Computers & Industrial Engineering 208 (2025) 111330

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Spatial decomposition-based iterative greedy algorithm for the multi-resource constrained re-entrant hybrid flow shop scheduling problem in semiconductor wafer fabrication

Feng-Shun Zhou [a,b], Rong Hu [a,b], Bin Qian [a,b,*], Qing-Xia Shang [a,b], Yuan-yuan Yang [a,b], Jian-Bo Yang [c]

[a] Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, PR China
[b] Higher Educational Key Laboratory for Industrial Intelligence and Systems of Yunnan Province, Kunming University of Science and Technology, Kunming 650500, PR China
[c] Alliance Manchester Business School, University of Manchester, Manchester M15 6PB, UK

ARTICLE INFO

Keywords:
Spatial decomposition
Iterated greedy algorithm
Semiconductor manufacturing
Wafer fabrication
Optimized scheduling

ABSTRACT

Wafer fabrication (WF) is the most expensive and complex process in semiconductor production, which critically impacts overall manufacturing costs and semiconductor delivery times. This paper considers a common multi-resource constrained re-entrant hybrid flow shop scheduling problem (MRCRHFSP) in WF. The mixed-integer programming (MIP) model of the MRCRHFSP is established for the first time to minimize the makespan, and a novel spatial decomposition-based iterated greedy (SDIG) algorithm is proposed to address it. In the SDIG, the decoding strategy is developed to attempt to obtain a compact scheduling solution corresponding to each individual. Meanwhile, the new spatial decomposition (SD) method is designed to reasonably decompose the solution space into a series of subspaces. Furthermore, the construction-based exploration is devised to guide the search to the promising regions in each subspace, and then the two-stage deep exploitation utilizing the problem's properties and the reset strategy are developed to execute in-depth and fast exploitation from these promising regions. The test results show that the proposed SDIG has better performance than the state-of-the-art algorithms.

1. Introduction

The semiconductor manufacturing industry is a pillar of the national economy and a crucial sector in the development of information technology. With the rapid development of artificial intelligence, the demand for semiconductors has exploded worldwide (Bang & Kim, 2011; Y. F. Lee et al., 2009). In this context, companies with high productivity, high efficiency, and high quality will gain the initiative in fierce international competition. As wafer fabrication is the most complex and cost-intensive stage of semiconductor production, involving various intricate processes such as photolithography, etching, and deposition (Y. H. Lee & Lee, 2022; Mönch et al., 2018). Therefore, scheduling optimization at this stage is of significant practical importance for enhancing the efficiency and capacity of the entire semiconductor manufacturing system.

In recent years, the modeling of wafer fabrication systems has been divided into two categories: the first type models the common shop configurations in wafer fabrication as reentrant hybrid flow shop scheduling problems (Dong & Ye, 2019; Hekmatfar et al., 2011; Jain et al., 2003). The second category simplifies the entire wafer fabrication process by defining the photolithography stage as the bottleneck stage and other stages as non-bottleneck stages. The total processing time of jobs in non-bottleneck stages is simplified into a fixed travel time, while the bottleneck stage is modeled as a multi-resource constrained parallel machine scheduling problem with sequence-dependent setup times (Kim & Lee, 2016; Y. H. Lee & Lee, 2022). Both categories have certain limitations: The first category lacks consideration for the special characteristics of the photolithography stage, while the second category oversimplifies the production configurations of non-photolithography stages. Therefore, we combined the characteristics of these two categories to establish a mathematical model for the multi-resource

* Corresponding author at: Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, PR China.
E-mail addresses: 20231104003@stu.kust.edu.cn (F.-S. Zhou), ronghu_da@kust.edu.cn (R. Hu), bin.qian@kust.edu.cn (B. Qian), jian-bo.yang@manchester.ac.uk (J.-B. Yang).

https://doi.org/10.1016/j.cie.2025.111330
Received 30 September 2024; Received in revised form 23 May 2025; Accepted 16 June 2025
Available online 7 July 2025

constrained reentrant hybrid flow shop scheduling problem (MRCRHFSP) to more accurately describe the production configurations of wafer fabrication workshops. Since the reentrant hybrid flow shop scheduling problem (RHFSP) has been proven to be NP-hard, and MRCRHFSP can be reduced to RHFSP, MRCRHFSP is also NP-hard. Hence, research on modeling and solving MRCRHFSP has significant theoretical value.

Due to the NP-hard nature of MRCRHFSP, traditional mathematical programming and heuristic methods struggle to balance solution quality and computational efficiency. To address these limitations, we propose a novel metaheuristic algorithm called the Space Decomposition-based Iterative Greedy (SDIG) algorithm for solving the problem. For MRCRHFSP, the solution space resembles a "big valley" terrain, containing numerous valleys of varying depths, with high-quality solutions of diverse structures located in different valleys. Therefore, we introduced a space decomposition (SD) method in SDIG that combines Uniform Manifold Approximation and Projection (UMAP) with k-means clustering to identify and leverage the valley structures in the solution space. This guides the algorithm to perform parallel exploration of multiple subspaces (valleys) and discover promising regions within them. Proper use of the SD method can avoid premature convergence caused by single or similar initial solutions in traditional iterative greedy (IG) algorithms (Chen et al., 2021), reducing the risk of getting stuck in local optima in a short time and improving solution diversity and quality. Furthermore, inspired by recent studies on critical path neighborhood search (H. Ding & Gu, 2020; He et al., 2023) and speed-up evaluation mechanisms for insert neighborhoods (Fernandez-Viagas, 2022; Taillard, 1990), as well as the research gap in MRCRHFSP, we propose the critical path theory and speed-up evaluation method for MRCRHFSP, and based on this, we design a two-stage deep exploitation approach to enable in-depth and rapid search of promising regions in subspaces. The effectiveness of the SDIG is validated through simulation experiments and algorithm comparisons. The main contributions of this paper are as follows:

1. The mixed-integer programming (MIP) model of the MRCRHFSP in wafer fabrication is established for the first time. The decoding strategy considering problem characteristics is developed to try to find a compact scheduling solution corresponding to each individual, ensuring that a smaller fitness value (i.e., the makespan) can be obtained for each individual with a higher probability.
2. The new spatial decomposition (SD) method is designed to perceive the MRCRHFSP's landscape over solution space and decompose the solution space into a series of subspaces. This method uses the UMAP technique to project all individuals in the population from high-dimensional space to low-dimensional space, and then performs the k-means clustering technique on these projected individuals to reasonably obtain several subspaces with certain differences.
3. Considering that the actual landscape over each subspace is different (e.g., flat or rugged), a novel search algorithm, i.e., the proposed SDIG, including the construction-based exploration on the job sequences as well as the two-stage deep exploitation on the operation sequences and their corresponding scheduling solutions, is developed to independently execute both broad and in-depth search in each subspace. This parallel multi-modal search can achieve a good balance between exploration and exploitation.

The remainder of this paper is orgized as follows: Section 2 reviews the relevant literature. Section 3 introduces the practical production background of MRCRHFSP and its MIP model. Section 4 presents the theoretical contributions based on the properties of the MRCRHFSP. Section 5 details the SDIG algorithm designed in this study. Section 6 provides extensive experimental results and discussions. Section 7 summarizes the coanntributions of the entire paper. The appendix presents a concrete example of MRCRHFSP and detailed proof of the theoretical contributions.

2. Literature review

2.1. Wafer fabrication scheduling problems

Wafer fabrication, as a technology-intensive stage in the semiconductor supply chain, can significantly impact the overall supply chain's throughput, cycle time, and responsiveness (Cakici & Mason, 2007). Therefore, researchers have developed various realistic mathematical models to approximate scheduling problems in wafer fabrication and designed effective algorithms to solve them.

Kong et al. (2024) developed a two-stage collaborative green scheduling model for the photolithography and etching stages in wafer fabrication. In this model, the photolithography stage is represented by parallel machines, and the etching stage is represented by batch processors, with considerations for waiting time and capacity constraints. To address this problem, they proposed a metaheuristic algorithm combining a hybrid genetic algorithm with the LPPT-cutting rule. For the photolithography bottleneck stage in wafer fabrication, Ghaedy-Heidary et al. (2024) developed a stochastic flexible job shop scheduling model considering machine processing capabilities, machine specificity, and mask constraints. C.-Y. Lee et al. (2023) proposed a mathematical model for unrelated parallel machine scheduling problems, while Bitar et al. (2016) constructed a mathematical model for unrelated parallel machine scheduling problems with dual resource constraints and sequence-dependent setup times. C.-T. Huang et al. (2025) modeled the wafer fabrication process as a parallel machine scheduling problem with sequence-dependent setup times. This model assumes that all stages of the wafer fabrication process can be completed on specific machines, and there are sequence-dependent setup times between different wafer products. To solve this problem, they proposed a data-driven multi-objective composite scheduling rule combined with NSGA-II. Yeong-Dae Kim et al. (2001) modeled the wafer production process as a hybrid flow line batch scheduling problem. In this model, multiple identical parallel machines form batch processing workstations, and wafer batches are processed sequentially through these workstations to produce the final wafer products. Their primary focus was on batch division and batch sequencing, for which they proposed three effective scheduling rules for solution and comparison. Dong & Ye (2019) developed a distributed reentrant hybrid flow shop model to describe production scheduling problems in wafer fabrication under a distributed manufacturing context and proposed an improved grey wolf algorithm for solving it. Y. H. Lee & Lee (2022) simplified the non-bottleneck stages in wafer fabrication by reducing the processing times of different wafer jobs at these stages into distinct travel times. The bottleneck stage (photolithography stage) was modeled as an identical parallel machine scheduling problem with sequence-dependent setup times. To address this problem, they proposed an end-to-end deep reinforcement learning algorithm for solution.

Based on the literature review, the existing models for scheduling problems in wafer fabrication have two main limitations: (1) Some researchers focus solely on a few specific production stages in wafer fabrication, particularly the photolithography bottleneck stage, often oversimplifying or even neglecting the production configurations of non-bottleneck stages. (2) Other researchers take a macro-level approach to address all production stages while overlooking the resource constraints specific to bottleneck stages. In reality, there is a significant interdependency between bottleneck and non-bottleneck stages. Bottleneck stages must be scheduled based on the completion times of jobs in non-bottleneck stages to avoid excessive job blocking. Similarly, non-bottleneck stages need to schedule jobs according to their release times from bottleneck stages to prevent excessive machine idleness. Both factors significantly impact the efficiency and responsiveness of wafer fabrication. Therefore, we combined the characteristics of existing scheduling models and developed a mathematical model for the reentrant hybrid flow shop scheduling problem that incorporates mask resource constraints, sequence-dependent setup times, and

bottleneck stages, that is, the MIP model of MRCRHFSP, and design an effective algorithm to solve it.

2.2. Dimension reduction visualization methods

Nonlinear dimensionality reduction visualization techniques provide researchers with intuitive insights into high-dimensional data, helping to reveal relationships and trends (Grebennik et al., 2023). For example, Liu et al. (2021) used t-distributed Stochastic Neighbor Embedding (t-SNE) to analyze groundwater geochemistry data. Hozumi et al. (2021) used UMAP and K-means clustering to analyze large-scale SARS-CoV-2 mutation datasets. Ding et al. (n.d) applied ISOMAP for dimensionality reduction and classification of hyperspectral images. Additionally, some researchers have used dimensionality reduction visualization techniques to display the distribution and evolutionary process of individuals in the solution space when solving various COPs. Lutton et al. (2012) employed the ScatterDice visualization tool to map all solutions generated by intelligent optimization algorithms onto a two-dimensional plane, analyzing the algorithms' search capabilities. Jornod et al. (2015) developed an open-source tool called SwarmViz for particle swarm optimization algorithms. This tool uses the Sammon mapping method to visualize and monitor the evolution direction of the particle swarm algorithm. Collins (2003) used principal component analysis (PCA) to develop a genetic algorithm visualization tool called Gonzo. Grebennik et al. (2023) utilized t-SNE to visualize permutation solutions in COPs, illustrating 5,040 solutions produced during the evolution of the algorithm for the traveling salesman problem (TSP) in the form of a heatmap on a two-dimensional plane. Michalak (2019) extended the t-SNE method to design a technique called Low-Dimensional Euclidean Embedding (LDEE), which visualizes combinatorial search spaces in two-dimensional Euclidean space, providing examples of visualizing all solutions generated by population-based intelligent optimization algorithms during their evolution when solving the four peaks problem, the firefighter problem, the knapsack problem, the quadratic assignment problem, and the TSP.

The above literature shows that current dimensionality reduction visualization techniques have only been used by some researchers to depict the evolutionary process of intelligent optimization algorithms, but they lack guidance on directing the search directions of these algorithms at the solution space level. To address this, we designed the new SD method in SDIG. This method employs UMAP, which is computationally efficient and adaptive to non-uniform data densities (McInnes et al., 2020), to perform dimensionality reduction on individuals from the initial population. Then, K-means clustering is applied to the reduced-dimension individuals to decompose the MRCRHFSP solution space (Fahim et al. 2006), guiding the algorithm to further explore and exploit different subspaces. Thus, the emergence of SDIG fills this research gap and provides a new perspective for the design of intelligent optimization algorithms.

2.3. Iterated greedy algorithm

The iterated greedy (IG) algorithm, known for its simple structure and strong embedding capabilities, was successfully applied to the set covering problem as early as 1995 (Jacobs & Brusco, 1995). In 2007, Ruiz & Stützle. (2007a) first employed the IG algorithm to solve flow shop scheduling problems, demonstrating exceptional performance. Since then, the IG algorithm has been improved by numerous researchers and applied to various scheduling problems.

Zou et al. (2021) proposed a solution speed-up evaluation mechanism to improve the efficiency of the IG algorithm for multi-workshop AGV scheduling problems in matrix manufacturing. Qin et al. (2022) designed a local perturbation strategy based on swap operators and a global perturbation strategy based on semi-swap operators to balance the global and local search capabilities of the IG algorithm. The improved IG algorithm was applied to solve energy-efficient blocking hybrid flow shop scheduling problems. J.-Y. Ding et al. (2015) introduced a series of Tabu lists into the construction scheme of the IG algorithm to prevent redundant searches, thereby enhancing solution diversity and improving search efficiency. Ozsoydan (2021) added a hyper-heuristic variable neighborhood descent local search phase to the IG algorithm and applied it to reconstructed solutions to enhance the algorithm's search depth. Fernandez-Viagas et al. (2018) developed a beam search initialization method to increase the diversity of initial solutions and reduce the risk of the IG algorithm becoming trapped in local optima in a short time. Rodriguez et al. (2013) defined heuristic rules to guide the reconstruction mechanism in the IG algorithm and employed acceptance criteria based on solution randomness to enhance the algorithm's ability to escape local optima, thereby improving its search performance. Qin et al. (2022) proposed two initialization strategies to improve the diversity and quality of initial solutions for distributed heterogeneous hybrid flow shop scheduling problems with blocking constraints. Additionally, a swap-based local search strategy was introduced to enhance the search depth of the IG algorithm.

From the above research, it is evident that researchers have proposed various improvement methods to address the issue of IG algorithms having a single initial solution, which tends to get trapped in local optima during the search process. Some researchers have enhanced the algorithm's ability to escape local optima by introducing perturbation mechanisms, while others have adopted multiple heuristic methods to generate initial solutions, thereby increasing solution diversity and preventing the algorithm from being trapped in local optima in a short time. However, due to a lack of awareness of the solution space landscape, these two methods, while improving the performance of IG algorithms to some extent, remain significantly limited. Perturbation strategies are often blind and unguided, often leading the algorithm into suboptimal solution regions, thereby wasting search resources. Moreover, strategies for generating initial solutions are highly dependent on specific problem structures, lacking sufficient generalizability. Therefore, the SDIG designed in this paper employs the proposed SD method to perceive the landscape of the solution space, decomposing it into subspaces with significant differences. The construction-based exploration strategy and the two-stage deep exploitation process are then designed to further optimize high-quality individuals in these subspaces. This approach ensures diversity in the search regions while avoiding redundant searches on similar individuals, thereby enhancing the algorithm's ability to discover high-quality solutions in complex solution spaces.

3. Problem description and modeling

This section provides a detailed introduction to the wafer fabrication process and establishes the MIP model of MRCRHFSP to minimize the makespan.

3.1. Practical background of the problem

The semiconductor wafer fabrication process comprises several critical steps: cleaning, coating, baking, photolithography, developing, etching, deposition, and ion implantation. Each step involves multiple identical machines and requires multiple cycles.

Cleaning Stage: The wafers undergo wet cleaning and deionized water cleaning to remove contaminants or residues from previous steps on the wafer surface.

Coating Stage: A small amount of photoresist is applied to the center of the wafer, and spin coating is used to evenly distribute the photoresist across the wafer.

Baking Stage: This step reduces the solvent content in the photoresist, making it thicker and more robust, thereby improving its adhesion to the wafer.

Photolithography Stage: A specific mask is used as a negative, and the wafer stage is continuously moved to achieve orderly exposure on

Table 1
Notations applied in the MIP model of MRCRHFSP.

Symbol	Description
Indices	
j, j'	The index of wafer job.
k, k'	The index of process stage.
k^*	The index of photolithography stage.
m	The index of machine.
l	The index of re-entries.
v, w	The index of mask.
Parameters	
WJ	The set of all jobs (i.e., $WJ = \{1, 2, ..., J\}$).
WK	The set of all processing stages (i.e., $WK = \{1, 2, ..., K\}$).
WM	The set of all machines (i.e., $WM = \{1, 2, ..., M\}$.
WM_k	The set of machines available for stage k (i.e., $WM_k = \{m'_k, ..., m_k\}, WM_k \in WM$). m'_k is the number of the first machine in stage k, and m_k is the number of the last machine in stage k.
WL	The set of all processing cycles (i.e., $WL = \{1, 2, ..., L\}$).
$p_{j,l,k}$	The processing time of job j in process stage k in the l-th cycle.
$t_{v,w}$	The switching time between masks v and w.
$r_{j,l}$	The mask number for the photolithography stage at the l-th cycle of job j.
G	The positive large number.
Auxiliary variables	
$S_{j,l,k}$	The continuous variable for the start processing time of job j in stage k of the l-th cycle.
$C_{j,l,k}$	The continuous variable for the completion time of job j in stage k of the l-th cycle.
Decision variables	
$X_{j,m,l,k}$	The binary variable is set to 1 if job j is processed on machine m at stage k of the l-th cycle, and 0 otherwise.
$Y_{j,j',m,l}$	The binary variable is set to 1 if the job j' is processed after the job j on machine m of the l-th cycle, and 0 otherwise.
Objective	
C_{max}	The maximum completion time (i.e., makespan).

the wafer surface.

Developing Stage: The exposed portions of the photoresist are dissolved and removed, leaving behind the desired pattern.

Etching Stage: Wet or dry etching is used to erode the wafer surface that is not covered by the photoresist, transferring the pattern onto the wafer.

Deposition Stage: Physical or chemical vapor deposition is employed to cover the wafer surface with a layer of metal or compound.

Ion Implantation Stage: High-energy ion beams of elements such as boron, phosphorus, and arsenic bombard the deposited surface, embedding ions into the material's lattice to alter its conductivity, forming semiconductors.

By repeatedly undergoing these processes, nanometer-scale transistors and circuits are constructed on the wafer surface.

3.2. Problem modeling

This section defines the symbols, describes the MRCRHFSP in detail, and proposes the corresponding MIP model.

3.2.1. Symbol definition

The definitions of the relevant mathematical symbols involved in the MIP model are listed in Table 1.

3.2.2. MIP model of the MRCRHFSP

Following the contents of Section 3.1, MRCRHFSP can be described as follows: There are N wafer jobs to be processed, which need to go through K processing stages with L re-entrant cycles to form different wafer products. Each processing stage k has M_k identical parallel machines to complete the processing tasks, with different machine configurations at each stage. The processing time of wafer job j in process stage k in l-th cycle is $p_{l,k,j}$. Furthermore, in the photolithography stage, wafer job j requires a specific photomask $r_{j,l}$ to complete the photolithography process during their l re-entrant cycles. Switching between different photomasks v and w involves a setup time $t_{v,w}$. as illustrated in Fig. 1.

The assumptions of MRCRHFSP are summarized as follows:

1. All jobs and machines are available at time 0 and have the same priority.
2. Job processing preemption and interruptions are not considered.
3. Jobs are processed sequentially through all stages, and at each stage, they can only be assigned to one machine for processing.
4. A machine can only process one job at a time.
5. The buffer capacity between different stages is infinite, and the transportation time between stages is not considered.

Based on the problem description, the MIP model for MRCRHFSP can be expressed as:

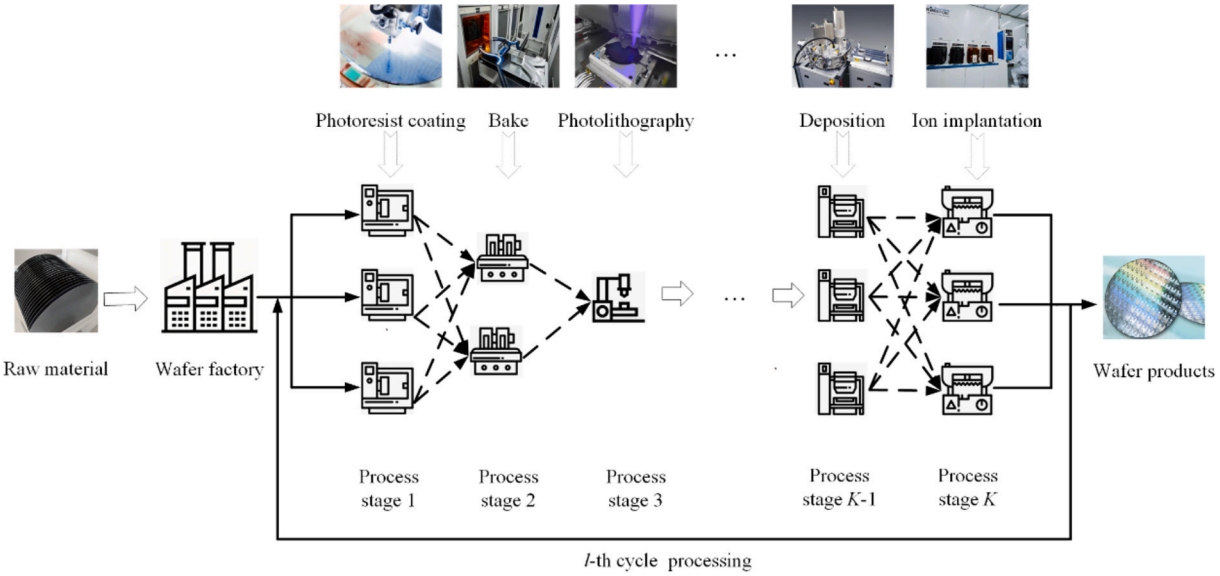


Fig. 1. Schematic diagram of MRCRHFSP in semiconductor wafer fabrication.

Table 2
The symbols applied in the proposed theory.

Symbol	Description
$\pi_{m,l,k}$	The processing operation sequence of jobs on machine m at stage k in the l-th cycle, $\pi_{m,l,k} = \{\pi_{m,l,k}(c) \mid c = 1,2,...,N_{m,l,k}\}$.
$N_{m,l,k}$	The total number of scheduled operations on machine m at stage k in the l-th cycle.
m_k	The total number of machines at stage k.
J	The total number of processing cycles.
L	The total number of processing stages.
K	The total number of jobs.
$o_{l,k,j}$	The processing operation of job j at stage k in the l-th cycle.
CO	The set of job operations on the critical path $CO = \{CO_{1,1}, CO_{1,2}, ..., CO_{1,K}, ..., CO_{1,K}\}$, where $CO_{l,k}$ represents the job operations belonging to the critical path in stage k of the l-th cycle.
N_CO	The set of job operations is not on the critical path, $N\_CO = \{N\_CO_{1,1}, N\_CO_{1,2}, ..., N\_CO_{1,K}, ..., N\_CO_{1,K}\}$, where $N\_CO_{l,k}$ represents the job operations not belonging to the critical path in stage k of the l-th cycle.
$C_{l,k,j}$	The completion time of job j at stage k in the l-th cycle in forward scheduling.
$\overline{C}_{l,k,j}$	The completion time of job j at stage k in the l-th cycle in backward scheduling.
π	A complete scheduling solution, $\pi = \{\pi_{1,1,1}, \pi_{2,1,1}, ..., \pi_{m_1,1,1}, ..., \pi_{m_K,1,K}, ..., \pi_{m_K,L,K}\}$.
π''	The new complete scheduling solution obtained after the operation adjustments
$C_{\max}(\pi)$	The makespan corresponding to scheduling solution π.
$C_{\max}^{new}(\pi'')$	The makespan corresponding to the new scheduling solution.

***Optimization objective*:**

$$Minimize C_{\max} \tag{1}$$

***Subject to*:**

$$C_{\max} \geq C_{j,l,k}, \quad j \in WJ, \quad l \in WL, \quad k \in WK \tag{2}$$

$$\sum_{m=m_k'}^{m_k} X_{j,m,l,k} = 1, j \in WJ, k \in WK, l \in WL \tag{3}$$

$$C_{j,l,k} = S_{j,l,k} + p_{j,l,k}, j \in WJ, l \in WL, k \in WK \tag{4}$$

$$S_{j,l,k} \geq C_{j,l,k-1}, j \in WJ, l \in WL, k \in WK \tag{5}$$

$$S_{j,l,k} \geq C_{j,l-1,k'}, j \in WJ, l \in WL, k, k' \in WK \tag{6}$$

$$S_{j',l,k} \geq C_{j,l,k} - G \times \left(3 - Y_{j,j',m,l} - X_{j,m,l,k} - X_{j',m,l,k}\right), \quad j, j' \in WL, m \in WM_k, l \in WL, k \in WK \cap k \neq k^* \tag{7}$$

$$S_{j,l,k} \geq C_{j',l,k} - G \times Y_{j,j',m,l} - G \times \left(2 - X_{j,m,l,k} - X_{j',m,l,k}\right), \quad j, j' \in WJ, m \in WM_k, l \in WL, k \in WK \cap k \neq k^* \tag{8}$$

$$S_{j',l,k} \geq C_{j,l,k} + t_{r_{j,l},r_{j',l}} - G \times \left(3 - Y_{j,j',m,l} - X_{j,m,l,k} - X_{j',m,l,k}\right), \quad j, j' \in WJ, m \in WM_k, l \in WL, k = k^* \tag{9}$$

$$S_{j,l,k} \geq C_{j',l,k} + t_{r_{j',l},r_{j,l}} - G \times Y_{j,j',m,l} - G \times \left(2 - X_{j,m,l,k} - X_{j',m,l,k}\right), \quad j, j' \in WJ, m \in WM_k, l \in WL, k = k^* \tag{10}$$

$$S_{j,l,k} \geq 0, j \in WJ, l \in WL, k \in WK \tag{11}$$

$$C_{j,l,k} \geq 0, j \in WJ, l \in WL, k \in WK \tag{12}$$

$$X_{j,m,l,k} \in \{0,1\}, j \in WJ, m \in WM_k, l \in WL, k \in WK \tag{13}$$

$$Y_{j,j',m,l} \in \{0,1\}, j, j' \in WJ, m \in WM, l \in WL \tag{14}$$

In this model, Constraint (1) defines the optimization objective as minimizing the makespan. Constraint (2) defines the makespan. Constraint (3) ensures that each job can only be processed on one machine at any given stage. Constraint (4) defines the method for calculating completion times, and Constraint (5) ensures that a job can only begin the next stage after completing the previous stage. Constraint (6) ensures that a job can only start processing in the next cycle after completing all stages of the previous cycle. Constraints (7) and (8) ensure that processing times for different jobs on the same machine do not overlap in non-lithography stages. Constraints (9) and (10) ensure that processing times for different jobs and mask-switching times do not overlap on the same machine in the lithography stages. Constraints (11) to (12) define the auxiliary variables. Constraints (13) to (14) define the decision variables.

To verify the correctness and applicability of the MIP model proposed in this paper, we validate it using the Gurobi solver in the appendix. Computational experiments show that the proposed MIP model accurately captures the constraints of the problem and generates the optimal feasible solution.

4. Proposed theoretical contributions

Before detailing the components of the SDIG algorithm, this section provides a thorough description of some theoretical results we have proposed, including forward scheduling and backward scheduling, critical path, speed-up evaluation, and other related theories, which form the foundation of the SDIG algorithm design. The symbols used in this section are shown in Table 2.

4.1. Forward scheduling and backward scheduling

Definition 1: (Forward Scheduling). For a complete scheduling solution π of the MRCRHFSP, the scheduling process that handles processing operation priorities sequentially in ascending order of stages and cycles is called forward scheduling, and the calculation process of the completion time $C_{j,l,k}$ for jobs at each cycle stage in forward scheduling is shown in Eqs. (15)–(19). A specific example of forward scheduling and its corresponding Gantt chart are provided in the appendix.

$$C_{\pi_{m,0,k}(c),0,k} = C_{\pi_{m,l,k}(c),0,k} = 0, m = 1,2,...,m_k, \ l = 1,2,\cdots,L, c = 1,2,...,N_{m,l,k}, k = 1,2,...,K \tag{15}$$

$$C_{\pi_{m,l,k}(0),l,k} = C_{\pi_{m,l-1,k}\left(N_{m,l-1,k}\right),l-1,k}, m = 1,2,...,m_k, \ l = 1,2,\cdots,L, k = 1,2,...,K \tag{16}$$

$$C_{\pi_{m,l,k}(c),l,0} = C_{\pi_{m,l,k}(c),l-1,K}, m = 1,2,...,m_k, \ l = 1,2,\cdots,L, c = 1,2,...,N_{m,l,k}, k = 1,2,...,K \tag{17}$$

$$C_{\pi_{m,l,k}(c),l,k} = \max\left\{C_{\pi_{m,l,k}(c),l,k-1}, C_{\pi_{m,l,k}(c-1),l,k}\right\} + p_{\pi_{m,l,k}(c),l,k}, \quad m = 1,2,...,m_k, l = 1,2,\cdots,L, c = 1,2,...,N_{m,l,k}, k = 1,2,...,K and k \neq k^* \tag{18}$$

$$C_{\pi_{m,l,k}(c),l,k} = \max\left\{C_{\pi_{m,l,k}(c),l,k-1}, C_{\pi_{m,l,k}(c-1),l,k} + t_{r_{\pi_{m,l,k}(c-1),l},r_{\pi_{m,l,k}(c),l}}\right\} + p_{\pi_{m,l,k}(c),l,k}, \quad m = 1,2,...,m_k, l = 1,2,\cdots,L, c = 1,2,...,N_{m,l,k}, k = k^* \tag{19}$$

Definition 2: (Backward Scheduling). For a complete scheduling solution π of the MRCRHFSP, the scheduling process that handles processing operation priorities in reverse order while descending by stages and cycles is called backward scheduling, and the calculation process of the completion time $\overline{C}_{j,l,k}$ for jobs at each cycle stage in backward scheduling is shown in Eqs. (20)–(24). A specific example of backward scheduling and its corresponding Gantt chart are provided in the appendix.

$$\overline{C}_{\pi_{m,L+1,k}(c),L+1,k} = \overline{C}_{\pi_{m,l,k}(c),L+1,k} = 0, m = 1,2,...,m_k, \ l = 1,2,\cdots,L, c = 1,2,...,N_{m,l,k}, k = 1,2,...,K \tag{20}$$

$$\overline{C}_{\pi_{m,l,k}\left(N_{m,l,k}+1\right),l,k} = \overline{C}_{\pi_{m,l+1,k}(1),l+1,k}, m = 1,2,...,m_k,\ l = 1,2,\cdots,L,k = 1,2,...,K \tag{21}$$

$$\overline{C}_{\pi_{m,l,k}(c),l,K+1} = \overline{C}_{\pi_{m,l,k}(c),l+1,1}, m = 1,2,...,m_k,\ l = 1,2,\cdots,L,c = 1,2,...,N_{m,l,k}, k = 1,2,...,K \tag{22}$$

$$\overline{C}_{\pi_{m,l,k}(c),l,k} = \max\left\{\overline{C}_{\pi_{m,l,k}(c),l,k+1}, \overline{C}_{\pi_{m,l,k}(c+1),l,k}\right\} + p_{\pi_{m,l,k}(c),l,k}$$
$$m = 1,2,...,m_k;\ l = 1,2,\cdots,L; c = 1,2,...,N_{m,l,k}; k = 1,2,...,K and k \neq k^* \tag{23}$$

$$\overline{C}_{\pi_{m,l,k}(c),l,k} = \max\left\{t_{r_{\pi_{m,l,k}(c),l},r_{\pi_{m,l,k}(c+1),l}} + \overline{C}_{\pi_{m,l,k}(c+1),l,k}, \overline{C}_{\pi_{m,l,k}(c),l,k+1},\right\} + p_{\pi_{m,l,k}(c),l,k}$$
$$m = 1,2,...,m_k;\ l = 1,2,\cdots,L; c = 1,2,...,N_{m,l,k}; k = k^* \tag{24}$$

4.2. Critical path

Definition 3 (Critical Path). For scheduling solution π, the longest path from the first operation starting at time 0 to the last operation with a completion time of $C_{\max}(\pi)$, without any idle time in between, is called the critical path.

Based on the above definition, this section presents a theorem and provides its proof.

Theorem 1: Given any instance of the MRCRHFSP and a scheduling solution π, there exists at least one critical path in both the forward and backward scheduling and the path lengths are the same.

Proof. Suppose that in the forward scheduling, the last operation $o_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K}$ in the final stage of the last processing cycle has the maximum completion time $C_{\max}(\pi)$, and its completion time is calculated according to Eq. (18) as shown in Eq. (25):

$$C_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K} = \max\left\{C_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K-1}, C_{\pi_{m,L,K}\left(N_{m,L,K}-1\right),L,K}\right\} + p_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K} \tag{25}$$

This implies that there is at least one operation ($o_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K-1}$ or $o_{\pi_{m,L,K}\left(N_{m,L,K}-1\right),L,K}$) preceding operation $o_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K}$ that seamlessly connects with it. We generalize this operation as $o_{\pi_{m,l,k}(c),l,k}$. If the operation $o_{\pi_{m,l,k}(c),l,k}$ is a non-lithography stage operation, according to Eq. (18), there must be at least one operation ($o_{\pi_{m,l,k}(c),l,k-1}$ or $o_{\pi_{m,l,k}(c-1),l,k}$) preceding the operation $o_{\pi_{m,l,k}(c),l,k}$ that seamlessly connects with it; if the operation $o_{\pi_{m,l,k}(c),l,k}$ is a lithography stage operation, according to Eq. (19), the operation $o_{\pi_{m,l,k}(c-1),l,k}$ combined with the mask switching operation $o_{r_{\pi_{m,l,k}(c-1),l},r_{\pi_{m,l,k}(c),l}}$ is considered as one operation $o_{\pi_{m,l,k}(c-1),l,k} + o_{r_{\pi_{m,l,k}(c-1),l},r_{\pi_{m,l,k}(c),l}}$, and thus, there must also be at least one operation $o_{\pi_{m,l,k}(c),l,k-1}$ or $o_{\pi_{m,l,k}(c-1),l,k} + o_{r_{\pi_{m,l,k}(c-1),l},r_{\pi_{m,l,k}(c),l}}$) preceding the operation $o_{\pi_{m,l,k}(c),l,k}$ that seamlessly connects with it. Through such recursive operations until $l = 1, k = 1, c = 1$, the forward completion time of the operation $o_{\pi_{m,1,1}(1),1,1}$ is calculated as shown in Eq. (26):

$$C_{\pi_{m,1,1}(1),1,1} = \max\left\{C_{\pi_{m,1,1}(1),1,0}, C_{\pi_{m,1,1}(0),1,1}\right\} + p_{\pi_{m,1,1}(1),1,1} \tag{26}$$

Then, according to Eqs. (15)–(17), it follows that $C_{\pi_{m,1,1}(1),1,0} = C_{\pi_{m,1,1}(1),0,K} = 0, C_{\pi_{m,1,1}(0),1,1} = C_{\pi_{m,0,1}(1),0,1} = 0$ in Eq. (26). This means that the start processing time of operation $o_{\pi_{m,1,1}(1),1,1}$ is $C_{\pi_{m,1,1}(1),1,1} - p_{\pi_{m,1,1}(1),1,1} = 0$. Thus, all the operations found during the recursion are seamlessly connected, so the sum of the times for all operations is $C_{\max}(\pi)$, satisfying the definition of the critical path (**Definition 3**).

Similarly, for backward scheduling, suppose that the first stage operation $o_{\pi_{m,1,1}\left(N_{m,1,1}\right),1,1}$ in the first processing cycle has the maximum completion time $C_{\max}(\pi)$. Using Eqs. (23)–(24), we recursively search for operations that seamlessly connect with it until $l = L, k = K, c = N_{m,L,K}$. Then, the backward completion time of the operation $o_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K}$ is calculated as shown in Eq. (27):

$$\overline{C}_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K} = \max\left\{\overline{C}_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K+1}, \overline{C}_{\pi_{m,L,K}\left(N_{m,L,K}+1\right),L,K}\right\} + p_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K} \tag{27}$$

Then, according to Eqs. (20)–(22), it follows that $\overline{C}_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K+1} = C_{\pi_{m,L,K}\left(N_{m,L,K}\right),L+1,1} = 0, \overline{C}_{\pi_{m,L,K}\left(N_{m,L,K}+1\right),L,K} = C_{\pi_{m,L+1,K}(1),L+1,K} = 0$ in Eq. (27). This means that the start processing time of operation $o_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K}$ is $\overline{C}_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K} - p_{\pi_{m,L,K}\left(N_{m,L,K}\right),L,K} = 0$. Thus, all the operations found during the recursion are seamlessly connected, so the sum of the times for all operations is $C_{\max}(\pi)$, satisfying the critical path definition (**Definition 3**).

Therefore, there exists at least one critical path in both the forward and backward scheduling and the path lengths are the same. Thus, **Theorem 1** is proven. A specific example of **Theorem 1** is provided in the appendix.

4.3. Speed-up evaluation

Theorem 2: Given a scheduling solution π for an instance of the MRCRHFSP, removing operation $o_{j,l,k}$ from π results in scheduling solution π' with a makespan of $C_{\max}(\pi')$. If the operation $o_{j,l,k}$ is inserted at position c of machine m in stage k of the l-th cycle, the new scheduling solution π'' is obtained. If k is a non-photolithography stage, the new makespan $C^{new}_{\max}(\pi'')$ is calculated according to Eq. (28); if k is a photolithography stage, the new makespan $C^{new}_{\max}(\pi'')$ is calculated according to Eq. (29).

$$C^{new}_{\max}(\pi'') = \max\left\{C_{\max}(\pi'), \max\left\{C_{j,l,k-1}, C_{\pi_{m,l,k}(c-1),l,k}\right\} + p_{j,l,k} + \max\left\{\overline{C}_{\pi_{m,l,k}(c),l,k}, \overline{C}_{j,l,k+1}\right\}\right\},$$
$$m = 1,2,...,m_k;\ l = 1,2,\cdots,L; c = 1,2,...,N_{m,l,k}; k = 1,2,...,K and k \neq k^* \tag{28}$$

$$C^{new}_{\max}(\pi'') = \max\left\{C_{\max}(\pi'), \max\left\{C_{j,l,k-1}, C_{\pi_{m,l,k}(c-1),l,k} + t_{r_{\pi_{m,l,k}(c-1),l},r_{j,l}}\right\} + p_{j,l,k} + t_{r_{j,l},r_{\pi_{m,l,k}(c),l}} + \max\left\{\overline{C}_{\pi_{m,l,k}(c),l,k}, \overline{C}_{j,l,k+1}\right\}\right\},$$
$$m = 1,2,...,m_k;\ l = 1,2,\cdots,L; c = 1,2,...,N_{m,l,k}; k = k^* \tag{29}$$

Proof. When operation $o_{j,l,k}$ is inserted at position c of machine m in stage k of the l-th cycle, two cases will arise:

1. After inserting the operation $o_{j,l,k}$, the remaining operations in the scheduling solution π' are unaffected.
2. After inserting the operation $o_{j,l,k}$, the start time of some operations in the scheduling solution π' will be delayed.

For the first case, If the insertion point of the operation $o_{j,l,k}$ is at the midpoint of the machine in the entire processing sequence. i.e., $c \neq N_{m,L,K} + 1$, the makespan $C_{\max}(\pi')$ will remain unaffected because other operations are not impacted. Hence $C^{new}_{\max}(\pi'') = C_{\max}(\pi')$, it clearly satisfies Eqs. (28) and (29); If the insertion point of the operation $o_{j,l,k}$ is at the end position of the machine in the entire processing sequence. i.e., $c = N_{m,L,K} + 1$, If after insertion, the completion time of the operation $o_{j,l,k}$ is $C_{j,L,K} > C_{\max}(\pi')$, then $C^{new}_{\max}(\pi'') = C_{j,L,K}$; if $C_{j,L,K} \leq C_{\max}(\pi')$, then $C^{new}_{\max}(\pi'') = C_{\max}(\pi')$; Since the stage in which the operation $o_{j,l,k}$ is inserted is the final stage, it must be a non-photolithography stage, The completion time of the operation $o_{j,l,k}$ can be calculated using Eq. (30) based on the forward scheduling Eq. (18). Moreover, according to the backward scheduling Eqs. (20)–(22), we get $\overline{C}_{\pi_{m,L,K}\left(N_{m,L,K}+1\right),L,K} = \overline{C}_{\pi_{m,L+1,k}(1),L+1,k} = 0$, $\overline{C}_{j,L,K+1} = \overline{C}_{j,L+1,1} = 0$. That is, Eq. (31) is equivalent to Eq. (30), Thus, the completion time of scheduling solution π'' after

inserting operation $o_{j,l,k}$ is calculated as shown in Eq. (32), which is equivalent to Eq. (28).

$$C_{j,L,K} = \max\{C_{j,L,K-1}, C_{\pi_{m,L,K}(N_{m,L,K}),L,K}\} + p_{j,L,k} \tag{30}$$

$$C_{j,L,K} = \max\{C_{j,L,K-1}, C_{\pi_{m,L,K}(N_{m,L,K}),L,K}\} + p_{j,l,k} + \max\{\overline{C}_{\pi_{m,L,K}(N_{m,L,K}+1),L,K}, \overline{C}_{j,L,K+1}\} \tag{31}$$

$$\begin{aligned} C^{new}_{\max}(\pi'') = \max\{C_{\max}(\pi'), C_{j,L,K}\} = \\ \max\Big\{C_{\max}(\pi'), \max\{C_{j,L,K-1}, C_{\pi_{m,L,K}(N_{m,L,K}),L,K}\} + p_{j,l,k} \\ +\max\{\overline{C}_{\pi_{m,L,K}(N_{m,L,K}+1),L,K}, \overline{C}_{j,L,K+1}\}\Big\} \end{aligned} \tag{32}$$

For the second case, if the operation $o_{j,l,k}$ is inserted at position c of machine m in stage k of the l-th cycle, it may delay the start times of operations $o_{\pi_{m,l,k}(c+1),l,k}$ or $o_{j,l,k+1}$. Similarly, it may also delay the start times of operations $o_{\pi_{m,l,k}(c+2),l,k}$ or $o_{j,l,k+2}$. This process continues recursively until $k = K$, $l = L$, $c = N_{m,L,K}$, If operations are consistently delayed during this process, it may result in the makespan of the new scheduling solution being either $C^{new}_{\max}(\pi'') = C_{\pi_{m,L,K}(N_{m,L,K}),L,K} > C_{\max}(\pi')$ or $C^{new}_{\max}(\pi'') = C_{\max}(\pi') \geq C_{\pi_{m,L,K}(N_{m,L,K}),L,K}$. If the first situation occurs, according to **Definition 3**, a new critical path containing operation $o_{j,l,k}$ will form in the scheduling solution π''. According to the forward and backward scheduling calculation rules, for the portion of operations before operation $o_{j,l,k}$ on the critical path, the forward calculation will produce a makespan of $\max\Big\{C_{j,l,k-1}, C_{\pi_{m,l,k}(c-1),l,k}\Big\}$ or $\max\Big\{C_{j,l,k-1}, C_{\pi_{m,l,k}(c-1),l,k} + t_{r_{\pi_{m,l,k}(c-1),l},r_{j,l}}\Big\}$. For the portion of operations after the operation $o_{j,l,k}$ on the critical path, the backward calculation will produce a makespan of $\max\Big\{\overline{C}_{k,\pi^m_k(c)}, \overline{C}_{j,l,k+1}\Big\}$ or $\max\Big\{t_{r_{j,l},r_{\pi_{m,l,k}(c),l}} + \overline{C}_{\pi_{m,l,k}(c),l,k}, \overline{C}_{j,l,k+1}\Big\}\Big\}$. Furthermore, according to the proof of **Theorem 1**, the operations on the critical path are seamlessly connected. Therefore, after inserting operation $o_{j,l,k}$, the makespan $C_{\pi_{m,L,K}(N_{m,L,K}),L,K}$ is calculated as shown in Eqs. (33) and (34). Therefore, the calculation of the makespan $C^{new}_{\max}(\pi'')$ for the new scheduling solution π'' still satisfies Eqs. (28) and (29).

$$\begin{aligned} C_{\pi_{m,L,K}(N_{m,L,K}),L,K} = \max\Big\{C_{j,l,k-1}, C_{\pi_{m,l,k}(c-1),l,k}\Big\} + p_{j,l,k} + \max\Big\{\overline{C}_{\pi_{m,l,k}(c),l,k}, \overline{C}_{j,l,k+1}\Big\} \\ m = 1,2,...,m_k;\ l = 1,2,\cdots,L; c = 1,2,...,N_{m,l,k}; k = 1,2,...,K and k \neq k^* \end{aligned} \tag{33}$$

$$\begin{aligned} C_{\pi_{m,L,K}(N_{m,L,K}),L,K} = \max\Big\{C_{j,l,k-1}, C_{\pi_{m,l,k}(c-1),l,k} + t_{r_{\pi_{m,l,k}(c-1),l},r_{j,l}}\Big\} + p_{j,l,k} + \\ \max\Big\{t_{r_{j,l},r_{\pi_{m,l,k}(c),l}} + \overline{C}_{\pi_{m,l,k}(c),l,k}, \overline{C}_{j,l,k+1}\Big\} \\ m = 1,2,...,m_k;\ l = 1,2,\cdots,L; c = 1,2,...,N_{m,l,k}; k = k^* \end{aligned} \tag{34}$$

If during the recursion process to $k = K$, $l = L$, $c = N_{m,L,K}$, there exists a stage where operations $o_{\pi_{m,l,k}(c+1),l,k}$ or $o_{j,l,k+1}$ are unaffected, then subsequent operations will also be unaffected. Therefore, The completion time $C_{\pi_{m,L,K}(N_{m,L,K}),L,K}$ of operation $o_{\pi_{m,L,K}(N_{m,L,K}),L,K}$ and the makespan $C_{\max}(\pi')$ of the original scheduling solution π' are unaffected, and thus $C^{new}_{\max}(\pi'') = C_{\max}(\pi')$, which still satisfies Eqs. (28) and (29). The proof of **Theorem 2** is complete. A specific example of **Theorem 2** is provided in the appendix.

Theorem 3: Given any scheduling solution π for MRCRHFSP, removing an operation that does not belong to the critical path in any stage results in a scheduling solution π', and then inserting this operation into any position on any machine in that stage forms a new scheduling solution π'', the makespan of the new scheduling solution π'' is $C^{new}_{\max}(\pi'') \geq C_{\max}(\pi)$.

Proof. According to the proof of **Theorem 1**, the operations on the critical path are seamlessly connected, and the sum of times their durations equals the makespan $C_{\max}(\pi)$. Therefore, if the operations on the critical path are not modified, the makespan $C_{\max}(\pi') = C_{\max}(\pi)$ of the scheduling solution π' after removing operations. Then, according to **Theorem 2**, the makespan $C^{new}_{\max}(\pi'')$ of the new scheduling solution π'' after inserting the operation can be calculated using Eqs. (28) and (29). Since $C_{\max}(\pi') = C_{\max}(\pi)$, the makespan after the operation insertion must satisfy $C^{new}_{\max}(\pi'') \geq C_{\max}(\pi') = C_{\max}(\pi)$, thus proving **Theorem 3**. A specific example of **Theorem 3** is provided in the appendix.

5. SDIG algorithm for the MRCRHFSP

Currently, a general framework is commonly followed when solving scheduling problems using metaheuristic algorithms: first, a MIP model is formulated to abstract various constraints and decision variables of the practical scheduling problem into mathematical expressions; then, a metaheuristic algorithm with efficient evolutionary mechanisms and neighborhood search strategies is designed to solve the problem (Fernandez-Viagas, 2022; Sadati & Çatay, 2021; Xu et al., 2024). However, unlike mathematical programming methods, metaheuristic algorithms typically operate on encoded sequences (i.e., permutations of numbers) to explore the solution space, whereas MIP models do not explicitly contain sequence-type variables. Instead, their decision variables are often represented as binary (0–1) variables. This fundamental difference in variable representation appears to have caused a certain degree of disconnection between metaheuristic algorithms and MIP models, and current literature has rarely clarified or explicitly addressed the linkage between the two.

In fact, the encoding–decoding mechanism serves as a key bridge between metaheuristic algorithms and MIP models. Specifically, during the decoding process, a series of decoding rules (see Section 5.1) are applied to map the encoding sequence to a set of decision variables that satisfy the constraints of the MIP model, thereby yielding a feasible solution. Through this mechanism, the 0–1 solution space defined by the original MIP model is transformed into a permutation-based solution space in which the encoding sequence resides. This transformation not only reduces the search space but also inherently ensures solution feasibility. This is also the fundamental reason why metaheuristic algorithms have demonstrated better performance than mathematical programming methods when solving various complex production scheduling problems under an acceptable running time. Therefore, based on the MIP model of the MRCRHFSP problem developed in Section 3, this study proposes a metaheuristic algorithm (i.e., the SDIG algorithm) that incorporates a well-designed encoding–decoding strategy and efficient neighborhood operation mechanisms to effectively solve the MRCRHFSP problem.

Specifically, the search framework of the SDIG algorithm proposed in this paper consists of two main components: the solution space decomposition and the parallel search within subspaces. As for the former part, the novel spatial decomposition (SD) method is developed to reasonably partition the solution space into a series of subspaces. For the latter part, the construction-based exploration is designed, which destructs and constructs the best-known optimal individuals (i.e., job sequences) in each subspace, and decodes them through heuristic rules, so as to drive the search to promising areas as soon as possible. As the solution quality obtained through heuristic decoding still leaves room for improvement, the two-stage deep exploitation approach is developed to conduct in-depth and rapid exploitation of the promising regions within each subspace. More specifically, in the first stage, the critical path-based multi-neighborhood exploitation is performed on the promising operation sequences decoded from the best-known individuals. In the second stage, the exploitation focuses on the complete scheduling solution determined by the optimal operation sequences identified in the first stage. To enhance the efficiency of the second stage, the specific Insert-based fast neighborhood search is designed,

Table 3
The symbols are applied in encoding and decoding.

Symbol	Description
π^*	A feasible encoded individual (job sequence). $\pi^* = \{\pi^*(j) \mid j = 1,2,...,J\}$.
π_o	The operation sequence is obtained after decoding the encoded individual. $\pi\_o = \{\pi\_o(a) \mid a = 1,2,...,J{\cdot}K{\cdot}L\}$.
π_m	The machine sequence is obtained after decoding the encoded individual. $\pi\_m = \{\pi\_m(b) \mid b = 1,2,...,J{\cdot}K{\cdot}L\}$.
$\pi_{m,l,k}$	The processing operation sequence on machine m at stage k in the l-th cycle is obtained after decoding, $\pi_{m,l,k} = \{\pi_{m,l,k}(c) \mid c = 1,2,...,N_{m,l,k}\}$.
π	The processing operation sequence on each machine corresponding to scheduling solution π, $\pi = \{\pi_{1,1,1}, \pi_{2,1,1}, ..., \pi_{m_1,1,1}, ..., \pi_{m_K,1,K}, ..., \pi_{m_K,L,K}\}$.
$C_{k,m}$	The completion time of machine m at stage k for processing tasks.
$C_{j,l,k}$	The completion time of job j at stage k in the l-th cycle.
M_k	The total number of machines at stage k.
L	The total number of processing cycles.
K	The total number of processing stages.
J	The total number of jobs.
$C_{\max}(\pi)$	The makespan corresponding to the scheduling solution π.

which utilizes the built-in block properties on the critical path to avoid invalid insertions and adopts the speed-up neighbor evaluation method to accelerate the search process. Moreover, when a stagnation condition is met, the reset strategy is added to restart the two-stage exploitation for diversifying exploitation.

5.1. Individual encoding and decoding strategy

The definitions of the relevant mathematical symbols involved in the encoding and decoding of the solution are shown in Table 3.

In the SDIG algorithm, the encoded individual π^* is formed by arranging the jobs in a specific order. During decoding, scheduling is performed based on the first available machine (FAM) rule and the first-out-first-in (FOFI) rule. The FOFI rule establishes the priority sequence of jobs for the next stage based on the completion times of jobs in the current stage. The priority sequence for the first stage is determined by the encoded individual (i.e., the randomly generated initial job sequence). This rule naturally ensures that a job must complete its operations in the previous stage before advancing to the next stage. Furthermore, in re-entrant scheduling scenarios, the rule ensures that the next cycle cannot commence until the final stage of the previous cycle is completed, effectively addressing the reentrant property of the problem. The FAM rule assigns jobs to the earliest available machine capable of processing them in the current stage based on the determined job priority sequence. If multiple machines are available simultaneously, one is selected randomly, as they share the same priority. Additionally, machine allocation strictly adheres to machine capability constraints, considering only machines capable of processing the current job. This ensures that constraints related to machines with varying capabilities are automatically satisfied during decoding. As outlined in the proposed MIP model for MRCRHFSP, the optimization objective is to minimize the makespan. Once the job sequence and machine allocation are determined, the start times for each job at every stage are scheduled as early as possible while satisfying all constraints. At this point, a compact scheduling solution π can be obtained, which includes sequence-dependent setup times and mask-related resource constraints, depending on the job sequence. This process is illustrated in Fig. 1* in the Appendix, with the detailed decoding procedure described in Algorithm 1.

Algorithm 1: The decoding strategy of the SDIG algorithm

Input: A Feasible encoded individual π^*.
  Initialize: intermediate variable $\pi'\_o = \pi$, $\pi'\_m = \varnothing$, $\pi'\_j = \varnothing$; $\pi_{m,l,k} = \varnothing$; $C_{k,m} = 0, \forall m,k$.
  for l = 1 to L **do**
    for k = 1 to K **do**
      for j = 1 to J **do**
        $m^* = \arg\min C_{k,m}, m \in \{1,2,...,M_k\}$.
        Calculate the completion time $C_{l,k,j}$ of job j assigned to machine m^*.
        $\pi'\_m \leftarrow$ Include m^* into π'_m.
        $C_{m^*} = C_{j,l,k}$.
      end for
      $\pi'\_j \leftarrow$ Arrange jobs in ascending order of their completion times $C_{j,l,k}, j \in \{1,2,...,J\}$.
      $\pi'\_o \leftarrow$ Include π'_j into π'_o.
    end for
  end for
  Operation sequence $\pi\_o = \pi'\_o$.
  Machine sequence $\pi\_m = \pi'\_m$.
  for i = 1 to $J \cdot K \cdot L$ **do**
    $j = \pi\_o(i)$, $m = \pi\_m(i)$, $l = i \,//\, (J \cdot K)$, $k = i \bmod J$.
    $\pi_{m,l,k} \leftarrow$ Include j into $\pi_{m,l,k}$.
  end for
  $\pi = \{\pi_{1,1,1}, \pi_{2,1,1}, ..., \pi_{m_1,1,1}, ..., \pi_{m_K,1,K}, ..., \pi_{m_K,L,K}\}$.
  Makespan $C_{\max}(\pi) = \max\{C_{L,K,j}\}, j \in \{1,2,...,J\}$.
Output: π_o, π_m, $C_{\max}(\pi)$, π.

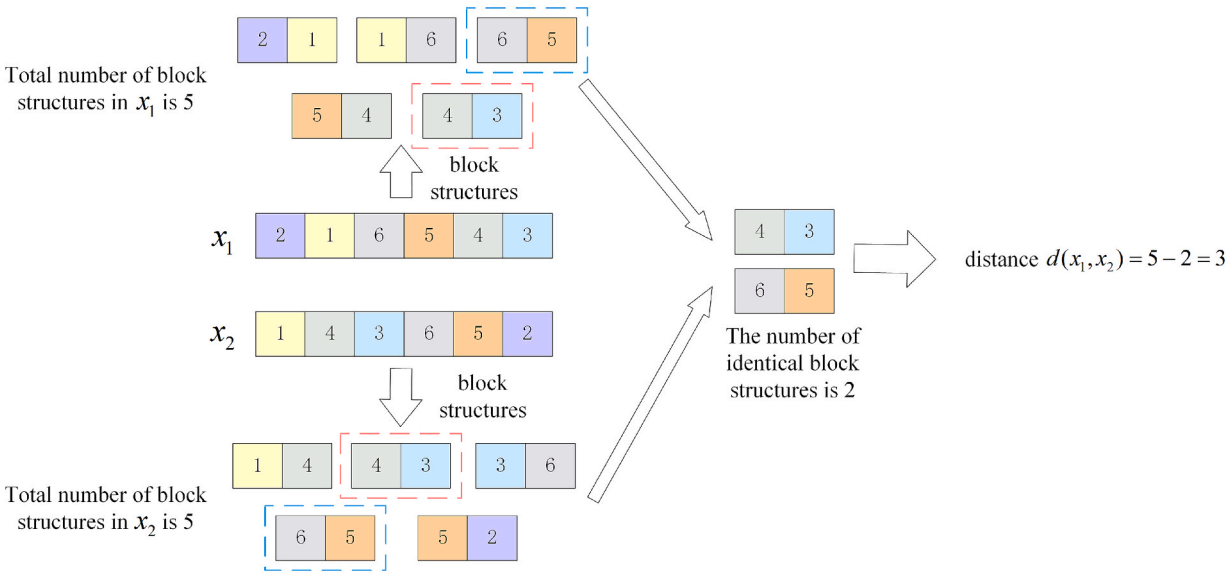


Fig. 2. Calculation of distance between two equal-length sequences.

5.2. Spatial decomposition

5.2.1. Uniform manifold approximation and projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP), proposed by McInnes et al. (2020), is a nonlinear dimensionality reduction method based on a graph-based algorithm. The method operates under three key assumptions: (1) the data are uniformly distributed on a Riemannian manifold, (2) the Riemannian metric is locally constant, and (3) the manifold is locally connected. The essential idea of UMAP is to create a predefined k-dimensional weighted UMAP graph representation of each of the original high-dimensional data points such that the edge-wise cross-entropy between the weighted graph and the original data is minimized. Finally, the k-dimensional eigenvectors of the UMAP graph are used to represent each of the original data points. In recent years, UMAP has been successfully applied to data analysis in fields such as biology (Hozumi et al., 2021) and environmental science (Yu et al., 2023). However, to date, no research has explored the integration of UMAP with metaheuristic algorithm design.

UMAP takes input data $X = \{x_1, x_2, \cdots, x_N\}$, $x_i \in \mathbb{R}^M$ and seeks an optimal low-dimensional representation $Y = \{y_1, y_2, \cdots, y_N\}$, $y_i \in \mathbb{R}^K$, where $K < M$.

The first stage of UMAP involves constructing a k-neighbor graph, and defining a distance metric $d : X \times X \to \mathbb{R}^+$, where k is a hyper-parameter, and $k << M$. Calculate the distance $d(x_i, x_j), 1 < j < k$ of the k-nearest neighbors of each x_i. For any given x_i, ρ_i and σ_i are defined as illustrated in Eqs. (35) and (36):

$$\rho_i = \min\{d(x_i, x_j) | 1 < j < k, d(x_i, x_j) > 0\} \tag{35}$$

$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0,\ d(x_i, x_j) - \rho_i)}{\sigma_i}\right) = \log_2 k \tag{36}$$

where σ_i is the length scale parameter and ρ_i ensures that at least one point with an edge weight of 1 is connected to x_i.

Define a finite weighted graph $\overline{G} = (V, E, \omega)$, where V is the set of vertices (i.e., X), E is the set of edges $E = \{(x_i, x_j) | 1 \le i \le N, 1 \le j \le k\}$, and w is the weight of the edges, computed as illustrated in Eq. (37).

$$w(x_i, x_j) = \exp\left(\frac{-\max(0,\ d(x_i, x_j) - \rho_i)}{\sigma_i}\right) \tag{37}$$

UMAP defines an undirected weighted graph G using the symmetry of $\overline{G}$. First, let A be the adjacency matrix of the graph G, and the symmetry matrix B can be obtained based on Eq. (38).

$$\boldsymbol{B} = \boldsymbol{A} + \boldsymbol{A}^{\boldsymbol{T}} - \boldsymbol{A} \otimes \boldsymbol{A}^{\boldsymbol{T}} \tag{38}$$

Where T is the transpose of the matrix and $\otimes$ is the Hadamard product. The undirected weighted graph G (UMAP graph) is then defined by the adjacency matrix B.

The UMAP employs gravitational repulsion along the boundary and vertices respectively to evolve an equivalent weighted graph H constructed from the set of points $\{y_i\}, i = 1, 2, \cdots, N$. The gravitational and repulsive forces exerted by vertices i and j at coordinates y_i and y_j are illustrated in Eqs. (39) and (40).

$$\frac{-2ab\|y_i - y_j\|_2^{2(b-1)}}{1 + \|y_i - y_j\|_2^2} w(x_i,\ x_j)\left(y_i - y_j\right) \tag{39}$$

$$\frac{2b}{(\varepsilon + \|y_i - y_j\|_2^2)(1 + a\|y_i - y_j\|_2^{2b})}\left(1 - w(x_i,\ x_j)\right)\left(y_i - y_j\right) \tag{40}$$

where a and b represent hyperparameters and ε is a small value that ensures the denominator is not zero.

Therefore, The objective of UMAP is to identify a low-dimensional equivalent weighted graph H comprising a point set $\{y_i\}, i = 1, 2, \cdots, N$, such that y_i minimizes the edge cross-entropy with the original data, while simultaneously producing a low-dimensional output that accurately reflects the topology of the original data.

The distance metric can be calculated using a variety of methods, including Euclidean distance, Manhattan distance, Minkowski distance, and Chebyshev distance, among others. For COPs, the distance metric between two solutions is often measured using Hamming distance (Bookstein, 2002) and Kendall-τ distance (Cicirello, 2020). Hamming distance is a measure of the number of differing characters at corresponding positions in two equal-length sequences x_1, x_2, as illustrated in Eq. (41). The Kendall-τ distance is defined as the number of pairwise disagreements between two equal-length sequences x_1, x_2, as illustrated in Eq. (42).

$$d_H(x_1,\ x_2) = \sum_{i=1}^{n} (x_1(i) \neq x_2(i)) \tag{41}$$

Table 4
The symbols are applied in the improved k-means algorithm.

Symbol	Description
K	The total number of clusters in clustering
x_i, x_j	Objects are to be clustered, and indexed as i and j.
C	The set of objects to be clustered $C = \{x_1, x_2, \cdots x_n\}$, n is the total number of objects to be clustered.
$x_{i,k}, x_{j,k}$	Objects indexed as i and j in the k-th cluster.
C_k	The k-th cluster set $C_k = \{x_{1,k}, x_{2,k}, \cdots x_{n_k,k}\}$, n_k is the total number of objects in the k-th cluster.
xc_k	The central object of the k-th cluster.
$d_{i,k}$	The Euclidean distance between object x_i and the central object xc_k of k-th cluster
$f(x_i)$, $f(x_{i,k})$	The fitness values (makespan) corresponding to x_i and $x_{i,k}$
α	A hyperparameter that determines the number of clusters K

$$d_\tau(x_1,x_2)=|\{(i,j):(j<i)\wedge[(x_1(i)<x_1(j)\wedge x_2(i)<x_2(j))] \vee(x_2(i)<x_2(j)\wedge x_1(i)<x_1(j))]\}| \quad i,j=1,2,\cdots,n \tag{42}$$

However, the aforementioned distance calculation methods lack consideration of the structural information of solutions in MRCRHFSP. In production scheduling problems, the sequence of adjacent jobs represents the most intuitive structure of the solution, and high-quality subspaces often exhibit similar structures. Therefore, based on the characteristics of such production scheduling problems, we define two adjacent positions in a sequence as a block structure and propose a distance metric method based on block structures, as shown in Eq. (43). Taking two equal-length sequences $x_1=[2,1,6,5,4,3]$ and $x_2=[1,4,3,6,5,2]$ of length $n=6$ in Fig. 2 as an example, both contain $n-1=5$ block structures. There are 2 identical block structures between the block structure $\boldsymbol{block}(x_1)=\{[2,1],[1,6],[6,5],[5,4],[4,3]\}$ in x_1 and the block structure $\boldsymbol{block}(x_2)=\{[1,4],[4,3],[3,6],[6,5],[5,2]\}$ in x_2, so the distance between them is $d(x_1, x_2)=5-2=3$.

$$d(x_1,\ x_2) = (n-1) - \sum_{i=1}^{n-1}\sum_{j=1}^{n-1}((x_1(i),x_2(i+1)) = (x_1(j),x_2(j+1))),i,j=1,\ 2,\ \cdots,\ n-1 \tag{43}$$

5.2.2. Improved k-means algorithm

The k-means algorithm is a typical clustering algorithm in data mining, first introduced by Macqueen (n.d). It is one of the simplest unsupervised learning algorithms. The algorithm randomly selects K initial centroids, assigns data points to the nearest centroid to form clusters, and then updates the centroid to the mean of the points within the cluster. This process iterates until the centroids no longer change significantly, ultimately dividing the data into K clusters, aiming to minimize the sum of squared distances within clusters (Na et al., 2010).

From the above description, it is clear that to apply the k-means clustering algorithm for a reasonable decomposition of the reduced solution space in combinatorial optimization problems, two key issues need to be addressed: 1. Determining the number of clusters K. 2. Determining the method for updating the centroid of each cluster. Therefore, this section improves the conventional k-means clustering algorithm from these two aspects. The symbol definitions corresponding to the improved k-means algorithm are shown in Table 4.

The solution space of the MRCRHFSP is "vast," containing various peaks (local maxima) and valleys (local minima). When solving MRCRHFSP, the objective is to find a sufficiently deep valley and its bottom. Ideally, when clustering and decomposing the solution space of MRCRHFSP, the number of clusters (K value) should correspond to the number of valleys, with the centroid of each cluster located at the bottom of the respective valley. However, in practice, if the K value is too large, it may result in the algorithm performing subsequent searches in relatively shallow valleys after space decomposition, wasting significant search resources and failing to find truly high-quality solutions. Therefore, We use the depth of the valleys, as shown in Eq. (44), as the objective function to determine the K value and select the individual with the lowest fitness in each cluster as the centroid to achieve a reasonable decomposition of the solution space.

$$\min\left(\max\left(f\left(x_{i,k}\right)\right)-f\left(xc_k\right)\right)\ \geq\left(\max\left(f\left(x_i\right)\right)-\min\left(f\left(x_j\right)\right)\right)/\alpha, \quad \forall x_i,x_j\in C,\ \forall x_{i,k}\in C_k, k=1,\ 2,\ \cdots,\ K, \alpha\in \mathrm{R} \tag{44}$$

Algorithm 2: Improved k-means clustering algorithm.

Input: Cluster objects $C=\{x_1,x_2,\cdots x_n\}$, hyperparameter α.
  Initialize: The number of clusters $K=0$.
  repeat
    The number of clusters $K=K+1$, random select K center points $\{xc_1, xc_2, \cdots, xc_K\}$.
    repeat
      Make $C_1=\phi, C_2=\phi, ..., C_K=\phi$.
      for i = 1 to n **do**
        Calculate the distance $d_{i,k}$ between object x_i and the center point xc_k : $d_{i,k}=\| x_i-xc_k \|_2, k\in\{1,2,\cdots,K\}$.
        Determine the cluster label of x_i as $k=\arg\min(d_{i,k}),\ k\in\{1,2,\cdots,K\}$.
        Assign the object x_i to the corresponding cluster: $C_k=C_k\cup\{x_i\}$.
      end for
      for k = 1 to K **do**
        Update the center points of each cluster $xc_k=\arg\min(f(x_{i,k})), x_{i,k}\in C_k$.
      end for
    until The center points of all clusters have not been updated.
  until $\min(\max(f(x_{i,k}))-f(xc_k))\le(\max(f(x_i))-\min(f(x_j)))/\alpha$, $\forall x_i,x_j\in C, \forall x_{i,k}\in C_k, k=1,2,\cdots,K, \alpha\in \mathbf{R}$.
Output: Clustering results $C_1, C_2, \cdots, C_K$.

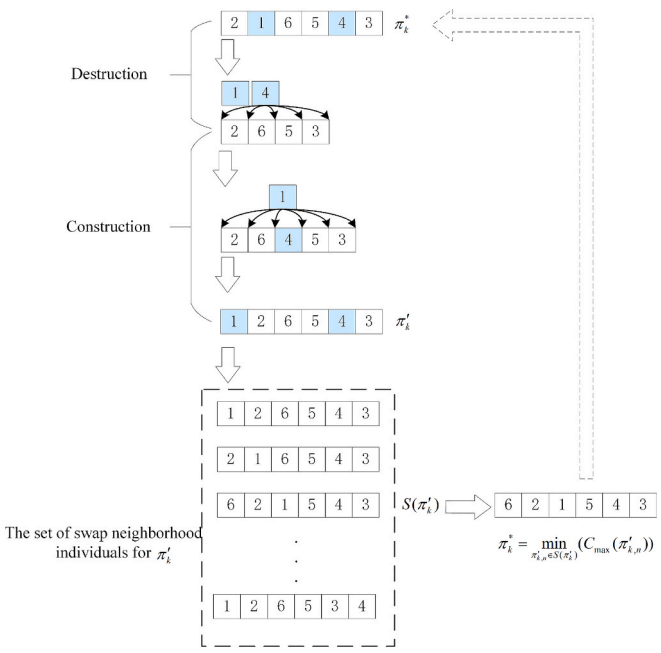


Fig. 3. The construction-based exploration operation process.

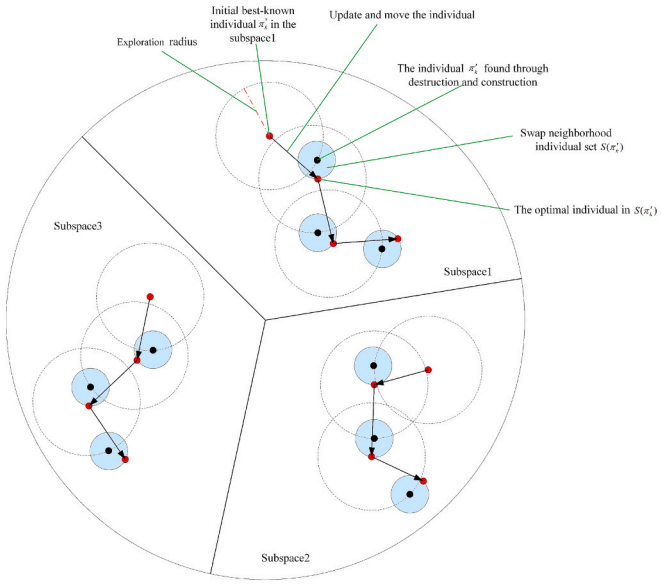


Fig. 4. Explore promising solution regions within the subspace.

Table 5
The symbols are applied in the construction-based exploration.

Symbol	Description
π^*_k	The best-known encoded individual in subspace k
π'_k	The new encoded individual is generated through destruction and construction.
$S(\pi'_k)$	The set of N neighboring individuals of π'_k generated by swap neighborhood operations $S(\pi'_k) = \{\pi'_k, \pi'_{k,1}, \pi'_{k,2}, \ldots, \pi'_{k,N}\}$.
β	The number of destroyed jobs
$C_{\max}(\pi^*_k)$, $C_{\max}(\pi'_k)$	The makespan corresponding to decoded individuals π^*_k and π'_k.

In Eq. (44), $\max(f(x_i)) - \min(f(x_j))$ represents the maximum fitness difference among the objects $C = \{x_1, x_2, \cdots x_n\}$ to be clustered, indicating the height difference between the highest peak and the lowest valley in the entire solution landscape. $(\max(f(x_i)) - \min(f(x_j)))/\alpha$ denotes the depth target of the valleys we aim to identify, and parameter α requires further determination through subsequent parameter experiments. Assuming that the objects $C = \{x_1, x_2, \cdots x_n\}$ to be clustered have been divided into K clusters (valleys), $\max(f(x_{i,k})) - f(xc_k)$ represents the difference in fitness between the object with the highest fitness in the k-th cluster and the cluster center (valley bottom), which is the depth of the k-th cluster (valley). Evidently, as the value of K increases, the depth of each cluster (valley) will gradually become shallower. Therefore, our objective is to find the largest possible value of K while ensuring that the depth of all clusters (valleys) remains greater than $(\max(f(x_i)) - \min(f(x_j)))/\alpha$. The specific process of the improved k-means clustering algorithm is illustrated in Algorithm 2.

5.3. Construction-based exploration

After decomposing the solution space of MRCRHFSP using the SD method described in Section 5.2, the algorithm has achieved an initial perception of the solution space but has not yet identified the truly promising regions within the subspaces. Further exploration of these decomposed subspaces is therefore necessary. The destruction and construction mechanism in the IG algorithm explores the solution space by breaking a candidate feasible solution and reconstructing it (Ruiz & Stützle, 2007). Currently, the IG algorithm has achieved favorable results in solving scheduling problems (J.-Q. Li et al., 2022; Ozsoydan, 2021; Zhao et al., 2022). Inspired by the destruction and construction mechanism of the IG algorithm, this section designs a construction-based exploration operation, as illustrated in Fig. 3, to explore promising regions within the decomposed subspaces, as shown in Fig. 4. The definitions of symbols involved in the construction-based exploration are provided in Table 5, and the detailed process is as follows:

First of all, the destruction and construction are applied to the best-known encoded individual π^*_k in the subspace to generate a new encoded individual π'_k, in order to achieve a rough exploration of the subspace on a large scale. If $C_{\max}(\pi'_k) \leq C_{\max}(\pi^*_k)$, it indicates that a new promising region has been discovered within the subspace. The swap neighborhood operation is used to construct a set $S(\pi'_k) = \{\pi'_{k,1}, \pi'_{k,2}, \ldots, \pi'_{k,N}\}$ that includes π'_k and all its neighboring individuals, and a small-scale detailed exploration is conducted on the promising region. Finally, the best-known individual π_k is moved to the optimal individual within $S(\pi'_k)$, i.e. $\pi^*_k = \min(C_{\max}(\pi'_{k,n}))$, $\pi'_{k,n} \in S(\pi'_k)$. As described in Section 5.2.1, The distance metric method based on block structures shows that the number of jobs destroyed β determines the exploration radius, which dictates the granularity of the subspace exploration. If the radius is too small, it is difficult to escape local optima; if it is too large, the exploration may become too coarse to identify the truly promising regions. Therefore, the number of jobs destroyed β will be determined through parameter experiments. In summary, the specific process of construction-based exploration is illustrated in Algorithm 3.

Algorithm 3: The construction-based exploration

Input: The best-known individuals in each subspace $\{\pi_1^*, \pi_2^*, \cdots, \pi_K^*\}$.
  Initialize: Number of jobs destroyed β. Intermediate variable $\pi\_d = \varnothing$, $\pi\_c = \phi$.
  for k =1 to K do
    $\pi\_c = \pi_k^*$.
    for j = 1 to β do %Destruction
      Remove a randomly selected job from π_c.
      Include the selected job into π_d.
    end for
    for j = 1 to β do %Construction
      Remove a randomly selected job from π_d.
      Insert the selected job into the optimal position of π_c.
    end for
    $\pi_k' = \pi\_c$.
    if $C_{\max}(\pi_k') < C_{\max}(\pi_k)$ **do**
      Using swap neighborhood operation to construct a set of all neighborhood individuals $S(\pi_k') = \{\pi_{k,1}', \pi_{k,1}', ..., \pi_{k,N}'\}$ for π_k'.
      $\pi_{k,n}^* = \arg\min C_{\max}(\pi_{k,n}')$, $\forall\ \pi_{k,n}' \in S(\pi_k')$.
      $\pi_k^* = \pi_{k,n}^*$.
    end if
  end for
Output: The best-known individuals $\{\pi_1^*, \pi_2^*, \cdots, \pi_K^*\}$ in each subspace.

5.4. Two-stage deep exploitation

This paper explores promising solution regions within the subspace through iterative destruction and construction of encoded individuals within the subspace during the construction-based exploration. By decoding the encoded individuals, the corresponding scheduling solutions are obtained. Although the FAM and FOFI rules in the decoding process can ensure the quality of the scheduling solutions to a certain extent, there is still room for improvement. Therefore, this section designs a two-stage deep exploitation method to thoroughly exploit promising solution regions within the subspace. Specifically, starting from the scheduling solution obtained by decoding the best-known encoded individual in the subspace, multi-neighborhood operations are utilized to search the neighboring solutions around this solution, thereby identifying high-quality solutions deeply embedded in this region. Meanwhile, as shown by Theorem 3 in Section 4.3, reducing the makespan (optimization objective) is only possible by adjusting the processing operations of jobs on the critical path. Therefore, all operations in the two-stage deep exploitation designed in this section are based on the critical path, effectively avoiding a large number of invalid neighborhoods and thereby improving the search efficiency of the algorithm.

Since this section is designed based on the theory proposed in Section 4, the definitions of symbols involved in this section are the same as those in Section 4.

5.4.1. Multi-neighborhood exploitation (First stage)

Studies have shown that efficient neighborhood operation strategies can significantly enhance the search depth of the algorithm (Qian et al., 2023). For example, H. Ding & Gu (2020) in their study on the flexible job shop scheduling problem, and He et al. (2023) in their work on multi-objective flow shop group scheduling problems effectively avoided numerous invalid neighborhood operations through critical path-based methods. Moreover, among commonly used neighborhood search operations, swap and insert neighborhood operations generate new solutions that are closest to the original ones (Wang & Qian, 2012), facilitating algorithmic search in a more compact solution space. Based on the above research findings and Theorem 3 proposed in Section 4.3 of this paper, the multi-neighborhood exploitation strategy was developed to perform multi-neighborhood searches on the operation sequence (corresponding to decision variable $Y_{j,j',m,l}$) obtained by decoding the optimal encoded individual in the subspace, thereby achieving the search for promising solution regions in the subspace. The specific process is shown in Algorithm 4.

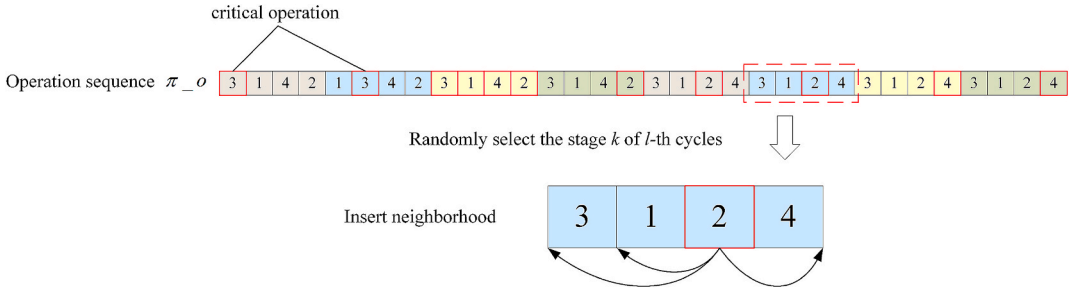


Fig. 5. The swap neighborhood for the operation sequence.

Algorithm 4: The multi-neighborhood exploitation

Input: operation sequence π_o, scheduling solution π and makespan $C_{\max}(\pi)$ of the decoding individual π^*, the set of job operations on the critical path $CO=\{CO_{1,1},CO_{1,2},...,CO_{l,K},...,CO_{l,K}\}$, the set of job operations is not on the critical path $N\_CO=\{N\_CO_{1,1},N\_CO_{1,2},...,N\_CO_{l,K},...,N\_CO_{l,K}\}$.
Initialize: $k=randomInt(1,K)$, $l=randomInt(1,L)$. Flag variable *flag = false.*
for o in $CO_{l,k}$ **do**
  for o' in $N\_CO_{l,k}$ **do**
    Swap the positions of operations o and o' in the operation sequence π_o to obtain a new sequence π'_o.
    Reassign machines to the processing operations based on the operation sequence π'_o and the FAM rule to obtain a new scheduling solution π''.
    Calculate the makespan $C^{new}_{\max}(\pi'')$ of the new scheduling solution π''.
    if $C^{new}_{\max}(\pi'')<C_{\max}(\pi)$ **do**
      $\pi=\pi''$, $\pi\_o=\pi'\_o$
      flag = true
      break
    end if
  end for
  if *flag = true* **do**
    flag = false
    break
  end if
end for
for o **in** $CO_{l,k}$ **do**
  for $i=(l\times K+k)\times J$ to $(l\times K+k+1)\times J$ **do**
    Remove operation o from π_o, and reinsert it into position i of π_o to obtain a new operation sequence π'_o.
    Reassign machines to the processing operations based on the operation sequence π'_o and the FAM rule to obtain a new scheduling solution π''.
    Calculate the makespan $C^{new}_{\max}(\pi'')$ of the new scheduling solution π''.
    if $C^{new}_{\max}(\pi'')<C_{\max}(\pi)$ **do**
      $\pi=\pi''$, . $\pi\_o=\pi'\_o$ ..
      flag = true.
      break
    end if
  end for
  if *flag = true* **do**
    flag = false.
    break
  end if
end for
Output: operation sequence π_o, scheduling solution π, makespan $C_{\max}(\pi)$.

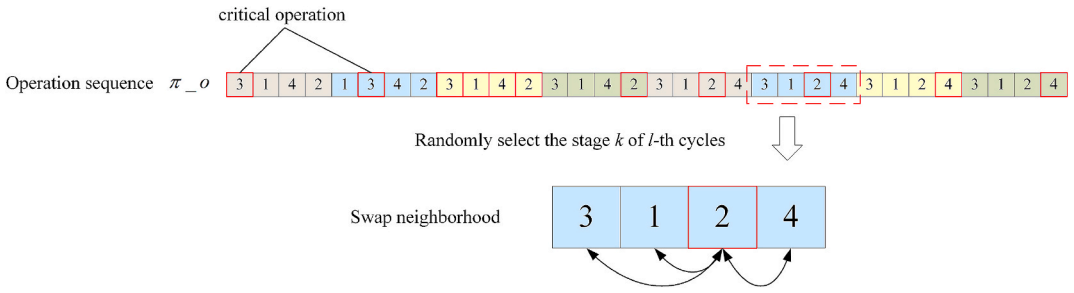


Fig. 6. The insert neighborhood for the operation sequence.

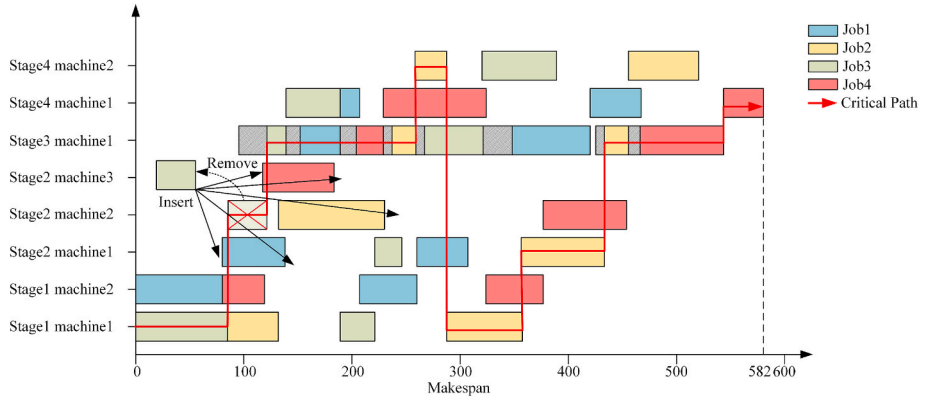


Fig. 7. The *Insert*-based fast neighborhood exploitation.

Swap neighborhood operation: randomly select a processing operation belonging to the critical set in stage k of l-th cycle, swap this operation with all non-critical path operations in stage k of l-th cycle, and reassign machines for the operations based on the FAM rule. Taking the operation sequence obtained from decoding in the appendix as an example, the swap neighborhood operation is illustrated in Fig. 5.

Insert neighborhood operation: Randomly select a processing operation belonging to the critical set in stage k of l-th cycle, remove this operation from the operation sequence, and reinsert it into all possible positions in stage k of l-th cycle. Taking the operation sequence obtained from decoding in the appendix as an example, the insert neighborhood operation is illustrated in Fig. 6.

5.4.2. *Insert-based fast neighborhood exploitation (Second stage)*

After the neighborhood search on the operation sequence in the first stage, we have optimized the operation sequence (corresponding to decision variable $Y_{j,j',m,l}$), but there are still shortcomings in the optimization of machine assignment for operations (corresponding to decision variable $X_{j,m,l,k}$). Therefore, we designed an Insert-based fast neighborhood exploitation method to further optimize the scheduling solution obtained after the first stage of exploitation from the perspective of machine assignment for operations. In this process, using the speed-up evaluation method proposed in Theorem 2 in Section 4.3, we can significantly reduce the complexity of calculating the makespan for the newly generated scheduling solution, thereby accelerating the search efficiency. The Insert-based fast neighborhood exploitation is described as follows: Traverse all the operations in the critical path sequentially, remove the operation from the current machine, and insert it into all possible positions on the machines of the same stage. Then, use the speed-up evaluation method to compute the makespan. If the makespan improves, update the critical path and restart from the first operation on the critical path until the critical path no longer updates and all operations on the critical are traversed. The specific process is shown in Algorithm 5. Using the complete scheduling solution in the appendix as an example, the Insert-based fast neighborhood exploitation is shown in Fig. 7.

5.4.3. *Reset strategy*

The scheduling problem's solution space determined by neighborhoods or operations often has a big-valley landscape, where a large number of local optima are densely scattered in the regions near the bottom of the big valley (Z. C. Li et al., 2019; Qian et al., 2023). In the proposed SDIG, two-stage deep exploitation utilizes several efficient neighborhood searches to try to guide the search from the current promising solution (i.e., the promising region found by the construction-based exploration) in each subspace to the bottom of the big valley. This means that the closer the searched or exploited position is to the bottom of the big valley, the easier it is for the exploitation to fall into some local optima.

Algorithm 5: The Insert-based fast neighborhood exploitation

Input: A complete scheduling solution $\pi = \{\pi_{1,1,1}, \pi_{2,1,1}, ..., \pi_{m_1,1,1}, ..., \pi_{m_K,1,K}, ..., \pi_{m_K,L,K}\}$, makespan $C_{\max}(\pi)$, the set of job operations on the critical path CO.
  Initialize: Flag variable *flag* = *true*.
  while *flag* = *true* **do**
    flag = *false*.
    for o in CO **do**
      Remove operation o from π to obtain a new scheduling solution π'.
      Calculate the forward scheduling of π' (i.e., obtain the completion time $C_{l,k,j}$ for each operation $o_{l,k,j}$).
      Calculate the backward scheduling of π' (i.e., obtain the completion time $\overline{C}_{l,k,j}$ for each operation $o_{l,k,j}$).
      for m = 1 to M_k **do**
        for c = 1 to $N_{m,l,k}$ **do**
          Insert operation o to position c of $\pi_{m,l,k}$ to obtain scheduling solution π''.
          Calculate the new makespan $C^{new}_{\max}(\pi'')$ using the speed-up evaluation method.
        end for
      end for
      $\pi^* = \text{argmin}\, C^{new}_{\max}(\pi''), \forall\, \pi''$.
      if $C^{new}_{\max}(\pi^*) < C_{\max}(\pi)$ **do**
        flag = *true*.
        $\pi = \pi^*$.
        Update the set of job operations on the critical path CO.
        break
      end if
    end for
  end while
Output: scheduling solution π and makespan $C_{\max}(\pi)$.

In the first stage, two-stage deep exploitation randomly selects critical and non-critical operations to construct neighborhood search, Thus, if this exploitation is performed multiple times from the same promising region, each time it is likely to reach the local optimal region at a different depth through a different downward search path. That is, in any subspace, when executing two-stage deep exploitation from the corresponding promising region and falling into a local optimum (i.e., no improvement is observed after the number of $J{\cdot}L{\cdot}K{\cdot}\rho$ iterations, where ρ is a variable parameter), the reset strategy should be adopted to reset the current search position to the original promising region, and then this exploitation should be executed again. This may boost the search to find a local optimal region at a deeper position through other paths, thereby obtaining a better solution. Obviously, combining the reset strategy with two-stage deep exploitation is an effective mode to prevent the proposed SDIG from getting stuck in local optima too early and to achieve deeper exploitation, which is beneficial for improving the SDIG's performance.

Based on these analyses, the above reset strategy is incorporated into the exploitation part of the SDIG.

5.5. Overall procedure of SDIG

Based on the above algorithm description, the proposed SDIG algorithm framework is shown in Fig. 8. The specific process is as follows:

Step 1: Initialization. A population of 200 encoded individuals is randomly generated, and the critical SDIG parameters are initialized. These include the SD parameter α, the subspace exploration radius parameter β, the exploration time ratio parameter γ, and the reset strategy parameter ρ.

Step 2: All encoded individuals in the population are decoded and evaluated. // Algorithm 1.

Step 3: Utilize UMAP to perform dimensionality reduction and visualize the population.

Step 4: Utilize the improved k-means clustering algorithm to cluster and decompose the dimensionally reduced population, achieving solution space decomposition and identifying the set $pop = \{\pi_1^*, \pi_2^*, \cdots, \pi_K^*\}$ formed by the best-known individual in each subspace. // Algorithm 2.

Step 5: Determine whether the exploration termination time has been met. If not, go to Step 6; if met, go to Step 7.

Step 6: Perform construction-based exploration on individuals in *pop*, update $pop = \{\pi_1^*, \pi_2^*, \cdots, \pi_K^*\}$, and return to Step 5. // Algorithm 3.

Step 7: Perform the multi-neighborhood exploitation (first stage exploitation) on the operation sequence corresponding scheduling solution π_n obtained by decoding the individual π_n^* in *pop*. // Algorithm 4.

Step 8: Determine whether the scheduling solution π_n has improved after the first stage. If the scheduling solution π_n has improved, go to Step 9; otherwise, go to Step 10.

Step 9: Perform the *Insert-based* fast neighborhood exploitation (second stage exploitation) on the improved scheduling solution π_n obtained in the first stage, and update the historical global best-known solution. // Algorithm 5.

Step 10: Determine whether π_n meets the reset condition. If this condition is met, proceed to Step 11; otherwise, return to Step 7.

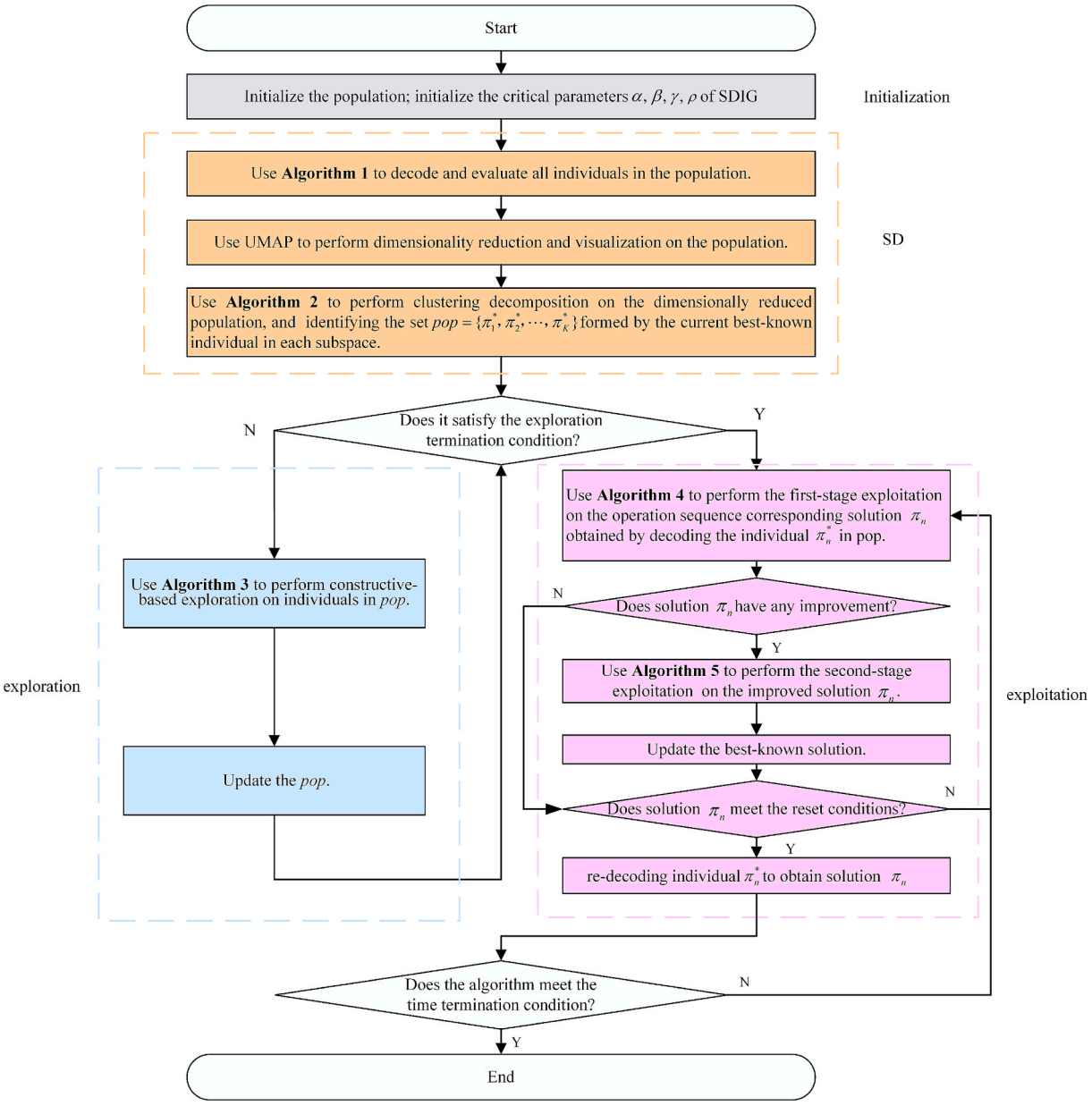


Fig. 8. The flowchart of SDIG for MRCRHFSP.

Table 6
Parameter ranges for test instances.

Parameter	Size
Number of jobs	10, 15, 20
Number of stages	5, 6, 7
Number of reentrances	4, 6, 10
Number of machines in a stage	2, 3
Processing times	*DU*[1,30]
Mask switch time	*DU*[1,5]

Step 11: re-decoding individual π_n^* in the corresponding subspace to obtain scheduling solution π_n.

Step 12: Check if the termination time condition is met. If not, return to Step 7; otherwise, terminate the algorithm and output the historical best scheduling solution.

Table 7
Values of five parameters for each factor level.

Parameters	Factor levels			
	1	2	3	4
α	1.0	1.3	1.6	1.9
β	2	4	6	8
γ	0.1	0.2	0.3	0.4
ρ	0.2	0.4	0.6	0.8

5.6. Time complexity and space complexity of the SDIG algorithm

Before analyzing the time complexity and space complexity of the proposed SDIG algorithm, the meanings of the following symbols need to be clarified: N represents the number of randomly generated initial encoding individuals, and N_k is the number of decomposed subspaces (number of clusters). d represents the dimensionality of the data to be clustered, and t is the number of iterations in the k-means algorithm. J

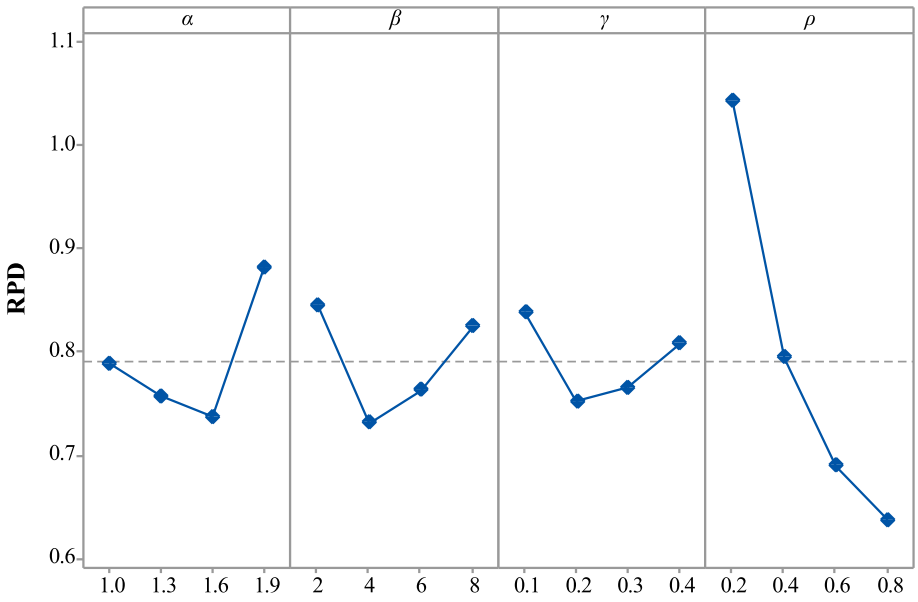


Fig. 9. Main effects plot of SDIG parameters.

Table 8
ANOVA results of SDIG parameters.

Source	*Sum of Squares*	*Df*	*Mean Square*	*F-ratio*	*P-value*
Main effects					
α	0.789061	3	0.26302	12.86	0.0000
β	0.538037	3	0.179346	8.77	0.0000
γ	0.301062	3	0.100354	4.91	0.0026
ρ	6.21302	3	2.07101	101.23	0.0000
Interactions					
α*β	0.301656	9	0.0335173	1.64	0.1069
α*γ	0.174174	9	0.0193527	0.95	0.4865
α*ρ	0.204652	9	0.0227391	1.11	0.3564
β*γ	0.246218	9	0.0273575	1.34	0.2201
β*ρ	0.121263	9	0.0134737	0.66	0.7455
γ*ρ	0.200802	9	0.0223114	1.09	0.3715
Residual	3.86652	189	0.0204578		
Total	12.9565	255			

denotes the number of jobs, K is the number of processing stages, L is the number of re-entries, and M is the total number of machines. β represents the number of jobs destroyed in the construction-based exploration. $G_{\max}$ is the number of iterations of the algorithm within the specified maximum runtime.

According to Fig. 8, the proposed SDIG algorithm consists of the following components: (1) Population initialization, (2) Space decomposition, (3) Construction-based exploration, and (4) Two-stage deep exploitation. In this algorithm, the time complexity of decoding and evaluating (computing the objective function value) for an individual is $O(J{\cdot}L{\cdot}K{\cdot}M)$, and the space complexity is $O(J{\cdot}L{\cdot}K{\cdot}M)$. Therefore, the time complexity of population initialization with N individuals is $O(N{\cdot}J{\cdot}L{\cdot}K{\cdot}M)$, and the space complexity is $O(N{\cdot}J{\cdot}L{\cdot}K{\cdot}M)$. For the space decomposition, the time complexity of the dimensionality reduction and visualization step using the UMAP algorithm is $O(N{\cdot}\log(N))$, and the space complexity is $O(N)$. The time complexity of the k-means clustering step is $O(N{\cdot}N_k{\cdot}t{\cdot}d)$, and the space complexity is $O(N)$. The time

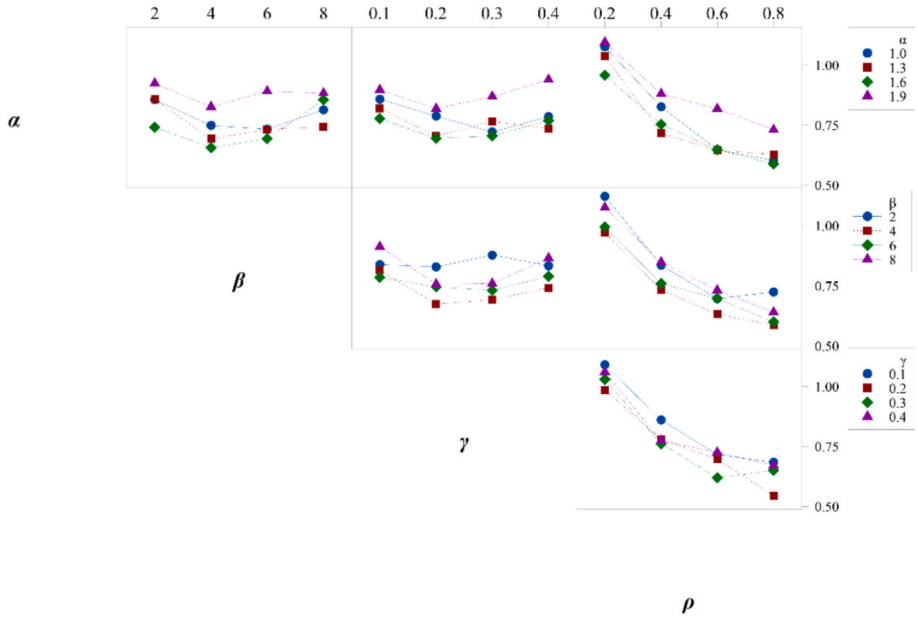
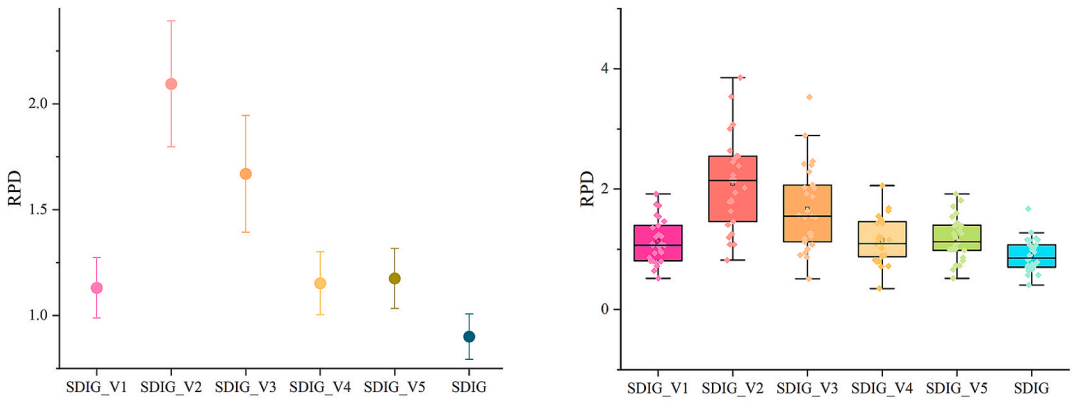


Fig. 10. Interaction effect plot of SDIG parameter pairs.

Table 9
Comparison results of the SDIG with the variants.

Instances	SDIG_v1			SDIG_v2			SDIG_v3			SDIG_v4			SDIG_v5			SDIG		
	BST	*WST*	*AVG*	*BST*	*WST*	*AVG*	*BST*	*WST*	*AVG*	*BST*	*WST*	*AVG*	*BST*	*WST*	*AVG*	*BST*	*WST*	*AVG*
10 × 5 × 4	0.47	2.31	**1.06**	2.56	3.47	2.64	1.86	3.70	2.01	0.93	2.56	1.46	**0.00**	2.55	1.20	**0.00**	**2.08**	**1.06**
10 × 5 × 6	0.49	2.45	1.21	1.80	3.27	2.55	0.82	2.45	1.57	0.16	**1.96**	1.16	0.49	2.61	1.39	**0.00**	**1.96**	**1.00**
10 × 5 × 10	0.62	1.53	0.64	1.75	2.45	1.82	0.92	2.45	1.22	0.82	2.15	1.20	**0.00**	1.43	0.66	**0.00**	**1.23**	**0.57**
10 × 6 × 4	0.44	2.40	1.46	2.40	5.24	3.54	1.31	2.40	1.92	0.66	2.84	1.64	0.44	2.84	1.81	**0.00**	**2.18**	**1.16**
10 × 6 × 6	0.29	1.62	0.81	1.17	3.52	2.23	0.88	1.91	1.26	0.29	1.32	0.90	0.29	2.50	0.98	**0.00**	**1.17**	**0.70**
10 × 6 × 10	0.55	3.94	1.72	1.19	3.76	2.55	0.64	2.84	1.53	0.09	2.94	1.52	0.55	2.39	1.54	**0.00**	**2.11**	**1.28**
10 × 7 × 4	0.36	1.62	0.72	0.54	2.71	1.41	0.54	1.08	0.90	**0.00**	1.08	0.90	0.54	1.44	0.81	**0.00**	**0.90**	**0.70**
10 × 7 × 6	0.42	1.54	1.09	1.26	2.66	1.80	0.98	1.96	1.26	**0.00**	1.82	1.09	0.42	**1.54**	1.07	0.28	1.82	**0.95**
10 × 7 × 10	**0.00**	1.54	0.52	0.16	1.46	0.82	0.16	0.81	0.51	0.08	1.05	**0.35**	**0.00**	0.81	0.52	**0.00**	**0.65**	0.41
15 × 5 × 4	0.17	1.52	0.95	0.51	1.69	1.08	1.52	3.55	2.42	0.51	1.52	1.01	0.17	1.69	1.00	**0.00**	**1.35**	**0.83**
15 × 5 × 6	0.93	2.32	1.36	1.28	3.02	2.15	1.39	3.14	2.46	0.70	2.32	1.68	0.93	2.09	1.44	**0.00**	**1.74**	**1.15**
15 × 5 × 10	0.30	1.80	0.86	0.75	1.65	1.20	0.97	2.09	1.55	**0.00**	1.72	0.79	0.52	1.50	0.99	**0.00**	**1.35**	**0.73**
15 × 6 × 4	0.34	2.39	0.97	1.19	4.09	2.45	0.17	1.70	1.12	0.17	2.04	1.02	0.34	1.87	1.12	**0.00**	**1.36**	**0.75**
15 × 6 × 6	0.92	2.64	1.09	1.49	2.75	2.14	0.92	**1.38**	1.08	0.69	1.72	1.09	0.92	1.72	1.26	**0.00**	1.72	**1.07**
15 × 6 × 10	0.63	1.67	0.93	0.56	1.95	1.25	1.11	1.95	1.53	0.28	1.74	0.88	0.63	1.46	1.00	**0.00**	**1.39**	**0.70**
15 × 7 × 4	0.33	1.99	1.40	2.66	4.98	3.85	1.16	2.49	1.88	0.33	3.49	1.43	0.33	2.49	1.71	**0.00**	**1.33**	**0.73**
15 × 7 × 6	0.33	2.52	1.54	1.20	3.61	2.39	0.88	2.08	1.64	0.44	3.06	1.43	0.33	2.08	1.40	**0.00**	**2.08**	**1.18**
15 × 7 × 10	0.07	1.68	1.05	0.94	2.02	1.46	0.34	1.82	1.18	**0.00**	1.48	0.71	0.40	1.82	1.02	**0.00**	**1.21**	**0.67**
20 × 5 × 4	0.27	1.49	0.80	0.81	3.53	2.51	0.54	1.49	0.87	0.27	**1.09**	0.80	0.27	1.36	0.73	**0.00**	1.36	**0.57**
20 × 5 × 6	1.31	2.71	1.92	2.10	4.20	3.00	2.19	4.47	3.53	0.44	3.15	2.06	1.31	2.71	1.92	**0.00**	**2.45**	**1.67**
20 × 5 × 10	0.50	1.96	1.75	1.06	2.86	1.79	2.30	3.53	2.89	**0.00**	2.02	1.17	0.50	1.90	1.32	0.17	**1.57**	**1.01**
20 × 6 × 4	1.07	2.01	1.25	2.28	4.43	3.07	1.48	2.82	2.07	0.67	2.15	1.21	1.07	3.36	1.60	**0.00**	**1.48**	**0.90**
20 × 6 × 6	0.86	2.39	1.22	0.96	2.87	1.94	1.82	2.87	2.40	0.29	1.82	1.55	0.86	2.11	1.39	**0.00**	**1.72**	**1.16**
20 × 6 × 10	0.60	1.64	1.07	1.04	2.13	1.63	1.42	2.57	2.03	0.22	1.69	1.02	0.60	1.58	1.00	**0.00**	**1.53**	**0.79**
20 × 7 × 4	0.76	2.28	1.57	1.39	2.78	2.02	1.64	3.03	2.29	0.88	2.15	1.49	0.76	2.02	1.26	**0.00**	**1.64**	**1.07**
20 × 7 × 6	**0.00**	1.42	0.80	1.88	2.76	2.15	0.89	1.87	0.94	0.63	1.51	**0.72**	**0.00**	1.60	0.73	0.98	**1.24**	0.85
20 × 7 × 10	0.54	1.13	0.77	0.48	1.51	1.08	0.70	1.34	1.01	0.48	1.18	0.82	0.54	1.29	0.86	**0.00**	**1.02**	**0.65**
Average	0.50	2.02	1.13	1.31	3.01	2.09	1.09	2.35	1.67	0.37	1.96	1.15	0.49	1.95	1.18	**0.05**	**1.55**	**0.90**



(*a*) Interval plots (*b*) Box plots

Fig. 11. Means plots with 95% Tukey's HSD confidence interval and box plots for SDIG against five variants.

complexity of the construction-based exploration is $O(\beta \cdot N_k \cdot J^2 \cdot L \cdot K \cdot M)$, and the space complexity is $O(N_k \cdot J \cdot L \cdot K \cdot M)$. The time complexities of the critical path-based swap neighborhood search and insert neighborhood search in the two-stage deep exploitation are both $O(J^2 \cdot L \cdot K \cdot M)$. Thus, the time complexity of the first-stage exploitation is $O(2 \cdot N_k \cdot J^2 \cdot L \cdot K \cdot M)$ and the space complexity is $O(N_k \cdot J \cdot L \cdot K \cdot M)$. The time complexity of using the speed-up evaluation method to compute the objective function for individuals in the insert-based fast neighborhood exploitation is $O(1)$. Therefore, the time complexity of the Insert-based fast neighborhood exploitation is $O(N_k \cdot J \cdot L \cdot K \cdot M)$, and the space complexity is $O(N_k \cdot J \cdot L \cdot K \cdot M)$.

In summary, the time complexity of the SDIG algorithm is $O(N \cdot J \cdot L \cdot K \cdot M) + O(N \cdot \log(N)) + O(N \times N_k \times t \times d) + G_{\max} \cdot \{O((\beta \cdot J + 2 \cdot J + 1) \cdot (N_k \cdot J \cdot L \cdot K \cdot M))\}$, and the space complexity is $O(N_k \cdot J \cdot L \cdot K)$.

The two-stage deep exploitation in the SDIG algorithm is designed based on the critical path, which effectively avoids a large number of invalid neighborhood operations. It reduces the time complexity of traditional swap and insert neighborhoods from $O(J^3 \cdot L \cdot K \cdot M)$ to $O(J^2 \cdot L \cdot K \cdot M)$. Additionally, the speed-up evaluation method (used only in the insert neighborhood) reduces the complexity of computing the objective function from $O(J \cdot L \cdot K \cdot M)$ to $O(1)$. The combination of the critical path and accelerated evaluation greatly enhances the search efficiency of the algorithm. As a result, the SDIG algorithm can perform more search operations within the same amount of time. This, in turn, demonstrates superior performance in the subsequent experimental sections.

6. Experimental comparisons and statistical analysis

In this section, numerical simulations and experiments on different scales of MRCRHFSP instances are conducted to verify the effectiveness of SDIG in solving the MRCRHFSP problem. First, a detailed experimental setup is introduced in Section 6.1. Subsequently, the impact of critical parameters in SDIG is discussed in Section 6.2. In Section 6.3, five variant experiments are designed to demonstrate the effectiveness of key components in SDIG. Finally, in Section 6.4, SDIG is compared with some current advanced algorithms through comparative statistical analysis.

6.1. Experimental setup

Due to the proprietary nature of real-world wafer production data and the difficulty of accessing such data, we generated test instances for MRCRHFSP by referring to the dataset generation methods of Cho (2011) for reentrant hybrid flow shop scheduling problems and Ham (2018) for dual-resource-constrained lithography scheduling problems. Based on the parameter ranges in Table 6, a total of 27 test instances were created.

To evaluate the performance of the algorithm, the relative percentage deviation (RPD) shown in Eq. (45) is used as a performance metric.

$$RPD = \frac{f_a - f_b}{f_b} \times 100 \tag{45}$$

Where f_b is the best response value generated by all comparison algorithms, and f_a is the response value obtained by the current algorithm. The RPD metric quantifies the discrepancy between the response values of all algorithms and the best response value obtained currently. The smaller the RPD value, the better the algorithm performance. To ensure the fairness of all experiments, the runtime of each algorithm is set to $J^3 \times K \times L \times 0.005$ seconds. Finally, all algorithms are developed using Python 3.9 based on PyCharm and run on a PC with Windows 11 OS, an Intel (R) Core (TM) i5-12400 2.50 GHz CPU, and 16 GB RAM.

6.2. Parameter calibration

Appropriate parameters play a crucial role in the solution quality and computational efficiency of SDIG. There are four key parameters in SDIG: SD parameter α, subspace exploration radius parameter β, and exploration time ratio parameter γ, as well as the reset strategy parameter ρ. This section uses the design of experiments (DOE) method to conduct parameter analysis experiments, thereby determining the parameter values for SDIG.

In this section, four instances with scales of $10 \times 5 \times 6$, $10 \times 6 \times 6$, $15 \times 5 \times 6$, and $10 \times 4 \times 6$ were selected from the MRCRHFSP test dataset for parameter experiments. The parameter level values are shown in Table 7. To statistically reveal the main effects and interaction effects of the parameters, a full factorial experimental design was carried out for all combinations, including $4^4 = 256$ parameter combinations. For each parameter combination, SDIG performed 10 independent experiments with a termination condition of 10 s, and the average fitness value (*AVF*) was calculated according to Eq. (46) as the response value for each parameter combination.

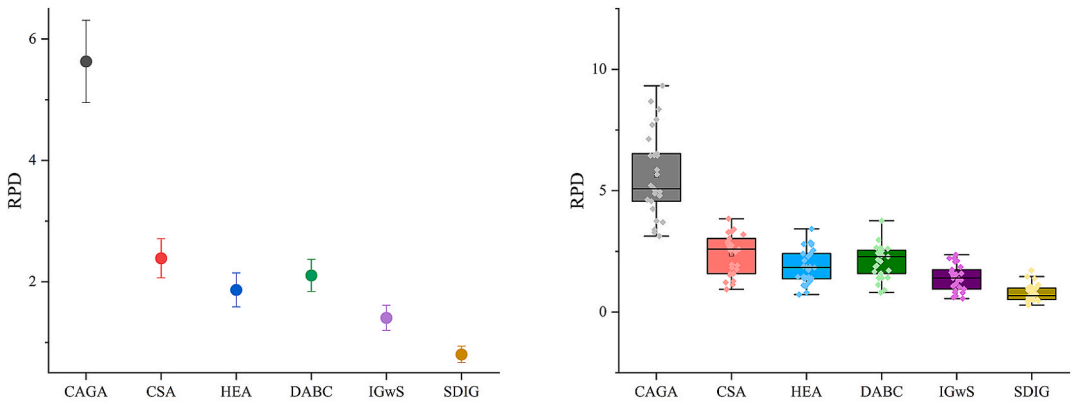
$$AVF = \frac{1}{R} \times \sum_{r=1}^{R} f_r \tag{46}$$

where f_r is the objective function value obtained by the algorithm under the given conditions in the r-th experiment for each instance, and R is the number of experiments.

Analysis of Variance (ANOVA), as a powerful tool for parameter analysis, is used to determine whether there are significant differences among multiple group means. In recent years, it has been widely applied

Table 10
Comparison results of SDIG with the state-of-the-art algorithms.

Instances	CAGA			CSA			HEA			DABC			IGwS			SDIG		
	BST	*WST*	*AVG*	*BST*	*WST*	*AVG*	*BST*	*WST*	*AVG*	*BST*	*WST*	*AVG*	*BST*	*WST*	*AVG*	*BST*	*WST*	*AVG*
10 × 5 × 4	7.87	9.26	8.68	1.16	4.17	2.75	1.62	3.47	2.55	1.85	3.70	2.55	1.16	2.08	1.50	**0.00**	**2.08**	**1.06**
10 × 5 × 6	5.71	7.18	6.46	0.82	2.45	1.76	0.16	2.28	1.45	0.82	2.45	1.71	0.82	1.96	1.32	**0.00**	**1.79**	**0.83**
10 × 5 × 10	7.28	8.00	7.71	1.54	4.00	2.82	0.72	2.56	1.87	1.44	3.18	2.08	1.44	**1.74**	1.60	**0.00**	1.85	**0.98**
10 × 6 × 4	8.73	9.83	9.32	0.00	4.37	1.53	**0.00**	**1.53**	**1.05**	0.66	2.40	1.42	1.31	2.62	1.86	**0.00**	2.40	1.46
10 × 6 × 6	6.30	7.76	7.13	0.73	2.34	1.83	0.88	1.90	1.43	0.88	2.34	1.60	0.29	1.76	0.95	**0.00**	**1.32**	**0.51**
10 × 6 × 10	5.70	6.89	6.44	1.19	3.31	2.47	1.47	2.48	1.84	1.47	3.31	2.51	1.75	3.03	1.75	**0.00**	**1.75**	**1.72**
10 × 7 × 4	3.97	5.96	5.00	0.18	1.44	0.94	0.00	1.08	0.72	0.54	1.62	0.90	0.72	0.90	0.79	**0.00**	**0.90**	**0.47**
10 × 7 × 6	5.74	7.00	6.44	0.84	2.38	1.58	0.84	1.96	1.29	0.56	1.82	1.40	**0.00**	1.68	**0.81**	0.14	**1.40**	0.95
10 × 7 × 10	3.43	4.40	3.75	0.57	1.55	1.22	0.73	1.71	1.17	0.82	2.04	1.40	1.22	1.63	1.40	**0.00**	**1.14**	**0.54**
15 × 5 × 4	4.37	5.55	4.79	2.18	4.54	3.29	2.18	3.70	2.40	1.68	3.87	2.54	1.18	3.53	2.22	**0.00**	**1.01**	**0.44**
15 × 5 × 6	4.84	5.65	5.21	2.08	4.04	3.32	2.31	3.34	2.84	1.38	4.27	2.28	1.73	2.42	2.11	**0.00**	**1.04**	**0.66**
15 × 5 × 10	4.14	4.89	4.63	2.11	3.46	2.89	1.88	3.08	2.41	2.18	2.78	2.65	1.73	2.33	2.08	**0.00**	**2.26**	**1.39**
15 × 6 × 4	7.69	9.06	8.36	1.37	3.76	2.75	1.71	3.25	2.80	1.54	2.91	2.34	0.68	1.71	1.32	**0.00**	**1.37**	**0.65**
15 × 6 × 6	4.23	5.26	4.81	1.03	2.06	1.59	0.69	1.83	1.39	1.14	2.63	1.87	0.34	1.37	**0.61**	**0.00**	**1.03**	0.74
15 × 6 × 10	3.61	4.73	4.25	2.08	3.06	2.54	0.97	1.88	1.47	1.67	3.06	2.34	1.18	2.08	1.63	**0.00**	**1.39**	**0.65**
15 × 7 × 4	6.78	8.43	7.93	1.65	3.31	2.60	1.65	3.31	2.78	0.99	3.97	2.51	0.50	2.15	1.07	**0.00**	**1.49**	**0.89**
15 × 7 × 6	5.01	6.09	5.67	0.87	3.05	1.93	1.20	2.61	1.86	1.09	2.72	1.92	0.54	1.96	1.16	**0.00**	**1.52**	**0.99**
15 × 7 × 10	5.62	7.03	6.53	1.07	2.48	1.96	0.80	1.61	1.10	0.80	1.20	0.80	0.47	1.20	0.89	**0.00**	**0.80**	**0.58**
20 × 5 × 4	2.85	3.66	3.13	0.68	2.44	1.27	0.68	1.76	1.41	0.54	1.90	1.12	0.27	1.22	0.66	**0.00**	**0.68**	**0.47**
20 × 5 × 6	4.67	5.45	5.08	2.59	4.75	3.85	**0.00**	3.20	1.74	3.03	4.24	3.76	1.82	2.68	2.36	**0.00**	**1.38**	**0.60**
20 × 5 × 10	3.10	3.82	3.39	1.66	3.32	2.79	**0.00**	2.71	1.38	1.77	2.43	2.16	0.83	1.77	1.36	0.06	**0.77**	**0.51**
20 × 6 × 4	4.15	6.29	5.86	2.28	4.55	3.32	2.41	4.02	3.43	1.74	4.15	2.97	0.54	2.14	1.55	**0.00**	**1.87**	**0.98**
20 × 6 × 6	4.49	5.45	4.96	2.58	4.40	3.40	2.58	3.35	2.87	1.72	3.54	2.43	1.91	2.39	2.22	**0.00**	**1.72**	**1.12**
20 × 6 × 10	3.26	4.18	3.70	2.50	3.53	3.03	1.47	2.07	1.89	2.17	3.15	2.63	1.09	2.17	1.52	**0.00**	**1.03**	**0.52**
20 × 7 × 4	4.26	5.64	4.89	2.26	4.14	3.20	0.00	3.26	2.29	1.75	3.63	2.64	0.88	2.13	1.59	**0.00**	**1.13**	**0.68**
20 × 7 × 6	3.47	5.25	4.57	2.23	3.65	2.66	1.60	2.49	2.13	2.23	3.12	2.61	0.45	1.87	1.02	**0.00**	**1.78**	**0.98**
20 × 7 × 10	2.89	3.58	3.28	0.80	1.55	1.13	0.27	1.12	0.80	0.86	2.35	1.66	0.32	0.86	0.56	**0.00**	**0.54**	**0.29**
Average	4.97	6.16	5.63	1.45	3.26	2.39	1.07	2.50	1.86	1.38	2.92	2.10	0.91	1.97	1.40	**0.03**	**1.40**	**0.80**



(*a*) Interval polts (*b*) Box polts

Fig. 12. Means plots with 95% Tukey's HSD confidence intervals and box plots for SDIG and five algorithms.

to parameter tuning for various scheduling problem-solving algorithms (Y.-Y. Huang et al., 2021; Pan et al., 2019; Zhang et al., 2022).

The main effects plot of SDIG for the 4 parameters and 4 levels on the test instances are shown in Fig. 9, and the ANOVA results are presented in Table 8. In the ANOVA table, the *P-value* is used to determine whether the parameter is significant, while the *F-ratio* describes the degree of significance. From Table 8, it can be seen that all parameters have *P-values* less than 0.05, indicating that they are significant parameters in the algorithm with statistical significance. Furthermore, based on the ranking of the *F-ratio* values in Table 8, the parameters affecting SDIG performance are ordered by significance from high to low as ρ, α, β, γ. This is consistent with the main effects plot.

Although the main effect plots can easily identify the optimal parameter combination, analyzing the parameter values is meaningless if there is a significant interaction between the parameters. Therefore, we further examined the interaction between the four parameters, as shown in Table 8. From Table 8, it can be seen that the *P*-Values for the interactions between each pair of parameters are all greater than 0.05, indicating that the interactions between the parameters are not significant. At the same time, Fig. 10 shows the interaction plots between the parameters, from which it can be seen that the interactions between the parameters are relatively weak, which is with the conclusions drawn from the table. In addition, the *P*-values for individual parameters are significantly larger than those for interactions, indicating that the interactions between parameters can be ignored. Based on this theoretical foundation and Fig. 9, it is clear that SDIG has the smallest response values when $\alpha = 1.6$, $\beta = 4$, $\gamma = 0.2$, $\rho = 0.8$. Therefore, it can be concluded that SDIG performs better with this parameter combination compared to others. Thus, this parameter combination will be used in the following experiments in this section.

6.3. Performance analysis of key components in the SDIG

To improve the search performance and efficiency of SDIG, four innovations are described in detail in Section 5, specifically, the SD in Section 5.2, the construction-based exploration in Section 5.3, the two-stage deep exploitation in Section 5.4, and the reset strategy in Section 5.5. To verify the effectiveness of these innovations, this section constructs some variants of SDIG for experimental analysis. SDIG_v1 excludes the SD, SDIG_v2 excludes the construction-based exploration, SDIG_v3 excludes the multi-neighborhood exploitation (First stage), SDIG_v4 excludes the *Insert*-based fast neighborhood exploitation (Second stage), and SDIG_v5 excludes the reset strategy.

To ensure the fairness of the experiments, it should be clarified that each variant algorithm only eliminates one phase, with all other parts remaining identical. All algorithms run independently 10 times on each example, and the average objective function value from Eq. (46) is used as the response value of the algorithm, and the relative percentage deviation (RPD) from Eq. (45) is used as the performance indicator of the algorithm. The experimental results are shown in Table 9, with the best-performing values highlighted in bold. To determine whether the differences between SDIG and the other variant algorithms are statistically significant, ANOVA was conducted in this section, and mean plots with 95 % Tukey's HSD confidence intervals and box plots of different algorithms were drawn, as shown in Fig. 11. As can be seen from Table 9, SDIG outperforms its variant algorithms in almost all cases. At the same time, from the interval plots in Fig. 11, it can be seen that the intervals of SDIG and the other variant algorithms do not overlap, indicating that there are significant differences between SDIG and the other variant algorithms, which means that SDIG is significantly better than the other variant algorithms in terms of the RPD performance indicator. In addition, from the box plots in Fig. 11, it can be seen that the box length of SDIG is significantly smaller than that of its variant algorithms, indicating that SDIG has better stability. In summary, all phases of the designed SDIG effectively improve the performance and stability of the algorithm.

The performance analysis of the key components of SDIG is as follows:

(1) The SD method: By decomposing the solution space, multiple subspaces with potential search values can be identified. On this basis, further searches are conducted in the subspaces, preserving solution diversity while only adding minimal computational costs, and avoiding the risk of getting trapped in a single local optimum for a short time. If deep searches are conducted only on the best-known individual in the initial population, it can significantly reduce computational costs but may lead to premature convergence, making the algorithm prone to a single local optimum. On the other hand, using a random method to decompose the solution space increases the diversity of solutions but lacks the perception of the solution space. This may result in further searches on multiple similar individuals that are not in the promising regions, greatly wasting search costs and making it difficult to find truly satisfactory solutions. Therefore, SDIG exhibits superior performance in comparison to SDIG_v1.

(2) The construction-based exploration: After the SD is achieved. The best-known individuals in each subspace may not necessarily be in the promising regions of the current subspace. Therefore, it is necessary to extensively explore the current subspace. The destruction and construction can roughly explore the solution regions near the best-known solution in the subspace while retaining some high-quality solution structure information.

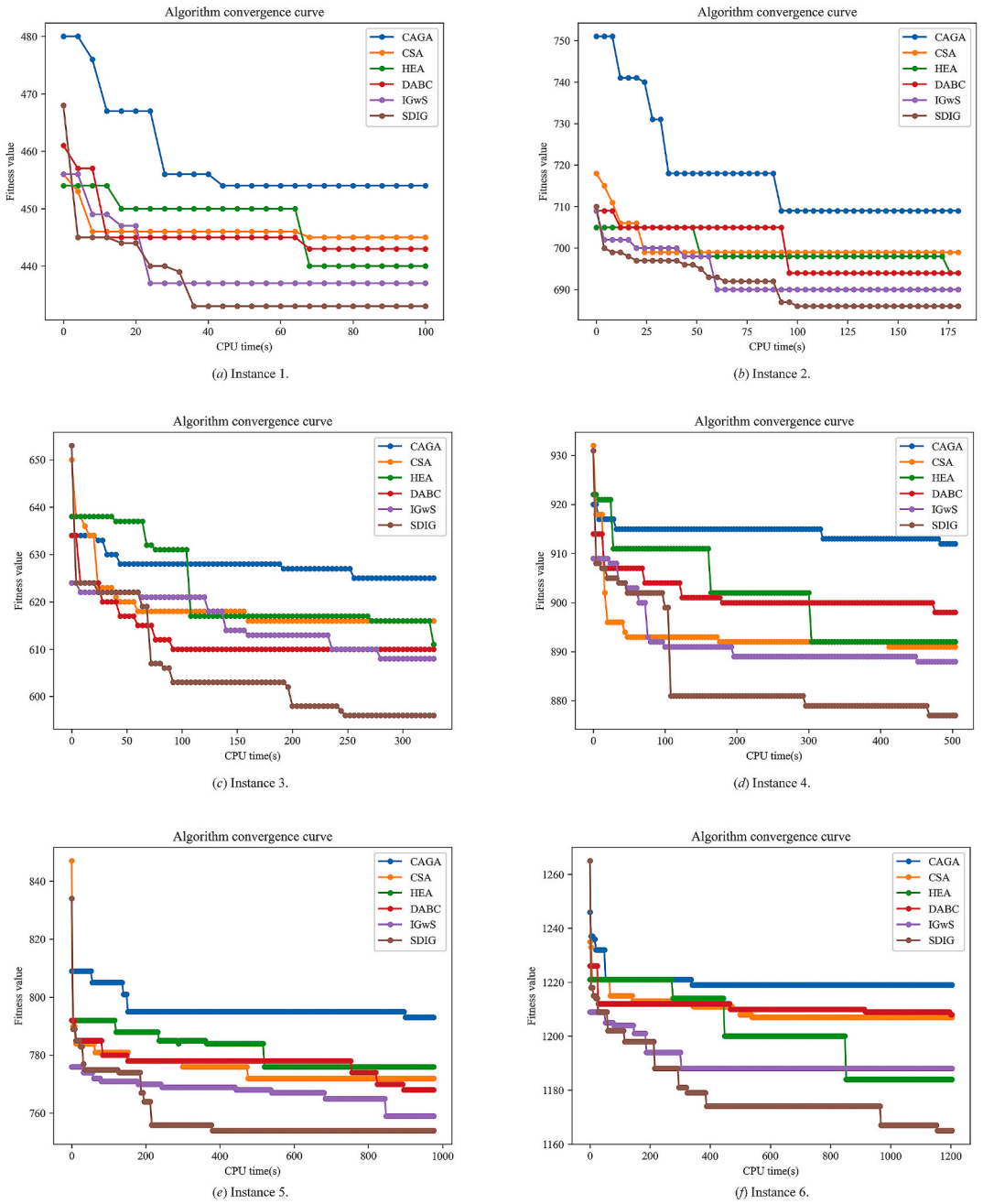


Fig. 13. Convergence traces of SDIG versus the other five algorithms on 6 different-scale instances.

Meanwhile, the swap-based neighborhood search allows for more detailed exploration, guiding individuals to move toward promising solution regions and laying the foundation for subsequent two-stage deep exploitation. Therefore, SDIG exhibits superior performance in comparison to SDIG_v2.

(3) The multi-neighborhood exploitation (first stage): Although the FOFI and FAM rules are used in decoding individuals to ensure the quality of the solution to a certain extent, the rules for operation sequencing and machine allocation are short-sighted. It can only make decisions based on the current availability of machines, without considering the impact on subsequent operations. Therefore, it is necessary to perform appropriate neighborhood searches on the operation sequence obtained after decoding the individuals to increase the search depth of the algorithm. At the same time, the multi-neighborhood exploitation is constructed based on the swap and insert neighborhood operations combined with the critical path characteristics, which avoids a large number of invalid neighborhood operations and accelerates the efficiency of the algorithm. Therefore, SDIG exhibits superior performance in comparison to SDIG_v3.

(4) The *Insert*-based fast neighborhood exploitation (second stage): Although the first stage has conducted in-depth searches on the operation sequence of the individuals, it still lacks search depth in the machine allocation aspect. Therefore, based on the speed-up neighbor evaluation method and critical path property, we designed a specific *Insert*-based fast neighborhood search for the scheduling solution determined by the best-known operation sequences obtained. This significantly reduces the computational complexity of the algorithm while enabling more detailed searches, further enhancing the algorithm's performance. Consequently, SDIG exhibits superior performance compared to SDIG_v4.

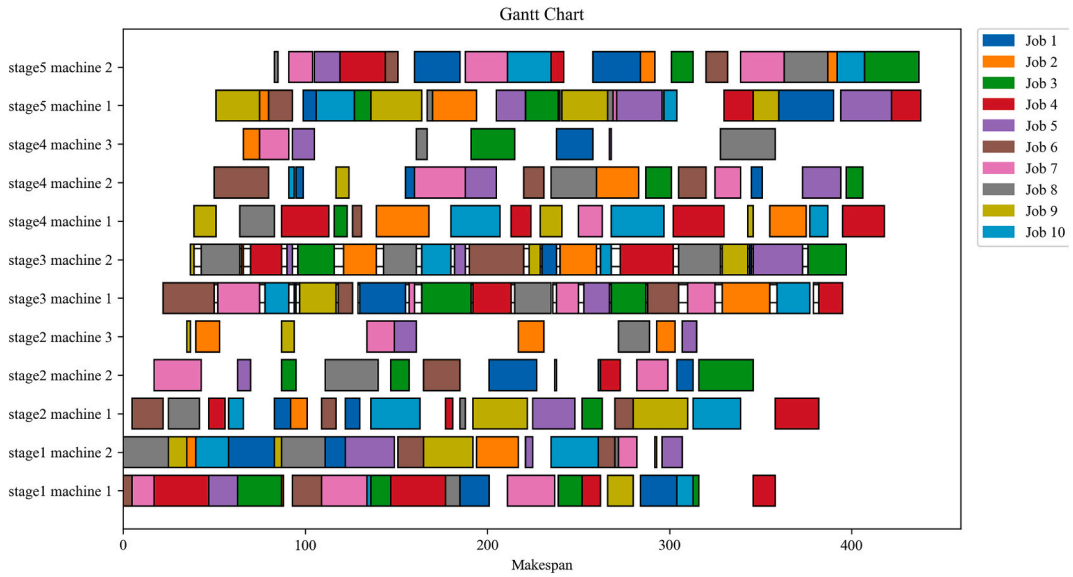


Fig. 14. Gantt chart of the best-known solution obtained by SDIG for instance $10 \times 5 \times 4$.

(5) The reset strategy: When the algorithm falls into a local optimum while conducting the in-depth search on the subspaces, The reset strategy can be used to return the current individual to its state before the exploitation, and then perform the two-stage deep exploitation process again. This strategy enriches the diversity of local optima, providing more options for further optimization. As a result, SDIG exhibits superior performance in comparison to SDIG_v5.

6.4. Comparison SDIG with the state-of-the-art methods

Currently, there are very few algorithms available for solving MRCRHFSP. Generally, for new or less studied problems, the mainstream academic approach is to select algorithms that solve similar problems for comparison to validate the effectiveness of the proposed algorithm. Therefore, this paper compares SDIG with the discrete artificial bee colony (DABC) algorithm(Pan et al., 2014) for HFSP, the chaos-enhanced simulated annealing (CSA) algorithm(Lin et al., 2021), the hybrid evolutionary algorithm (HEA) (Fan et al., 2023), the iterated greedy algorithm with speed-up mechanism (IGwS) (Fernandez-Viagas, 2022), and the cooperative adaptive genetic algorithm (CAGA) for solving RHFSP with sequence-dependent setup times and limited buffers (Zheng et al., 2023). For experimental fairness, the main frameworks and encoding/decoding methods of the aforementioned algorithms were not modified; they were directly applied to solve MRCRHFSP, with parameter settings identical to their original papers. All algorithms were run independently 20 times on each instance.

The average objective function value calculated by Eq. (46) was used as the response value, and the relative percentage deviation (RPD) defined by Eq. (45) was used as the performance measure. The experimental results are shown in Table 10, with the best results highlighted in bold. To make the experimental results more convincing, a multifactorial ANOVA was conducted, similar to Section 6.3, was performed to verify that the results were statistically significant. The 95 % Tukey HSD confidence interval mean plots and box plots for SDIG and the compared algorithms at different CPU run times are shown in Fig. 12.

As shown in Table 10, SDIG outperforms the other compared algorithms in almost all instances. The mean plots show that there is no overlapping region between SDIG and the other algorithms, and the RPD is smaller, indicating that SDIG's performance is statistically superior. In addition, the box plots show that SDIG's results are more concentrated with smaller variances, indicating better stability and robustness compared to the other algorithms. Furthermore, to better demonstrate the convergence of the algorithms, six instances were selected for testing: $10 \times 5 \times 4$, $10 \times 6 \times 6$, $15 \times 5 \times 4$, $15 \times 5 \times 6$, $20 \times 5 \times 4$, and $20 \times 5 \times 6$ for testing. The convergence curves of SDIG and the compared algorithms are shown in Fig. 13. SDIG exhibits superior convergence compared to the other algorithms. In addition, there is a distinct step in the SDIG convergence curve, which marks the transition from exploration to exploitation. This indicates that once the promising regions are found in the subspace exploration process, the exploitation phase can quickly converge to a higher-quality solution.

Finally, this paper presents the Gantt chart of the solution obtained by the SDIG algorithm for the $10 \times 5 \times 4$ instance, as shown in Fig. 14, where the makespan is 438. From Fig. 14, it can be seen that almost all jobs in the bottleneck stage are seamlessly connected, and the jobs in the non-bottleneck stage are scheduled as compactly as possible. This indicates that the proposed SDIG can effectively solve the MRCRHFSP and obtain high-quality scheduling schemes.

The performance differences of various algorithms in solving MRCRHSSP fundamentally stem from their encoding and decoding methods and evolutionary mechanisms. Among these algorithms, CAGA and CSA perform the worst, primarily because they adopt an operation-based encoding–decoding approach rather than a job-based approach. This expands the solution search space $J!$ to $(J \times K \times L)!$, where J represents the number of jobs, K is the number of stages, and L is the number of cycles. Although operation-based encoding–decoding can explore the solution space in greater detail, it requires substantial computational resources, making it difficult to converge to high-quality solution regions within a limited time frame. For the second tier of algorithms, HEA and DABC, a job-based encoding–decoding method is employed. By leveraging heuristic rules to order operations, these algorithms can ensure a certain level of solution quality while narrowing the search space. However, since both algorithms utilize population-based evolutionary mechanisms and lack acceleration strategies based on problem characteristics, they exhibit higher time complexity, which leads to significantly poorer search efficiency compared to the IGWS and SDIG algorithms. As shown in Fig. 13, even at the deadline, HEA has not yet converged, there is still considerable room for improvement in the optimization objective value. The top-performing algorithms are IGWS and SDIG. Both use job-based encoding–decoding methods. IGWS employs a single-individual evolutionary mechanism, initializing the solution with the NEH heuristic and further optimizing it. This approach ensures fast convergence, as demonstrated in Fig. 13. However, due to

the limited diversity of its single individual, IGWS tends to fall into local optima and struggles to explore deep, high-quality solutions across multiple regions. In contrast, the SDIG algorithm proposed in this study decomposes the solution space during the initial stage and iteratively optimizes the best-known solutions within subspaces. While this adds slight computational complexity, it enriches the search region. Additionally, as discussed in Sections 5.4.1 and 5.4.2, the multi-neighborhood exploitation based on the critical path and the Insert-based fast neighborhood exploitation enhance the algorithm's search efficiency. As a result, the SDIG algorithm demonstrates significant advantages in comparative experiments.

In summary, CAGA and CSA adopt operation-based encoding–decoding methods with population-based evolutionary mechanisms. HEA and DABC employ job-based encoding–decoding with population-based evolutionary mechanisms. IGWS uses job-based encoding–decoding combined with a single-individual evolutionary mechanism. SDIG also uses job-based encoding–decoding but integrates a limited-individual evolutionary mechanism, and designs two methods to improve the search efficiency of the algorithm. These differences in encoding–decoding methods and evolutionary mechanisms fundamentally explain the variation in performances among the algorithms.

7. Conclusions

Based on the reentrant hybrid flow shop scheduling problem, We further consider practical factors such as mask resource constraints and sequence-dependent setup times during the photolithography stage (bottleneck stage) in semiconductor wafer fabrication, It establishes the mixed-integer programming (MIP) model for the multi-resource constrained re-entrant hybrid flow shop scheduling problem (MRCRHFSP) for the first time to minimize makespan by scheduling the machine assignment and operation processing sequence of jobs. We propose the space decomposition-based iterative greedy (SDIG) algorithm for solving MRCRHFSP. Computational experiments and statistical results show that SDIG outperforms the compared algorithms in terms of efficiency, quality, and convergence when solving MRCRHFSP.

The superiority of the SDIG lies in two aspects: The new SD method and a parallel multimodal search mechanism. The SD method can perceive the landscape of the MRCRHFSP solution space and decompose it into several subspaces with certain differences, guiding the algorithm to perform subsequent searches in the subspaces, avoiding the risk of quickly falling into local optima. The parallel multi-modal search mechanism implements extensive and in-depth parallel searches for each subspace through construction-based exploration and two-stage deep exploitation, thus achieving a good balance between exploration and exploitation.

In future research, the following valuable directions exist: (1) further extending the MRCRHFSP by incorporating buffer capacities between stages and wafer transportation time into the scheduling framework, to enhance the realism of the wafer fabrication workshop scheduling problem model. (2) Consider order-driven MRCRHFSP and design dynamic scheduling strategies to solve it. In actual production activities, dynamic orders are the source of production demand, which makes research on order-driven MRCRHFSP more practical. (3) Apply SDIG to solve other combinatorial optimization problems (COPs) to verify its generality and effectiveness.

CRediT authorship contribution statement

Feng-Shun Zhou: Writing – original draft, Validation, Software, Methodology, Investigation, Data curation. **Rong Hu:** Writing – review & editing, Supervision, Methodology, Funding acquisition. **Bin Qian:** Writing – review & editing, Methodology, Funding acquisition, Conceptualization. **Qing-Xia Shang:** Writing – review & editing, Supervision. **Yuan-yuan Yang:** Writing – review & editing, Investigation. **Jian-Bo Yang:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Numbers: U24A20273, 62173169 and 62463013), the Basic Research Key Project of Yunnan Province (Grant Number: 202201AS070030), and the Construction Project of Higher Educational Key Laboratory for Industrial Intelligence and Systems of Yunnan Province (Grant Number: KKPH202403003).

Appendix 1

The appendix includes the six materials required for the paper titled "Spatial Decomposition-Based Iterative Greedy Algorithm for the Multi-Resource Constrained Re-entrant Hybrid Flow Shop Scheduling Problem in Semiconductor Wafer Fabrication."

Part 1. A simple example of the encoding and decoding strategy

To gain a deeper understanding of the encoding and decoding strategy of the SDIG algorithm for MRCRHFSP, a simple example is provided below. The number of jobs $J = 4$, the number of processing stages $K = 4$, and the number of machines in each processing stage $M_1 = 2$, $M_2 = 3$, $M_3 = 1$, $M_4 = 2$ with re-entry cycles $L = 4$. The processing times of the jobs are illustrated in Table 1*, the required masks for the jobs are illustrated in Table 2*, and the mask switch setup times are illustrated in Table 3*. The encoded individual $\pi^* = [3, 1, 4, 2]$, after decoding as described in Section 5.1, results in the operation sequence $\pi\_o = [3, 1, 4, 2, 1, 3, 4, 2, 3, 1, 4, 2, 3, 1, 4, 2, 3, 1, 2, 4, 3, 1, 2, 4, 3, 1, 2, 4, 3, 1, 2, 4]$, machine sequence $\pi\_m = [1, 2, 2, 1, 1, 2, 3, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1]$, and a complete scheduling solution Π are generated, with a makespan of 582. The entire decoding process and the Gantt chart corresponding to the obtained scheduling solution after decoding are shown in Fig. 1*.

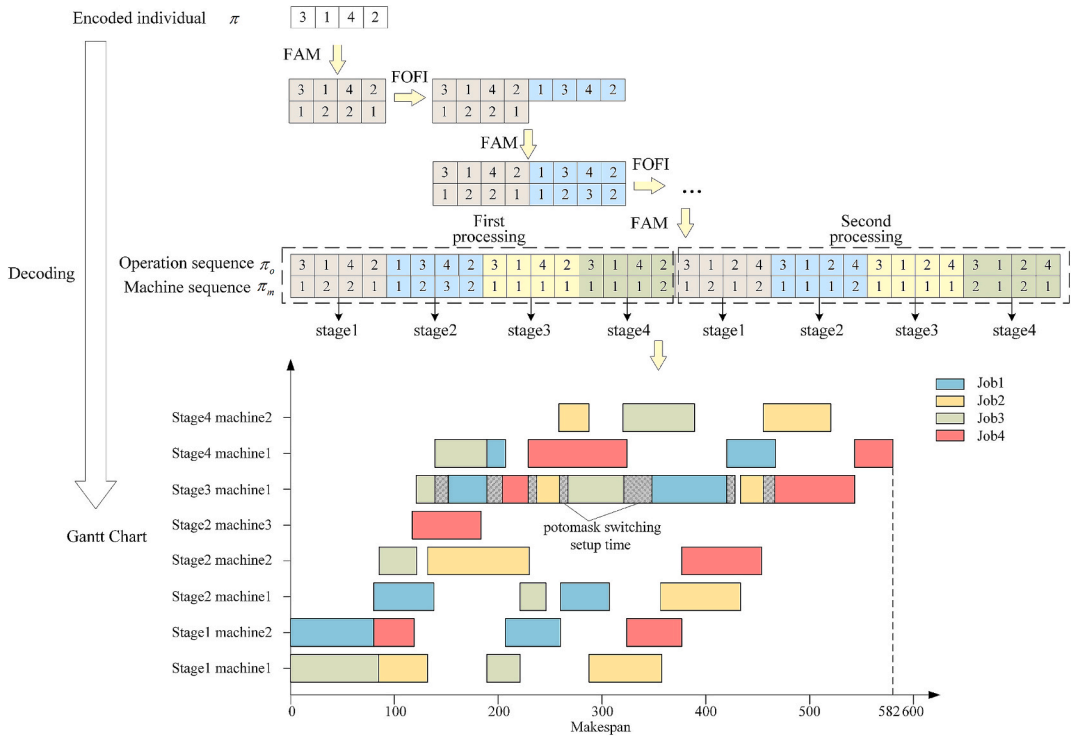


Fig. 1*. Numerical example of the decoding strategy.

Table 1*
Job processing time.

Job	Process 1				Process 2			
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 1	Stage 2	Stage 3	Stage 4
1	80	58	37	18	53	47	72	47
2	47	98	22	29	70	77	22	65
3	85	36	18	50	32	25	54	69
4	39	66	25	95	53	77	77	37

Table 2*
Mask required for the job.

Job	Process 1	Process 2
1	4	1
2	8	6
3	5	3
4	2	7

Table 3*
Mask switching setup time.

Mask	1	2	3	4	5	6	7	8
1	0	1	29	1	8	8	16	29
2	21	0	5	13	24	5	25	8
3	27	15	0	30	8	1	20	24
4	14	15	12	0	8	23	30	10
5	22	28	24	13	0	13	18	7
6	7	27	18	13	19	0	11	14
7	20	15	5	25	5	25	0	17
8	2	25	8	1	2	25	23	0

Part 2. Validation of the proposed MIP model in the paper using the Gurobi solver

To validate the correctness and applicability of the proposed MIP model in this paper, we used the example in Part 1 and verified the model with the Gurobi solver. Computational experiments showed that the proposed MIP model accurately captured the problem constraints and produced the optimal feasible schedule as illustrated in Fig. 2*, with a makespan of 580.

Part 3. Examples of forward scheduling and backward scheduling

To gain a deeper understanding of the forward and backward scheduling processes, we use the decoded scheduling solution from Part 1 as an example. The Gantt chart corresponding to the forward scheduling solution, calculated based on Eqs. (14)–(18) in the main text, is shown in Fig. 3*. The Gantt chart corresponding to the backward scheduling solution, calculated based on Eqs. (19)–(23), is shown in Fig. 4*.

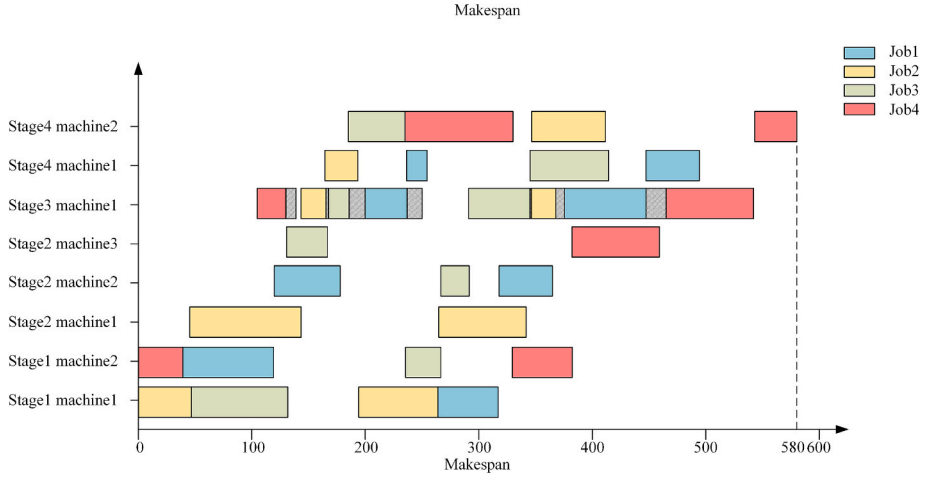


Fig. 2*. The Gantt chart corresponding to the optimal schedule solution obtained by the Gurobi solver.

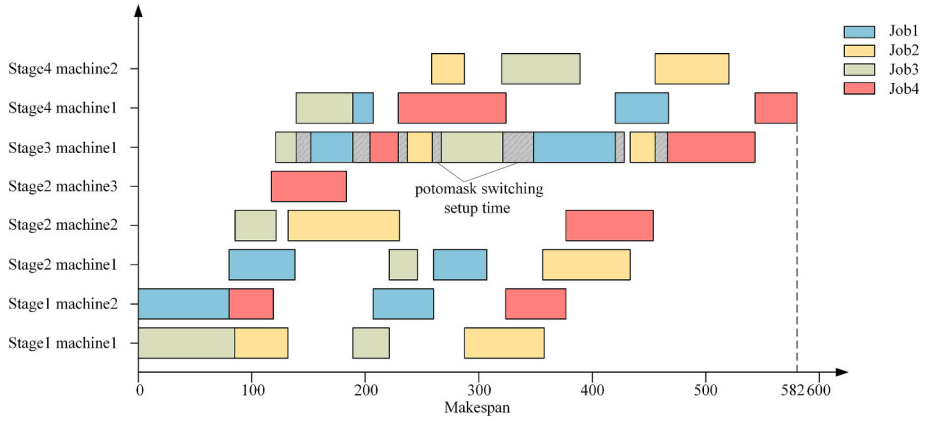


Fig. 3*. Forward Schedule Gantt chart.

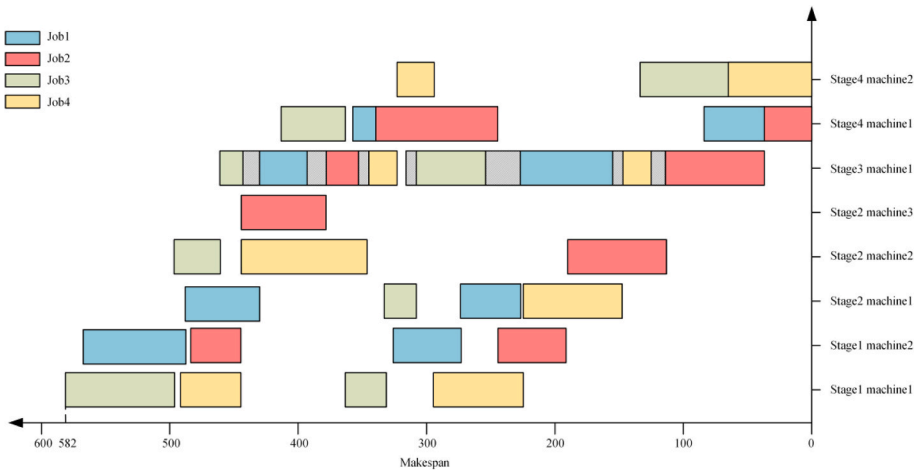


Fig. 4*. Backward schedule Gantt chart.

Part 4. Examples of the Theorem 1

Taking the scheduling solution provided in Part 1 as an example, the critical paths for forward and backward scheduling (**Theorem 1**) are shown in Fig. 5* and Fig. 6*.

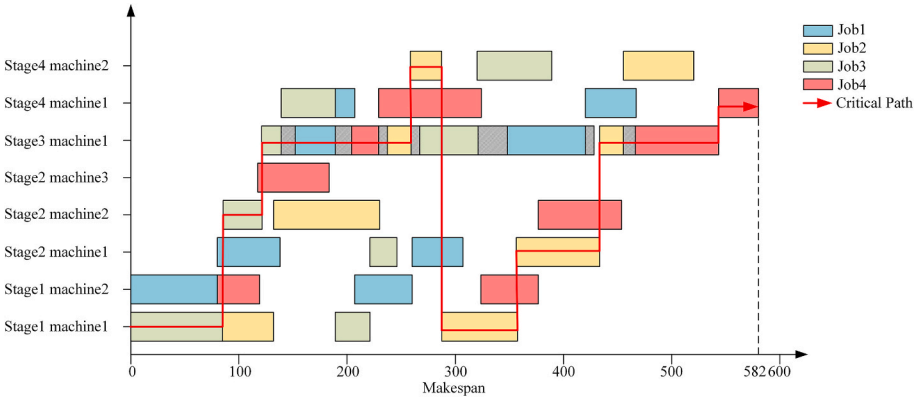


Fig. 5*. The forward schedule critical path.

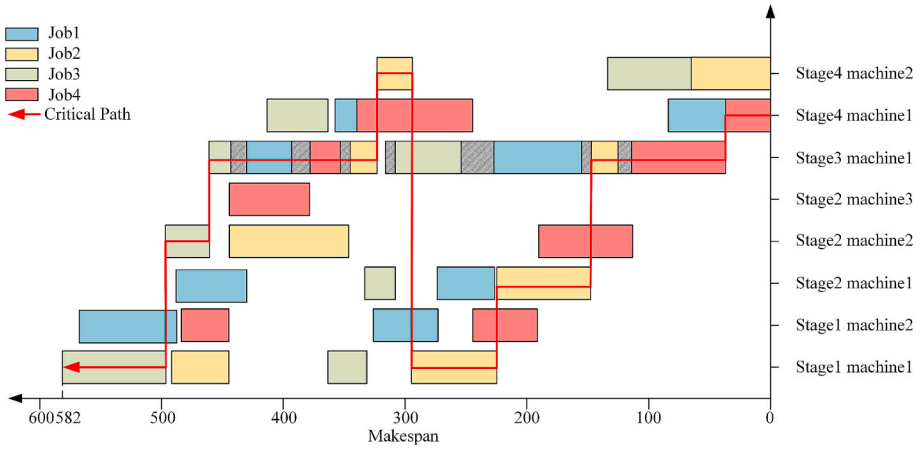
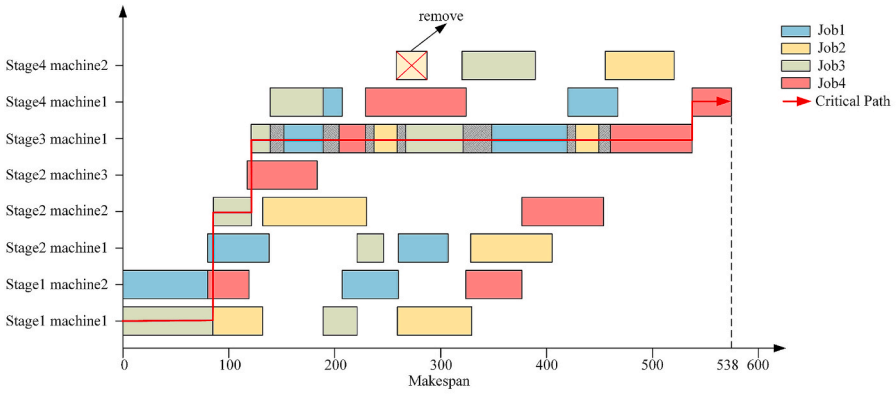


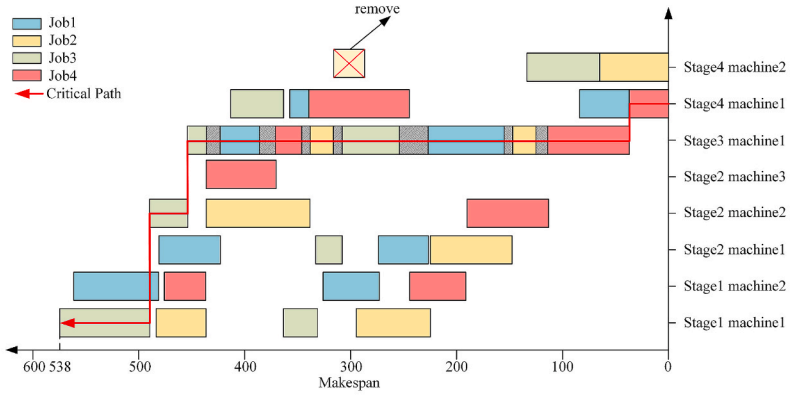
Fig. 6*. The backward schedule critical path.

Part 5. Examples of the Theorem 2

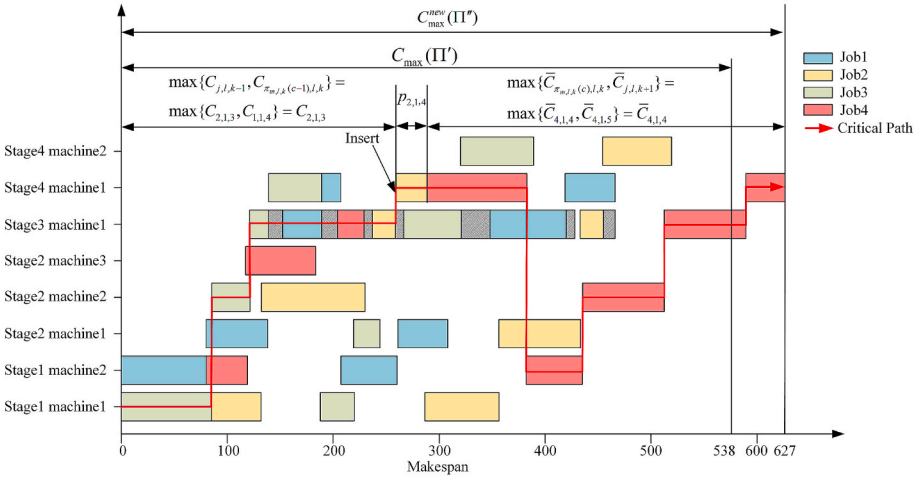
Taking the scheduling solution provided in Part 1 as an example, the schematic diagrams for the speed-up evaluation of the insertion neighborhood (**Theorem 2**) in the photolithography and non-photolithography stages are shown in Fig. 7* and Fig. 8*.



(*a*) Gantt chart of the forward scheduling after removing non-bottleneck stage operation

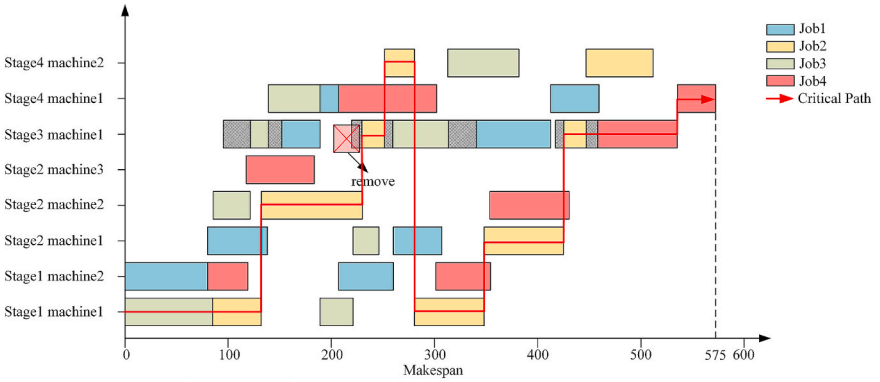


(*b*) Gantt chart of the backward scheduling after removing non-bottleneck stage operation

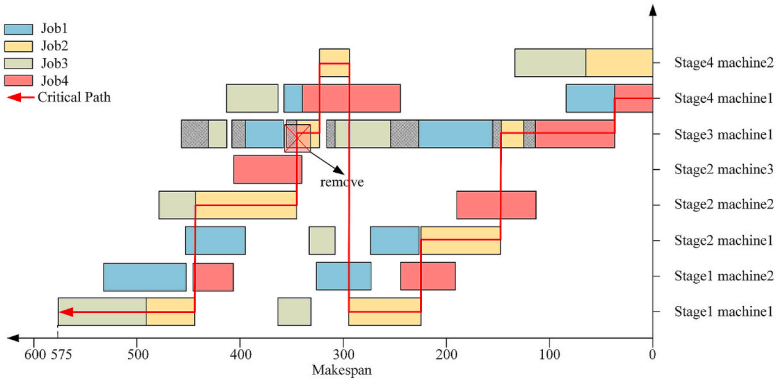


(*c*) Speed-up evaluation Gantt chart after the insertion of non-bottleneck stage operation

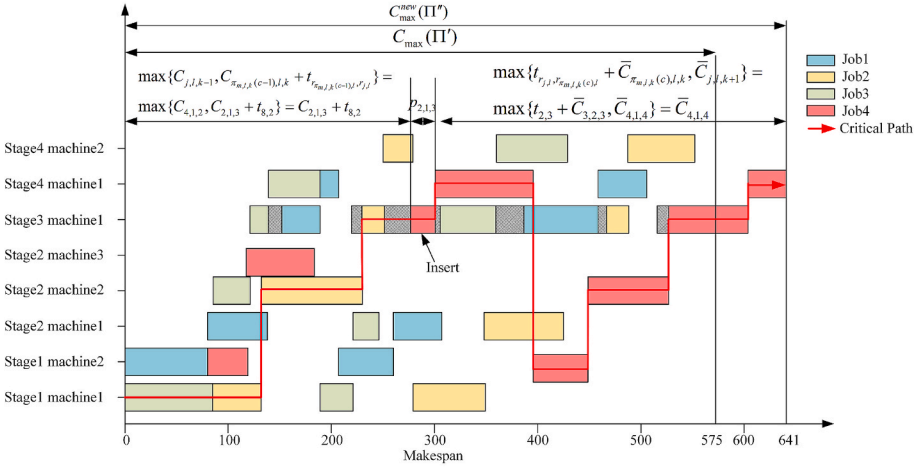
Fig. 7*. Speed-up evaluation for non-bottleneck stages.



(*a*) Gantt chart of the forward scheduling after removing bottleneck stage operation



(*b*) Gantt chart of the backward scheduling after removing bottleneck stage operation



(*c*) Speed-up evaluation Gantt chart after the insertion of bottleneck stage operation

Fig. 8*. Speed-up evaluation for the bottleneck stage.

Part 6. Examples of the Theorem 3

Taking the scheduling solution provided in Part 1 as an example, the schematic diagrams for **Theorem 3** are shown in Fig. 9*.

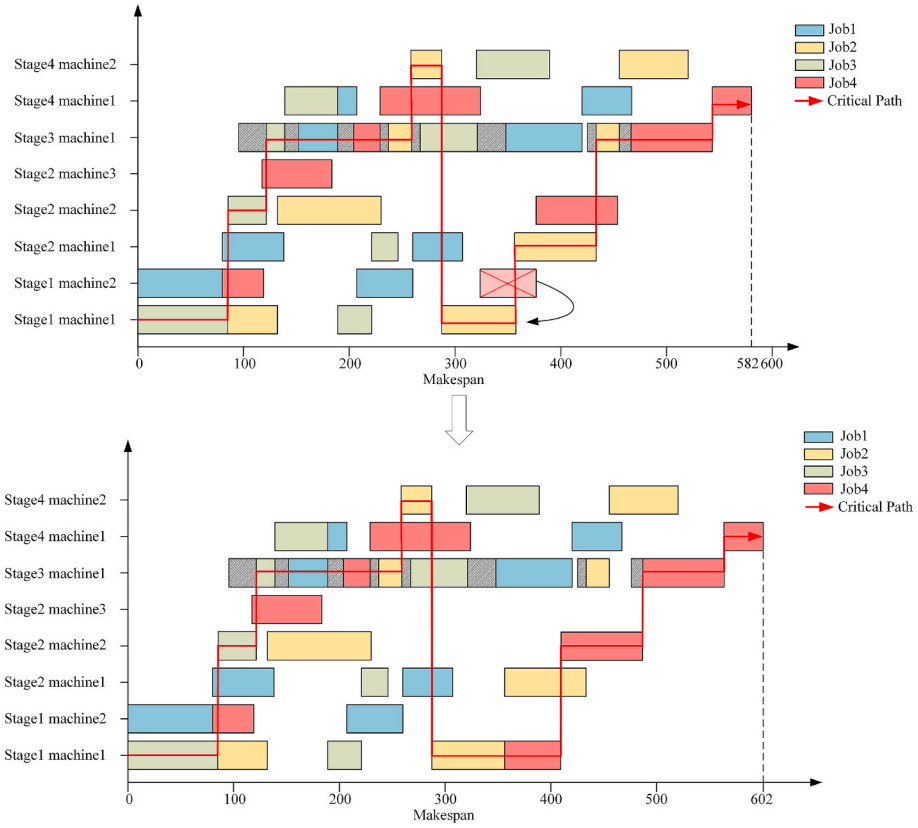


Fig. 9*. Schematic diagram of Theorem 3.

Data availability

Data will be made available on request.

References

Bang, J.-Y., & Kim, Y.-D. (2011). Scheduling algorithms for a semiconductor probing facility. *Computers and Operations Research, 38*(3), 666–673. https://doi.org/10.1016/j.cor.2010.08.010

Bitar, A., Dauzère-Pérès, S., Yugma, C., & Roussel, R. (2016). A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling, 19*(4), 367–376. https://doi.org/10.1007/s10951-014-0397-6

Bookstein, A. (2002). Generalized hamming distance. *Information Retrieval, 5*, 353–375.

Cakici, E., & Mason, S. J. (2007). Parallel machine scheduling subject to auxiliary resource constraints. *Production Planning & Control, 18*(3), 217–225. https://doi.org/10.1080/09537280601035836

Chen, S., Pan, Q.-K., Gao, L., & Sang, H.-Y. (2021). *A population-based iterated greedy algorithm to minimize total flowtime for the distributed blocking flowshop scheduling problem, 104*, Article 104375.

Cho, H.-M. (2011). Bi-objective scheduling for reentrant hybrid flow shop using Pareto genetic algorithm. *Industrial Engineering*.

Cicirello, V. A. (2020). Kendall Tau sequence distance: Extending Kendall Tau from ranks to sequences. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems, 7*(23), Article 163925. https://doi.org/10.4108/eai.13-7-2018.163925

Collins, T. D. (2003). Applying software visualization technology to support the use of evolutionary algorithms. *Journal of Visual Languages & Computing, 14*(2), 123–150. https://doi.org/10.1016/S1045-926X(02)00060-5

Ding, H., & Gu, X. (2020). Improved particle swarm optimization algorithm based novel encoding and decoding schemes for flexible job shop scheduling problem. *Computers and Operations Research, 121*, Article 104951. https://doi.org/10.1016/j.cor.2020.104951

Ding, J.-Y., Song, S., Gupta, J. N. D., Zhang, R., Chiong, R., & Wu, C. (2015). An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. *Applied Soft Computing, 30*, 604–613. https://doi.org/10.1016/j.asoc.2015.02.006

Ding, S., Keal, C. A., Zhao, L., & Yu, D. (n.d.). *Dimensionality reduction and classification for hyperspectral image based on robust supervised ISOMA*.

Dong, J., & Ye, C. (2019). Research on collaborative optimization of green manufacturing in semiconductor wafer distributed heterogeneous factory. *Applied Sciences, 9*(14), 2879. https://doi.org/10.3390/app9142879

Fahim, A. M., Salem, A. M., Torkey, F. A., & Ramadan, M. A. (2006). An efficient enhanced k-means clustering algorithm. *Journal of Zhejiang University-SCIENCE A, 7*(10), 1626–1633. https://doi.org/10.1631/jzus.2006.A1626

Fan, J., Li, Y., Xie, J., Zhang, C., Shen, W., & Gao, L. (2023). A hybrid evolutionary algorithm using two solution representations for hybrid flow-shop scheduling problem. *IEEE Transactions on Cybernetics, 53*(3), 1752–1764. https://doi.org/10.1109/TCYB.2021.3120875

Fernandez-Viagas, V. (2022). A speed-up procedure for the hybrid flow shop scheduling problem. *Expert Systems with Applications, 187*, Article 115903. https://doi.org/10.1016/j.eswa.2021.115903

Fernandez-Viagas, V., Valente, J. M. S., & Framinan, J. M. (2018). Iterated-greedy-based algorithms with beam search initialization for the permutation flowshop to minimise total tardiness. *Expert Systems with Applications, 94*, 58–69. https://doi.org/10.1016/j.eswa.2017.10.050

Ghaedy-Heidary, E., Nejati, E., Ghasemi, A., & Torabi, S. A. (2024). A simulation optimization framework to solve stochastic flexible job-shop scheduling problems—Case: Semiconductor manufacturing. *Computers and Operations Research, 163*, Article 106508. https://doi.org/10.1016/j.cor.2023.106508

Grebennik, I., Chorna, O., & Urniaieva, I. (2023). Visualizing Feasible Regions for Optimization Problems on High-Dimensional Permutations using Dimensionality Reduction Methods. In *2023 13th International Conference on Advanced Computer Information Technologies (ACIT)* (pp. 126–130). https://doi.org/10.1109/ACIT58437.2023.10275497

Ham, A. (2018). Scheduling of dual resource constrained lithography production: Using CP and MIP/CP. *IEEE Transactions on Semiconductor Manufacturing, 31*(1), 52–61. https://doi.org/10.1109/TSM.2017.2768899

He, X., Pan, Q.-K., Gao, L., Wang, L., & Suganthan, P. N. (2023). A greedy cooperative co-evolutionary algorithm with problem-specific knowledge for multiobjective flowshop group scheduling problems. *IEEE Transactions on Evolutionary Computation, 27*(3), 430–444. https://doi.org/10.1109/TEVC.2021.3115795

Hekmatfar, M., Fatemi Ghomi, S. M. T., & Karimi, B. (2011). Two stage reentrant hybrid flow shop with setup times and the criterion of minimizing makespan. *Applied Soft Computing, 11*(8), 4530–4539. https://doi.org/10.1016/j.asoc.2011.08.013

Hozumi, Y., Wang, R., Yin, C., & Wei, G.-W. (2021). UMAP-assisted K-means clustering of large-scale SARS-CoV-2 mutation datasets. *Computers in Biology and Medicine, 131*, Article 104264. https://doi.org/10.1016/j.compbiomed.2021.104264

Huang, C.-T., Hsieh, T.-J., & Lin, B. M. T. (2025). Data-driven scheduling for the photolithography process in semiconductor manufacturing. *Journal of Industrial and Management Optimization, 21*(3), 1946–1963. https://doi.org/10.3934/jimo.2024157

Huang, Y.-Y., Pan, Q.-K., Huang, J.-P., Suganthan, P., & Gao, L. (2021). An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem. *Computers and Industrial Engineering, 152*, Article 107021. https://doi.org/10.1016/j.cie.2020.107021

Jacobs, L. W., & Brusco, M. J. (1995). Note: A local-search heuristic for large set-covering problems. *Naval Research Logistics, 42*(7), 1129–1140. https://doi.org/10.1002/1520-6750(199510)42:7<1129::AID-NAV3220420711>3.0.CO;2-M

Jain, V., Swarnkar, R., & Tiwari, M. K. (2003). Modelling and analysis of wafer fabrication scheduling via generalized stochastic Petri net and simulated annealing. *International Journal of Production Research, 41*(15), 3501–3527. https://doi.org/10.1080/0020754031000118152

Jornod, G., Di Mario, E., Navarro, I., & Martinoli, A. (2015). SwarmViz: An open-source visualization tool for Particle Swarm Optimization. *IEEE Congress on Evolutionary Computation (CEC), 2015*, 179–186. https://doi.org/10.1109/CEC.2015.7256890

Kim, S. H., & Lee, Y. H. (2016). Synchronized production planning and scheduling in semiconductor fabrication. *Computers and Industrial Engineering, 96*, 72–85. https://doi.org/10.1016/j.cie.2016.03.019

Kong, M., Zhang, Y., Xu, J., Wang, W., Lu, S., & Fathollahi-Fard, A. M. (2024). A green scheduling model for two-stage photo-etching and acid-etching collaboration in semiconductor manufacturing. *Journal of Industrial Information Integration, 41*, Article 100655. https://doi.org/10.1016/j.jii.2024.100655

Lee, C.-Y., Wu, C.-M., Hsu, C.-Y., Xie, H.-H., & Fang, Y.-H. (2023). Lithography reticle scheduling in semiconductor manufacturing. *Engineering Optimization, 1–19*. https://doi.org/10.1080/0305215X.2023.2285416

Lee, Y. F., Jiang, Z. B., & Liu, H. R. (2009). Multiple-objective scheduling and real-time dispatching for the semiconductor manufacturing system. *Computers and Operations Research, 36*(3), 866–884. https://doi.org/10.1016/j.cor.2007.11.006

Lee, Y. H., & Lee, S. (2022). Deep reinforcement learning based scheduling within production plan in semiconductor fabrication. *Expert Systems with Applications, 191*, Article 116222. https://doi.org/10.1016/j.eswa.2021.116222

Li, J.-Q., Du, Y., Gao, K.-Z., Duan, P.-Y., Gong, D.-W., Pan, Q.-K., & Suganthan, P. N. (2022). A hybrid iterated greedy algorithm for a crane transportation flexible job shop problem. *IEEE Transactions on Automation Science and Engineering, 19*(3), 2153–2170. https://doi.org/10.1109/TASE.2021.3062979

Li, Z. C., Qian, B., Hu, R., Chang, L. L., & Yang, J. B. (2019). An elitist nondominated sorting hybrid algorithm for multi-objective flexible job-shop scheduling problem with sequence-dependent setups. *Knowledge-Based Systems, 173*, 83–112. https://doi.org/10.1016/j.knosys.2019.02.027

Lin, S.-W., Cheng, C.-Y., Pourhejazy, P., Ying, K.-C., & Lee, C.-H. (2021). New benchmark algorithm for hybrid flowshop scheduling with identical machines. *Expert Systems with Applications, 183*, Article 115422. https://doi.org/10.1016/j.eswa.2021.115422

Liu, H., Yang, J., Ye, M., James, S. C., Tang, Z., Dong, J., & Xing, T. (2021). Using t-distributed Stochastic Neighbor Embedding (t-SNE) for cluster analysis and spatial zone delineation of groundwater geochemistry data. *Journal of Hydrology, 597*, Article 126146. https://doi.org/10.1016/j.jhydrol.2021.126146

Lutton, E., Foucquier, J., Perrot, N., Louchet, J., & Fekete, J.-D. (2012). Visual Analysis of Population Scatterplots. In J.-.-K. Hao, P. Legrand, P. Collet, N. Monmarché, E. Lutton, & M. Schoenauer (Eds.), *Artificial Evolution, 7401* pp. 61–72). Berlin Heidelberg: Springer. https://doi.org/10.1007/978-3-642-35533-2_6.

Macqueen, J. (n.d.). Some methods for classification and analysis of multivariate observations. *Multivariate Observations*.

McInnes, L., Healy, J., & Melville, J. (2020). *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction* (No. arXiv:1802.03426). arXiv. http://arxiv.org/abs/1802.03426.

Michalak, K. (2019). Low-dimensional Euclidean Embedding for visualization of search spaces in combinatorial optimization. *IEEE Transactions on Evolutionary Computation, 23*(2), 232–246. https://doi.org/10.1109/TEVC.2018.2846636

Mönch, L., Uzsoy, R., & Fowler, J. W. (2018). A survey of semiconductor supply chain models part I: Semiconductor supply chains, strategic network design, and supply chain simulation. *International Journal of Production Research, 56*(13), 4524–4545. https://doi.org/10.1080/00207543.2017.1401233

Na, S., Xumin, L., & Yong, G. (2010). Research on k-means clustering algorithm: An improved k-means clustering algorithm. *Third International Symposium on Intelligent Information Technology and Security Informatics, 2010*, 63–67. https://doi.org/10.1109/IITSI.2010.74

Ozsoydan, F. B. (2021). *Iterated greedy algorithms enhanced by hyper-heuristic based learning for hybrid flexible flowshop scheduling problem with sequence dependent setup times: A case study at a manufacturing plant.*

Pan, Q.-K., Gao, L., Xin-Yu, L., & Jose, F. M. (2019). Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem. *Applied Soft Computing, 81*, Article 105492. https://doi.org/10.1016/j.asoc.2019.105492

Pan, Q.-K., Wang, L., Li, J.-Q., & Duan, J.-H. (2014). A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega, 45*, 42–56. https://doi.org/10.1016/j.omega.2013.12.004

Qian, B., Zhang, Z.-Q., Hu, R., Jin, H.-P., & Yang, J.-B. (2023). A matrix-cube-based estimation of distribution algorithm for no-wait flow-shop scheduling with sequence-dependent setup times and release times. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 53*(3), 1492–1503. https://doi.org/10.1109/TSMC.2022.3198829

Qin, H.-X., Han, Y.-Y., Liu, Y.-P., Li, J.-Q., Pan, Q.-K., & Xue-Han. (2022). A collaborative iterative greedy algorithm for the scheduling of distributed heterogeneous hybrid flow shop with blocking constraints. *Expert Systems with Applications, 201*, Article 117256. https://doi.org/10.1016/j.eswa.2022.117256

Qin, H.-X., Han, Y.-Y., Zhang, B., Meng, L.-L., Liu, Y.-P., Pan, Q.-K., & Gong, D.-W. (2022). An improved iterated greedy algorithm for the energy-efficient blocking hybrid flow shop scheduling problem. *Swarm and Evolutionary Computation, 69*, Article 100992. https://doi.org/10.1016/j.swevo.2021.100992

Rodriguez, F. J., Lozano, M., Blum, C., & García-Martínez, C. (2013). An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers and Operations Research, 40*(7), 1829–1841. https://doi.org/10.1016/j.cor.2013.01.018

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049. https://doi.org/10.1016/j.ejor.2005.12.009

Sadati, M. E. H., & Çatay, B. (2021). A hybrid variable neighborhood search approach for the multi-depot green vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review, 149*, Article 102293. https://doi.org/10.1016/j.tre.2021.102293

Taillard, E. (1990). *Some efficient heuristic methods for the flow shop sequencing problem, 1* (47), 65–74.

Wang, L., & Qian, B. (2012). *Hybrid differential evolution and scheduling algorithm.*

Xu, Y., Zhang, M., Yang, M., & Wang, D. (2024). Hybrid quantum particle swarm optimization and variable neighborhood search for flexible job-shop scheduling problem. *Journal of Manufacturing Systems, 73*, 334–348. https://doi.org/10.1016/j.jmsy.2024.02.007

Kim, Y.-D., Kim, J.-G., Choi, B., & Kim, H.-U. (2001). Production scheduling in a semiconductor wafer fabrication facility producing multiple product types with distinct due dates. *IEEE Transactions on Robotics and Automation, 17*(5), 589–598. https://doi.org/10.1109/70.964660

Yu, T.-T., Chen, C.-Y., Wu, T.-H., & Chang, Y.-C. (2023). Application of high-dimensional uniform manifold approximation and projection (UMAP) to cluster existing landfills on the basis of geographical and environmental features. *The Science of the Total Environment, 904*, 16701.

Zhang, Z.-Q., Hu, R., Qian, B., Jin, H.-P., Wang, L., & Yang, J.-B. (2022). A matrix cube-based estimation of distribution algorithm for the energy-efficient distributed assembly permutation flow-shop scheduling problem. *Expert Systems with Applications, 194*, Article 116484. https://doi.org/10.1016/j.eswa.2021.116484

Zhao, Z., Zhou, M., & Liu, S. (2022). Iterated greedy algorithms for flow-shop scheduling problems: A tutorial. *IEEE Transactions on Automation Science and Engineering, 19*(3), 1941–1959. https://doi.org/10.1109/TASE.2021.3062994

Zheng, Q., Zhang, Y., Tian, H., & He, L. (2023). A cooperative adaptive genetic algorithm for reentrant hybrid flow shop scheduling with sequence-dependent setup time and limited buffers. *Complex & Intelligent Systems*. https://doi.org/10.1007/s40747-023-01147-8

Zou, W.-Q., Pan, Q.-K., & Tasgetiren, M. F. (2021). An effective iterated greedy algorithm for solving a multi-compartment AGV scheduling problem in a matrix manufacturing workshop. *Applied Soft Computing, 99*, Article 106945. https://doi.org/10.1016/j.asoc.2020.106945