	75 Jobs-No Waiting	100 Jobs	100 Jobs-No Waiting
1	0.2%	0%	1.5%	0.5%	0.3%	3.2%	0.7%	10.7%
2	0.4%	0.9%	0.2%	0.4%	0.1%	2.5%	0.4%	11%
3	0.5%	0.7%	1.2%	0.3%	0.2%	2.1%	0.6%	10.4%
4	0.4%	0.4%	0.7%	0.6%	0.4%	2.3%	0.2%	10.3%
5	0.4%	0.1%	0.7%	0.1%	0.1%	2.4%	0.2%	10.2%
6	0.5%	0.5%	1%	0.6%	0.4%	2.6%	0.7%	8.7%
7	0.5%	0%	1.3%	0.1%	0.3%	2.4%	0.5%	10.3%
8	0.6%	0.4%	1%	0.4%	0%	2.4%	0.7%	9.4%
9	0.2%	0.2%	1.1%	0.3%	0.2%	2.7%	0.7%	10.5%
10	0.4%	0.5%	0.8%	0.4%	0.2%	3.1%	0.4%	12.6%
11	0.4%	0.6%	0.4%	0.4%	0.1%	1.9%	0.4%	10.8%
12	0.5%	0.3%	1.2%	0.1%	0.4%	2.8%	0.6%	9.2%
13	0.2%	0.8%	0.8%	0.4%	0.4%	3.6%	0.3%	8.9%
14	0.2%	0.5%	0.9%	0.5%	0.1%	3.1%	0.4%	10.2%
15	0.3%	0.3%	0.8%	0.3%	0.2%	2.9%	0.4%	10.8%
16	0.6%	0.6%	0.7%	0.3%	0.2%	3.7%	0.5%	11.4%
17	0.2%	0.6%	1.5%	0.1%	0.2%	2.2%	0.3%	10.9%
18	0.5%	0.3%	1.3%	0.1%	0.4%	3.2%	0.6%	11.6%
19	0.3%	0.5%	0.8%	0.5%	0.2%	3.7%	0.7%	10.3%
20	0.3%	0.7%	0.6%	0.1%	0.1%	3.3%	0.6%	10.5%
21	0.7%	0.1%	0.8%	0.4%	0.1%	3.2%	0.3%	10.2%
22	0.2%	0.5%	0.7%	0.9%	0.1%	2.7%	0.3%	9.9%
23	0.4%	0.5%	1.1%	0.5%	0.25%	3.1%	0.4%	10.6%
24	0%	0.6%	0.5%	0.7%	0.2%	2.5%	0.5%	10.1%
25	0.5%	0.4%	0.8%	0%	0.3%	3.2%	0.8%	10.7%
26	0.6%	0.2%	1.3%	0.5%	0.4%	2.3%	0.2%	11.1%
27	0.2%	1%	1%	0.4%	0.2%	2.2%	0.6%	9%
28	0.3%	0.5%	0.5%	0.2%	0.2%	3.1%	0.8%	11.6%
29	0.7%	0.7%	0.9%	0.3%	0.2%	2.8%	0.4%	10.5%
30	0.1%	0.2%	0.5%	0.3%	0.1%	2.6%	0.4%	11.7%
Ave.	0.38%	0.45%	0.89%	0.36%	0.22%	2.79%	0.49%	10.47%

Table 3 illustrates the probability of generating unadjustable individuals under different situations, distinguished by problem size and whether queue times are set to zero. The probabilities for all cases with the predetermined queue times are below 1%. At this level, such occurrences are unlikely to have a noticeable impact on the searching process of swarm intelligence algorithms.

Moreover, in small-sized problems, even if queue times are set as zero, the average probabilities consistently remain below 1%. Interestingly, when the number of jobs is 50, the average probability even decreases when queue times are set as zero compared with predetermined queue times. This implies that setting queue times as zero has minimal impact on the average probability for small-sized problems.

However, as the problem size increases, the difference between the probabilities of generating unadjustable individuals by considering predetermined queue times and queue times to be set as zero becomes significant. In large-sized problems, when the predetermined queue times are reset to zero, the average probability increases significantly from 0.22% to 2.79% and from 0.49% to 10.47%, respectively. In practice, queue times play a crucial role by providing a time window for jobs to wait between two operations, such that they can be processed in their workflow. It is uncommon to set the queue time to zero due to its essential function in optimizing the production process. Here, the purpose is to explore the most extreme strict situation by setting queue times as zero to indicate that the impact of unadjustable individuals on the search effectiveness of DBSA appears to be limited.

C. Parameter Settings

To ensure a fair and rigorous comparison with other metaheuristic algorithms, the parameters of DBSA are calibrated prior to experimental evaluation, as algorithm performance may vary significantly under different configurations. The proposed DBSA involves three key parameters: group size, the maximum number of iterations for DBSA, and the number of local search iterations.

The Taguchi method is adopted for parameter tuning, and the detailed procedure is provided in the Appendix A.3. The tuning experiments are conducted across all considered problem scales (30, 50, 75, and 100 jobs), rather than on a single tuning instance. For each parameter, three levels are evaluated (Table A2), and an L9(3^3) orthogonal array (Table A3) is adopted to systematically assess parameter combinations.

The final parameter configuration is determined based on aggregated performance across scales using response variation analysis. As illustrated in Fig. 6, the selected settings are: group size = 100, iterations for DBSA = 150, and iterations for local search = 40.

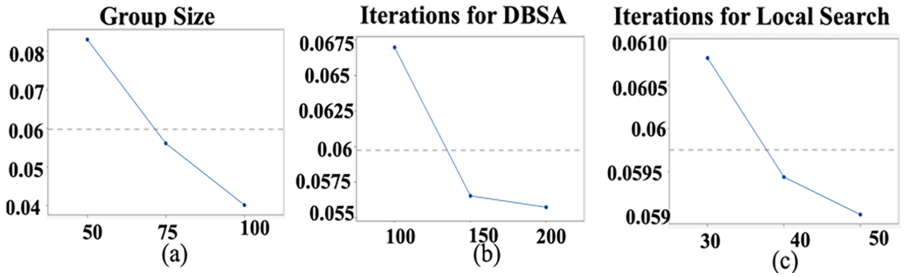


Figure 6: Factor level trends of three key parameters. (**a**) Effect of group size. (**b**) Effect of iterations for DBSA. (**c**) Effect of iterations for local search.

D. Solution Quality Analysis

After determining the parameter settings of DBSA, we first conduct an ablation study to independently evaluate the contribution of the proposed regrouping strategy, which represents the main structural enhancement introduced in this work. Specifically, DBSA with and without the regrouping mechanism are compared under identical settings, where all other components, including population initialization, search operators, and termination criteria, are kept unchanged. In the variant without regrouping, infeasible individuals are assigned a static penalty, which effectively suppresses their survival during the evolutionary selection process, whereas the proposed DBSA additionally applies the rule-based regrouping strategy to restore feasibility under QTL constraints.

Fig. 7 reports the convergence behaviors of the two variants across different problem scales. The results reveal a clear and consistent trend. DBSA without the regrouping strategy converges prematurely in all tested cases and stagnates at inferior solution quality, indicating that infeasible individuals caused by QTL

violations cannot be effectively corrected by search operators alone, particularly as the problem size increases. In contrast, DBSA equipped with the regrouping strategy exhibits sustained improvement and converges to significantly better solutions on all instances. The performance gap becomes more pronounced in large-scale problems (75 and 100 jobs), where QTL-induced infeasibility is more frequent and severe. These observations demonstrate that the performance gain of DBSA does not merely originate from the search framework itself but critically relies on the proposed regrouping strategy, which effectively restores feasibility while preserving the established processing sequences.

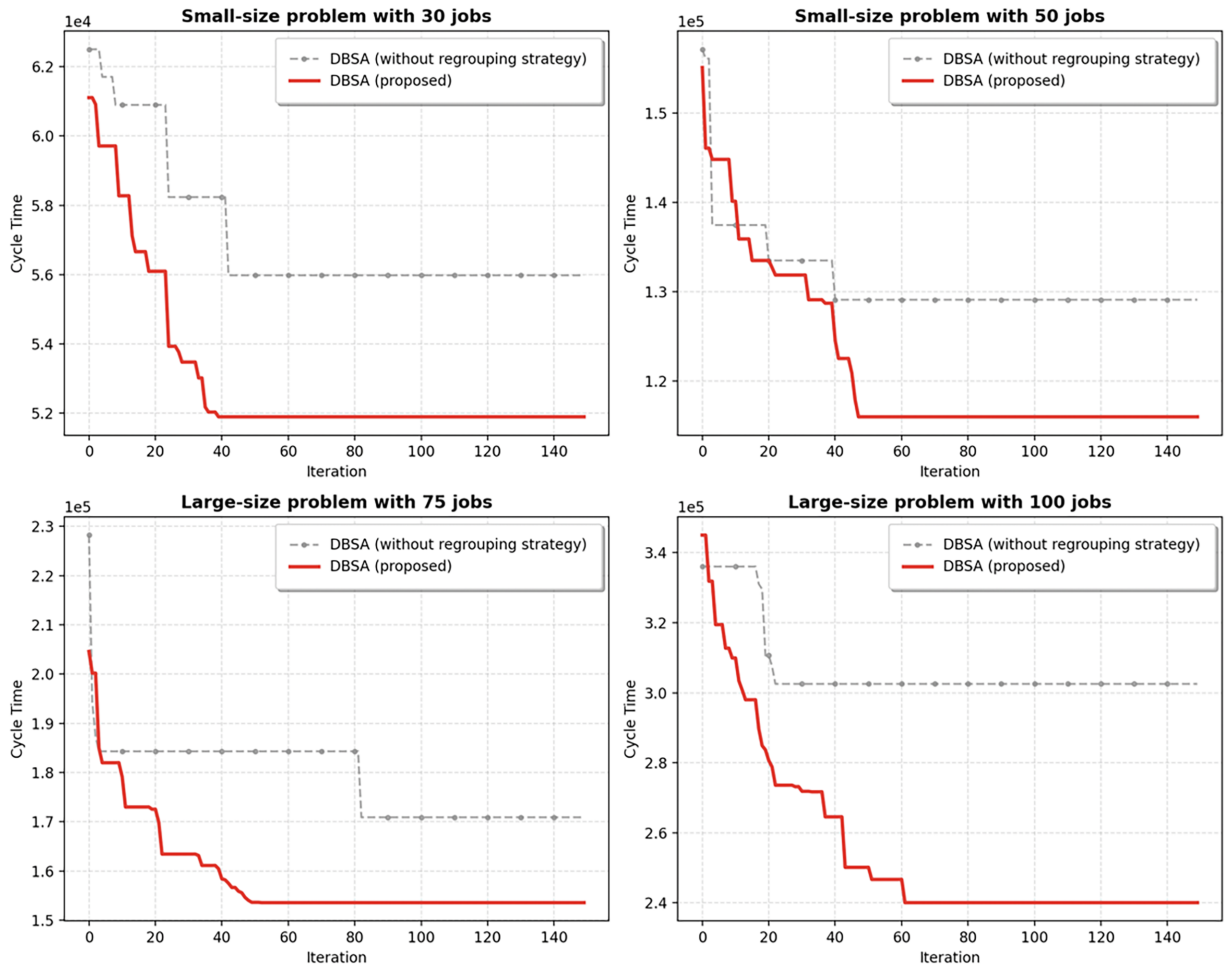


Figure 7: Impact of the regrouping strategy on the convergence behavior of DBSA under different problem scales.

Having verified the essential role of the regrouping strategy, we next compare the proposed DBSA with several representative metaheuristic algorithms, including GA, MixPSO, and the *Grey Wolf Optimizer* (GWO), to further assess its overall effectiveness. It should be noticed that the searching mechanisms of all these metaheuristic algorithms focus on generating high-quality Part-Is. Under this situation, once a Part-I is generated, the local search operation will be applied to it such that an individual can be finally determined. To guarantee fair comparisons among these four algorithms, the running time for each algorithm should be roughly the same. To do so, the running time of DBSA serves as the termination criterion. Besides, the parameter settings for group size and iterations for local search applied for all compared metaheuristic

algorithms are the same as the ones in DBSA. For each algorithm of each case, 30 independent experiments are run.

Comprehensive experimental results are provided in Tables A4 and A5 in the Appendix A, with representative outcomes summarized in Tables 4 and 5. In terms of solution quality, DBSA consistently achieves the best average fitness across all test cases. Specifically, for small-sized scenarios (30 and 50 jobs), it outperforms GA, MixPSO, and GWO by margins ranging from 10.31% to 15.48%. For larger instances (75 and 100 jobs), the improvements remain significant, reaching up to 19.19% against MixPSO. To assess robustness, 95% confidence intervals were calculated based on 30 independent runs. The confidence intervals of DBSA are consistently lower and non-overlapping with those of the comparative algorithms, indicating that the superiority of DBSA is statistically significant. This is further visually corroborated by the box plots in Fig. 8, where DBSA exhibits a lower median and a narrower distribution range (indicating superior stability) compared to the other three metaheuristic algorithms. Notably, in several cases (e.g., Fig. 8c), the worst fitness value of DBSA is even better than the best values obtained by the comparative methods, strongly validating its dominance.

Table 4: A summary of experimental results for small-sized problems.

k	DBSA		GA		MixPSO		GWO	
	Fitness Value (min)		Fitness Value (min)		Fitness Value (min)		Fitness Value (min)	
	30 Jobs	50 Jobs	30 Jobs	50 Jobs	30 Jobs	50 Jobs	30 Jobs	50 Jobs
Max	53,089	129,782	58,861	142,125	60,812	143,795	58,805	135,715
Min	46,660	104,633	53,368	128,772	54,849	132,301	53,284	126,303
Ave.	49,827.6	116,380.6	56,852.7	135,411.6	57,752.6	137,695.3	55,555.1	130,742.8
Std.	1552.7	5022.5	1483.4	3797.1	1207.8	2371.8	1479.6	2567.9
95% CI	[49,272.0, 50,383.2]	[114,583.3, 118,177.9]	[56,321.9, 57,383.5]	[134,052.8, 136,770.4]	[57,320.4, 58,184.8]	[136,846.6, 138,544.0]	[55,025.6, 56,084.6]	[129,823.9, 131,661.7]
Running time	4743.5 s	7117.1 s						
Performance for both 30 and 50 jobs: DBSA > GWO > GA > MixPSO								

Table 5: A summary of experimental results for large-size problems.

k	DBSA		GA		MixPSO		GWO	
	Fitness Value (min)		Fitness Value (min)		Fitness Value (min)		Fitness Value (min)	
	75 Jobs	100 Jobs	75 Jobs	100 Jobs	75 Jobs	100 Jobs	75 Jobs	100 Jobs
Max	160,053	269,963	188,708	329,380	190,095	332,274	185,800	313,294
Min	136,713	231,542	166,561	268,567	173,123	273,052	160,946	263,197
Ave.	147,658.6	253,236.7	179,555.3	289,473.1	182,728.9	292,913.7	175,753.4	279,256.3
Std.	7041.8	10,289.2	5437.6	14,807.0	4726.8	12,150.0	6639.9	9795.6
95% CI	[145,138.7, 150,178.5]	[249,554.8, 256,918.6]	[177,609.5, 181,501.1]	[284,174.5, 294,771.7]	[181,037.4, 184,420.4]	[288,565.9, 297,261.5]	[173,377.3, 178,129.5]	[275,751.0, 282,761.6]
Running time	8813.9 s	12,722.6 s						
Performance for both 75 and 100 jobs: DBSA > GWO > GA > MixPSO								

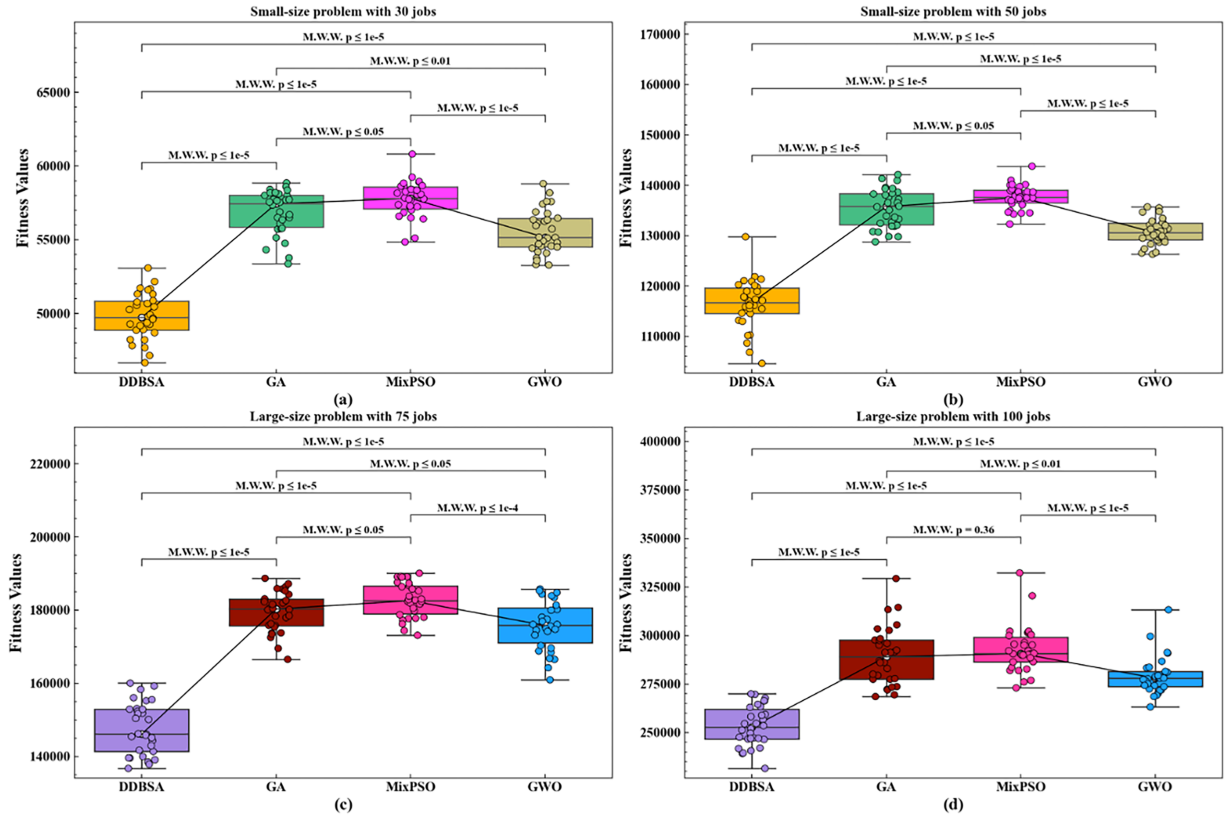


Figure 8: Box plots for different sizes of problems (**a**) 30-job instance. (**b**) 50-job instance. (**c**) 75-job instance. (**d**) 100-job instance.

To increase the trustworthiness of the performance differences among these algorithms, significance tests were conducted to compare any two different metaheuristic algorithms. The significance tests are performed using the Mann-Whitney U-test, a non-parametric method for comparing the medians of two independent samples. This method is suitable for cases in which data do not follow a normal distribution. In statistical analysis, a smaller p-value (p-value $\leq 1e^{-5}$) generally indicates higher statistical significance and makes the observed differences more trustworthy. The outcomes of significance tests are also shown in Fig. 8. The results consistently demonstrate a highly significant difference (p-value $\leq 1e^{-5}$) between DBSA and each of the other algorithms concerning their average fitness values. These results provide strong evidence to support the superior effectiveness of the DBSA algorithm.

Regarding computational efficiency, it is crucial to note that all comparative algorithms were executed under the same computational time budget (strictly aligned with the running time of DBSA, e.g., approx. 3.5 h for 100 jobs, as listed in Tables 4 and 5) to ensure a fair comparison. Under these identical temporal constraints, DBSA consistently identifies significantly better schedules. While the computation time for large-scale instances is longer than simple heuristics, this investment is fully justified within the context of offline daily planning in semiconductor manufacturing. In modern fabs, scheduling is typically organized in a hierarchical manner. The proposed algorithm operates at the high-level planning layer, where schedules are generated offline before the production cycle begins. Because the generated baseline plan usually covers a horizon of multiple days, an offline computation time of several hours is a reasonable investment to ensure a high-quality schedule with rigorous constraint satisfaction, rather than prioritizing real-time speed.

E. Industrial Implications for Diffusion-Area Scheduling in Semiconductor Fabs

The proposed DBSA framework is designed for diffusion-area batch-processing systems with strict QTLs, which represent one of the most challenging and yield-critical stages in modern semiconductor manufacturing. The experimental results presented in Section 4.D demonstrate that DBSA consistently achieves superior scheduling quality and robust performance across different production scales. These advantages translate directly into several important practical benefits for real-world semiconductor fabs.

First, from a production efficiency perspective, DBSA provides substantial reductions in overall cycle time, which is a key performance indicator in wafer fabs. Diffusion areas are typically operated as bottlenecks due to long processing times and batch constraints. By jointly optimizing batch formation, processing sequences, and starting time points under QTL constraints, the proposed method effectively shortens manufacturing lead times and improves batch-processing throughput stability. In practical fab operations, such cycle time reductions directly contribute to higher equipment utilization, improved delivery performance, and better workload balancing across furnace groups.

Second, the explicit integration of QTL constraints into the scheduling and fitness evaluation framework offers strong yield-protection capability. In diffusion processes, excessive waiting times between sensitive operations may lead to native oxide regrowth and surface contamination, which in turn increases the risk of wafer re-cleaning or scrapping. By rigorously enforcing QTL compliance at the planning stage, DBSA systematically limits inter-operation waiting times and ensures that wafer lots are processed within allowable time windows. In industrial practice, this mechanism helps stabilize process quality, reduce yield loss, and lower manufacturing costs associated with rework and scrap.

Third, the proposed framework provides an intelligent decision-support tool for fab-level production planning. In modern semiconductor manufacturing, scheduling is typically organized in a hierarchical manner within *advanced planning and scheduling* (APS) systems, where high-level plans are generated offline and executed by online dispatching and control layers. DBSA is well suited for deployment at the offline planning layer, where it can generate high-quality baseline schedules over multi-day horizons under complex technological and operational constraints. The resulting baseline plans serve as reliable references for subsequent online adjustments, enabling fabs to respond more effectively to disturbances such as tool failures and urgent order insertions.

5 Conclusions

This work aims to solve the scheduling problem of the diffusion area in semiconductor manufacturing with the consideration of QTLs. To do so, we construct the DBSA, which is a metaheuristic algorithm to minimize the sum of cycle times of all jobs. To improve the productivity of the diffusion area, efficient approaches are proposed for optimizing both job assignments and job batch processing sequences. Such approaches involve designing the double crossover operation and mutation for job assignments, and the local search operation for job batch processing sequences. Additionally, an LP model is established to determine the optimal starting time point for each job batch under the given job assignments and job batch processing sequences such that QTLs are met. Once a schedule is found infeasible, a proposed approach is used to regroup jobs to form job batches and adjust the processing sequences of job batches to potentially make it feasible. Comparative experiments on four problem sizes show that DBSA consistently outperforms GA, MixPSO, and GWO, with average cycle time reductions of up to 17.75%, 19.19%, and 9.21%, respectively. Crucially, DBSA achieves the best trade-off between solution quality and algorithmic robustness, yielding the lowest standard deviation across all tested scenarios.

This work is based on known job arrivals and assumes no tool failures or job cancellations, generating a static baseline schedule before the production horizon (e.g., a multi-day shift). Such a centralized offline planning layer provides a robust and consistent reference for performance evaluation and algorithmic comparison. However, by design, it is not intended to directly handle real-time disturbances such as unexpected machine failures or urgent order insertions. While the proposed DBSA establishes a high-quality baseline for OWB-MF scheduling under these assumptions, its computational cost and offline nature may limit responsiveness in large-scale and highly dynamic production environments. Specifically, the computation time exhibits a notable increase with problem size (e.g., reaching approximately 3.5 h for large-scale instances). This growth is primarily attributed to the combinatorial explosion of batch configurations and the increasing number of decision variables in large-scale systems. Furthermore, the fitness evaluation mechanism of DBSA relies on solving an LP model for each individual, which, although enabling rigorous QTL enforcement, introduces a non-negligible computational overhead. Consequently, the overall efficiency is sensitive to the LP model size. Future research may explore decomposition techniques or surrogate-assisted approximations to further reduce this computational burden.

To address the limitations regarding responsiveness, future research will also focus on developing real-time scheduling and dispatching strategies that complement the static baseline within an APS system. Possible extensions include the integration of DBSA into a rolling-horizon architecture or hybrid online–offline frameworks. By combining the global optimality of the offline baseline with the flexibility of real-time decision layers, the system can achieve adaptive responses to evolving shop-floor conditions such as tool failures and urgent order insertions. In addition, similar to many meta-heuristic approaches, the decision logic of DBSA can be perceived as a black box by human operators. To improve interpretability and trust, we plan to incorporate Explainable AI (XAI) techniques, such as feature attribution analysis, to extract transparent decision rules from the evolved schedules. This dual focus aims to ensure that future scheduling systems are not only algorithmically efficient but also adaptive, interpretable, and transparent for practical deployment in manufacturing environments.

Acknowledgement: The authors would like to thank the Institute of Systems Engineering and the Collaborative Laboratory for Intelligent Science and Systems, Macau University of Science and Technology, for providing computational resources and technical support. The authors also appreciate the assistance from IKAS Holdings (Beijing) Co., Ltd. and AscenPower Semiconductors Co., Ltd. for sharing domain expertise and industrial data related to diffusion scheduling.

Funding Statement: This research was funded by Science and Technology Development Fund (FDCT), Macau SAR (file Nos. 0199/2024/AGJ, 0120/2024/RIA2).

Author Contributions: Conceptualization, Liangchao Chen and Yan Qiao; methodology, Liangchao Chen; algorithm development, Liangchao Chen; software, Siwei Zhang; validation, Siwei Zhang, Yonghua Shao and Sijun Zhan; data curation, Siwei Zhang and Bin Liu; investigation, Liangchao Chen and Bin Liu; resources, Bin Liu; writing—original draft preparation, Liangchao Chen; writing—review and editing, Yan Qiao; visualization, Liangchao Chen; supervision, Yan Qiao; project administration, Yan Qiao; funding acquisition, Yan Qiao. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Data available within the article.

Ethics Approval: Not applicable.

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

Appendix A

Appendix A.1 Linear Programming Model for Fitness Evaluation

Model Formulation

Before presenting the LP model, two key parameters are defined. $\boldsymbol{OMB}_{mbl}$ is used to represent a set of operations that are performed by the l-th processed job batch of type b at Tool m, where $m \in \boldsymbol{M}$, $b \in \boldsymbol{B}_m$, $l \in \{1, \ldots, N_{mb}\}$. In fact, once the processing sequence of batch types is determined, the processing sequence of job batches is established accordingly. Under this situation, we use $\boldsymbol{\varepsilon}_{mh}$ to represent a set of operations performed by the h-th processed job batch at Tool m, where $m \in \boldsymbol{M}$ and $h \in \{1, \ldots, \sum_{b}^{|\boldsymbol{B}_m|} N_{mb}\}$. The established LP model is formulated as follows, with two types of constraints: 1) job-specific constraints, as shown by Constraints (A2)–(A4), and 2) tool-specific constraints, as shown by Constraints (A5)–(A7).

$$\text{Minimize}\Gamma = \sum_{j=1}^{|J|} T_j. \tag{A1}$$

Subject to:

$$t_{sj1} \geq ART_j, \forall j \in \boldsymbol{J}_t, \tag{A2}$$

$$t_{sj(i+1)} \geq t_{eji}, \forall j \in \boldsymbol{J}_t, i \in \{1, \ldots, NO_j - 1\}, \tag{A3}$$

$$t_{eji} + q_{ji} \geq t_{sj(i+1)}, \forall j \in \boldsymbol{J}_t, i \in \{1, \ldots, NO_j - 1\}, \tag{A4}$$

$$t_{sji} \geq AVT_m + C_{rr'}, \forall m \in \boldsymbol{M}, j \in \boldsymbol{J}_t, i \in \{1, \ldots, NO_j\}, O_{ji} \in \boldsymbol{\varepsilon}_{m1}, r = r_{ji}, r' = r_m \tag{A5}$$

$$t_{sj1} \geq ARTj, \forall j \in \boldsymbol{J}_t, t_{sji} \geq t_{ej'i'} + C_{rr'}, \forall m \in \boldsymbol{M}, h \in \{1, \ldots, \sum_{b}^{|\boldsymbol{B}_m|} N_{mb} - 1\}, j, j' \in \boldsymbol{J}_t, i \in \{1, \ldots, NO_j\},$$
$$i' \in \{1, \ldots, NO_{j'}\}, O_{j'i'} \in \boldsymbol{\varepsilon}_{mh}, O_{ji} \in \boldsymbol{\varepsilon}_{m(h+1)}, r = r_{ji}, r' = r_{j'i'} \tag{A6}$$

$$t_{sj1} \geq ART_j, \forall j \in \boldsymbol{J}_t, t_{sji} = t_{sj'i'}, \forall m \in \boldsymbol{M}, b \in \boldsymbol{B}_m, l \in \{1, \ldots, N_{mb}\}, j, j' \in \boldsymbol{J}_t, i \in \{1, \ldots, NO_j\},$$
$$i' \in \{1, \ldots, NO_{j'}\}, O_{ji}, O_{j'i'} \in \boldsymbol{OMB}_{mbl}. \tag{A7}$$

1) Job-specific constraints (Process Flow and Yield Assurance):

Constraints (A2) and (A3) establish the fundamental physical timeline: a job cannot start before its arrival (release time) and should strictly follow its predefined operation sequence. Constraint (A4) addresses the critical QTLs. Unlike standard scheduling bounds, this constraint enforces a hard upper limit on the waiting time between sensitive steps. Industrially, this serves as a critical yield-protection mechanism: strict adherence effectively prevents native oxide regrowth and surface contamination, thereby significantly reducing the need for wafer re-cleaning or scrapping in diffusion areas.

2) Tool-specific constraints (Resource Availability and Batching):

Constraints (A5) and (A6) govern machine availability and setup requirements, ensuring that processing only begins after the tool becomes available and any required recipe switching is completed. Finally, Constraint (A7) enforces batch synchronization, mandating that all jobs within a formed batch start processing simultaneously. In practice, this logic prevents resource deadlocks and eliminates unnecessary idle time caused by fragmented batch sequencing, ensuring high throughput stability on batch-processing equipment.

With Constraints (A2)–(A7), the sum of cycle times of all jobs can be calculated by the objective function, where the cycle time for each job is calculated as the completion time point of its last operation minus its arrival time point, given by $T_j = t_{ejNOj} - ART_j$, $\forall\, j \in \boldsymbol{J}_t$.

Theoretical Justification of the LP Formulation

In the proposed framework, each individual generated by DBSA determines the discrete structure of a schedule, including job-to-tool assignments and the processing sequence of job batch types on each tool. Once these discrete decisions are fixed, the remaining problem reduces to determining feasible start times for all operations subject to technological precedence, machine availability, setup transitions, batching synchronization, and QTL constraints.

Under this fixed partial schedule $\mathbb{C}$, the LP model optimizes only continuous start-time variables. Constraints (A2)–(A7) explicitly characterize all temporal feasibility conditions, including precedence relations, QTL upper bounds, machine capacity and setup requirements, and batch synchronization. Therefore, the feasible region of the LP coincides with the admissible timing space under $\mathbb{C}$.

If a feasible schedule exists under $\mathbb{C}$, its timing variables necessarily satisfy Constraints (A2)–(A7), and thus constitute a feasible solution to the LP. Conversely, any feasible LP solution directly defines a complete schedule consistent with $\mathbb{C}$ and satisfying all technological and QTL constraints. Hence, the LP neither excludes feasible schedules nor admits infeasible ones under the given structure.

Since both the objective function (minimizing the total cycle time, i.e., the sum of job cycle times) and all constraints are linear, the LP defines a polyhedral feasible region over continuous variables. Consequently, any optimal solution returned by a linear programming solver is globally optimal within this region. Because QTL requirements are enforced as hard constraints, the optimization is carried out strictly within the QTL-feasible region. Therefore, the LP yields the minimum total cycle time among all schedules consistent with the fixed discrete decisions encoded by the individual.

Table A1: Job classification based on the job family, recipe, and priority.

Job:	(Job family, Recipe)	Priority
1	(1, 1)	10
2	(1, 2)	9
3	(2, 2)	6
4	(1, 1)	7
5	(2, 1)	8
6	(1, 2)	1
7	(1, 2)	2
8	(2, 2)	4
9	(1, 1)	8
10	(1, 2)	6
$CM_m = 2$, $\boldsymbol{B}_m = \{1, 2, 3, 4\}$, $\boldsymbol{J}_{m1} = \{1, 4, 9\}$, $N_{m1} = 2$, $\boldsymbol{J}_{m2} = \{2, 6, 7, 10\}$, $N_{m2} = 2$, $\boldsymbol{J}_{m3} = \{3, 8\}$, $N_{m3} = 1$, $\boldsymbol{J}_{m4} = \{5\}$, $N_{m4} = 1$		

Appendix A.2 Example of How Jobs Are Grouped into Job Batches

Suppose that there are ten jobs to be processed at Tool m, i.e., Jobs 1–10. The priorities of jobs are represented by using Arabic numbers and a higher numerical value indicates a higher priority. Also, the job family and recipe of each job are shown in Table A1. It follows from the table that there should be four batch types such that $\boldsymbol{B}_m = \{1, 2, 3, 4\}$. Specifically, Jobs 1, 4, and 9 belong to batch type 1 ($\boldsymbol{J}_{m1} = \{1, 4, 9\}$), Jobs 2, 6, 7, and 10 belong to batch type 2 ($\boldsymbol{J}_{m2} = \{2, 6, 7, 10\}$), Jobs 3 and 8 belong to batch type 3 ($\boldsymbol{J}_{m3} = \{3, 8\}$), and Job 5 belong to batch type 4 ($\boldsymbol{J}_{m4} = \{5\}$). The maximal number of jobs that the Tool m can process at a time

is two, i.e., $CM_m = 2$. As a result, the number of job batches corresponding to batch types 1, 2, 3, and 4 can be determined, and we have $N_{m1} = 2$, $N_{m2} = 2$, $N_{m3} = 1$, and $N_{m4} = 1$, respectively. Since $N_{m1} > 1$ and $N_{m2} > 1$, the jobs in $\boldsymbol{J}_{m1}$ and $\boldsymbol{J}_{m2}$ should be sequenced in descending order based on their priorities. Then, for job batch type 1, Jobs 1 and 9 together form the first job batch, while Job 4 forms the second job batch. For job batch type 2, Jobs 2 and 10 form the first job batch, while Jobs 7 and 6 form the second job batch.

Appendix A.3 Parameter Settings by Applying Taguchi Method

For the proposed DBSA, the parameters that need to be set are three: Group size, Iterations for DBSA, and Iterations for local search. Three different levels of these three parameters are shown in Table A2. To find the most suitable parameters, the Taguchi method is applied. To do so, an L9(3^3) orthogonal array can be generated by Taguchi experimental design as shown in Table A3. There are nine combinations of different factors of parameters in total and each combination is called a trial. Consequently, a combination experiment can be defined as running these nine trials once. Moreover, DBSA runs 10 times for this combination experiment for each case independently. There are 360 experiments in total. For each case, relative percentage deviations (RPDs) among the results in each combination experiment are calculated by the following.

$$\mathrm{RPD} = \left(Best_{val} - Best_{best}\right) / Best_{best} \times 100\%. \tag{A8}$$

Table A2: Factors and their levels.

	Level		
Parameter	**1**	**2**	**3**
Population size	50	75	100
Iterations for DBSA	100	150	200
Iterations for local search	30	40	50

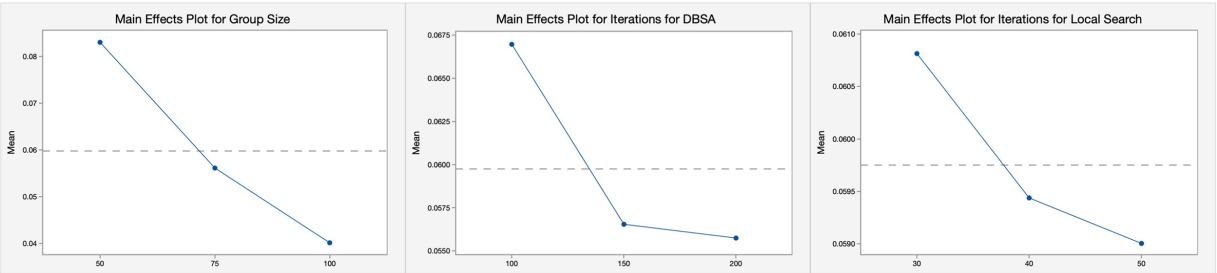
Table A3: Orthogonal arrays.

Tail	**Factor**		
	Population Size	**Iterations for DBSA**	**Iterations for Local Search**
1	50	100	30
2	50	150	40
3	50	200	50
4	75	100	40
5	75	150	50
6	75	200	30
7	100	100	50
8	100	150	30
9	100	200	40

In (A8), $Best_{val}$ represents the result obtained by DBSA in each trial, while $Best_{best}$ indicates the best result across the corresponding combination experiment. Subsequently, the average RPD value can be calculated for each trial by running the experiment for ten times, defined as *Response Variation* (RV). Further,

an average RV value for each trial can be calculated, representing the average RVs of four cases. It is important to note that a smaller average RV value indicates a better performance of the DBSA.

Note that the combination of parameters with the best performance of DBSA may not be anyone of these nine trials. Therefore, after the average RV of four cases for each trial is obtained, the most suitable factor of each parameter can be determined by the *analysis of means* (ANOM). ANOM is used to describe the central tendency of numerical data which is known as the calculation of arithmetic average. The mean of the level of each parameter can be achieved by a software called Minitab Express 1.5. By doing so, the factor level trend of the three parameters is shown in Fig. A1. It can be observed from Fig. A1b that there is not a significant difference in the mean values between iterations for DBSA set as 150 and 200. To better save computational resources, the decision was made to select 150 as the iterations for DBSA. As a result, the parameter settings for the DBSA are: group size = 100, iterations for DBSA = 150, and iterations for local search = 40.



(a) (b) (c)

Figure A1: Factor level trends of three key parameters (**a**) Effect of group size. (**b**) Effect of iterations for DBSA. (**c**) Effect of iterations for local search.

Appendix A.4 Experimental Results

Table A4: A summary of experimental results for small-sized problems.

k	DBSA		GA		MixPSO		GWO	
	Fitness Value (min)		Fitness Value (min)		Fitness Value (min)		Fitness Value (min)	
	30 Jobs	50 Jobs	30 Jobs	50 Jobs	30 Jobs	50 Jobs	30 Jobs	50 Jobs
1	50,560	106,823	56,886	140,962	57,772	136,398	57,424	132,602
2	48,867	110,272	58,861	138,302	58,060	134,950	56,775	132,143
3	50,777	110,167	57,354	137,649	55,110	143,795	56,133	130,700
4	51,319	117,033	55,712	142,125	57,226	137,469	55,955	131,403
5	48,907	121,850	54,322	138,081	56,413	139,065	58,181	130,932
6	49,792	108,659	58,013	139,581	58,904	137,114	54,574	128,313
7	51,685	104,633	58,111	138,634	58,662	134,519	57,587	126,548
8	47,820	114,605	57,729	141,338	57,838	138,708	53,320	135,526
9	53,089	115,882	53,771	138,390	57,074	134,286	54,177	134,686
10	51,723	129,782	53,368	136,526	58,257	132,301	54,093	128,694
11	49,921	121,358	57,998	137,210	57,673	137,820	56,254	132,566
12	49,680	121,069	58,403	131,847	57,371	140,152	53,754	131,309
13	49,260	119,800	58,182	135,717	57,015	137,506	56,470	132,037
14	49,178	114,506	56,498	130,800	58,640	140,053	55,148	126,303

(Continued)

Table A4 (continued)

k	DBSA		GA		MixPSO		GWO	
	Fitness Value (min)		Fitness Value (min)		Fitness Value (min)		Fitness Value (min)	
	30 Jobs	50 Jobs	30 Jobs	50 Jobs	30 Jobs	50 Jobs	30 Jobs	50 Jobs
15	52,166	118,923	57,538	136,525	58,838	138,396	55,735	132,884
16	51,313	117,689	58,197	132,678	57,237	134,648	56,348	129,099
17	49,610	120,905	57,690	133,310	57,288	139,740	53,284	129,668
18	47,151	113,182	55,755	132,041	54,849	138,670	55,115	135,715
19	46,660	116,130	56,701	129,791	56,838	138,924	54,797	126,671
20	50,678	112,989	56,499	132,457	58,163	136,980	54,565	130,868
21	50,271	120,112	56,092	131,869	56,489	138,680	54,420	129,872
22	50,874	117,839	55,140	128,772	57,742	137,529	54,920	134,806
23	47,683	120,261	55,773	129,811	58,954	136,957	54,501	127,388
24	50,450	117,118	54,755	130,715	58,412	141,013	56,363	130,451
25	51,599	115,530	58,603	133,912	58,086	136,124	54,709	129,344
26	49,374	116,394	57,720	133,662	56,593	139,273	53,588	133,447
27	49,282	117,371	57,642	135,094	60,812	137,698	56,885	128,745
28	48,213	116,111	57,561	139,319	58,354	134,477	55,174	130,120
29	48,229	118,957	56,347	139,376	58,668	140,168	57,598	129,937
30	48,696	115,469	58,360	135,853	59,241	137,447	58,805	129,507
Running time	4743.5 s	7117.1 s						

Table A5: A summary of experimental results for large-sized problems.

k	DBSA		GA		MixPSO		GWO	
	Fitness Value (min)		Fitness Value (min)		Fitness Value (min)		Fitness Value (min)	
	75 Jobs	100 Jobs	75 Jobs	100 Jobs	75 Jobs	100 Jobs	75 Jobs	100 Jobs
1	160,053	256,700	182,316	314,519	187,593	301,465	184,379	290,968
2	153,082	263,028	188,708	302,681	186,714	332,274	183,453	313,294
3	158,328	269,742	184,296	297,749	190,095	320,507	175,004	299,597
4	155,227	254,710	181,943	313,402	189,135	301,956	180,771	274,196
5	152,920	252,006	175,790	329,380	182,321	286,475	183,938	291,280
6	152,355	247,655	173,762	305,543	189,104	282,028	175,606	270,186
7	151,819	249,712	185,817	295,109	181,723	296,388	174,514	277,672
8	146,056	258,248	186,455	286,169	189,262	302,223	180,181	276,531
9	145,796	266,347	187,198	291,588	189,032	292,103	185,800	278,362
10	159,248	246,455	182,647	303,479	181,931	276,098	184,811	278,562
11	141,341	269,963	185,907	292,437	177,690	290,540	179,999	286,739
12	151,704	267,766	185,271	291,221	182,740	295,520	174,934	272,002
13	138,477	266,433	183,108	295,967	182,889	283,571	176,078	272,491

(Continued)

Table A5 (continued)

k	DBSA		GA		MixPSO		GWO	
	Fitness Value (min)		Fitness Value (min)		Fitness Value (min)		Fitness Value (min)	
	75 Jobs	100 Jobs	75 Jobs	100 Jobs	75 Jobs	100 Jobs	75 Jobs	100 Jobs
14	144,295	263,425	177,961	286,952	186,409	299,973	185,327	283,341
15	145,418	264,553	180,551	286,050	181,681	290,220	178,105	281,176
16	150,403	246,667	180,171	277,456	187,331	286,366	173,189	278,912
17	136,713	251,274	175,834	272,269	182,449	290,624	176,086	271,714
18	146,200	253,551	180,095	269,492	185,259	281,962	177,471	269,182
19	139,071	239,225	172,535	277,448	183,814	302,295	181,376	277,291
20	142,930	239,501	178,365	273,695	177,233	273,052	168,429	273,632
21	145,122	240,643	173,458	271,962	173,123	282,735	166,547	278,579
22	152,891	241,781	175,405	279,486	179,823	295,066	168,823	278,036
23	139,959	242,050	169,546	291,255	177,771	290,830	160,946	275,156
24	156,046	231,542	182,286	297,593	176,225	292,143	176,927	268,604
25	141,678	254,573	166,561	298,396	178,749	295,139	169,621	283,686
26	155,531	259,359	173,801	283,038	174,391	300,410	166,780	281,517
27	139,549	258,925	176,271	273,423	178,029	290,041	174,183	263,197
28	137,857	247,267	180,490	277,776	180,541	276,930	170,306	274,301
29	150,094	246,982	178,504	268,567	185,763	289,953	174,775	278,090
30	139,594	247,018	181,606	280,092	183,046	288,524	164,242	279,396
Running time	8813.9 s	12,722.6 s						

References

1. Psarommatis F, Azamfirei V. Zero Defect Manufacturing: a complete guide for advanced and sustainable quality management. J Manuf Syst. 2024;77:764–79. doi:10.1016/j.jmsy.2024.10.022.
2. May MC, Oberst J, Lanza G. Managing product-inherent constraints with artificial intelligence: production control for time constraints in semiconductor manufacturing. J Intell Manuf. 2024;35(8):4259–76. doi:10.1007/s10845-024-02472-6.
3. Park IB, Huh J. A genetic algorithm with feasibility-agnostic encoding and three-phase decoding for scheduling semiconductor manufacturing facilities under queue time limits. Expert Syst Appl. 2026;301(4):130310. doi:10.1016/j.eswa.2025.130310.
4. Mathirajan M, Vimalarani M. Scheduling a BPM with incompatible job-families and dynamic job-arrivals. In: Proceedings of the 2012 IEEE International Conference on Industrial Engineering and Engineering Management. Piscataway, NJ, USA: IEEE; 2012. p. 622–6.
5. Rani MV, Mathirajan M. Meta-heuristics for dynamic real time scheduling of diffusion furnace in semiconductor manufacturing industry. Int J Ind Syst Eng. 2020;34(3):365. doi:10.1504/ijise.2020.105737.
6. Guo C, Jiang Z, Hu H. A hybrid ant colony optimization method for scheduling batch processing machine in the semiconductor manufacturing. In: 2010 IEEE International Conference on Industrial Engineering and Engineering Management; 2010 Dec 7–10; Macao, China. Piscataway, NJ, USA: IEEE; 2010. p. 1698–701. doi:10.1109/IEEM.2010.5674588.
7. Rocholl J, Mönch L, Fowler J. Bi-criteria parallel batch machine scheduling to minimize total weighted tardiness and electricity cost. J Bus Econ. 2020;90(9):1345–81. doi:10.1007/s11573-020-00970-6.

8. Jia ZH, Wang C, Leung JYT. An ACO algorithm for makespan minimization in parallel batch machines with non-identical job sizes and incompatible job families. Appl Soft Comput. 2016;38:395–404. doi:10.1016/j.asoc.2015.09.056.
9. Wu K, Huang E, Wang M, Zheng M. Job scheduling of diffusion furnaces in semiconductor fabrication facilities. Eur J Oper Res. 2022;301(1):141–52. doi:10.1016/j.ejor.2021.09.044.
10. Rani MV, Mathirajan M. Real time scheduling of nonidentical multiple batch processors with machine eligibility restriction. Int J Math Eng Manag Sci. 2021;6(6):1460–86. doi:10.33889/ijmems.2021.6.6.088.
11. Ham M, Raiford M, Dillard F, Risner W, Knisely M, Harrington J, et al. Dynamic wet-furnace dispatching/scheduling in wafer fab. In: The 17th Annual SEMI/IEEE ASMC 2006 Conference; 2006 May 22–24; Boston, MA, USA. Piscataway, NJ, USA: IEEE; 2006. p. 144–7. doi:10.1109/ASMC.2006.1638739.
12. Yugma C, Dauzère-Pérès S, Artigues C, Derreumaux A, Sibille O. A batching and scheduling algorithm for the diffusion area in semiconductor manufacturing. Int J Prod Res. 2012;50(8):2118–32. doi:10.1080/00207543.2011.575090.
13. Yurtsever T, Kutanoglu E, Johns J. Heuristic based scheduling system for diffusion in semiconductor manufacturing. In: Proceedings of the 2009 Winter Simulation Conference (WSC); 2009 Dec 13–16; Austin, TX, USA. Piscataway, NJ, USA: IEEE; 2010. p. 1677–85. doi:10.1109/WSC.2009.5429171.
14. Jung C, Pabst D, Ham M, Stehli M, Rothe M. An effective problem decomposition method for scheduling of diffusion processes based on mixed integer linear programming. IEEE Trans Semicond Manuf. 2014;27(3):357–63. doi:10.1109/TSM.2014.2337310.
15. Scholl W, Domaschke J. Implementation of modeling and simulation in semiconductor wafer fabrication with time constraints between wet etch and furnace operations. IEEE Trans Semicond Manuf. 2000;13(3):273–7. doi:10.1109/66.857935.
16. Rajasekaran N, Arjunwadkar V, Man R. Empirical relationship between cycle time impact and batching on furnaces in semiconductor foundry. In: 2020 31st Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC); 2020 Aug 24–26; Saratoga Springs, NY, USA. Piscataway, NJ, USA: IEEE; 2020. p. 1–4. doi:10.1109/asmc49169.2020.9185224.
17. Schmidt K, Rose O. Simulation analysis of semiconductor manufacturing with small lot size and batch tool replacements. In: 2008 Winter Simulation Conference; 2008 Dec 7–10; Miami, FL, USA. Piscataway, NJ, USA: IEEE; 2008. p. 2118–26. doi:10.1109/WSC.2008.4736309.
18. Li M, Lei D. A shuffled frog-leaping algorithm with competition for parallel batch processing machines scheduling in fabric dyeing process. Comput Model Eng Sci. 2025;143(2):1789–808. doi:10.32604/cmes.2025.064886.
19. Zhang P, Jin M, Wang M, Zhang J, He J, Zheng P. A clustering-aided multi-agent deep reinforcement learning for multi-objective parallel batch processing machines scheduling in semiconductor manufacturing. Meas Control. 2025;58(5):614–31. doi:10.1177/00202940241269643.
20. Kong M, Wang W, Deveci M, Zhang Y, Wu X, Coffman D. A novel carbon reduction engineering method-based deep Q-learning algorithm for energy-efficient scheduling on a single batch-processing machine in semiconductor manufacturing. Int J Prod Res. 2024;62(18):6449–72. doi:10.1080/00207543.2023.2252932.
21. Li K, Wang Z, Xie F, Chen J, Xu L. Resource-constrained scheduling of unrelated parallel machines with release times and setup times. Eng Optim. 2025. doi:10.1080/0305215X.2025.2547946.
22. Kim D, Choi Y, Moon K, Lee M, Lee K, Pinedo M. Iterated greedy algorithm with constraint programming for scheduling steelmaking-continuous casting process. Eur J Oper Res. 2026;331(3):706–18. doi:10.1016/j.ejor.2025.11.023.
23. Geibinger T, Mischek F, Musliu N. Investigating constraint programming and hybrid methods for real world industrial test laboratory scheduling. J Sched. 2024;27(6):607–22. doi:10.1007/s10951-024-00821-0.
24. Chen L, Qiao Y, Wu N, Zhang S, Li J, Liu B. Scheduling analysis of diffusion area in semiconductor manufacturing. In: 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE); 2023 Aug 26–30; Auckland, New Zealand. Piscataway, NJ, USA: IEEE; 2023. p. 1–6. doi:10.1109/CASE56687.2023.10260420.
25. Boussaïd I, Lepagnot J, Siarry P. A survey on optimization metaheuristics. Inf Sci. 2013;237:82–117. doi:10.1016/j.ins.2013.02.041.

26. Mönch L, Roob S. A matheuristic framework for batch machine scheduling problems with incompatible job families and regular sum objective. Appl Soft Comput. 2018;68(2):835–46. doi:10.1016/j.asoc.2017.10.028.
27. Phanden RK, Jain A, Verma R. A genetic algorithm-based approach for flexible job shop scheduling. Appl Mech Mater. 2011;110–116:3930–7. doi:10.4028/www.scientific.net/amm.110-116.3930.
28. Qiao Y, Wu N, He Y, Li Z, Chen T. Adaptive genetic algorithm for two-stage hybrid flow-shop scheduling with sequence-independent setup time and no-interruption requirement. Expert Syst Appl. 2022;208:118068. doi:10.1016/j.eswa.2022.118068.
29. Civicioglu P. Backtracking Search Optimization Algorithm for numerical optimization problems. Appl Math Comput. 2013;219(15):8121–44. doi:10.1016/j.amc.2013.02.017.
30. Wu CH, Chien WC, Chuang YT, Cheng YC. Multiple product admission control in semiconductor manufacturing systems with process queue time (PQT) constraints. Comput Ind Eng. 2016;99(1):347–63. doi:10.1016/j.cie.2016.04.003.
31. Chen L, Zhang S, Wu N, Qiao Y, Zhong Z, Chen T. Optimization of inventory space in smart factory for integrated periodic production and delivery scheduling. IEEE Trans Comput Soc Syst. 2023;10(6):3488–511. doi:10.1109/TCSS.2022.3205699.
32. Caldeira RH, Gnanavelbabu A. An improved backtracking search algorithm for the flexible job shop rescheduling problem with new job insertions. Eur J Ind Eng. 2022;16(1):41. doi:10.1504/ejie.2022.119365.