A Cooperative Coevolution Algorithm with a Heterogeneous-Island-Based Hyper-Heuristic for Cloud–Edge–Device Collaborative Scheduling

Jianheng Hu[a], Yuanjun Laili[a,b,*], Lei Ren[a,b], ...[a], ...[a]

[a]*School of Automation Science and Electrical Engineering, Beihang University, 37 Xueyuan Road, Beijing, 100191, China*
[b]*Hangzhou International Innovation Institute, Beihang University, Hangzhou, 311115, Zhejiang, China*

Abstract

...

Keywords: Cloud–edge–device collaborative scheduling, Distributed manufacturing systems, Structure-aware adaptive operator selection, Robust scheduling, Cooperative coevolution.

1. Introduction

An Industrial Internet-of-Things (IIoT) platform integrates computational resources from both cloud datacenters and nearby edge nodes to help heterogeneous devices on the shop floor perform collaborative supervision, computation, decision-making, and control. It has been applied to many discrete manufacturing processes for flexible production line reconfiguration and efficient customized production.

In this scenario, large-scale manufacturing jobs, their constituent manufacturing operations, and associated computational tasks arise with complex communication and material-flow dependencies. For example, the production of an electronic product requires dozens of machining and assembly tasks, each of which can be further divided into dozens of operations involving material or component transportation. In addition, each manufacturing job is associated with a job-level computational task that involves several

*corresponding author. E-mail: lailiyuanjun@buaa.edu.cn

data exchanges with the shop floor. When dozens or even hundreds of orders for different product types are submitted to the IIoT platform, thousands of hybrid tasks must be performed simultaneously.

Cloud–edge–device collaborative scheduling is therefore essential for the efficient operation of the IIoT platform [1]. It determines where each computational task is executed, which device processes each manufacturing operation, and in what order these tasks are carried out. Under cross-layer task and resource constraints, the main objectives are to reduce makespan and energy consumption.

Existing methods for solving the cloud–edge–device collaborative scheduling problem can be broadly grouped into mathematical programming methods, evolutionary and metaheuristic methods, and learning-based methods.

The existing EAs usually encode all decision variables into a single vector and apply evolutionary operators to search for feasible solutions [2, 3]. As the number of decision variables increases, population-based iterative search becomes increasingly inefficient. To reduce the repeated random exploration of EAs in such a large solution space, RL has been introduced as a high-level heuristic for parameter setting and operator selection [1, 4, 5]. However, the effectiveness of evolutionary operators and heuristics varies across decision variables.

To further reduce scheduling time, RL has also been used to map current resource states to scheduling decisions in a step-by-step manner [6]. In such settings, policy, value, actor, and critic networks are typically trained in a simulation environment where resources and tasks are fixed [7]. As the number of decision variables (i.e., action spaces) increases, training becomes harder to converge. Moreover, when the scheduling scenario changes, a pretrained RL policy may not generalize well to the new setting.

In summary, the effectiveness of evolutionary operators, heuristics, and RL-based policies varies across decision subspaces such as computational-task offloading, manufacturing-operation sequencing, and device assignment. Using a single operator for all decision variables in the cloud–edge–device collaborative scheduling problem can reduce search efficiency in each subspace. A collaborative search strategy is therefore needed to optimize this large-scale scheduling problem in parallel while maintaining high search efficiency.

To this end, this paper proposes a Cooperative Coevolution algorithm with a Heterogeneous-Island-Based Hyper-Heuristic (CCHIHH) to solve the large-scale cloud–edge–device collaborative task scheduling problem. A divide-and-conquer strategy is used to decompose the decision variables into three

decision blocks: computational-task offloading, manufacturing-operation sequencing, and device assignment. These decision blocks are then optimized cooperatively within a heterogeneous island model through parallel iterative search. The same operator pools are available to all islands, but independently updated localized contextual bandits allow different islands to develop different operator-selection distributions and search trajectories. A stability gate conditioned on search stagnation is further introduced to stabilize cooperative coevolution across islands.

The main contributions are as follows:

1. **A divide-and-conquer cooperative coevolution strategy** is introduced to decompose the joint optimization problem into three decision blocks according to its natural decision structure. Each decision block is evolved cooperatively within parallel islands, while cross-block coupling is preserved through full-solution evaluation.
2. **A contextual bandit-based hyper-heuristic** is proposed for operator selection in each decision block at each evolutionary generation. Thus, the operator preferences learned by each bandit reflect the search dynamics of its corresponding block, rather than being diluted by mixed signals from unrelated decision variables, thereby improving the reliability of credit assignment and the search efficiency in heterogeneous subspaces.
3. **A stability-aware gate mechanism** is introduced to regulate the use of high-perturbation operators during cooperative coevolution. Specifically, disruptive resampling operators are suppressed while the search is still improving and are activated only after sustained stagnation, thereby preserving useful partial structures during convergence.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 formulates the cloud–edge–device collaborative scheduling problem. Section 4 presents the overall framework and technical details of CCHIHH. Section 5 reports the experimental results. Section 6 concludes the paper.

2. Related Work

This section reviews related work from two aspects: the evolution of problem formulation for cloud–edge–device collaborative scheduling, and the design of solution algorithms for this problem.

2.1. Task Scheduling Models for Cloud–Edge–Device Collaborative Scheduling

Research related to cloud–edge–device collaborative scheduling has developed from several earlier lines, including cloud-only scheduling, edge–cloud collaborative scheduling, and device–edge–cloud cooperative computing [8, 9, 10]. These studies show the growing need to coordinate heterogeneous computing and manufacturing resources across multiple layers, but they usually focus on only part of the full scheduling process.

DSAC-DE [1] explicitly formulates cloud–edge–device collaborative scheduling as a three-layer problem involving cloud datacenters, edge nodes, and end devices. In this formulation, computational tasks and manufacturing operations are scheduled jointly, so the problem is no longer limited to task offloading alone. This three-layer formulation provides the modeling basis adopted in the present paper.

After this formulation was introduced, subsequent studies mainly extended it in three directions. First, more constraints were considered, such as cross-layer collaboration modes, task precedence relations, resource-capacity limits, deadlines, and device-eligibility constraints [11, 12, 13]. Second, the objectives expanded from makespan-only optimization to joint criteria such as makespan, energy consumption, and service-related indicators [11, 12]. Third, different model construction methods were adopted for different scenarios, including mathematical programming for offline optimization and sequential decision formulations for learning-based scheduling [10, 6].

This paper follows the same three-layer formulation and does not modify the problem definition itself. Instead, it focuses on an algorithmic difficulty that becomes prominent in large-scale instances. Under this formulation, the decision variables form three coupled decision blocks: computational-task offloading, manufacturing-operation sequencing, and device assignment. These blocks differ in meaning, size, and search behavior. As the problem scale grows, a single shared adaptive mechanism becomes less effective across all three blocks. This paper addresses this large-scale algorithmic difficulty.

2.2. Algorithm Design for Cloud–Edge–Device Collaborative Scheduling

Existing methods for cloud–edge–device collaborative scheduling fall into three groups: mathematical programming [10, 14], adaptive evolutionary algorithms, and deep reinforcement learning.

Mathematical programming gives interpretable baselines and, in principle, exact solutions. However, its computing cost grows quickly with the

problem size, especially when offloading, sequencing, and device assignment are optimized at the same time.

Adaptive evolutionary methods include bandit-based adaptive operator selection (AOS) [15, 16, 17, 18] and reinforcement-learning-configured evolution such as DSAC-DE [1]. These methods adjust operators online to make the search more flexible. However, their adaptation is usually done at the population level. When decision blocks with different structures share a single bandit, the reward signal mixes improvements from different sources, and operator credit assignment becomes less reliable.

Island models [19, 20, 21] and cooperative coevolution [22, 23] address large-scale search in different ways. Island models maintain several islands in parallel to preserve diversity. Cooperative coevolution decomposes the problem into smaller blocks and has been applied to scheduling problems such as flexible job-shop scheduling [24] and distributed manufacturing [25]. However, neither approach by itself explains how adaptive operator selection should be organized across decision blocks with different structures.

Deep reinforcement learning [7, 26, 6] can learn context-dependent scheduling policies. However, it usually needs a lot of offline training, and it often generalizes poorly when the action space is large, tightly coupled, and heterogeneous.

In short, existing methods address different aspects of the problem, such as exact modeling, online adaptation, parallel search, or learned scheduling policies. However, under the existing three-layer formulation, large-scale instances still raise a specific algorithmic difficulty: the offloading, sequencing, and device-assignment decisions are tightly coupled, but they have different search characteristics. Existing studies rarely combine decision decomposition with block-specific adaptive operator selection in one framework. This paper focuses on this algorithmic gap.

2.3. Summary and Positioning

Table 1 compares typical method families along four dimensions: whether the decision space is decomposed, how fine the adaptation is, whether parallel search trajectories are used, and whether any stability protection is included. Rule-based methods and classic evolutionary algorithms have none of the four. Global AOS and bandit-based methods are more adaptive, but they still learn from feedback that is mixed over the whole decision space. Island models keep parallel search trajectories, but they do not solve the credit-assignment problem when decision blocks have different structures.

Cooperative coevolution decomposes the problem, but it usually does not give each block its own adaptation. The label "Partial" in Table 1 means that some methods separate variables or stages only in an implicit or heuristic way, not through an explicit block with local adaptation. None of the listed method families has all four properties at the same time.

Table 1: Positioning of CCHIHH relative to representative method families.

Method	**Decision decomposition**	**Adaptation granularity**	**Parallel trajectory**	**Stability protection**
Rule-based / classic EA	No	None	No	No
Global AOS / bandit	Partial	Global	No	Weak
Island models	No	None	Yes	No
CC-based methods	Yes	None	No	No
DSAC-DE [1]	No	Global	No	No
CCHIHH (ours)	**Yes**	**Local**	**Yes**	**Yes**

To fill this gap, this paper combines cooperative coevolution, localized operator adaptation, heterogeneous islands, and a stability gate in one framework. The core idea is to learn operator preferences separately for each decision block, so that the feedback used by each learner comes from the subspace that the learner is actually optimizing.

3. Problem Formulation

This section introduces the mathematical model of the cloud–edge–device collaborative scheduling problem.

3.1. System Architecture

The IIoT platform consists of three resource layers, as illustrated in Fig. 1. The *cloud layer* $\mathcal{C} = \{c_0, \ldots, c_{N_c-1}\}$ provides high computing capacity at the cost of communication latency. The *edge layer* $\mathcal{E} = \{e_0, \ldots, e_{N_e-1}\}$ offers a trade-off between computing efficiency and proximity to production cells. The *device layer* $\mathcal{D} = \{d_0, \ldots, d_{N_d-1}\}$ executes shop-floor operations with minimal communication latency but limited computing capability.

The first category consists of computational tasks $\mathcal{T}_{CE} = \{T_1^{(CE)}, \ldots, T_{N_{CE}}^{(CE)}\}$, each of which is offloaded to either cloud or edge resources. In this study, each manufacturing job is associated with one computational task, i.e., $N_{CE} =$

N_j. Each computational task $T_j^{(CE)}$ is characterized by computation workload $comp_j$, communication size $comm_j$, precedence set P_j, successor data-dependency set $\mathcal{L}_j$, and an ordered tuple of feasible edge servers:

$$\mathcal{A}_j^{(E)} = \left(a_0^{(j)}, a_1^{(j)}, \ldots, a_{n_j^E-1}^{(j)}\right), \qquad n_j^E = \left|\mathcal{A}_j^{(E)}\right|. \tag{1}$$

The second category consists of manufacturing jobs $\mathcal{J} = \{j_1, \ldots, j_{N_j}\}$, where each job j contains an ordered operation set $\mathcal{O}_j = \{o_{j,1}, \ldots, o_{j,n_j}\}$. The computational task $T_j^{(CE)}$ is defined at the job level and is associated with the whole manufacturing job j rather than with an individual operation. Each operation $o_{j,k}$ has processing time $p_{j,k}$ and a fixed ordered tuple of eligible devices, as shown in Eq. (2).

$$D_{j,k} = \left(d_0^{(j,k)}, d_1^{(j,k)}, \ldots, d_{q_{j,k}-1}^{(j,k)}\right), \qquad q_{j,k} = \left|D_{j,k}\right|. \tag{2}$$

In addition, three kinds of cross-layer synchronization relations between computational tasks and manufacturing jobs are defined. Each manufacturing job is associated with one computational task, and this job-level computational task may synchronize with specified operations within the same job. Let $\mathcal{S}_{\mathrm{FS}}$, $\mathcal{S}_{\mathrm{SS}}$, and $\mathcal{S}_{\mathrm{FF}}$ denote the Type I (Finish-to-Start), Type II (Start-to-Start), and Type III (Finish-to-Finish) synchronization relation sets, respectively. Type I requires the computational task of a job to finish before a specified operation of the same job starts. Type II requires the computational task of a job to start no later than a specified operation of the same job. Type III requires the computational task of a job to finish no later than a specified operation of the same job.

3.2. Decision Variables

Let $N_{\mathrm{ops}} = \sum_{j=1}^{N_j} n_j$ denote the total number of manufacturing operations. For each job-level computational task, two continuous variables are used: one variable determines the execution layer, and the other determines the selected server. For each manufacturing operation, two continuous variables are also used: one variable determines the execution order, and the other determines the assigned device. Therefore, a candidate solution is encoded as a continuous vector $\mathbf{x} \in [0,1]^D$ with

$$D = 2N_{CE} + 2N_{\mathrm{ops}}, \tag{3}$$

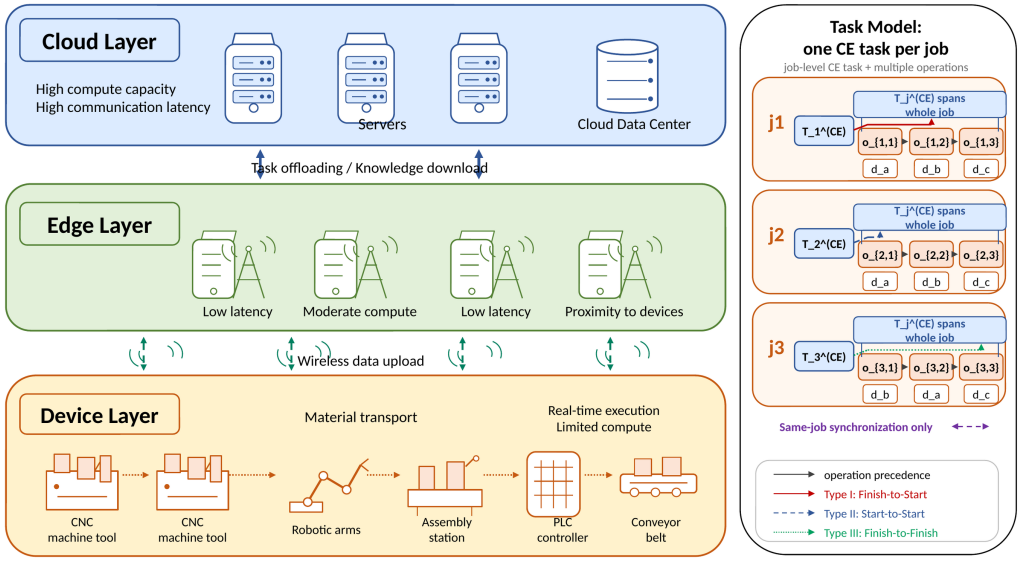

Figure 1: Cloud–edge–device three-layer system architecture (left) and task model (right). Each manufacturing job is associated with one job-level computational task, which is offloaded to cloud or edge resources. The job-level computational task may synchronize with specified manufacturing operations in the same job through three cross-layer synchronization types: Type I (Finish-to-Start), Type II (Start-to-Start), and Type III (Finish-to-Finish).

where the first $2N_{CE}$ dimensions are associated with computational-task decisions and the remaining $2N_{\text{ops}}$ dimensions are associated with manufacturing-operation decisions. The encoded variables are written in Eq. (4).

$$\mathbf{x} = \big[\underbrace{\lambda_1, \ldots, \lambda_{N_{CE}}}_{\text{layer indicator}}, \underbrace{r_1, \ldots, r_{N_{CE}}}_{\text{server selection}}, \underbrace{\pi_1, \ldots, \pi_{N_{\text{ops}}}}_{\text{operation priorities}}, \underbrace{m_1, \ldots, m_{N_{\text{ops}}}}_{\text{device assignment}}\big]. \tag{4}$$

Layer indicator. For each job-level computational task $T_j^{(CE)}$, the continuous variable λ_j is converted into a binary layer indicator $\ell_j \in \{0, 1\}$ by the threshold defined in Eq. (5).

$$\ell_j = \begin{cases} 0, & \lambda_j < 0.5, \\ 1, & \lambda_j \geq 0.5, \end{cases} \tag{5}$$

where $\ell_j = 0$ means that $T_j^{(CE)}$ is executed in the cloud layer, and $\ell_j = 1$

means that it is executed in the edge layer. This threshold rule maps the continuous search variable $\lambda_j \in [0,1]$ to a binary execution-layer decision.

Server selection. Given the fixed order in $\mathcal{A}_j^{(E)}$, the selected server s_j is defined in Eq. (6).

$$s_j = \begin{cases} c_{\min\{\lfloor r_j N_c \rfloor, N_c - 1\}}, & \ell_j = 0, \\ a^{(j)}_{\min\{\lfloor r_j n_j^E \rfloor, n_j^E - 1\}}, & \ell_j = 1. \end{cases} \tag{6}$$

If $\ell_j = 0$, task $T_j^{(CE)}$ is assigned to a cloud server; otherwise, it is assigned to an edge server. The floor operator maps the continuous variable $r_j \in [0,1]$ to a discrete server index by partitioning the interval $[0,1]$ into equal bins. The min operator is used to avoid an out-of-range index when $r_j = 1$.

Operation sequencing. The variables $\pi_1, \ldots, \pi_{N_{\text{ops}}}$ are priority values for manufacturing operations. Sorting these priority values in ascending order gives a permutation σ as shown in Eq. (7).

$$\pi_{\sigma(1)} \leq \pi_{\sigma(2)} \leq \cdots \leq \pi_{\sigma(N_{\text{ops}})}. \tag{7}$$

The resulting execution order is $o_{\sigma(1)}, o_{\sigma(2)}, \ldots, o_{\sigma(N_{\text{ops}})}$. Therefore, a smaller priority value means that the corresponding operation is scheduled earlier.

Device assignment. The device-assignment variables $m_1, \ldots, m_{N_{\text{ops}}}$ are indexed by execution rank rather than by fixed operation identity. This rank-conditioned encoding is intentionally adopted to couple device assignment with the decoded operation sequence. Specifically, after the priority vector induces the operation permutation σ, the variable m_i is used to assign a feasible device to the operation placed at rank i, namely $o_{\sigma(i)}$. Let $D\big(o_{\sigma(i)}\big) = D_{j(i),k(i)}$ denote the eligible-device tuple of this operation:

$$D\big(o_{\sigma(i)}\big) = \big(d_0^{(i)}, d_1^{(i)}, \ldots, d_{q_i-1}^{(i)}\big), \qquad q_i = \big|D\big(o_{\sigma(i)}\big)\big|. \tag{8}$$

Then the device assigned to $o_{\sigma(i)}$ is calculated by Eq. (9).

$$\mu_i = d_{k_i}^{(i)}, \qquad k_i = \min\{\lfloor m_i q_i \rfloor, q_i - 1\}, \qquad i = 1, \ldots, N_{\text{ops}}. \tag{9}$$

Additionally, the floor operator maps the continuous variable $m_i \in [0,1]$ to a discrete device index by partitioning $[0,1]$ into q_i equal bins. The min operator is used to avoid an out-of-range index when $m_i = 1$. Because m_i is attached to execution rank i, any change in the permutation σ changes

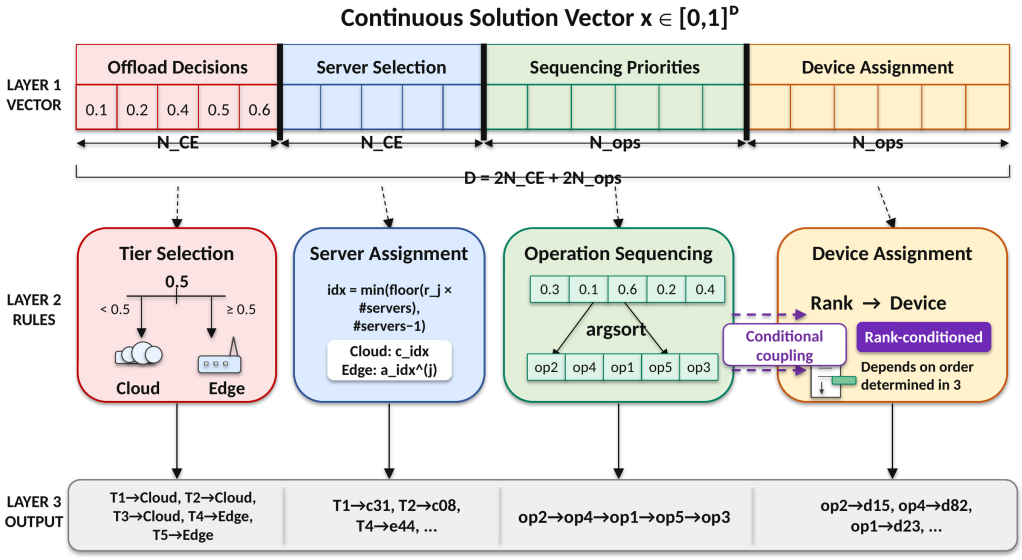

Figure 2: Solution encoding and decoding. The continuous vector $\mathbf{x} \in [0,1]^D$ is decoded into discrete decisions via thresholding (layer indicator), index mapping on ordered tuples (server and device), and the permutation induced by priority values (sequencing). The conditional dependence between sequencing and device assignment (purple arrow) highlights the heterogeneous coupling structure.

the operation associated with m_i. Therefore, the device-assignment block is conditionally dependent on the sequencing block.

Let Ω denote the corresponding decoded discrete search space. Its size is upper-bounded by

$$|\Omega| \leq \prod_{j=1}^{N_{CE}} (N_c + n_j^E) \times \frac{N_{\text{ops}}!}{\prod_{j=1}^{N_j} n_j!} \times \prod_{i=1}^{N_{\text{ops}}} q_i. \tag{10}$$

The second term represents the number of possible interleavings of job-level operation chains when the internal operation order of each job is preserved. This expression highlights that the search space still grows rapidly even after considering within-job precedence constraints. The encoding and decoding process is illustrated in Fig. 2.

3.3. Constraints

For any candidate schedule decoded from $\mathbf{x}$, let $ST_\cdot$ and $ET_\cdot$ denote the start time and end time of the corresponding task or operation, respectively.

Let $T_{\text{comm}}^{(CE)}(i,k)$ denote the communication time required to transfer output data from computational task $T_i^{(CE)}$ to its data-dependent successor $T_k^{(CE)}$. The scheduling process is subject to the following constraints.

Computational-task precedence constraint. If $T_k^{(CE)} \in P_i$, then $T_k^{(CE)}$ must finish before $T_i^{(CE)}$ starts, as shown in Eq. (11).

$$ST_i^{(CE)} \geq ET_k^{(CE)}, \qquad \forall\, T_k^{(CE)} \in P_i. \tag{11}$$

Operation precedence constraint within a job. Within each manufacturing job, operation $o_{j,k+1}$ cannot start before operation $o_{j,k}$ is finished, as shown in Eq. (12).

$$ST_{j,k+1} \geq ET_{j,k}, \qquad j = 1, \ldots, N_j,\ k = 1, \ldots, n_j - 1. \tag{12}$$

Device capacity constraint. Each device can process at most one manufacturing operation at any time. Equivalently, for any two operations $o_{j,k}$ and $o_{j',k'}$ assigned to the same device, as shown in Eq. (13).

$$ET_{j,k} \leq ST_{j',k'} \quad \text{or} \quad ET_{j',k'} \leq ST_{j,k}. \tag{13}$$

Computational-task data-dependency constraint. If $T_k^{(CE)} \in \mathcal{L}_i$, then $T_k^{(CE)}$ cannot start until the required data generated by $T_i^{(CE)}$ has been transferred, as given in Eq. (14).

$$ET_i^{(CE)} + T_{\text{comm}}^{(CE)}(i,k) \leq ST_k^{(CE)}, \qquad \forall\, T_k^{(CE)} \in \mathcal{L}_i. \tag{14}$$

Cross-layer synchronization constraints. Each pair in $\mathcal{S}_{\text{FS}} \cup \mathcal{S}_{\text{SS}} \cup \mathcal{S}_{\text{FF}}$ couples the computational task of a job with a specified manufacturing operation in the same job. The three relation sets impose the following temporal requirements:

$$ET_j^{(CE)} \leq ST_{j,k}, \qquad \forall\, (T_j^{(CE)}, o_{j,k}) \in \mathcal{S}_{\text{FS}}, \tag{15}$$

$$ST_j^{(CE)} \leq ST_{j,k}, \qquad \forall\, (T_j^{(CE)}, o_{j,k}) \in \mathcal{S}_{\text{SS}}, \tag{16}$$

$$ET_j^{(CE)} \leq ET_{j,k}, \qquad \forall\, (T_j^{(CE)}, o_{j,k}) \in \mathcal{S}_{\text{FF}}. \tag{17}$$

Type I (Eq. (15)) requires the computational task of a job to be completed before the specified manufacturing operation of the same job starts.

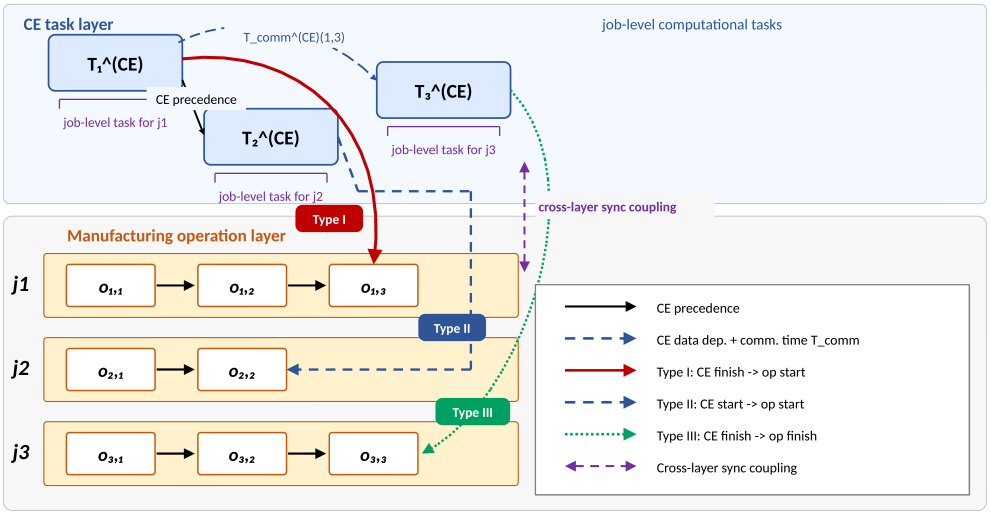

Figure 3: Task dependency graph. Upper layer: job-level computational tasks. Lower layer: manufacturing operations. Red solid, blue dashed, and green dotted arrows denote Type I (Finish-to-Start), Type II (Start-to-Start), and Type III (Finish-to-Finish) cross-layer synchronization, respectively. Purple brackets/arrows indicate the job-level computational-task span within each job.

This type applies to cases in which the operation consumes the result generated by the job-level computational task. Type II (Eq. (16)) requires the computational task of a job to start no later than the specified manufacturing operation of the same job starts. This type applies to cases in which the computational process must be initiated before or at the beginning of the physical operation. Type III (Eq. (17)) requires the computational task of a job to finish no later than the specified manufacturing operation of the same job finishes. This type applies to cases in which the computational result, such as support, monitoring, or quality-related information, must be available before the physical operation is completed.

Device feasibility is guaranteed by the decoding procedure because each operation is assigned only to an eligible device. All temporal feasibility conditions, including computational-task precedence, job and operation ordering, device capacity, computational-task data dependencies, and cross-layer synchronization, are checked during schedule construction and objective evaluation. An example is shown in Fig. 3.

3.4. Objectives

Two objectives are considered in this study, i.e., makespan and energy consumption.

Makespan. The makespan is defined as the latest completion time among all computational tasks in the schedule, as shown in Eq. (18).

$$f_1(\mathbf{x}) = \max_{i=1,\ldots,N_{CE}} ET_i^{(CE)}. \tag{18}$$

This expression takes the maximum end time because the schedule is completed only when the last computational task is finished.

Energy consumption. The energy objective includes communication energy and server energy. Let $t_i^{(\mathrm{comm})}$ denote the communication time of computational task $T_i^{(CE)}$, let P_{comm} denote the communication power coefficient, let u_s denote the discretized utilization level of server s, let $P(u_s)$ denote the power consumption rate of server s at utilization level u_s, and let $T_s^{(\mathrm{act})}$ denote the total active time of server s during the scheduling horizon. The total energy consumption is given in Eq. (19).

$$f_2(\mathbf{x}) = P_{\mathrm{comm}} \sum_{i=1}^{N_{CE}} t_i^{(\mathrm{comm})} + \sum_{s \in \mathcal{C} \cup \mathcal{E}} P(u_s) \frac{T_s^{(\mathrm{act})}}{1000}. \tag{19}$$

The first term represents communication energy, and the second term represents the energy consumed by cloud and edge servers.

To obtain a scalar fitness for single-objective search, the two objectives are normalized and combined as shown in Eq. (20).

$$f(\mathbf{x}) = \alpha \frac{f_1(\mathbf{x})}{f_1^{(\mathrm{ref})}} + (1-\alpha) \frac{f_2(\mathbf{x})}{f_2^{(\mathrm{ref})}}, \qquad \alpha \in (0,1), \tag{20}$$

where $f_1^{(\mathrm{ref})}$ and $f_2^{(\mathrm{ref})}$ are instance-specific reference values used for normalization, and α controls the trade-off between makespan and energy consumption.

4. The Cooperative Coevolution Algorithm with a Heterogeneous-Island-Based Hyper-Heuristic

This section presents the proposed CCHIHH method in a top-down manner. We first introduce the overall framework and workflow of CCHIHH and then describe its main components in detail. The method is motivated by

four requirements: (i) decomposition is needed because the decision space contains heterogeneous blocks; (ii) localized adaptation is needed because shared feedback obscures operator credit assignment; (iii) parallel islands are needed to maintain multiple search trajectories in high-dimensional spaces; and (iv) a stability gate is needed to prevent disruptive perturbations during the active-improvement phase, when the best fitness is still improving.

These requirements are realized through cooperative coevolution, localized contextual bandits, behaviorally heterogeneous islands, and stability-gated diversity regulation, respectively.

For clarity, let

$$\mathcal{B} = \{\text{off}, \text{seq}, \text{dev}\}$$

denote the set of decision blocks. The corresponding block dimensions are

$$d_{\text{off}} = 2N_{CE}, \qquad d_{\text{seq}} = N_{\text{ops}}, \qquad d_{\text{dev}} = N_{\text{ops}}.$$

An individual on island m is a complete solution vector

$$\mathbf{x}_i^{(m)} = \left(\mathbf{x}_{i,\text{off}}^{(m)}, \mathbf{x}_{i,\text{seq}}^{(m)}, \mathbf{x}_{i,\text{dev}}^{(m)}\right),$$

where only one block is modified during a block-wise update and the other two blocks are treated as read-only context. The population on island m is denoted by $P^{(m)}$, and its size is $n_m = N/M$.

4.1. Framework Overview and Overall Workflow

Fig. 4 gives a framework-level overview of CCHIHH. The algorithm follows a top-down divide-and-conquer design. A complete solution is first decomposed into three decision blocks, namely the offload block, the sequence block, and the device block. These blocks are optimized cooperatively through full-solution evaluation, so that cross-block coupling is still considered during selection.

On each island, every decision block is equipped with its own localized contextual bandit. The bandit selects operators from the block-specific operator pool according to the current search state. The selected operator is then checked by the stability gate, which suppresses high-perturbation resampling operators during the active-improvement phase. Meanwhile, heterogeneous islands maintain multiple search trajectories, and ring migration periodically transfers useful individuals between neighboring islands.

The overall procedure is as follows:

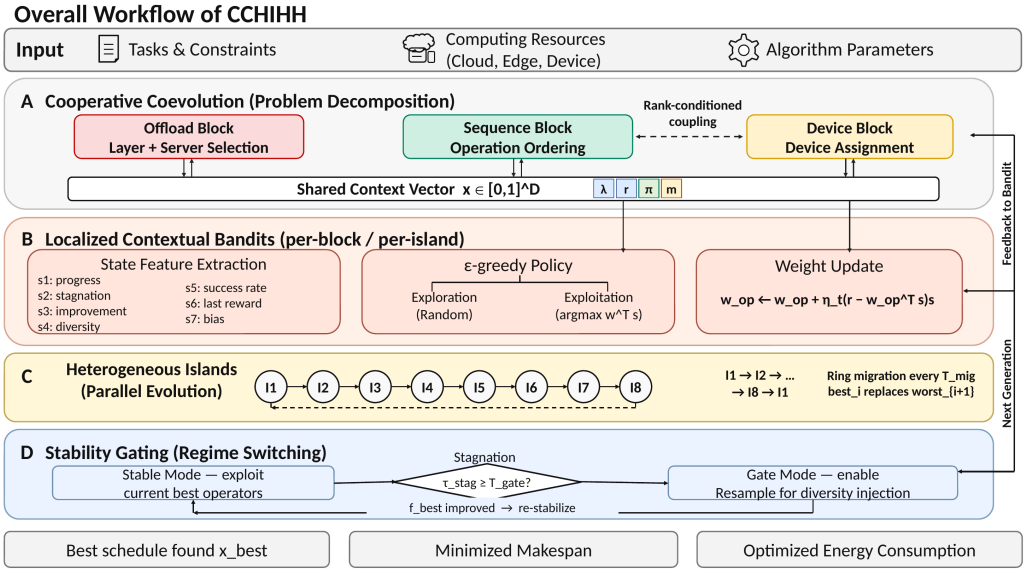

Figure 4: Overall workflow of CCHIHH. The three blocks are optimized sequentially via cooperative coevolution. Within each block, the contextual bandit selects operators subject to the stability gate, and ring migration periodically transfers the best individuals between neighboring islands.

1. **Initialization**: Generate N individuals, partition them into M islands, and initialize all bandit weights to zero.
2. **Block-wise evolution**: Repeat the following process on every island until the total evaluation budget $E_{\max}$ is exhausted. In each outer generation, the three decision blocks are updated once in the fixed order offload $\rightarrow$ sequence $\rightarrow$ device. For the current decision block, the corresponding localized contextual bandit selects one operator, and the stability gate filters it if necessary. The executed operator generates offspring, environmental selection is performed, and the resulting full solution is evaluated. Here, $E_{\max}$ denotes the total number of full-solution evaluations over all decision block updates.
3. **Migration**: Every T_{mig} outer generations, perform ring migration.
4. **Output**: Return the best solution found across all islands.

The following subsections describe the main components of CCHIHH in detail, including problem decomposition, localized operator selection, heterogeneous islands, stability-gated diversity regulation, and the operator pools.

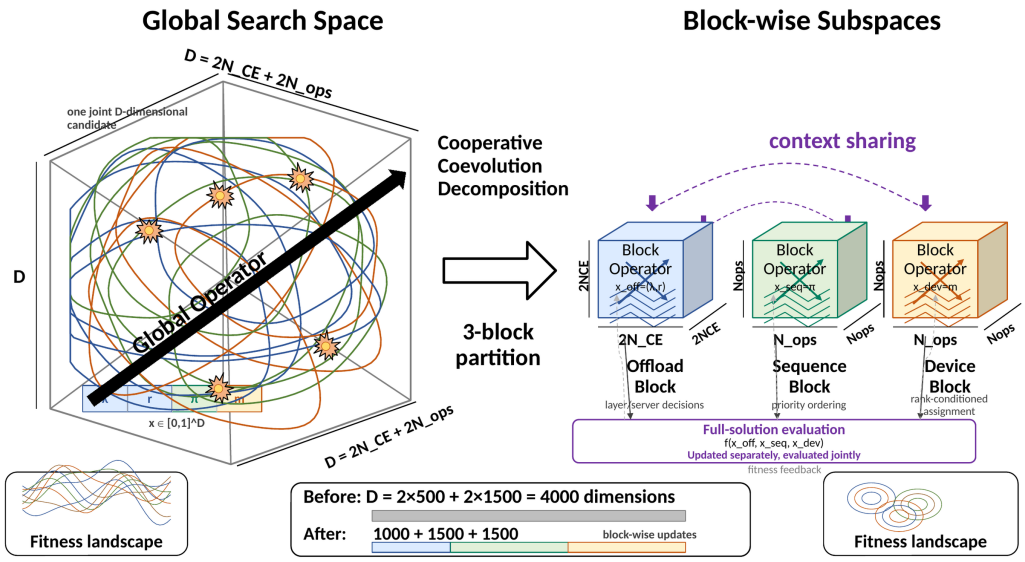

Figure 5: Cooperative coevolution decomposition. Left: the global search space of dimension $D = 2N_{CE} + 2N_{\text{ops}}$. Right: three decision blocks connected through full-solution evaluation.

4.2. Cooperative Coevolution for Problem Decomposition

As shown in the overall framework in Fig. 4, the solution vector is divided into three decision blocks according to the encoding defined in Section 3.2:

- **Offload block** $\mathbf{x}_{\text{off}} = (\boldsymbol{\lambda}, \mathbf{r}) \in [0,1]^{2N_{CE}}$: variables for execution-layer selection and server selection of computational tasks.
- **Sequence block** $\mathbf{x}_{\text{seq}} \in [0,1]^{N_{\text{ops}}}$: priority variables that determine the execution order of manufacturing operations.
- **Device block** $\mathbf{x}_{\text{dev}} \in [0,1]^{N_{\text{ops}}}$: rank-conditioned device-assignment variables for manufacturing operations.

The decomposition is illustrated in Fig. 5. For $N_{CE} = 500$ and $N_{\text{ops}} = 1500$, a single 4000-dimensional problem is reduced to blocks of 1000, 1500, and 1500 dimensions.

During optimization, the three decision blocks are updated sequentially [22, 23]. When one decision block is updated, the other two decision blocks are kept fixed and serve as the context for full-solution evaluation. The updated

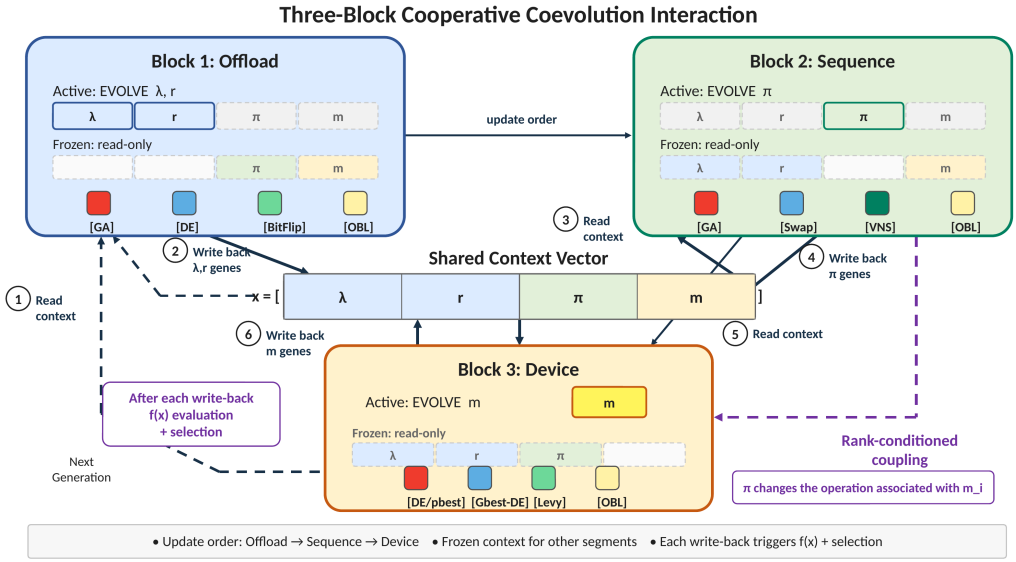

Figure 6: Three-block interaction protocol. Each block evolves its own portion of the solution vector while other segments are frozen as read-only context. The rank-conditioned coupling between the sequence and device blocks (purple arrow) reflects the conditional dependence in decoding.

decision block is then inserted into the full solution, and the resulting solution is evaluated by the fitness function in Section 3.4. Therefore, cross-block interactions are always assessed at the full-solution level.

Effect of inter-block coupling. The strongest inter-block coupling occurs between the sequence block and the device block because the device-assignment variables are rank-conditioned. A change in $\mathbf{x}_{\text{seq}}$ changes the decoded operation order and may therefore change the operation associated with each component of $\mathbf{x}_{\text{dev}}$. This conditional dependence can make alternating block updates unstable if the blocks are evaluated independently. CCHIHH reduces this risk in three ways: (i) every offspring block is inserted back into a complete solution before evaluation; (ii) migration transfers complete solution vectors rather than isolated blocks; and (iii) the stability gate suppresses high-disruption resampling while the island-best solution is still improving.

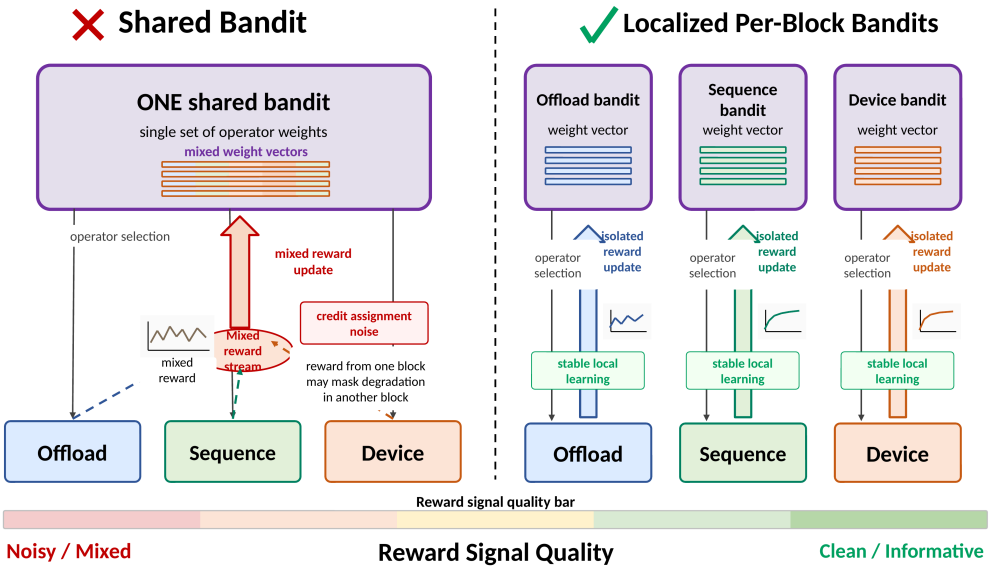

Figure 7: Shared bandit vs. localized bandit. A single shared bandit receives mixed rewards from all blocks and produces noisy updates (left). Localized bandits receive isolated rewards and enable stable operator preference learning (right).

4.3. Localized Contextual Bandit for Operator Selection

The hyper-heuristic component of CCHIHH is implemented as a set of localized contextual bandits. Specifically, each decision block on each island is equipped with a localized linear contextual bandit for operator selection. Thus, operator selection is performed separately for the offload block, the sequence block, and the device block. Because each localized contextual bandit observes only the update of its own decision block, the reward it receives is not mixed with improvements from other decision blocks. The state vector combines global progress information with island-level and block-level search indicators. Fig. 7 compares this design with a shared bandit.

State vector. For the current decision block on one island, let

$$\mathbf{s} = [s_1, s_2, \ldots, s_7]^T \in \mathbb{R}^7$$

denote the state vector used by the contextual bandit. Its seven components are listed in Table 2. All components are normalized to $[0, 1]$.

Table 2: Components of the state vector $\mathbf{s}$.

Index	Symbol	Definition
1	s_1	Search progress, $e/E_{\max}$
2	s_2	Stagnation level, $\min(\tau_{\text{stag}}/T_{\text{gate}}, 1)$
3	s_3	Recent improvement rate over the latest window
4	s_4	Normalized population diversity of the current decision block on the current island
5	s_5	Success rate of the current decision block over the latest window
6	s_6	Most recent reward after tanh normalization
7	s_7	Constant bias term, $s_7 \equiv 1$

Operator scoring and selection. Each operator *op* has a weight vector $\mathbf{w}_{op} \in \mathbb{R}^7$. Its estimated value under state $\mathbf{s}$ is

$$\text{score}(op, \mathbf{s}) = \mathbf{w}_{op}^T \mathbf{s}. \tag{21}$$

Operator selection follows an ϵ-greedy rule with decaying exploration:

$$\epsilon(e) = \max\left(\epsilon_{\min},\ \epsilon_0 \cdot \exp\left(-\epsilon_k \cdot \frac{e}{E_{\max}}\right)\right). \tag{22}$$

Reward definition. After the selected operator is applied to the current decision block, let f_{old} and f_{new} denote the best full-solution fitness values before and after the update, respectively. Let D_{old} and D_{new} denote the corresponding population-diversity values. The fitness-improvement ratio is defined as

$$I = \frac{f_{\text{old}} - f_{\text{new}}}{\max\{|f_{\text{old}}|, \varepsilon\}}, \tag{23}$$

where $\varepsilon > 0$ is a small constant used to avoid division by zero. The diversity change is defined as

$$\Delta D = D_{\text{new}} - D_{\text{old}}. \tag{24}$$

The reward is defined as

$$r = \text{clip}(I + \lambda \Delta D,\ -c,\ c), \tag{25}$$

where λ controls the contribution of diversity preservation, $c > 0$ is the clipping threshold, and

$$\text{clip}(z, -c, c) = \max\{-c, \min(z, c)\}. \tag{26}$$

This definition combines fitness improvement and diversity preservation in a bounded reward signal.

Weight update. The weight vector of the selected operator is updated by

$$\mathbf{w}_{op} \leftarrow \mathbf{w}_{op} + \eta_t\big(r - \mathbf{w}_{op}^T\mathbf{s}\big)\mathbf{s}, \tag{27}$$

where $r - \mathbf{w}_{op}^T\mathbf{s}$ is the prediction error. This update moves the estimated operator score of the selected operator toward the observed reward under the current state. To improve stability in later search stages, the learning rate is decayed as

$$\eta_t = \eta_0 \exp(-\eta_k t), \tag{28}$$

where t is the update count of the executed operator. If the stability gate vetoes an *OBL Resample* proposal and replaces it with a fallback operator, the reward is assigned only to the fallback operator that is actually executed. The vetoed *OBL Resample* operator receives no weight update in that block update. This avoids assigning credit to an operator whose offspring has not been evaluated.

Algorithm 1 gives the operator-proposal step of the localized contextual bandit. The actual operator execution, reward computation, and weight update are integrated with the stability gate in Algorithm 2. Fig. 8 summarizes the closed-loop learning process.

Algorithm 1 Contextual Bandit Operator Proposal (per block, per island)

Require: island population P_{isl}, evaluation counter e, budget $E_{\max}$, stagnation counter τ_{stag}, last reward r_{last}, operator weights $\mathbf{W}$
Ensure: proposed operator op_{bandit} and state vector $\mathbf{s}$

Compute state $\mathbf{s}$ from P_{isl}, $e/E_{\max}$, τ_{stag}, r_{last}; compute $\epsilon(e)$
if rand() $< \epsilon$ **then**
  $op_{\text{bandit}} \leftarrow \text{RandomChoice}(\mathcal{O}_b)$
else
  $op_{\text{bandit}} \leftarrow \arg\max_{i \in \mathcal{O}_b} \mathbf{w}_i^T\mathbf{s}$
end if
return $op_{\text{bandit}}, \mathbf{s}$

Why localized learners help. A shared bandit mixes rewards from different decision blocks, which makes operator credit assignment noisier as block size and inter-block coupling increase. In contrast, localized contextual

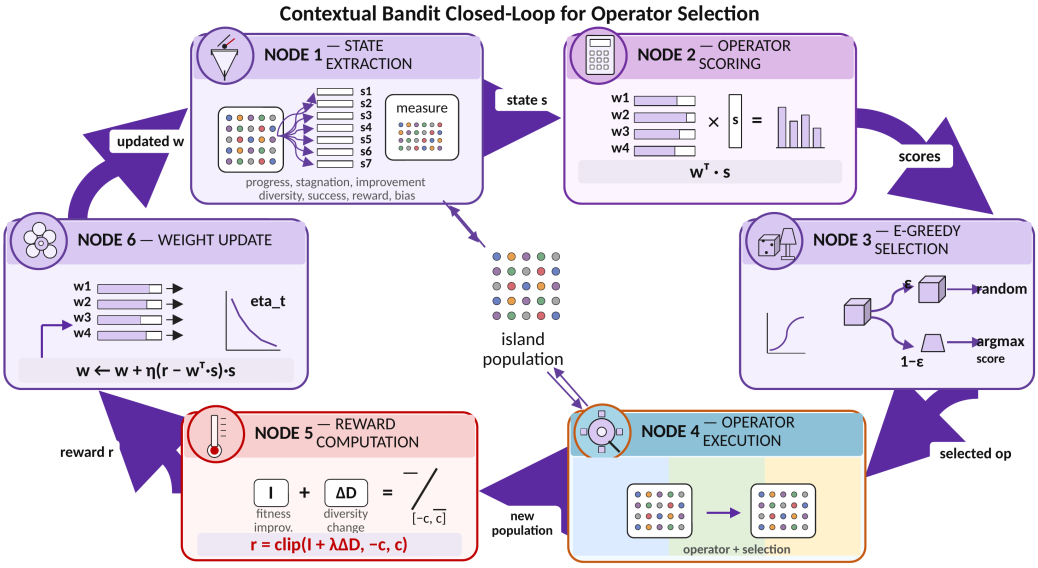

Figure 8: Contextual bandit closed-loop: state extraction → operator scoring ($\mathbf{w}^T\mathbf{s}$) → ε-greedy selection → operator execution → reward computation → weight update.

bandits receive more isolated feedback and can learn more stable operator preferences. This advantage is particularly important for the strongly coupled sequence and device blocks. Section 5.3 provides empirical evidence.

4.4. Heterogeneous Islands with Ring Migration

The population is partitioned into M islands, and each island maintains its own localized bandit weights for the three decision blocks. In this paper, island heterogeneity refers to *behavioral heterogeneity*: all islands share the same candidate operator pools and base parameter settings, but their bandit weights, local population states, stagnation counters, and reward histories are updated independently. Consequently, different islands can form different operator-selection distributions and search trajectories during evolution. This design preserves implementation simplicity while still allowing island-level search specialization.

Every T_{mig} outer generations, one-hop ring migration is performed. Let $\mathbf{x}_{\text{best}}^{(m)}$ and $\mathbf{x}_{\text{worst}}^{(m)}$ denote the best and worst individuals on island m, respectively. The migration rule is

$$\mathbf{x}_{\text{worst}}^{(m+1)} \leftarrow \mathbf{x}_{\text{best}}^{(m)}, \qquad m = 1, \ldots, M, \tag{29}$$

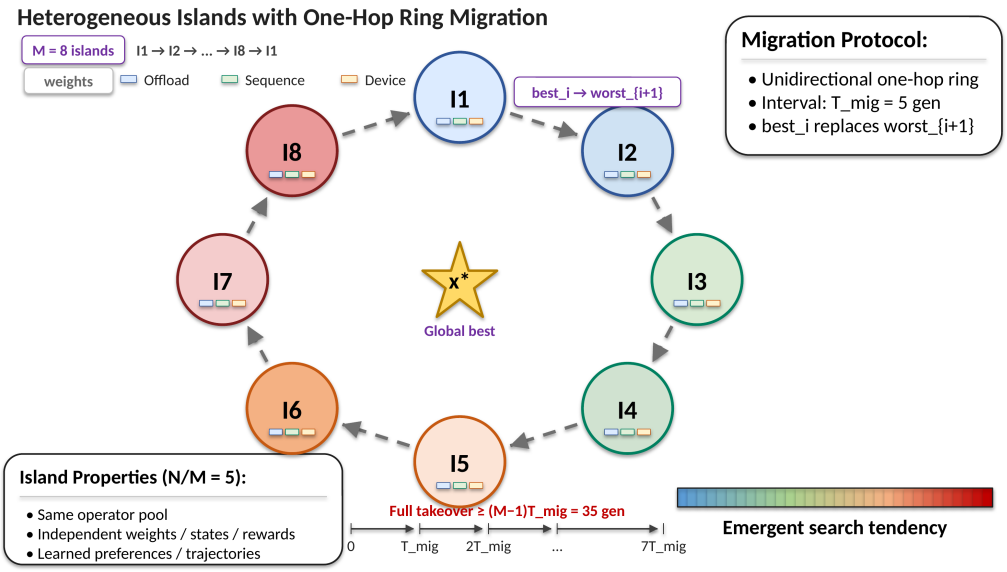

Figure 9: Heterogeneous island model with ring migration.

where island $M+1$ is identified with island 1. The migrated individual is a complete solution vector, not an isolated block. This one-hop migration gradually spreads useful full-solution structures while preserving island-level behavioral heterogeneity.

Diversity preservation. In a unidirectional ring with M islands, a solution requires at least $M-1$ migration events to reach all islands. Therefore, the minimum full-takeover time is $(M-1)\cdot T_{\text{mig}}$. For example, when $M=8$ and $T_{\text{mig}}=5$, full takeover requires at least 35 generations. During this period, each island evolves independently with its own bandit weights.

4.5. Stability-Gated Diversity Regulation

Each decision block includes a high-perturbation *OBL Resample* operator to help the search escape deep local optima. However, applying this operator during periods of continued improvement may destroy useful partial structures. To avoid this problem, a stability gate is introduced. The gate is controlled by an island-local stagnation counter $\tau_{\text{stag}}^{(m)}$, which measures whether the best solution on island m has improved recently. Let T_{gate} denote the gating threshold, let $\mathcal{O}_b$ denote the operator set of the current decision block, and let $\mathcal{O}_{\text{res}}$ denote the set containing the *OBL Resample* operator.

The admissible operator set is defined as

$$\mathcal{O}_{\mathrm{adm}}(\tau_{\mathrm{stag}}^{(m)}) = \begin{cases} \mathcal{O}_b \setminus \mathcal{O}_{\mathrm{res}}, & \tau_{\mathrm{stag}}^{(m)} < T_{\mathrm{gate}}, \\ \mathcal{O}_b, & \tau_{\mathrm{stag}}^{(m)} \geq T_{\mathrm{gate}}. \end{cases} \tag{30}$$

Therefore, *OBL Resample* is enabled only after sufficient stagnation has been observed. In implementation, if the localized contextual bandit selects *OBL Resample* when $\tau_{\mathrm{stag}} < T_{\mathrm{gate}}$, a fixed fallback operator is executed instead. The fallback operator is DE/current-to-$p$best/1 for the offload block, Swap+Segment for the sequence block, and Gbest-DE for the device block. The stagnation counter is reset to zero whenever the best fitness improves by more than 10^{-12}; otherwise, it is increased by one. Within each decision block update, the offspring generated by the selected operator compete with the current population through environmental selection. This step is referred to as "selection update" in Algorithm 2. Algorithm 2 gives the gating procedure, and Fig. 10 shows the corresponding state transition.

Algorithm 2 Stability-Gated Block Update with Bandit Learning

Require: island population $P^{(m)}$, block ID b, evaluation counter e, budget $E_{\max}$, island stagnation counter $\tau_{\text{stag}}^{(m)}$, gate threshold T_{gate}, last reward $r_{\text{last}}^{(m,b)}$, operator weights $\mathbf{W}_b^{(m)}$

Ensure: executed operator op_{exec}, updated $P^{(m)}$, updated $\mathbf{W}_b^{(m)}$, updated $\tau_{\text{stag}}^{(m)}$, updated $r_{\text{last}}^{(m,b)}$

Obtain op_{bandit} and state vector $\mathbf{s}^{(m,b)}$ from Algorithm 1
if $op_{\text{bandit}} \in \mathcal{O}_{\text{res}}$ **and** $\tau_{\text{stag}}^{(m)} < T_{\text{gate}}$ **then**
    $op_{\text{exec}} \leftarrow op_{\text{fallback}}(b)$ {veto resampling during active improvement}
else
    $op_{\text{exec}} \leftarrow op_{\text{bandit}}$
end if
$f_{\text{old}} \leftarrow \text{BestFitness}(P^{(m)})$
$D_{\text{old}} \leftarrow \text{Div}_b(P^{(m)})$
Apply op_{exec} to block b and generate full trial solutions by Eq. (**??**)
Evaluate the generated full trial solutions
Update $P^{(m)}$ by environmental selection in Eq. (**??**)
$f_{\text{new}} \leftarrow \text{BestFitness}(P^{(m)})$
$D_{\text{new}} \leftarrow \text{Div}_b(P^{(m)})$
Compute reward r by Eq. (25)
Update only the weight vector of the executed operator $\mathbf{w}_{op_{\text{exec}}}$ by Eq. (27)
$r_{\text{last}}^{(m,b)} \leftarrow r$
if $f_{\text{new}} < f_{\text{old}} - 10^{-12}$ **then**
    $\tau_{\text{stag}}^{(m)} \leftarrow 0$
else
    $\tau_{\text{stag}}^{(m)} \leftarrow \tau_{\text{stag}}^{(m)} + 1$
end if
return op_{exec}, $P^{(m)}$, $\mathbf{W}_b^{(m)}$, $\tau_{\text{stag}}^{(m)}$, $r_{\text{last}}^{(m,b)}$

Design rationale. During periods of continued improvement (low τ_{stag}), low-perturbation operators are preferred because they better preserve useful partial structures. During stagnation (high τ_{stag}), stronger perturbations become more useful for escaping local optima. The stability gate follows this idea by delaying high-disruption operators until stagnation occurs. The ablation results in Section 5.2 support this design.

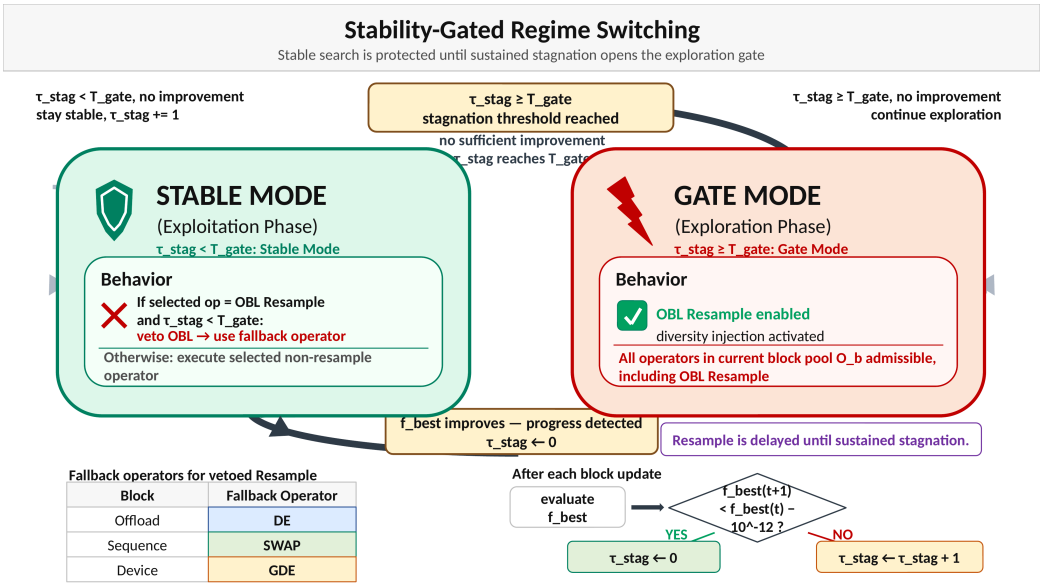

Figure 10: Stability-gating state transition. In Stable Mode, Resample selections are vetoed and replaced by fallback operators. When $\tau_{\text{stag}} \geq T_{\text{gate}}$, the system enters Gate Mode, enabling Resample. Any fitness improvement resets τ_{stag} and returns the system to Stable Mode.

4.6. Operator Pools

Each decision block has four candidate operators (Table 3). The operators are implementation choices, whereas the main contribution lies in how localized adaptation is organized across decision blocks. All operators use fixed parameters to maintain stable bandit learning. The parameter settings are given in Section 5.1.

Table 3: Operator pool for each block. Op1–Op3 are block-specific, whereas Op4 is the shared OBL Resample operator controlled by the stability gate [27].

Block	**Op1**	**Op2**	**Op3**	**Op4**
Offload	GA (SBX)	DE/current-to-$p$best/1	BitFlip+Cauchy	OBL Resample
Sequence	GA (SBX)	Swap+Segment	Hetero. VNS	OBL Resample
Device	DE/current-to-$p$best/1	Gbest-DE	Lévy Flight	OBL Resample

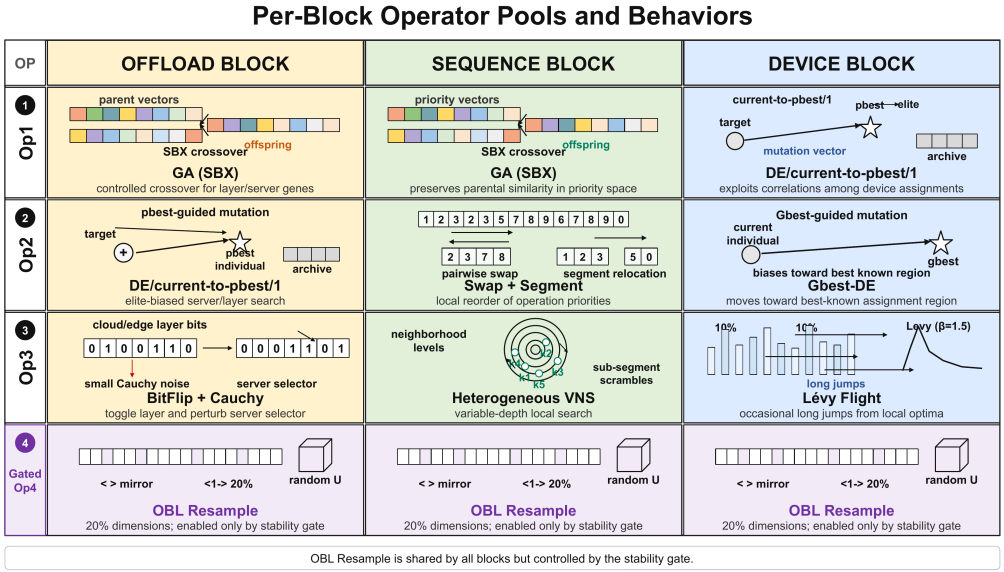

Figure 11: Per-block operator behavior panel. Each column corresponds to one block and each row to one operator. All blocks share OBL Resample controlled by the stability gate.

The **offload block** involves coupled discrete decisions. GA with SBX [28] provides controlled offspring generation; DE/current-to-$p$best/1 [29] biases mutation toward elite individuals; BitFlip toggles layer decisions while perturbing the server subvector with a Cauchy step [30].

The **sequence block** searches over priority vectors that determine operation order. SBX-based GA preserves partial ordering through parental similarity. Sequence Swap applies pairwise exchanges together with segment moves in the spirit of Or-opt [31]. Heterogeneous VNS [32] employs five neighborhood structures of increasing perturbation strength.

The **device block** operates on continuous variables decoded into eligible devices. DE/current-to-$p$best/1 with archive exploits correlations across device assignments. Gbest-DE biases mutation toward the best known region. Lévy Flight with $\beta = 1.5$ introduces heavy-tailed jumps for escaping local optima.

For all three blocks, OBL Resample selects $\lceil 0.2 \cdot D_{\text{block}} \rceil$ dimensions at random. Each selected dimension is mapped to its opposite point, $x_i' = 1 - x_i$, with probability 0.5, and reinitialized from $\mathcal{U}(0, 1)$ otherwise. A visual summary is provided in Fig. 11.

4.7. Computational Complexity and Overhead Separation

After introducing the overall workflow and the main algorithmic components, this subsection provides a formal accounting of the computational costs to demonstrate that the adaptive mechanisms do not dominate the overall complexity. Because each block update is evaluated on the concatenated full solution, the budget $E_{\max}$ counts total full-solution evaluations rather than outer generations.

Fitness evaluation. A single fitness evaluation involves (a) decoding the continuous vector into discrete decisions, where constructing the permutation σ in Eq. (7) requires sorting the priority vector and costs $\mathcal{O}(N_{\text{ops}} \log N_{\text{ops}})$, and (b) discrete-event simulation over the decoded schedule, costing $\mathcal{O}(N_{\text{ops}}\,\bar{d} + N_{CE}\,\bar{p})$, where $\bar{d}$ is the average number of eligible devices per operation and $\bar{p}$ is the average number of predecessors per computational task. The total per-evaluation cost is

$$T_{\text{eval}} = \mathcal{O}(N_{\text{ops}} \log N_{\text{ops}} + N_{\text{ops}}\,\bar{d} + N_{CE}\,\bar{p}). \tag{31}$$

Adaptive control overhead. Each localized contextual bandit update (Eq. (27)) involves computing a score $\mathbf{w}_{op}^{T}\mathbf{s}$ for each of $|\mathcal{O}_b| = 4$ operators in a $p = 7$-dimensional state space and updating one weight vector. The per-step cost is therefore $\mathcal{O}(|\mathcal{O}_b| \cdot p)$. Because both the state dimension and the operator-pool size are small constants, this overhead remains negligible relative to the cost of full-solution fitness evaluation.

Migration overhead. Ring migration occurs periodically and involves copying M individuals (one per island), each requiring $\mathcal{O}(D)$ time. Since migration is infrequent and only transfers one individual per island, its overhead is small relative to repeated schedule evaluation.

Stability gate overhead. The gate check (Eq. (30)) is an $\mathcal{O}(1)$ comparison per operator invocation, contributing negligible cost.

Overhead summary. The dominant computational cost remains full-solution fitness evaluation, whose per-evaluation complexity is given by Eq. (31). In comparison, the adaptive mechanisms are lightweight: the localized contextual bandit update operates in a fixed low-dimensional state space, migration transfers only a small number of individuals periodically, and the gate check is $\mathcal{O}(1)$. Therefore, the additional overhead of localized adaptation does not dominate the overall runtime on the tested instances.

5. Experiments and Discussion

This section evaluates CCHIHH through four experimental questions. First, we examine whether each major component of CCHIHH contributes to solution quality and whether the localized contextual bandit is more effective than a shared-bandit design (Sections 5.2–5.3). Second, we analyze the sensitivity of CCHIHH to the objective weight α and further inspect the makespan–energy trade-off induced by different scalarization weights (Section 5.4). Third, we test the robustness of the framework under resource and communication degradation (Section 5.5). Finally, we compare CCHIHH with representative evolutionary baselines under the same full-solution evaluation budget, while reporting PPO only as a cross-paradigm reference (Section 5.6).

5.1. Experimental Settings

Problem instances. Three instance scales are used in the main experiments: T100 (100 computational tasks, 100 cloud servers, 100 edge servers, and 300 devices), T200 (200/100/100/300), and T500 (500/200/200/800). Here, Tn denotes an instance with n job-level computational tasks; the corresponding manufacturing operations are generated by the same job-template and synchronization-generation rules used in the base configuration. The scale study in Section 5.3 further considers instances from T50 to T500. T100 is used as the base configuration because it reflects realistic resource distributions and synchronization patterns. The larger instances are generated by scaling this configuration while preserving the coupling density and the resource heterogeneity ratios.

Algorithm parameters. The population size is $N = 40$, and the number of islands is $M = 8$, so each island contains 5 individuals. All methods use the same total evaluation budget, $E_{\max} = 4 \times 10^5$, where $E_{\max}$ counts full-solution evaluations. The migration interval is $T_{\text{mig}} = 5$ outer generations, and the gating threshold is $T_{\text{gate}} = 15$. The bandit parameters are $\epsilon_0 = 0.2$, $\epsilon_{\min} = 0.02$, $\epsilon_k = 0.01$, $\eta_0 = 0.05$, $\eta_k = 0.002$, clipping threshold $c = 0.2$, and diversity weight $\lambda = 0.1$. The operator parameters are as follows: DE uses $F = 0.5$ and $CR = 0.5$; GDE uses $F \sim \mathcal{U}(0.2, 0.8)$ and $CR \sim \mathcal{U}(0.1, 0.6)$; BitFlip uses $p_{\text{flip}} = 0.1$; VNS uses $k_{\max} = 5$ and $n_s = 5$; Lévy Flight uses $p_d = 0.1$, $\beta = 1.5$, and $\alpha_{\text{levy}} = 0.01$; the resample rate is $\rho = 0.2$; and the GA mutation rate is $p_m = 0.1$.

Baselines. Six evolutionary baselines and one reinforcement-learning reference are considered. The evolutionary baselines are CGA [2], IMOMA [3], DSAC-DE [1], RDE [33], NL-SHADE-LBC [4], and L-SRTDE [5]. PPO [34], which has been used in cloud–edge scheduling studies such as [6], is included only as a cross-paradigm reference. Its result is discussed separately in Section 5.6.

Ablation variants. Five variants isolate the role of each component: CCHIHH-noCC (no cooperative coevolution), CCHIHH-noHI (single population), CCHIHH-noCB (no contextual bandit), CCHIHH-noMig (no migration), and CCHIHH-noGate (no stability gating).

Degradation scenarios. Four degraded conditions are tested on the large-scale instances: cloud-capacity reduction, edge-capacity reduction, device-capacity reduction, and communication-time inflation. For the resource-side scenarios, three degradation levels are used: 10%, 20%, and 30% reduction. For the communication-side scenario, three inflation levels are used: 20%, 40%, and 60%.

Fairness and reproducibility. All methods use the same encoding/decoding scheme and the same discrete-event simulation engine. For the main convergence, ablation, robustness, and baseline-comparison experiments, each method is run with 10 independent random seeds. The same seed index generates the same instance realization and initial population across compared methods whenever applicable. The reference values $f_1^{(\mathrm{ref})}$ and $f_2^{(\mathrm{ref})}$ are computed once for each instance and then fixed for all algorithms. Statistical significance is evaluated using a two-sided Wilcoxon signed-rank test at the 0.05 level for pairwise comparisons. No additional wall-clock stopping criterion is used. Therefore, the runtime values reported in Fig. 24 are the actual execution times required by each method to exhaust the same evaluation budget $E_{\max}$. All experiments are conducted on a machine with an Intel Core Ultra 7 255HX CPU (20 cores / 20 threads) and 32 GB RAM. All algorithms are implemented in C++ and compiled with g++ `-O2`.

5.2. Ablation Study

Table 4 and Fig. 12 report the performance loss caused by removing each component. All ablation variants are evaluated under the same full-solution evaluation budget $E_{\max}$ as the full method. The performance loss is computed as

$$\mathrm{Loss} = \frac{f_{\mathrm{variant}} - f_{\mathrm{full}}}{f_{\mathrm{full}}} \times 100\%, \tag{32}$$

where f_{full} and f_{variant} denote the final best fitness values of CCHIHH and the corresponding ablation variant, respectively. Since the objective is minimized, a larger loss indicates a larger degradation after removing the component.

Table 4: Performance loss (%) when removing individual components. The largest loss in each scale is shown in bold.

Ablation variant	**T100**	**T200**	**T500**
CCHIHH-noCC	**9.27**	34.92	**51.27**
CCHIHH-noHI	2.62	25.41	35.41
CCHIHH-noCB	3.58	24.62	6.68
CCHIHH-noMig	2.89	**35.45**	42.24
CCHIHH-noGate	0.66	12.23	4.21

Two observations can be made. First, the structural components, namely cooperative coevolution, heterogeneous islands, and migration, become more important as the problem scale increases. On T500, removing cooperative coevolution causes a 51.27% performance loss, and removing migration causes a 42.24% loss. Second, the contextual bandit component and the stability gate show a different trend. Their effects are strongest on T200 (24.62% and 12.23%) and remain positive on T500 (6.68% and 4.21%). This suggests that adaptive operator learning is most effective after the search structure has been established. Therefore, the performance of CCHIHH comes from the joint effect of structural decomposition, parallel search, and localized adaptation.

Why the noCB loss is lower on T500 than on T200. Removing the contextual bandit component causes a 24.62% loss on T200 but only a 6.68% loss on T500. A plausible explanation is that localized bandits require a relatively stable reward signal to learn useful operator preferences. On T500, each island contains only 5 individuals, and the decision dimension is much higher. These two factors increase the variance of the reward obtained after each update. As a result, operator preferences become less stable, so removing the contextual bandit component causes a smaller loss. On T200, the reward signal is more stable, allowing the contextual bandit to produce a larger gain.

Why the gate benefit is not monotonically increasing. Removing the stability gate causes a 12.23% loss on T200 but only a 4.21% loss on

T500. A plausible explanation is that the dominant difficulty on T500 is structural. In this case, cooperative coevolution and migration account for most of the performance gain, and the search may spend less time in the stage where premature resampling is most harmful. On T200, useful partial structures are more likely to form before the search stagnates. The stability gate is therefore more beneficial on this instance.

Localized vs. shared bandit. Replacing the three localized bandits with a single shared bandit, while keeping the operator pool, island model, and evaluation budget unchanged, yields similar results on T100 but consistently worse results on T200 and T500 (Fig. 16). This result is consistent with the analysis in Section 5.3. The gain does not come from adaptation alone; it comes from matching localized adaptation to the decision structure.

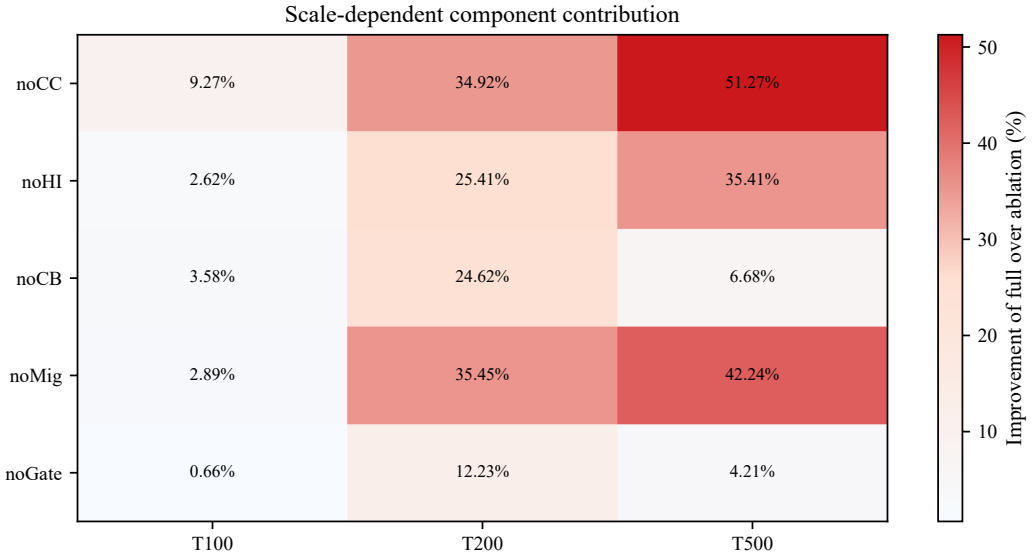

Figure 12: Scale-dependent contribution of each component.

Convergence behavior of ablation variants. Fig. 13 further shows the convergence curves of CCHIHH and selected ablation variants on T100, T200, and T500. The structural removals, especially noCC and noMig, lead to increasingly severe degradation as the instance scale grows. This trend is consistent with the loss values in Table 4. The noCB variant is reported numerically in Table 4; it is omitted from Fig. 13 for visual clarity.

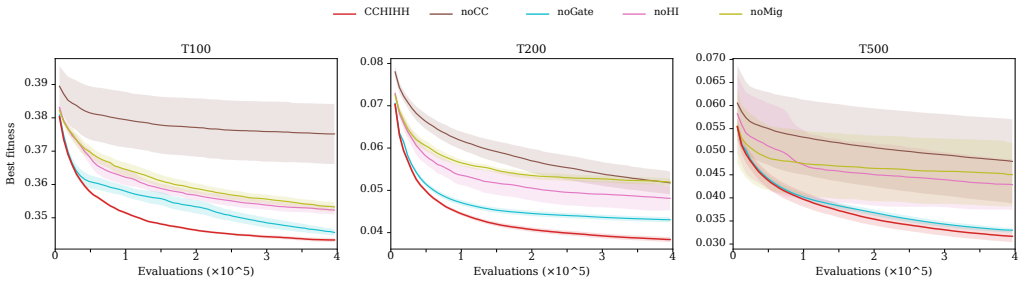

Figure 13: Convergence of CCHIHH-full and selected ablation variants on T100, T200, and T500. Structural removals such as noCC, noHI, and noMig cause increasingly severe degradation at larger scales. Shaded regions: 95% CI over 10 runs.

Operator selection dynamics. Table 5 and Figs. 14–15 report the operator-selection statistics on T500. Here, "early" and "late" refer to the first 20% and the last 20% of the total evaluation budget $E_{\max}$, respectively. The probabilities are computed as normalized selection counts aggregated over islands and random seeds. In the early stage, more exploitative operators are selected more often in the offload and device blocks, namely DE in the offload block (52.9%) and GDE in the device block (47.7%). In contrast, the sequence block relies more on VNS (43.0%) to explore neighborhoods of operation priorities. In the later stage, the operator preferences shift, and the frequency of OBL Resample increases moderately in all three blocks. This trend is consistent with the intended behavior of the stability gate: strong perturbations are suppressed during active improvement and become more likely only after stagnation accumulates.

Table 5: Operator selection probabilities on T500: early (first 20% of $E_{\max}$) vs. late (last 20%). The largest probability within each block and stage is shown in bold.

Block	Operator	Early	Late	Change
Offload	GA (SBX)	0.3680	**0.5770**	0.2090
Offload	DE/$p$best	**0.5290**	0.3570	−0.1720
Offload	BitFlip+Cauchy	0.0970	0.0400	−0.0570
Offload	OBL Resample	0.0060	0.0250	0.0190
Sequence	GA (SBX)	0.3690	**0.5410**	0.1720
Sequence	Swap+Segment	0.1970	0.2310	0.0340
Sequence	Hetero. VNS	**0.4300**	0.1980	−0.2320
Sequence	OBL Resample	0.0040	0.0290	0.0250
Device	DE/$p$best	0.3590	**0.6790**	0.3200
Device	Gbest-DE	**0.4770**	0.2820	−0.1950
Device	Lévy Flight	0.1610	0.0210	−0.1400
Device	OBL Resample	0.0030	0.0180	0.0150

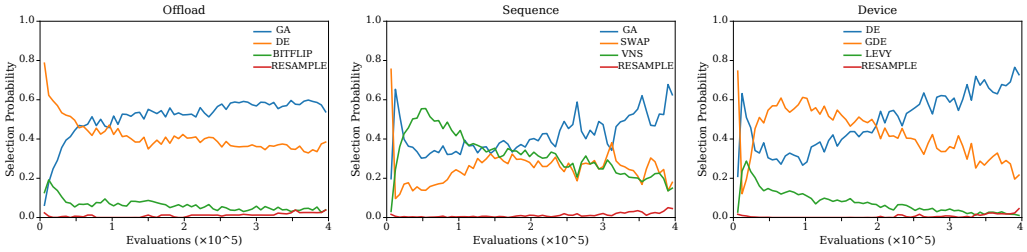

Figure 14: Operator selection probability over the evaluation budget on T500.

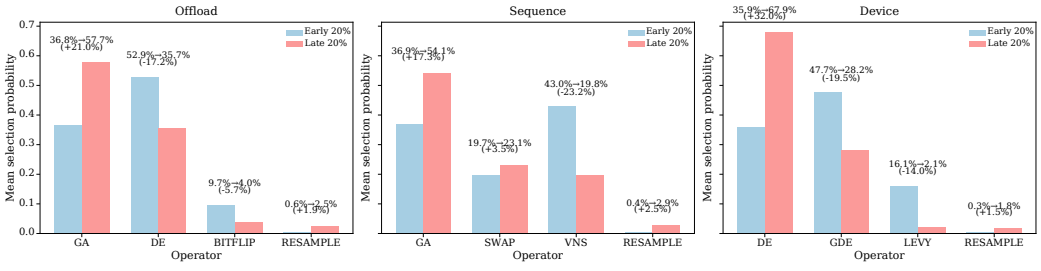

Figure 15: Early vs. late operator selection on T500. The early and late stages correspond to the first and last 20% of the evaluation budget $E_{\max}$, respectively.

5.3. Effect of Problem Scale on Structure-Aware Adaptation

To test whether the benefit of localized adaptation grows with problem scale, CCHIHH-full is compared with the shared-bandit variant over instance sizes from T50 to T500, using the same operator pool, island structure, and evaluation budget.

Fig. 16 shows the convergence behavior on T100, T200, and T500. On T100, the two methods reach similar objective values. On T200 and T500, CCHIHH converges faster and reaches better final solutions, whereas the shared-bandit variant exhibits increasingly clear plateaus.

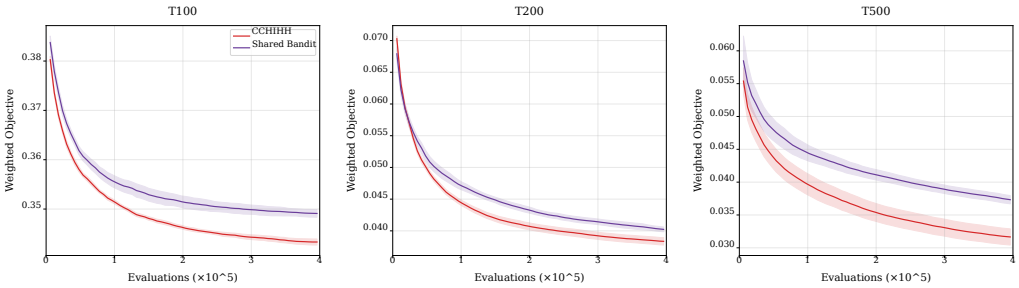

Figure 16: Convergence: CCHIHH (localized bandit) vs. shared bandit on T100, T200, and T500. Shaded regions: 95% CI over 10 runs.

Fig. 17 reports the relative improvement of CCHIHH over the shared-bandit variant as a function of problem size. The relative improvement is defined as

$$\mathrm{RI} = \frac{f_{\mathrm{shared}} - f_{\mathrm{CCHIHH}}}{f_{\mathrm{shared}}} \times 100\%, \tag{33}$$

where f_{shared} and f_{CCHIHH} denote the final best fitness values of the shared-bandit variant and CCHIHH, respectively. The advantage increases from -0.7% at T50 to 18% at T500. On very small instances, the benefit of cleaner block-wise credit assignment is limited, while maintaining separate learners introduces additional exploration overhead. As the problem scale increases, reward dilution under the shared bandit becomes more severe, and the advantage of localized adaptation becomes more apparent.

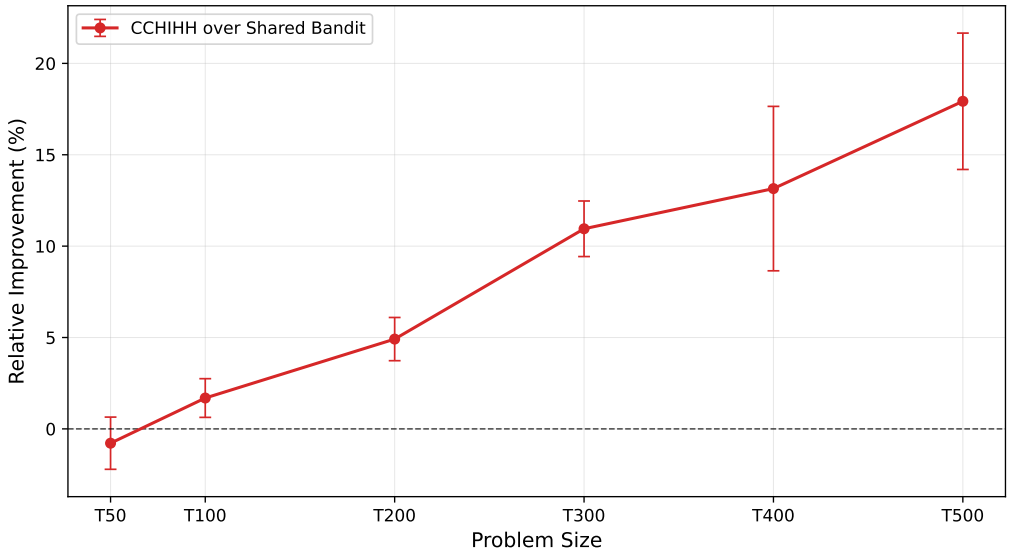

Figure 17: Relative improvement of CCHIHH over the shared-bandit variant across problem sizes, computed by Eq. (33). Error bars: 95% CI over 10 runs.

5.4. Sensitivity Analysis of α

Fig. 18 compares CCHIHH with the baselines under $\alpha \in \{0.2, 0.5, 0.8\}$, corresponding to energy-oriented, balanced, and makespan-oriented optimization. On T200 and T500, CCHIHH consistently outperforms the baselines across all three settings. On T100, it remains competitive but does not always achieve the top rank. This result is consistent with the scale-dependent pattern observed in Sections 5.2–5.3.

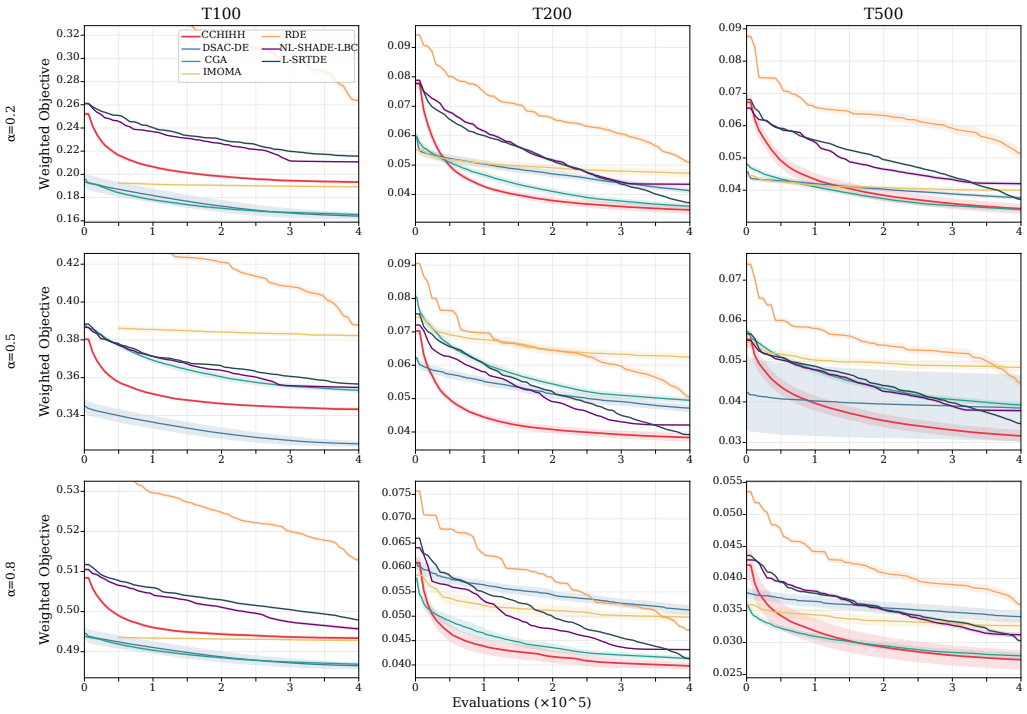

Figure 18: Sensitivity of α on T100/T200/T500. Mean with 95% CI over 10 runs.

Scalarization-induced non-dominated set. To further examine the makespan–energy trade-off beyond a single scalar fitness value, all seven evolutionary algorithms are evaluated on T200 under $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$, with 5 seeds for each setting. This analysis should be interpreted as a post-processing analysis of weighted-sum runs, rather than as a dedicated multi-objective optimization experiment. For each algorithm, the non-dominated subset across all (α, seed) pairs is extracted and plotted in the makespan–energy plane (Fig. 19).

Three observations can be made. First, on T200, the scalarization-induced non-dominated set of CCHIHH lies closer to the lower-left region of the plot than those of the compared baselines, indicating a more favorable trade-off between makespan and energy consumption on the tested setting. Second, the plotted non-dominated CCHIHH points are relatively flat: energy varies by less than 8 units (approximately 58–66) across the displayed points. By contrast, several baselines show steeper trade-off curves, indicating that improving one objective tends to cause a larger deterioration in the other objective. Third, within the sampled α values, no strong non-convex

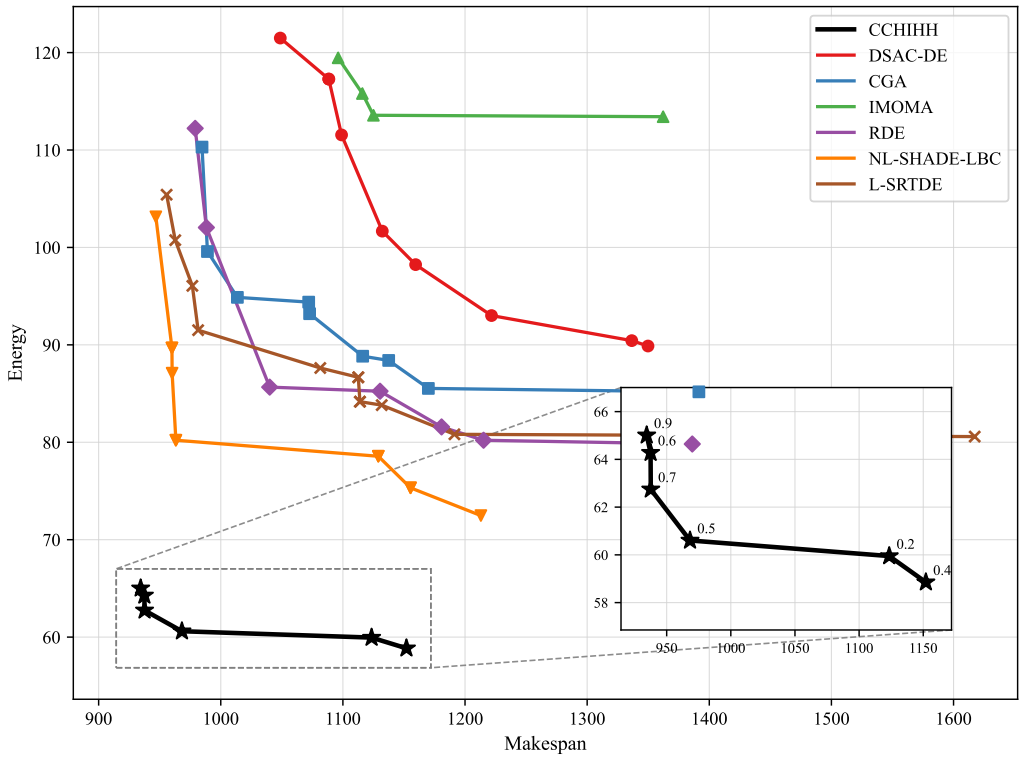

Figure 19: Scalarization-induced non-dominated sets on T200 ($\alpha \in \{0.1, 0.2, \ldots, 0.9\}$, 5 seeds each). The inset zooms into the CCHIHH region with α labels. On the tested T200 setting, the CCHIHH set lies closer to the lower-left region than those of the compared baselines.

gap is observed around the CCHIHH points in the inset of Fig. 19. This suggests that the weighted-sum scalarization provides a reasonable trade-off sample for this tested setting, although it does not replace a dedicated multi-objective search.

To provide a quantitative summary of these scalarization-induced non-dominated sets, Table 6 reports per-seed Hypervolume (HV) on T200. HV is computed in the normalized objective space $(f_1/f_1^{(\mathrm{ref})}, f_2/f_2^{(\mathrm{ref})})$ using the same reference point for all algorithms, chosen to upper-bound all observed normalized objectives across algorithms and seeds. CCHIHH achieves the highest mean HV (1.1796 ± 0.0143). This result is consistent with the visualization in Fig. 19 and suggests that the trade-off advantage of CCHIHH is stable across repeated runs.

Table 6: Per-seed Hypervolume of scalarization-induced non-dominated sets on T200 under $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$ (mean $\pm$ std over 5 seeds). Best in **bold**.

Algorithm	**HV (mean $\pm$ std)**
CCHIHH	**1.1796 $\pm$ 0.0143**
DSAC-DE	0.7207 $\pm$ 0.0352
CGA	0.8820 $\pm$ 0.0172
IMOMA	0.5836 $\pm$ 0.0221
RDE	0.8909 $\pm$ 0.0486
NL-SHADE-LBC	1.0326 $\pm$ 0.0166
L-SRTDE	0.9481 $\pm$ 0.0152

5.5. Robustness under Resource and Communication Degradation

CCHIHH and the evolutionary baselines are evaluated under four degraded conditions: cloud-capacity reduction, edge-capacity reduction, device-capacity reduction, and communication-time inflation, each at three stress levels. Two indicators are reported: absolute best fitness and a normalized retention metric,

$$\mathrm{PRR} = \frac{f_{\mathrm{deg}}}{f_{\mathrm{nom}}}, \tag{34}$$

where f_{deg} and f_{nom} are the best fitness under degraded and nominal settings, respectively. Because degradation is expected to make the scheduling problem more difficult, PRR is generally no smaller than 1 in the tested settings. A value closer to 1 indicates that the algorithm retains more of its nominal performance under degradation, whereas a larger value indicates stronger performance deterioration.

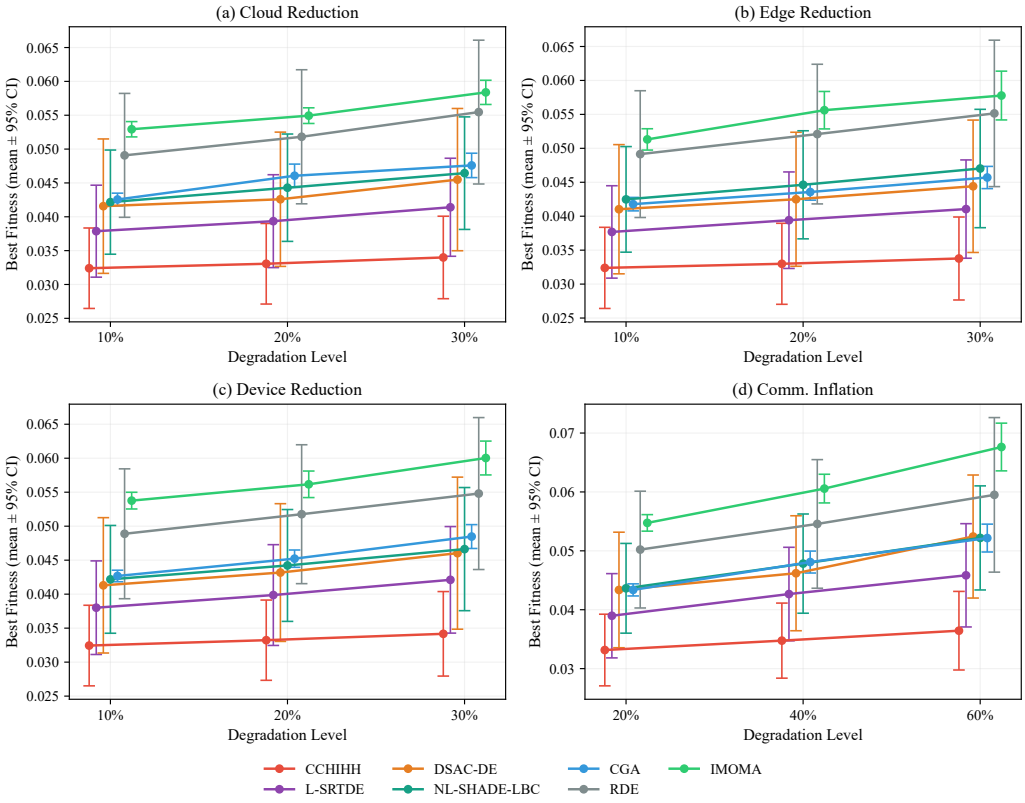

Figure 20: Absolute best fitness under four degradation families. Lower is better.

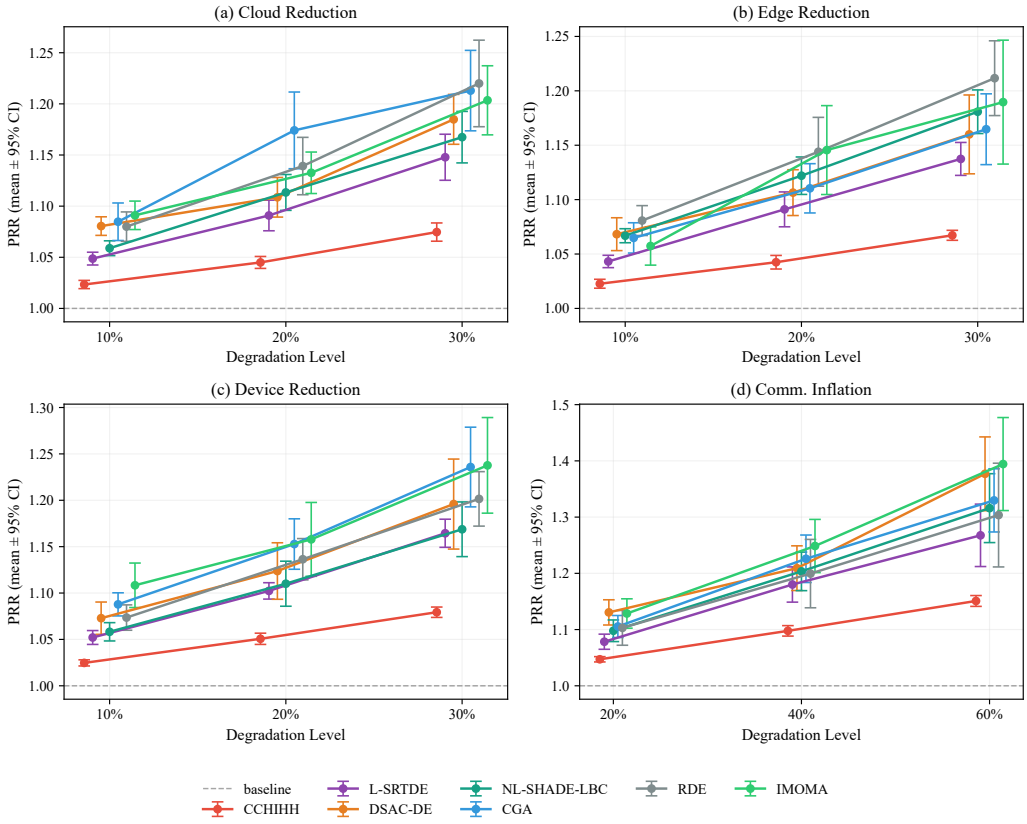

Figure 21: Performance retention ratio (PRR) under system degradation. Values closer to 1 indicate slower deterioration.

Fig. 20 shows that CCHIHH achieves the best absolute fitness across all tested degradation families and stress levels. Fig. 21 shows that its PRR values are also closest to 1 in most cases, which means that its performance degrades less under the same stress. This pattern is especially clear under device-side reduction and communication-time inflation, where the coupling becomes stronger. These results suggest that localized adaptation helps maintain performance under degradation.

5.6. Comparison with Baselines

Solution quality. Table 7 reports the relative improvements over six evolutionary baselines. The improvement over a baseline is computed as

$$\text{Impr.} = \frac{f_{\text{baseline}} - f_{\text{CCHIHH}}}{f_{\text{baseline}}} \times 100\%, \tag{35}$$

where a positive value means that CCHIHH obtains a lower final fitness. On T200 and T500, CCHIHH obtains lower mean fitness than all six evolutionary baselines. The pairwise Wilcoxon test is significant at the 0.05 level for all pairs except the comparison with L-SRTDE on T200. On T500, the gains reach 34.75% over IMOMA, 29.30% over RDE, 19.31% over CGA, 17.75% over DSAC-DE, 16.30% over NL-SHADE-LBC, and 8.70% over L-SRTDE. On T100, DSAC-DE obtains a lower mean fitness than CCHIHH by 5.63%, and on T50 the shared-bandit variant is marginally better (−0.7%, Fig. 17). These results indicate that the proposed framework is less advantageous on small instances, where the additional exploration overhead of localized learners is not yet offset by cleaner block-wise credit assignment.

Applicability boundary. Localized adaptation introduces two forms of overhead. First, maintaining $3 \times M = 24$ localized bandit learners (3 blocks $\times$ 8 islands) requires additional exploration before each learner converges to useful operator preferences. Second, with $N = 40$ and $M = 8$, each island contains only 5 individuals, which limits the statistical quality of the reward signal. On compact instances, where the search space is still manageable by a single shared bandit, these costs can exceed the benefit of cleaner credit assignment. In our experiments, the crossover point lies between T50 and T100 (Fig. 17). Below this threshold, lighter methods such as success-history-based DE or RL-configured DE may be preferable. Above it, reward dilution under the shared bandit becomes dominant, and localized adaptation yields progressively larger gains.

This boundary depends on problem-specific factors, including coupling density, block-dimension ratios, and the number of islands. A systematic characterization of this threshold is left for future work.

Table 7: CCHIHH vs. evolutionary baselines: relative improvement (%) computed by Eq. (35). Impr. > 0 means that CCHIHH obtains a lower final fitness. The largest improvement in each scale is shown in bold.

Baseline	**T100**	**T200**	**T500**
DSAC-DE	−5.63	18.66	17.75
CGA	2.82	22.54	19.31
IMOMA	10.19	**38.61**	**34.75**
RDE	**11.50**	24.20	29.30
NL-SHADE-LBC	3.20	8.90	16.30
L-SRTDE	3.70	2.20	8.70

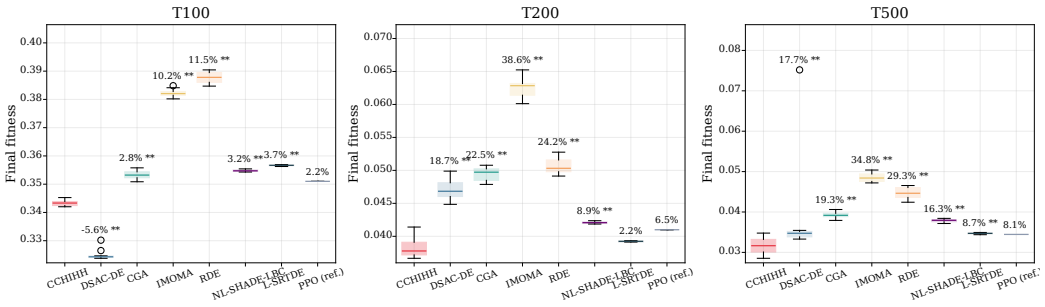

Figure 22: Baseline comparison overview. ** denotes $p < 0.05$. PPO is descriptive only.

Cross-paradigm reference. PPO [34] is included only as a descriptive cross-paradigm reference because its offline training cost is not reflected in the shared evaluation budget $E_{\max}$. Under the evaluation-only comparison, CCHIHH obtains better mean fitness than PPO on all three scales, with relative margins of 2.2%, 6.5%, and 8.1% on T100, T200, and T500, respectively. However, none of these differences reaches statistical significance at the 0.05 level. These numbers should therefore be interpreted only as contextual evidence rather than as a formal claim that CCHIHH dominates PPO. A fair wall-clock comparison would require amortizing the training cost over the number of instances solved by a single trained policy, which is outside the scope of this study. Therefore, PPO is reported separately in Fig. 22 and is not included in the main statistical ranking.

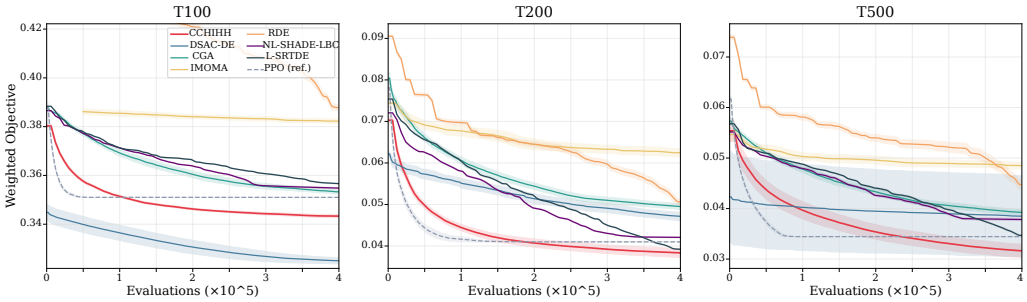

Figure 23: Convergence curves: CCHIHH vs. all baselines across three scales. Shaded regions: 95% CI over 10 runs.

Stability. Table 8 reports the coefficient of variation (CV) over 10 runs. L-SRTDE and NL-SHADE-LBC achieve the lowest CV values (e.g., 0.0047 and 0.0096 on T500), but they converge to substantially worse objective

Table 8: Run-to-run dispersion (CV = Std/Mean) over 10 runs. The lowest CV in each scale is shown in bold.

Scale	CCHIHH	DSAC-DE	CGA	IMOMA	RDE	NL-SHADE-LBC	L-SRTDE
T100	0.0029	0.0058	0.0040	0.0035	0.0053	0.0011	**0.0006**
T200	0.0246	0.0324	0.0210	0.0230	0.0235	0.0040	**0.0021**
T500	0.0629	0.3184	0.0211	0.0216	0.0323	0.0096	**0.0047**

values. Among the methods with competitive solution quality, CCHIHH maintains a CV of 0.0629 on T500, which is much lower than that of DSAC-DE (0.3184).

Efficiency. Under $E_{\max} = 4 \times 10^5$, the lightweight DE variants finish in 14–119 s, whereas CCHIHH, CGA, IMOMA, and PPO require 30–300 s. Under the same full-solution evaluation budget, CCHIHH achieves the best solution quality on the medium and large instances. Runtime details are shown in Fig. 24.

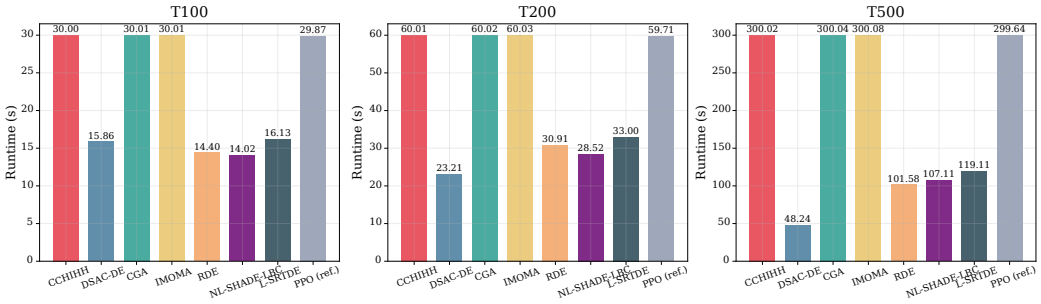

Figure 24: Wall-clock time comparison (mean over 10 runs).

Statistical ranking. Fig. 25 reports the Friedman–Nemenyi ranking over 9 configurations (3 scales × 3 α values). CCHIHH attains the best average rank. The Friedman test indicates a significant overall difference among the algorithms ($p = 7.64 \times 10^{-5}$), and the corresponding Nemenyi critical difference at $\alpha = 0.05$ is $CD = 3.003$. Under this criterion, CCHIHH is significantly better than the algorithms whose average-rank differences from CCHIHH exceed the critical difference in Fig. 25.

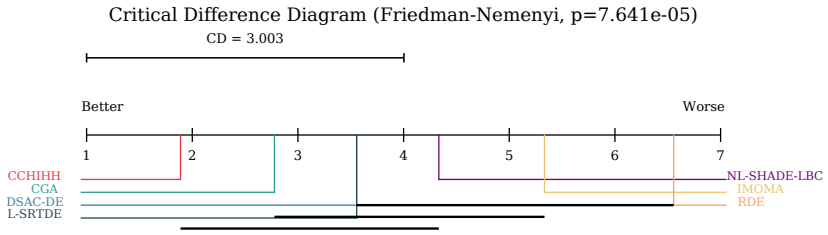

Figure 25: Critical difference diagram (Friedman–Nemenyi test, significance level 0.05, 9 configurations). Bold bars connect algorithms that are not significantly different.

Overall trade-off among evaluation indicators. Fig. 26 summarizes the relative behavior of the compared methods from four perspectives: solution quality, stability, efficiency, and convergence speed. The radar plot is intended as a compact visual summary; the exact numerical evidence for these dimensions is provided in Tables 7–8 and Figs. 23–24. CCHIHH is strongest in solution quality and convergence behavior on medium and large instances, while lightweight DE variants have advantages in runtime and run-to-run stability.

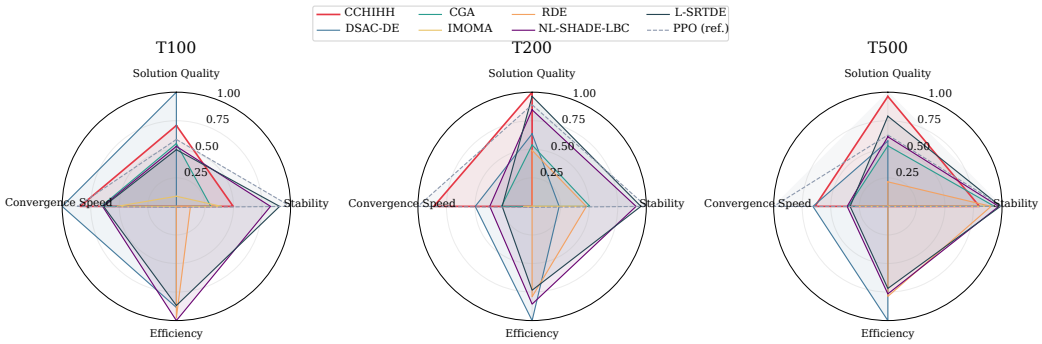

Figure 26: Multi-dimensional comparison: solution quality, stability, efficiency, and convergence speed.

Schedule visualization. To illustrate the decoded schedule produced by CCHIHH, Fig. 27 presents the best cloud–edge allocation on T200. The figure shows that computational tasks are distributed across both cloud and edge resources, with single-use servers merged for readability. This visualization is not used as an additional quantitative metric; it is provided to demonstrate that the final solution can be interpreted at the schedule level.

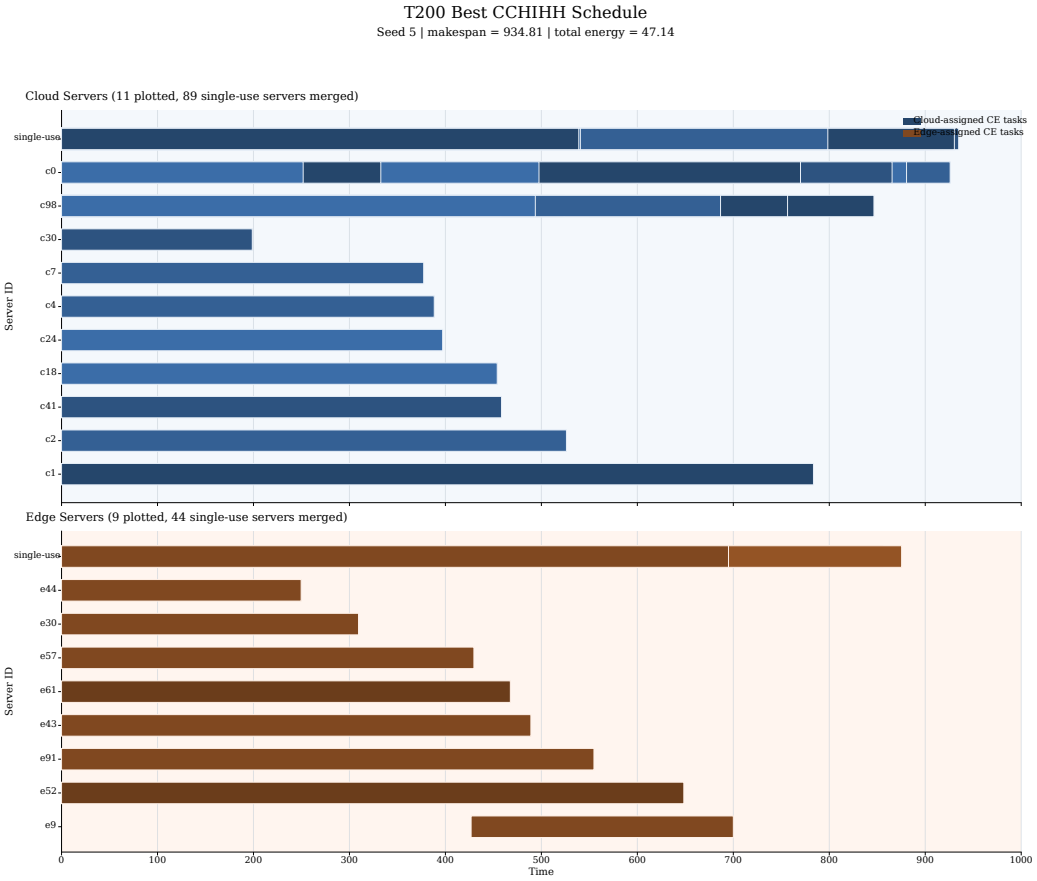

Figure 27: Gantt chart of the best cloud–edge schedule produced by CCHIHH on T200 (Seed 5, makespan = 934.81, total energy = 47.14). Blue: cloud-assigned; brown: edge-assigned. Device-layer operations are omitted for clarity.

6. Conclusion

This paper studied cloud–edge–device collaborative scheduling under a coupled three-block decision structure consisting of offloading, sequencing, and device assignment. In large-scale instances, these three blocks differ in search behavior, which makes a single shared adaptive mechanism less effective.

To address this issue, we proposed CCHIHH, which combines cooperative coevolution, localized contextual bandits, heterogeneous islands, and a stability gate. Experiments on instances with up to 500 computational tasks and 800 devices show that the benefit of localized adaptation grows

with problem scale. The proposed method outperforms representative baselines on medium and large instances and remains stable under resource and communication degradation.

These results suggest that adaptive operator selection is more effective when aligned with the natural decision structure of the problem. Future work will extend the framework to dynamic rescheduling settings and further examine when the overhead of localized adaptation is justified.

References

[1] Y. Laili, X. Wang, L. Zhang, L. Ren, Dsac-configured differential evolution for cloud–edge–device collaborative task scheduling, IEEE Transactions on Industrial Informatics 20 (2) (2023) 1753–1763.

[2] S. Wang, Y. Li, S. Pang, Q. Lu, S. Wang, J. Zhao, A task scheduling strategy in edge-cloud collaborative scenario based on deadline, Scientific Programming 2020 (1) (2020) 3967847.

[3] G. Cui, W. Zhang, W. Xu, H. Bao, Efficient workflow scheduling using an improved multi-objective memetic algorithm in cloud-edge-end collaborative framework, Scientific Reports 15 (1) (2025) 29754.

[4] V. Stanovov, S. Akhmedova, E. Semenkin, Nl-shade-lbc algorithm with linear parameter adaptation bias change for cec 2022 numerical optimization, in: 2022 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2022, pp. 01–08.

[5] V. Stanovov, E. Semenkin, Success rate-based adaptive differential evolution l-srtde for cec 2024 competition, in: 2024 IEEE Congress on evolutionary computation (CEC), IEEE, 2024, pp. 1–8.

[6] T. Zheng, J. Wan, J. Zhang, C. Jiang, Deep reinforcement learning-based workload scheduling for edge computing, Journal of Cloud Computing 11 (1) (2022) 3.

[7] Y. Gu, Z. Liu, S. Dai, C. Liu, Y. Wang, S. Wang, G. Theodoropoulos, L. Cheng, Deep reinforcement learning for job scheduling and resource management in cloud computing: An algorithm-level review, arXiv preprint arXiv:2501.01007 (2025).

[8] A. Avan, A. Azim, Q. H. Mahmoud, A state-of-the-art review of task scheduling for edge computing: A delay-sensitive application perspective, Electronics 12 (12) (2023) 2599.

[9] R. Latip, J. Aminu, Z. M. Hanafi, S. Kamarudin, D. Gabi, Metaheuristic task offloading approaches for minimization of energy consumption on edge computing: a systematic review, Discover Internet of Things 4 (1) (2024) 35.

[10] B. Wang, J. Cheng, J. Cao, C. Wang, W. Huang, Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve sla satisfaction, PeerJ Computer Science 8 (2022) e893.

[11] K. Li, Energy-constrained dag scheduling on edge and cloud servers with overlapped communication and computation, Journal of Grid Computing 22 (3) (2024) 60.

[12] W. Zhang, H. Ou, Reinforcement learning based multi objective task scheduling for energy efficient and cost effective cloud edge computing, Scientific Reports 15 (1) (2025) 41716.

[13] H. Yin, X. Huang, E. Cao, A cloud-edge-based multi-objective task scheduling approach for smart manufacturing lines, Journal of Grid Computing 22 (1) (2024) 9.

[14] B. B. Naik, B. Priyanka, M. S. A. Ansari, Energy-efficient task offloading and efficient resource allocation for edge computing: a quantum inspired particle swarm optimization approach, Cluster Computing 28 (3) (2025) 155.

[15] L. Sun, K. Li, Adaptive operator selection based on dynamic thompson sampling for moea/d, in: International conference on parallel problem solving from nature, Springer, 2020, pp. 271–284.

[16] V. R. de Carvalho, E. Özcan, J. S. Sichman, Comparative analysis of selection hyper-heuristics for real-world multi-objective optimization problems, Applied Sciences 11 (19) (2021) 9153.

[17] L. DaCosta, A. Fialho, M. Schoenauer, M. Sebag, Adaptive operator selection with dynamic multi-armed bandits, in: Proceedings of the 10th

annual conference on Genetic and evolutionary computation, 2008, pp. 913–920.

[18] G. Karafotias, M. Hoogendoorn, Á. E. Eiben, Parameter control in evolutionary algorithms: Trends and challenges, IEEE Transactions on Evolutionary Computation 19 (2) (2014) 167–187.

[19] Y. Gong, A. Fukunaga, Distributed island-model genetic algorithms using heterogeneous parameter settings, in: 2011 IEEE congress of evolutionary computation (CEC), IEEE, 2011, pp. 820–827.

[20] L. A. da Silveira, T. A. de Lima, J. B. de Barros, J. L. Soncco-Álvarez, C. H. Llanos, M. Ayala-Rincón, On the behavior of parallel island models, Applied Soft Computing 148 (2023) 110880.

[21] D. Whitley, S. Rana, R. B. Heckendorn, The island model genetic algorithm: On separability, population size and convergence, Journal of computing and information technology 7 (1) (1999) 33–47.

[22] M. N. Omidvar, X. Li, Y. Mei, X. Yao, Cooperative co-evolution with differential grouping for large scale optimization, IEEE Transactions on evolutionary computation 18 (3) (2013) 378–393.

[23] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, Z. Zhu, A survey on cooperative co-evolutionary algorithms, IEEE Transactions on Evolutionary Computation 23 (3) (2018) 421–441.

[24] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, Q. Pan, A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems, IEEE/CAA journal of automatica sinica 6 (4) (2019) 904–916.

[25] L. Sun, L. Lin, M. Gen, H. Li, A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling, IEEE Transactions on Fuzzy Systems 27 (5) (2019) 1008–1022.

[26] S. Mangalampalli, G. R. Karri, M. Ratnamani, S. N. Mohanty, B. A. Jabr, Y. A. Ali, S. Ali, B. S. Abdullaeva, Efficient deep reinforcement learning based task scheduler in multi cloud environment, Scientific Reports 14 (1) (2024) 21850.

[27] H. R. Tizhoosh, Opposition-based learning: a new scheme for machine intelligence, in: International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06), Vol. 1, IEEE, 2005, pp. 695–701.

[28] K. Deb, R. B. Agrawal, et al., Simulated binary crossover for continuous search space, Complex systems 9 (2) (1995) 115–148.

[29] J. Zhang, A. C. Sanderson, Jade: adaptive differential evolution with optional external archive, IEEE Transactions on evolutionary computation 13 (5) (2009) 945–958.

[30] T. J. Choi, C. W. Ahn, An improved lshade-rsp algorithm with the cauchy perturbation: ilshade-rsp, Knowledge-Based Systems 215 (2021) 106628.

[31] I. Or, TRAVELING SALESMAN TYPE COMBINATORIAL PROBLEMS AND THEIR RELATION TO THE LOGISTICS OF REGIONAL BLOOD BANKING., Northwestern University, 1976.

[32] P. Hansen, N. Mladenović, Variable neighborhood search: Principles and applications, European journal of operational research 130 (3) (2001) 449–467.

[33] S. Tao, R. Zhao, K. Wang, S. Gao, An efficient reconstructed differential evolution variant by some of the current state-of-the-art strategies for solving single objective bound constrained problems, arXiv preprint arXiv:2404.16280 (2024).

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).