



ARTICLE

A Discrete Multi-Objective Squirrel Search Algorithm for Energy-Efficient Distributed Heterogeneous Permutation Flowshop with Variable Processing Speed

Liang Zeng^{1,2,3}, Ziyang Ding¹, Junyang Shi¹ and Shanshan Wang^{1,2,3,*}

¹School of Electrical and Electronic Engineering, Hubei University of Technology, Wuhan, 430068, China

²Hubei Key Laboratory for High-Efficiency Utilization of Solar Energy and Operation Control of Energy Storage System, Hubei University of Technology, Wuhan, 430068, China

³Xiangyang Industrial Institute of Hubei University of Technology, Xiangyang, 441100, China

*Corresponding Author: Shanshan Wang. Email: wangshanshan@hbut.edu.cn

Received: 01 July 2024 Accepted: 12 September 2024 Published: 15 October 2024

ABSTRACT

In the manufacturing industry, reasonable scheduling can greatly improve production efficiency, while excessive resource consumption highlights the growing significance of energy conservation in production. This paper studies the problem of energy-efficient distributed heterogeneous permutation flowshop problem with variable processing speed (DHPFSP-VPS), considering both the minimum makespan and total energy consumption (TEC) as objectives. A discrete multi-objective squirrel search algorithm (DMSSA) is proposed to solve the DHPFSP-VPS. DMSSA makes four improvements based on the squirrel search algorithm. Firstly, in terms of the population initialization strategy, four hybrid initialization methods targeting different objectives are proposed to enhance the quality of initial solutions. Secondly, enhancements are made to the population hierarchy system and position updating methods of the squirrel search algorithm, making it more suitable for discrete scheduling problems. Additionally, regarding the search strategy, six local searches are designed based on problem characteristics to enhance search capability. Moreover, a dynamic predator strategy based on Q-learning is devised to effectively balance DMSSA's capability for global exploration and local exploitation. Finally, two speed control energy-efficient strategies are designed to reduce TEC. Extensive comparative experiments are conducted in this paper to validate the effectiveness of the proposed strategies. The results of comparing DMSSA with other algorithms demonstrate its superior performance and its potential for efficient solving of the DHPFSP-VPS problem.

KEYWORDS

Distributed heterogeneous permutation flowshop problem; squirrel search algorithm; multi-objective optimization; energy-efficient; variable processing speed

1 Introduction

The permutation flow-shop scheduling problem (PFSP) is a typical scheduling problem with a wide range of applications, commonly used in large-scale production. In 1954, Johnson first proposed a PFSP scheduling problem [1]. The core idea of PFSP is to divide the production process into multiple



stages and allocate each stage to specifically designed production lines, aiming to optimize production efficiency. The work tasks for each production line are specially designed and optimized. Upon completing a stage, a production line transfers materials or products to the next line and proceeds with the subsequent task. Therefore, PFSP can significantly improve production efficiency. Nonetheless, the problem has been proven to be NP-hard when the number of machines exceeds three [2]. Therefore, designing effective algorithms for PFSP holds important academic value and practical significance [3].

On the one hand, the quantity of orders for enterprises has significantly expanded due to the advancement of industrial technology and worldwide trade. Increasingly, managers are recognizing the challenges faced by traditional single factories in meeting market and customer demands within a dynamic production environment [4]. Distributed scheduling, enabling collaborative production among multiple factories, has garnered significant attention for its high production efficiency and optimal resource utilization [5]. Naderi et al. [6] first proposed the distributed permutation flow-shop scheduling problem (DPFSP) in 2010. DPFSP is an extension of PFSP that involves job assignment to factories and the determination of job processing sequences within each factory, rendering it more intricate than PFSP. Gao et al. [7] improved GA and designed effective crossover and mutation operators for DPFSP. Gao et al. [8] also improved TS and proposed a method to generate neighborhood by exchanging working subsequences, which improved the performance of the algorithm for DPFSP. In 2016, Rifai et al. [9] proposed multi-objective DPFSP, taking makespan, total cost and average delay as optimization objectives, and designed an adaptive neighborhood search algorithm to solve this problem. Lin et al. [10] proposed Iterated Cocktail Greedy (ICG) algorithm for DPFSP without waiting, which includes destruction mechanism and two self-coordination mechanisms. Shao et al. [11] proposed a distributed estimation algorithm based on Pareto frontiers and created three probability models to solve sequence-related DPFSP. Lu et al. [12] proposed a multi-objective cooperative optimization algorithm based on Pareto Front, which achieved good convergence in solving DPFSP with buffers. Zhao et al. [13] improved the water wave algorithm and designed four local search methods, exhibiting exceptional performance in optimizing blocked DPFSP scenarios. However, a comprehensive review of prior research reveals that the majority of studies have concentrated on homogeneous factories, characterized by identical machines and processing times. However, in actual production, each factory may have different machine types and processing times, which are called heterogeneous factories. When studying the distributed heterogeneous permutation flowshop problem (DHPFSP), it is essential to consider not only job allocation across multiple factories but also the disparities in machines and processing times, thereby augmenting the problem's complexity. Additionally, the inclusion of machine and processing time heterogeneity brings the production process closer to real-world requirements. Therefore, numerous scholars have designed corresponding algorithms for DHPFSP. Ruiz et al. [14] presented an iterative greedy algorithm with enhanced initialization, showcasing its effectiveness in optimizing the makespan in DHPFSP. Bargaoui et al. [15] devised a highly efficient heuristic initialization method utilizing chemical reaction optimization (CRO) and incorporating greedy and crossover strategies, exhibiting superior performance on the Taillard benchmarks. Li et al. [16] introduced an improved artificial bee colony algorithm (IABC) for the makespan of DHPFSP, which utilized a workload-balanced initialization method and local search. The effectiveness of IABC was tested on famous benchmarks. Lin et al. [17] improved the backtracking search algorithm and designed effective encoding-decoding schemes and heuristic rules, the results show the strong superiority of the algorithm. These methods mentioned above for DHPFSP have shown excellent performance, but they only optimize the objective of makespan.

On the other hand, as the global economy develops and resource consumption exceeds sustainable levels, the issue of global warming is escalating in severity. Environmental organizations

and government institutions have begun to advocate energy-efficient and consumption reduction in various manufacturing sectors. Improving environmental sustainability has emerged as a pressing concern [18]. Statistical data reveals that the manufacturing industry comprises roughly 33% of global energy consumption and contributes approximately 38% to carbon emissions [19]. In light of this, the manufacturing industry has started to attach great importance to reducing energy consumption. Hence, when devising production plans, integrating conventional economic indicators like energy efficiency and completion time is of paramount importance, alongside the development of effective energy-efficient measures and technologies. Researchers have undertaken studies on energy-efficient scheduling problems in DHPFSP. In 2017, Wang et al. [20] considered energy consumption for the first time in DPFSP, and proposed Energy-efficient Distributed Permutation Flow Shop Problem, EDPFSP. Afterward, Wang et al. [21] suggested a cooperative memetic algorithm that designed knowledge-driven operators based on the properties of non-dominated solutions, which effectively optimized completion time and TEC in DHPFSP. Li et al. [22] improved the NSGA-II algorithm through the introduction of a novel population generation method and the design of operators targeting completion time and energy consumption in DHPFSP. Zhao et al. [23] designed a self-learning Jaya algorithm and proposed an evaluation criterion that combines TEC and makespan. The higher performance of the suggested method was confirmed by experimental data. Fathollahi-Fard et al. [24] established a new multi-objective heuristic algorithm based on the Social Engineering Optimizer and built a mixed-integer linear model. Experimental outcomes verified the efficacy of the suggested algorithm in reducing energy consumption and model efficiency.

However, a comprehensive analysis of previous research reveals that numerous scholars in the field of DHPFSP have neglected the influence of variable processing speeds on scheduling, or assuming constant speeds. That is, in terms of research objects, there are relatively few researches on distributed workshops with energy-saving performance, and the vast majority of researches still aim at minimizing the makespan, which cannot meet the current diverse scheduling requirements. The Distributed Heterogeneous Permutation Flowshop with variable processing speed (DHPFS-VPS) allows products to be processed on multiple production lines and dynamically scheduled based on real-time conditions to improve production efficiency and reduce costs. Some relevant studies have shown that using variable-speed DPFSP algorithms can effectively reduce waiting and queueing times in the manufacturing process [25]. Considering the need for more efficient workshop scheduling solutions in modern industrial production, DHPFS-VPS holds substantial research value, prompting a growing number of scholars to investigate this problem. Huang et al. [26] proposed a bi-roles co-evolution algorithm (BRCE) for DHPFS-VPS, which designed various local search strategies for energy consumption and machine speeds and proposed an effective energy-efficient strategy. Wang et al. [27] proposed an improved multi-objective whale swarm algorithm (MOWSA) for DHPFS-VPS, which aimed to improve energy efficiency without compromising production efficiency and introduced an updated utilization mechanism for completion time and machine speeds. Generally, research on DHPFS-VPS remains relatively limited, and its exploration is still in the early stages compared to scheduling methods employed for single production lines. Although there are still many challenges in practical applications, such as considering logistics and collaboration between production lines, the concept of DHPFS-VPS has been receiving increasing attention and exploration [28]. Consequently, undertaking further in-depth research in this field holds immense significance and presents promising prospects for practical application.

It can be seen that with the deepening of research and the development of industry, the types of DPFSP have gradually become diversified. At the same time, various swarm intelligence algorithms

have gradually become the mainstream methods for solving various types of DPFSP. Table 1 summarizes the state of the art of DPFSP research in detail.

Table 1: The key details of DPFSP research

Author	Schedule problem	Optimization objective	Core algorithm	Date
Naderi et al. [6]	DPFSP	Makespan	Heuristic algorithm	2010
Gao et al. [7]	DPFSP	Makespan	GA	2011
Gao et al. [8]	DPFSP	Makespan	TS	2013
Rifai et al. [9]	DPFSP	Makespan, total cost	LNS	2016
Lin et al. [10]	DPFSP	Makespan	ICG	2016
Shao et al. [11]	DPFSP	Makespan	EDA	2019
Lu et al. [12]	DPFSP	Makespan, TEC	CO	2022
Zhao et al. [13]	DPFSP	Makespan	WVO	2022
Ruiz et al. [14]	DHPFSP	Makespan	GA	2019
Bargaoui et al. [15]	DHPFSP	Makespan	CRO	2019
Li et al. [16]	DHPFSP	Makespan	ABC	2019
Lin et al. [17]	DHPFSP	Makespan	BSA	2017
Wang et al. [21]	DHPFSP	Makespan, TEC	CMAF	2022
Li et al. [22]	DHPFSP	Makespan, TEC	NSGA-II	2021
Zhao et al. [23]	DHPFSP	Makespan, TEC	SD-Jaya	2021
Fathollahi et al. [24]	DHPFSP	Makespan, TEC	SEO	2021
Huang et al. [26]	DHPFS-VPS	Makespan, TEC, MS	BRCE	2023
Wang et al. [27]	DHPFS-VPS	Makespan, TEC, MS	MOWOA	2020

The Squirrel Search Algorithm (SSA) is a novel type of swarm intelligence optimization algorithm [29]. Since SSA was proposed, it has been applied to many fields in engineering, including single-objective optimization and multi-objective optimization, demonstrating its practicality and competitiveness in solving engineering problems. It is renowned for its simplistic structure and exceptional performance, making it highly esteemed within the academic community [30]. Currently, there have been multiple studies applying SSA to different fields, such as MEMS vector hydrophone arrays [31], classification of X-ray images [32], and multi-objective economic environmental scheduling. Sakthivel et al. [33] utilized SSA to minimize the total fuel cost and emissions of multi-regional generating units to effectively manage power generation and realize multi-regional economic and environmental scheduling. Wang et al. [34] proposed a dynamic multi-objective SSA and successfully applied it to multi-objective FJSP. Jaishankar et al. [35] used a multi-purpose SSA to secure an efficient blockchain network of healthcare data to generate intelligent and secure healthcare systems, ultimately resulting in 26.87% higher throughput, 34.67% higher efficiency, 22.97% lower latency, 37.03% lower compute overhead, and 34.29% higher storage costs. Guha et al. [36] combined quasi-opposition learning and SSA to develop the controller of wind turbine. Ishwarya et al. [37] combined convolutional neural network and SSA to accurately predict human posture. Maden et al. [38] used SSA to optimize the model parameters of photovoltaic power generation and improve the effectiveness of the photovoltaic system. Nevertheless, research on the discretization of SSA remains limited, and its application to solving the DHPFSP has not been explored. Therefore, this study aims to enrich

the application research of SSA and expand the solution methods for DHPFSP-VPS, which holds significant theoretical and practical significance.

The remaining structure of the paper is as follows: In [Section 2](#), the problem is defined, and a multi-objective mixed integer programming model is presented. [Section 3](#) provides an introduction to the standard Squirrel Search Algorithm. In [Section 4](#), the proposed Discrete Multi-objective Squirrel Search Algorithm is described in detail. [Section 5](#) discusses the experimental process and analyzes the results obtained. Finally, [Section 6](#) concludes the paper and suggests potential research extensions for future studies.

2 Problem Description

2.1 Definition of DHPFSP-VPS

DHPFSP-VPS is a highly complex multi-objective combinatorial optimization problem, posing significant challenges to manufacturing production scheduling. The description of DHPFSP-VPS is as follows: There are n ($i = 1, \dots, n$) jobs and F ($f = 1, \dots, F$) factories in the problem. Each factory faces a permutation flowshop scheduling problem and is equipped with m ($j = 1, \dots, m$) machines. Consequently, every job comprises m stages of operations. In the DHPFSP-VPS, the initial step involves assigning each job to a distinct factory. Subsequently, the processing sequence of jobs within each factory needs to be determined. Moreover, the selection of processing speeds for each operation is required. The processing time for each operation of a job is different in different factories, exhibiting heterogeneity. Once a job is assigned to a factory, all its operations are completed within that factory. All jobs start processing on the first machine and finish processing on the last machine. By scheduling the processing sequences within each factory and selecting different processing speeds, distinct makespan, and TEC values can be obtained. Each operation O_{ij} has a standard processing time $t_{i,j}$ on each machine. If the processing speed for O_{ij} is V_v , its actual processing time is $P_{i,j} = t_{i,j}/V_v$, and the energy consumption $W_{i,j}$ is proportional to both $P_{i,j}$ and V_v^2 . Therefore, increasing the processing speed leads to a reduction in processing time at the expense of higher power consumption by the machine.

The assumptions of DHPFSP-VPS are as follows:

- (1) All machines in all factories are available and ready for processing at time zero, and all factories commence processing at time zero.
- (2) Each machine is capable of handling only one operation at a time.
- (3) When a job is assigned to a factory, all its operations are carried out exclusively within that factory.
- (4) Subsequent operations of a job can only begin once their preceding operations have been completed.
- (5) Potential interruptions during job processing and machine failures are not accounted for.

2.2 MILP model for DHPFSP-VPS

To facilitate the description of the model, the notations employed throughout this study are defined as follows:

Indices:

i : set for jobs, $i \in \{1, \dots, n\}$;

j : set of machines, $j \in \{1, \dots, m\}$;

f : set of factories, $f \in \{1, \dots, F\}$;

v : set of speed, $v \in \{1, \dots, s\}$;

Parameters:

n : the number of all jobs;

m : the number of machines in each factory;

F : the number of factories;

s : the number of processing speeds;

l : job positions in a sequence, $l \in \{1, \dots, n\}$;

V_v : the v th processing speed;

O_{ij} : operation of job i on machine j ;

Decision variables:

P_{ij} : the standard processing time of O_{ij} ;

\hat{P}_{ij} : the actual processing time of O_{ij} ;

$PP_{f,j,v}$: the energy consumption per unit time of machine j in factory f running at speed v ;

$SP_{f,j}$: the energy consumption of the machine j at stand-by mode per unit time in factory f ;

$PEC_{f,j}$: processing energy consumption of machine j in factory f ;

$SEC_{f,j}$: idle energy consumption of machine j in factory f ;

TEC : total energy consumption;

C_{max} : the makespan of a schedule;

$C_{i,j,f}$: the completion time of position l on machine j in factory f ;

$C(f)$: the makespan of factory f ;

$x_{i,l,f}$: a binary variable is defined as 1 when job i occupies position l in factory f , and 0 otherwise.

$z_{i,j,v}$: a binary variable is defined as 1 when job i process on machine j at speed v , and 0 otherwise

DHPFSP-VPS includes two objectives: makespan and TEC.

The first objective, makespan, is represented by Eq. (1), where the makespan is the sum of the completion times of all jobs.

$$\text{Makespan} = C_{max} = \max_f C(f) \quad \forall f \quad (1)$$

The second objective, TEC, is represented by Eq. (2), where the energy consumption includes the processing energy consumption (PEC) of all machines in the factories and the idle energy consumption (SEC). The calculation methods for PEC and SEC are demonstrated in Eqs. (3) and (4), respectively.

$$TEC = \sum_{f=1}^F \sum_{j=1}^m (PEC_{f,j} + SEC_{f,j}) \quad (2)$$

$$PEC_{f,j} = \sum_{i=1}^n \sum_{l=1}^n \left(x_{i,l,f} \cdot P_{ij} \cdot \sum_{v=1}^s z_{i,j,v} \cdot PP_{f,j,v} \right) \quad \forall f, j \quad (3)$$

$$SEC_{f,j} = \left(C(f) - \sum_{i=1}^n p_{ij} \cdot \sum_{i=1}^n x_{i,l,f} \right) \cdot SP_{f,j} \quad \forall f, j \quad (4)$$

Makespan and TEC are the two most important indicators in production, and also they are two conflicting objectives. A smaller makespan implies a shorter completion time, which requires higher processing speeds. However, higher processing speeds lead to increased energy consumption. Therefore, to achieve a better balance between the two objectives, it is essential to employ effective scheduling strategies and make appropriate speed selections.

Based on the aforementioned assumptions and the actual production scenario, the constraints of DHPFSP-VPS are as follows:

$$\sum_{i=1}^n \sum_{l=1}^F x_{i,l,f} = 1 \quad \forall i \quad (5)$$

$$\sum_{i=1}^n \sum_{l=1}^F x_{i,l,f} = 1 \quad \forall l \quad (6)$$

$$\sum_{v=1}^s z_{i,j,v} = 1 \quad \forall i, j \quad (7)$$

$$p_{ij} = t_{ij} \cdot \sum_{v=1}^s \frac{z_{i,j,v}}{V_v} \quad \forall i, j \quad (8)$$

$$C_{l,j,f} = \sum_{i=1}^n x_{i,l,f} \cdot p_{ij} \quad l = 1, j = 1, \forall f \quad (9)$$

$$C_{l,i,f} \geq C_{l,j-1,f} + \sum_{i=1}^n x_{i,l,f} \cdot p_{ij} \quad \forall l, j, f \quad (10)$$

$$C_{l,i,f} \geq C_{l,j-1,f} + \sum_{i=1}^n x_{i,l,f} \cdot p_{ij} \quad \forall l > 1, j, f \quad (11)$$

$$C(f) \geq C_{l,j,f} \quad \forall l, f \quad (12)$$

$$C_{l,j,f} \geq 0 \quad \forall l, j, f \quad (13)$$

$$C_{\max} = \max_f C(f) \quad \forall f \quad (14)$$

$$x_{i,l,f} \in \{0, 1\} \quad \forall i, l, f \quad (15)$$

$$z_{i,j,v} \in \{0, 1\} \quad \forall i, j, v \quad (16)$$

Eq. (5) restricts each job to be assigned to a single factory. Eq. (6) guarantees that each machine handles only one job concurrently. Eq. (7) makes sure that each operation is executed at a single speed.

Eq. (8) represents the actual processing time for operation O_{ij} . Eq. (9) guarantees that the starting time for the first job on the first machine in factory f is zero. Eq. (10) ensures that each operation of a job can only begin after its preceding operation has finished. Eq. (11) ensures that each machine can only process the next operation after completing the previous one. Eqs. (12) and (13) are processing time constraints for each factory. The makespan objective is defined as Eq. (14). Eqs. (15) and (16) specify the range of binary variables.

2.3 Encoding and Decoding

Encoding method in scheduling occupied an important position in scheduling solution [39]. DHPFSP-VPS, includes three sub-problems: (i) determining the factories for processing each job; (ii) determining the job processing sequence within each factory; (iii) selecting processing speeds for operations. To better utilize DMSSA for solving DHPFSP-VPS, this study adopts a three-stage encoding, which includes job sequence (JS), factory assignment (FS), and variable speed encoding (VS). An example of DHPFSP-VPS with 7 jobs, 2 factories, and 3 speed options is presented in Table 2. In this example, four jobs are processed in Factory 1 with the sequence of 1, 5, 3, 7, while three jobs are processed in Factory 2 with the sequence of 6, 2, 4. Each job consists of three operations, and each factory has three machines dedicated to processing these operations. Assuming a standard processing time of 3 h for each operation, the schedule's Gantt chart, generated based on the scheduling scheme in Table 2, is depicted in Fig. 1.

Table 2: An example of DHPFSP-VPS

$FS_i(J,F,V)$	JS	1	5	2	7	6	3	4
	FA	1	1	2	1	2	1	2
	VS	3	1	3	1	2	3	1
		1	1	1	3	2	2	2
		2	2	2	1	1	2	2

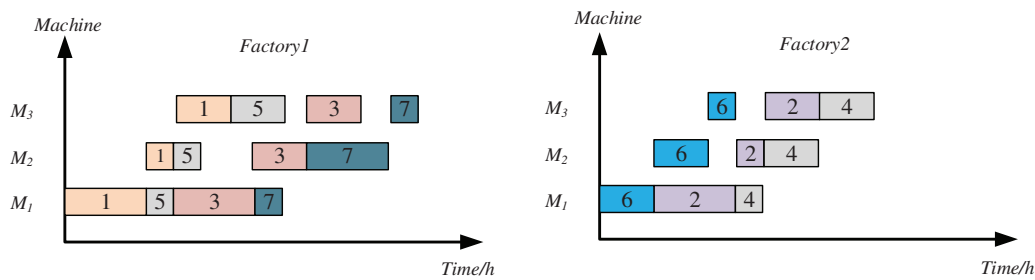


Figure 1: Gantt chart according to Table 2

During the decoding process, jobs are initially assigned to factories based on FA. Subsequently, the job processing sequence within each factory is arranged according to JS. Lastly, the processing speeds for each operation of the jobs are determined based on VS. The actual processing time is calculated by dividing the standard processing time by the processing speed. The processing energy consumption can be calculated based on the principle that it is positively correlated with the square of the processing speed.

3 Squirrel Search Algorithm

3.1 Description of the Squirrel Search Algorithm

Squirrel Search Algorithm (SSA) is a new swarm intelligence algorithm proposed in recent years, which has been proved to be highly competitive in solving real-world problems. However, there is no research on the application of multi-objective squirrel search algorithm in distributed job shop scheduling. SSA is a three-population based algorithm, which has excellent population cooperation ability and can avoid blind crossover and mutation in scheduling optimization, which is very important for complex combinatorial optimization problems. In addition, energy-efficient distributed job-shop scheduling usually involves a large number of job and resource problems, and the data dimension of the optimization problem is high, while the excellent performance of SSA in dealing with similar high-dimensional data makes it may be a suitable choice to deal with such problems. In view of this, as well as the advantages of SSA compared with other algorithms in convergence speed and avoiding local optimum, this paper chooses SSA algorithm to solve the optimal energy-saving distributed job shop scheduling problem.

SSA proposed by Jain et al. [29] in 2018 based on the dynamic feeding and gliding behavior of flying squirrels. Flying squirrels possess a parachute-like membrane that enables them to alter lift and air resistance effortlessly during gliding, exhibiting a complex and energy-efficient aerodynamic motion. During warm seasons, squirrels glide continuously among forest trees in search of food and to store excess provisions. However, in winter, reduced food availability and leaf scarcity increase the risk of predation, resulting in decreased squirrel activity. Squirrels only regain their activity levels at the end of winter. Three assumptions are formulated to enhance the simulation of squirrel behavior:

(1) Each squirrel in the forest occupies a tree individually.

(2) The forest comprises three types of trees: hickory, oak, and normal trees. Hickory trees offer the highest-quality food, oak trees provide regular food, while normal trees do not have any food resources.

(3) The forest consists of a single hickory tree and three oak trees, while the remaining trees are normal trees.

Under the given assumptions, the positions of each squirrel are randomly initialized. Then, the fitness of each squirrel is evaluated, and the individuals are sorted in ascending order based on their fitness. The squirrel with the best fitness is assigned to the hickory tree, and the squirrels ranked 2nd to 4th in terms of fitness are assigned to the oak trees. The remaining squirrels are allocated to the normal trees.

3.2 Individual Location Update Methods

Squirrels update their positions by gliding across different trees. The updating of squirrel positions can be categorized into three stages, determined by the initial and final positions of their glides.

Step 1: Squirrel located on oak tree moves towards the hickory tree using the position update method described by Eq. (17).

$$FS_{at}^{t+1} = \begin{cases} FS_{at}^t + d_g \times G_c \times (FS_{ht}^t - FS_{at}^t) & R \geq P_{dp} \\ \text{random location} & \text{otherwise} \end{cases} \quad (17)$$

where FS_{at} is the location of the squirrel on the oak tree and FS_{ht} is the location of the squirrel on the hickory tree. d_g represents a random gliding distance, G_c is a gliding parameter, and R is a random number in $[0,1]$. P_{dp} represents the probability of the presence of a predator.

Step 2: Squirrel located on normal tree migrates to the oak tree using the position update method depicted in Eq. (18):

$$FS_{nt}^{t+1} = \begin{cases} FS_{nt}^t + d_g \times Gc \times (FS_{at}^t - FS_{nt}^t) & R \geq P_{dp} \\ \text{random location} & \text{otherwise} \end{cases} \quad (18)$$

Step 3: Squirrel located on normal tree relocates to the hickory tree using the location update method illustrated in Eq. (19):

$$FS_{nt}^{t+1} = \begin{cases} FS_{nt}^t + d_g \times Gc \times (FS_{ht}^t - FS_{nt}^t) & R \geq P_{dp} \\ \text{random location} & \text{otherwise} \end{cases} \quad (19)$$

3.3 Season Detection Condition

To avoid local optima, a seasonality detection condition is introduced after the movement of all individuals. Eq. (20) is used to determine the seasonality constant, and Eq. (21) is used to determine the minimum value:

$$S_c^t = \sqrt{\sum_{k=1}^d (FS_{at,k}^t - FS_{ht,k}^t)^2} \quad t = 1, 2, 3 \quad (20)$$

$$S_{\min} = \frac{10e^{-6}}{(365)^{t/(T/2.5)}} \quad (21)$$

where T represents the maximum number of iterations, t represents the current iteration. When $S_c^t < S_{\min}$, the seasonality condition is satisfied, indicating the end of winter. At this stage, squirrels that have not found food yet are randomly repositioned according to Eq. (22). In the formula, $Lévy(n)$ is the step size that follows a levy distribution in the random walk model, and FS_U and FS_L represent the upper and lower bounds of the individual's position, respectively.

$$FS_{nt}^{new} = FS_L + Lévy(n) \times (FS_U - FS_L) \quad (22)$$

4 The DMSSA Is Designed Specifically for the DHPFS-VPS Problem

4.1 Hybrid Initialization Strategies

Employing effective initialization methods can substantially enhance the quality of the initial population, benefiting both the algorithm's rapid convergence and the final solution's quality. Hence, designing an effective initialization strategy for DHPFS-VPS holds immense significance. Four hybrid initialization strategies are proposed, each based on distinct optimization objectives, as follows:

(1) The principle of makespan prioritizing: To minimize the maximum completion time of the schedule, the processing speeds of all machines are set to the maximum. Firstly, JS and FA are randomly initialized, and then the VS of all operations is set to the maximum. This step aids in minimizing the makespan.

(2) The principle of TEC prioritizing: To minimize the energy consumption of the schedule, the processing speeds of all machines are set to the minimum. Similarly, JS and FA are randomly initialized first, and then the VS of all operations is set to the minimum. This step aids in reducing energy consumption during processing.

(3) To prevent one factory from processing an excessive number of jobs, which could lead to a longer overall makespan, the factory with the lowest current workload is selected when assigning jobs

to factories. Firstly, JS and VS are randomly initialized, and then the processing times of all machines in each factory are evaluated. The factory with the shortest processing time for a job is chosen to generate FA. If multiple factories have an equal workload, one of them is randomly selected.

(4) Random initialization: The diversity of the initial population is crucial for achieving the final scheduling solution [39]. Therefore, as part of this study, a portion of the individuals adopt a random initialization strategy that involves the random initializing of JS, FA, and VS. This step will help enrich the diversity of the population.

To effectively implement the aforementioned four strategies, a quarter of the individuals are generated for each strategy. The adoption of a hybrid initialization strategy allows for a thorough exploration of the extreme spaces of the two objectives and ensures diversity in the initial population. This approach is beneficial for increasing the distribution of solutions and accelerating the convergence speed of the algorithm.

4.2 Improved Strategies for Making DMSSA Applicable to Discrete Scheduling Problems

4.2.1 Improved Population Hierarchy

In the standard SSA, the population is categorized into three categories: one squirrel perched on a hickory tree, three squirrels on oak trees, and the remaining individuals on normal trees. During foraging, collaboration is required. However, the basic SSA exhibits slow convergence speed and premature convergence. This is because the SSA relies only on the guidance of four squirrels, which weakens the information exchange and cooperation among the population, to some extent limiting the algorithm’s search capability. In the process of collective search, having too few guiding individuals weakens the group’s collaborative ability and leads to a linear centralization of population decision-making. This is reflected in the convergence process, where the algorithm cannot escape the domain of local optimal solutions and experiences premature convergence.

This study improves the population hierarchy system of squirrels by adjusting the distribution proportions of the population individuals as follows: 10%: 30%: 60%, as shown in Fig. 2, where “ps” represents the population size. Following the improvement, the number of individuals in categories FS_{ht} and FS_{ot} increases, enhancing the guiding role of the head individuals. As this research deals with a multi-objective problem, a sorting method akin to the non-dominated sorting and crowding distance employed in the NSGA-II algorithm is utilized to rank the individuals. The top 10% of the sorted individuals are assigned to hickory trees, followed by assigning 10% to 40% of individuals to oak trees and allocating the remaining 60% of individuals to normal trees.

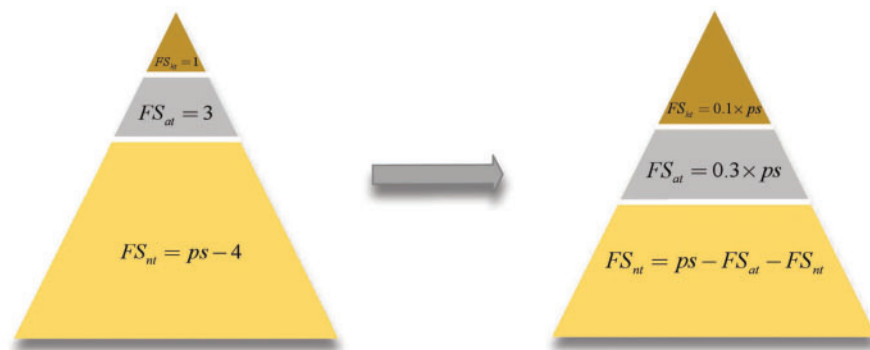


Figure 2: Improvement of the algorithm population hierarchy

4.2.2 Improved Position Updates

DHPFS-VPS is a discrete problem, while SSA is continuous, so it is necessary to improve the position update method. The improved DMSSA categorizes the squirrel movement into three stages, corresponding to the three types:

Step 1: The squirrels on the oak tree (FS_{at}) move towards the hickory tree (FS_{ht}). In the absence of predators ($R \geq P_{dp}$), individuals on the hickory trees guide the individuals on the oak trees using discrete Partial Match Crossover (PMX) for JS. Fig. 3 illustrates an example of PMX. Firstly, squirrels located on the hickory trees and oak trees are selected, respectively. Then, two different points are randomly selected, and the permutations between these two points are exchanged to generate two offspring FS_1 and FS_2 . These generated individuals may contain illegal solutions and require mapping.

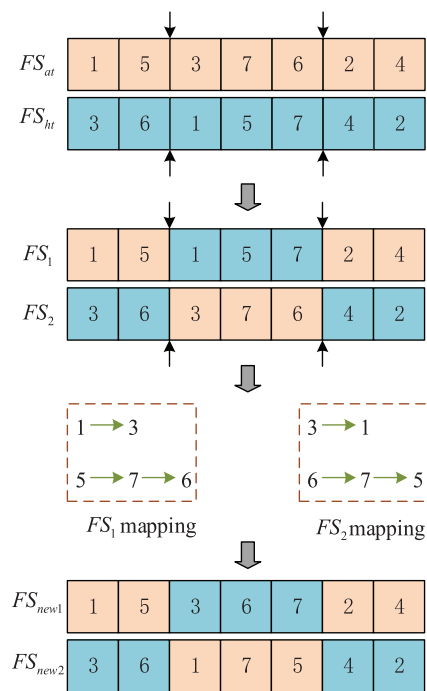


Figure 3: PMX

As shown in Fig. 3, there are repetitions in Jobs 1 and 7 in FS_1 . Similarly, there are repetitions in Jobs 3 and 6 in FS_2 . For FS_1 , the repeated job 1 is replaced with the corresponding Job 3 in FS_2 . Likewise, Job 5 is replaced with Job 7, and the extra Job 7 is then replaced with Job 6. For FS_2 , after applying the same replacement operation, the final two new individuals FS_{new1} and FS_{new2} are generated.

Since there is no correlation between the factory assignment and speed settings of each job, and speed settings do not have permutation constraints, Uniform Crossover (UX) is used for FA and VS. UX randomly selects corresponding encodings of two individuals for exchange. Fig. 4 illustrates a schematic diagram of UX.

When a predator is present ($R < P_{dp}$), the squirrel’s movement is not guided by superior individuals, and in the standard SSA, the squirrel randomly moves to other positions. However, in this case, as FS_{at} is already a better individual, randomly moving to other positions would result in the loss of its previous memory, which is unfavorable for algorithm convergence. Thus, in this paper, the random

movement of FS_{at} is improved by importing the local search methods introduced in Section 4.3.1. Local search are conducted near the current position, which not only retains the previous memory but also prevents the algorithm from getting trapped in local optima. The movement of FS_{at} after the improvement of Eq. (17) is shown in Eq. (23), where \otimes denotes crossover.

$$FS_{at}^{t+1} = \begin{cases} FS_{at}' \otimes FS_{ht}' & R \geq P_{dp} \\ FS_{at}' \text{ local search} & \text{otherwise} \end{cases} \quad (23)$$

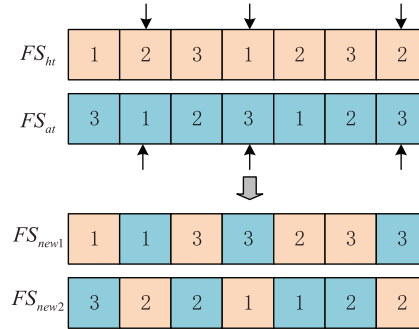


Figure 4: UX

Step 2: The squirrels on the normal tree (FS_{nt}) move towards the hickory tree (FS_{ht}). In the absence of natural enemies ($R \geq P_{dp}$), individuals on the hickory trees guide individuals on the normal trees, following a similar approach as described in Eq. (23). For JS, PMX guidance is still used, while UX guidance is employed for FA and VS. When natural enemies are present ($R < P_{dp}$), a mutation approach is used instead of the “random location” described in Eq. (18), which is advantageous for increasing the population’s diversity. For JS, the mutation involves randomly swapping the positions of two items. For FA, the mutation entails randomly selecting an item and reassigning it to a different factory. For VS, the mutation entails randomly selecting an operation and changing its speed. The improved movement approach for Eq. (18) is described in Eq. (24).

$$FS_{nt}^{t+1} = \begin{cases} FS_{nt}' \otimes FS_{ht}' & R \geq P_{dp} \\ FS_{nt}' \text{ mutation} & \text{otherwise} \end{cases} \quad (24)$$

Step 3: The squirrels on the normal tree (FS_{nt}) move towards oak trees (FS_{at}). The movement approach in this step follows the same approach as in Step 2, and the improved movement approach for Eq. (19) is described in Eq. (25).

$$FS_{nt}^{t+1} = \begin{cases} FS_{nt}' \otimes FS_{at}' & R \geq P_{dp} \\ FS_{nt}' \text{ mutation} & \text{otherwise} \end{cases} \quad (25)$$

4.3 Search Strategy

4.3.1 Local Search Strategy

In the standard SSA, the superior individuals on the oak trees may undergo random initialization to other positions, as shown in Eq. (17). This causes the superior individuals to completely abandon their prior memory, resulting in a considerable loss of the previous iterative outcomes. To address this issue, an improvement is proposed in this study. When $R < P_{dp}$, a local search strategy is applied to the individuals on the oak trees. Designing an effective local search strategy can greatly enhance the individuals’ search ability and have a positive impact on scheduling outcomes [40]. Therefore, based

on the characteristics of DHPFS-VPS, this study introduces the following six local search strategies for the individuals on the oak trees. Each local search algorithm needs to be executed sequentially, ensuring equal triggering frequency for all six algorithms. For detailed implementation, please refer to Algorithm 1.

LS1: Randomly select two jobs and swap their positions, as shown in Fig. 5a. This operation helps increase the diversity of solutions.

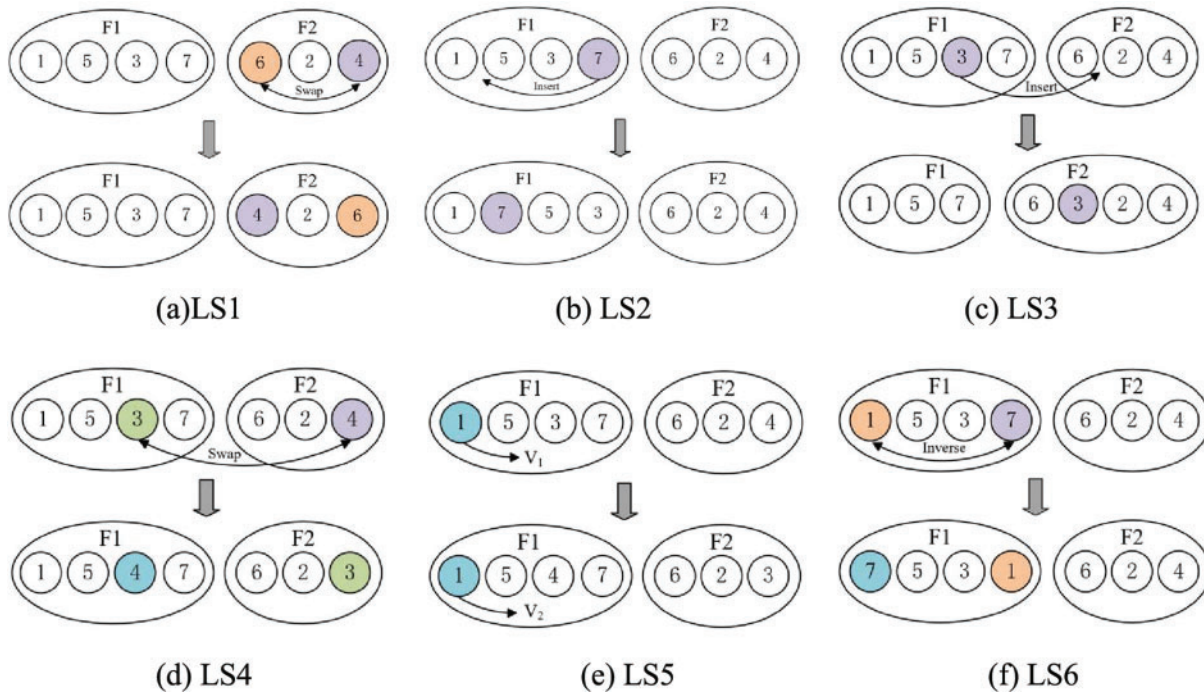


Figure 5: Local search

LS2: Perform an insertion operation within the critical factory. Within a critical factory, randomly select a critical operation and insert it at another possible position in the factory, as shown in Fig. 5b.

LS3: Perform an insertion operation outside the critical factory. Within a critical factory, randomly select a critical operation and insert it into a possible position in another factory, as shown in Fig. 5c. This helps reduce the makespan.

LS4: Exchange operation between factories. Randomly select two factories, then randomly select two operations and exchange their positions, as shown in Fig. 5d.

LS5: Perform a speed increase operation within critical factories. Within a critical factory, randomly select a critical operation and increase its processing speed, as shown in Fig. 5e. This helps reduce the makespan.

LS6: Perform an exchange operation within critical factories. Within a critical factory, randomly select two critical operations and swap their positions, as shown in Fig. 5f.

Algorithm 1: Local search method

Input: Randomly select an individual in FS_{at}^t , $k = 1$

(Continued)

Algorithm 1 (continued)**Output:** The updated individual**While** the stopping requirement is not met **do** **If** $k = 1$

LS1;

Else If $k = 2$

LS2;

Else If $k = 3$

LS3;

Else If $k = 4$

LS4;

Else If $k = 5$

LS5;

Else If $k = 6$

LS6;

End If $k = k + 1$; **If** $k > 6$ then set $k = 0$;**End While**

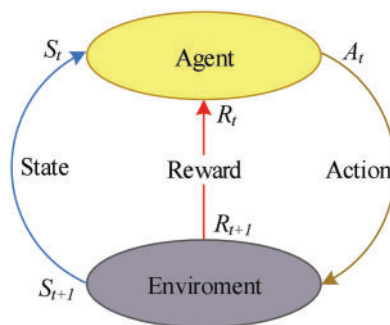
If the current solution dominate the original one, update the solution.

4.3.2 Dynamic Predator Strategy based on Q-Learning

(1) An introduction to Q-Learning

Watkins et al. [41] first introduced Q-learning in 1992, which has now become one of the most famous reinforcement learning methods. It assists algorithms in selecting optimal parameters throughout the iteration process and enhancing algorithm performance. Q-learning is comprised of a quintuple (A, E, C, S, R), representing the Agent, Environment, Action set, State set, and Reward. Referring to Fig. 6, the Agent performs action A_t based on its current state S_t at time t in the Environment, then receives reward R_{t+1} , and its state transitions to a new state S_{t+1} . Eq. (26) is used to update the Q-value as follows:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha [R_t + \gamma \max (Q(S_{t+1}, A_t) - Q(S_t, A_t))] \quad (26)$$

**Figure 6:** Q-learning

$Q(S_t, A_t)$ represents the Q -value when taking action A_t in state S_t , R_t represents the reward signal, α is the learning rate, and γ is the discount factor. This formula means that after taking action A_t , we receive an immediate reward R_t and add it to the maximum Q -value of the next state S_{t+1} , updating the corresponding element in the Q -table. This operation is controlled by α , which determines the learning speed, but it is also influenced by γ , which represents the future rewards in the next stage of the Q -value. When α tends to 0, the immediate reward has minimal impact on the current Q -value, leading to slower learning but more stable learning. Conversely, larger values of α enable faster learning but may lead to unstable Q -values.

(2) Definition of agent and action

In SSA, the probability of predator existence, denoted as P_{dp} , affects the individual's movement direction. In the standard SSA, P_{dp} is a fixed value of 0.1, which is obviously not conducive to dynamically selecting the movement mode of the algorithm during iteration [42]. To optimize the selection of individual movement directions at different stages of iteration and obtain a better Pareto solution set, a dynamic P_{dp} based on Q-learning is designed. The predator existence probability for each generation is abstracted as an agent, representing the successful parameter selection. Three candidate values for the predator existence probability are set: $P_{dp} = 0.3, 0.5, \text{ and } 0.7$.

(3) State setting

The changes in the Agent's state provide feedback to Q-learning to determine whether the action taken can improve the quality of the Pareto Front (PF). To describe the state, convergence (CV) and diversity (DV) metrics are defined, as shown in Eqs. (27) and (28):

$$CV(P, P^*) = \frac{\sqrt{\sum_{y \in P} \min_{x \in P^*} dis(x, y)^2}}{|P|} \quad (27)$$

$$DV = \frac{\sum_{i=1}^{N-1} |d_i - \bar{d}|}{(N-1)\bar{d}} \quad (28)$$

The calculation of CV and DV is similar to LI , etc. [43]. Here, P represents the PF obtained by the algorithm, and P^* is a predefined reference point. d_i denotes the Euclidean distance between adjacent points in the PF , and \bar{d} is the average of d_i values. A smaller CV is better, while a larger DV is better. To describe the changes in the state, Eqs. (29) and (30) provide the calculation methods for the variations of CV and DV at each iteration.

$$\Delta CV = CV_{i-1} - CV_i \quad (29)$$

$$\Delta DV = DV_i - DV_{i-1} \quad (30)$$

According to the different values of ΔCV and ΔDV during the iteration process, four states are defined, namely:

$$(1) \Delta CV > 0, \Delta DV > 0;$$

$$(2) \Delta CV > 0, \Delta DV \leq 0;$$

$$(3) \Delta CV \leq 0, \Delta DV > 0;$$

$$(4) \Delta CV \leq 0, \Delta DV \leq 0;$$

Specifically, (1) and (2) occur in the early stages of iteration, where the population is relatively scattered. (3) and (4) occur in the later stages of iteration, where the population becomes more clustered.

(4) Definition of reward

Following the execution of an action, the agent is either rewarded or punished. The selected action will be rewarded if it increases the PF's diversity; otherwise, it will be punished. For this study, the maximum reward value is predetermined as 10, as used in previous research. To mitigate the impact of punishment on previous rewards, the punishment is set to 0. The definition of rewards in this paper is shown in Eq. (31):

$$Reward = \begin{cases} 10 \Delta DV > 0 \\ 0 \Delta DV \leq 0 \end{cases} \quad (31)$$

(5) Based on Q-learning parameter selection

Algorithm 2 describes the details of parameter selection based on Q-learning. Firstly, the Q-table is initialized with all zeros, and CV and DV as well as the state S_i are initialized. Then, the action A_i is executed, resulting in PF, and ΔCV and ΔDV are updated. The agent transitions to the next state S_{i+1} . Lastly, the (S_i, A_i) in the Q-table is updated using Eq. (26).

Algorithm 2: Dynamic predator strategies based on Q-learning

Input: Population, α, γ

Output: Q-table

Q-table(4,3)=0; $CV_i = CV_{i-1} = DV_i = DV_{i-1} = 0$;

While the stopping requirement is not met **do**

$CV_i = CV_{i-1}$; $DV_i = DV_{i-1}$;

Calculate agent's state S_i ;

Select the max (S_i, A_i) action A_i , $i = 1, 2, 3, 4$;

Execute action A_i for SSA to update population and get PF;

Calculate C_i and i by Eqs. (27) and (28);

Calculate C_i and i by Eqs. (29) and (30);

Get action A_i 's reward (S_i, A_i) by Eq. (20);

Calculate the new solution state S_{i+1} ;

Update the Q-table by Eq. (26);

End While

4.4 Speed Control Energy-Efficient Strategy

TEC, an additional crucial objective of DHPFS-VPS. Designing effective energy-efficient techniques is a key technology for energy scheduling. However, in DHPFS-VPS, makespan and TEC are two conflicting sub-objectives, where improving one objective may lead to the deterioration of the other. Based on this characteristic, this paper presents two speed control energy-efficient strategies that specifically address idle time. These strategies can adjust the processing speed of the machine, reducing energy consumption and minimizing idle time with no impact on makespan.

Speed reduction strategy be applicated to when the start time of an operation on the same machine is greater than the end time of the previous operation, idle time is generated. Therefore, reducing the

processing speed of the operation before the idle time period can effectively decrease idle time, as shown in Fig. 7. After implementing this operation, the processing speeds of operations $O_{5,2}$, $O_{5,3}$, and $O_{3,3}$ are slowed down, and the idle time T_1 and T_2 disappear, T_3 reducing by half to T'_3 . The critical path, representing the longest path from the start to the end, determines the makespan of this schedule. To ensure that the makespan is not affected, this operation must be applied solely to non-critical operations outside the critical path.

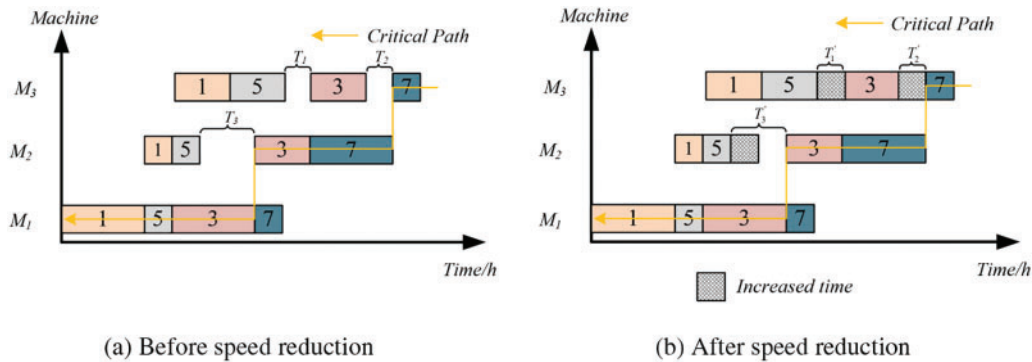


Figure 7: Speed reduction strategy

Speed increase strategy be applied when idle time is generated due to the excessive processing time of a preceding operation, as in Fig. 7a where the operation $O_{3,2}$ has excessive processing time, causing the start time of $O_{3,2}$ to be later than the end time of $O_{5,2}$, a period of idle time is created. Thus, by appropriately increasing the processing speed of $O_{3,1}$, and reducing its processing time, the start time of $O_{3,2}$ can be advanced, consequently reducing idle time. As shown in Fig. 8, the implementation of the speed increase strategy not only reduces idle time but also decreases the makespan.

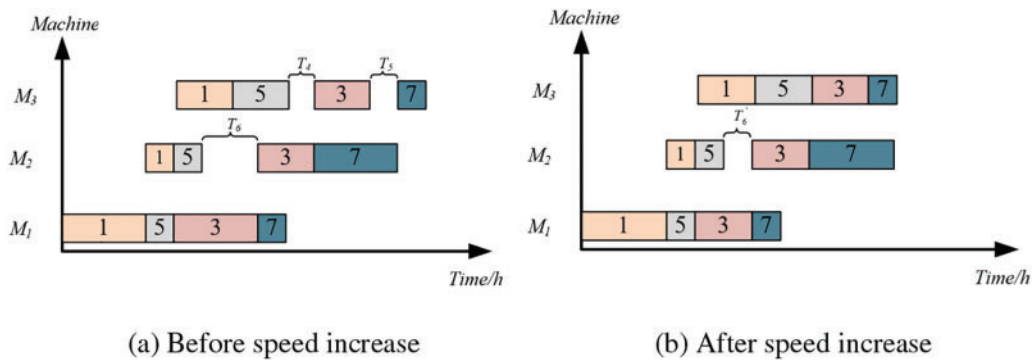


Figure 8: Speed increase strategy

The speed reduction strategy extends the completion time of the preceding operation before idle time, whereas the speed increase strategy advances the start time of the succeeding operation after idle time. Both strategies can reduce energy consumption without increasing the makespan. To balance the number of machines, a random selection probability of 0.5 is set for each of the two strategies. Furthermore, to minimize runtime, we specify that the execution of the strategies should only occur when the working time exceeds 80% of the total time. This strategy serves as a comprehensive alternative to the seasonal detection condition discussed in Section 3.3 of SSA.

4.5 Framework of DMSSA

The overall framework of DMSSA, incorporating the search components mentioned above, including hybrid initialization, local search, population position updates, parameter selection, and energy-efficient strategy, is illustrated in Fig. 9. Firstly, high-quality solutions are generated using hybrid initialization. Secondly, individuals are subjected to a non-dominated sorting based on crowding distance, and positions are allocated to all individuals. Next, the Q-learning is employed to select the optimal P_{dp} . Subsequently, individual position information is updated using an improved position update approach. Finally, the speed control energy-efficient strategy is applied to effectively reduce TEC.

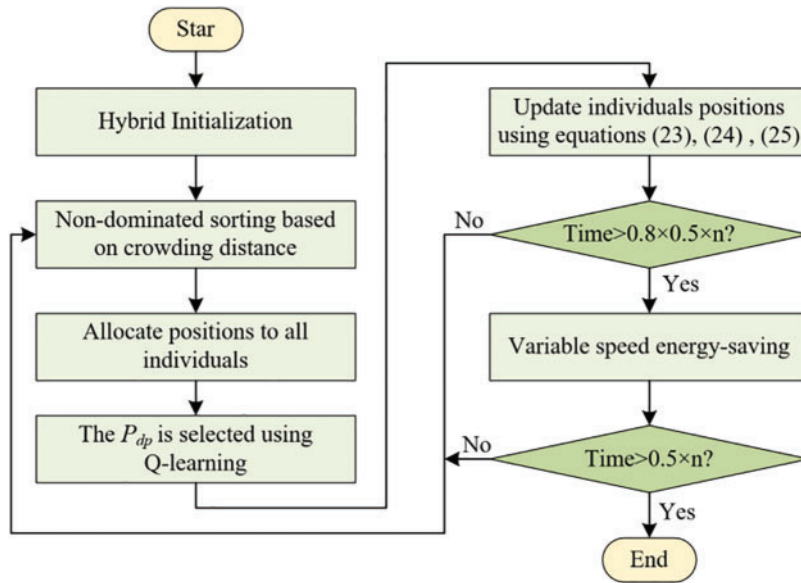


Figure 9: Framework of the DMSSA for DHPFSP-VPS

5 Experimental Results and Discussion

5.1 Performance Metrics

(1) Hypervolume (HV) [44]: HV can evaluate the distribution of Pareto solutions in the objective space. A larger HV value indicates better overall performance of the algorithm. Eq. (32) shows how to calculate it:

$$HV(P, r) = \bigcup_{x \in P} v(x, r) a \tag{32}$$

In the equation, P represents the Pareto solution set obtained by each algorithm, r is the reference point, and let r be (1.1, 1.1). x represents a normalized Pareto solution, and v represents the volume of the hypercube.

(2) Inverse Generational Distance (IGD) [45]: It primarily evaluates the convergence and distribution of an algorithm. A smaller IGD value indicates better performance. The calculation formula

is shown in Eq. (33):

$$IGD = \frac{\sqrt{\sum_{i=1}^n d_i^{*2}}}{n} \quad (33)$$

In the equation, d_i represents the Euclidean distance between each point in the true Pareto front (PF^*) and the nearest point obtained by the algorithm's Pareto front. n represents the number of points in PF^* . Since the true values of PF^* are unknown, in this study, the PF^* for each problem is obtained from the runs of all the algorithms.

(3) Spread [46]: It measures the diversity of a solution set by quantifying its extent of spread. A smaller *Spread* value indicates better performance. The mathematical expression is provided in Eq. (34):

$$Spread = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N - 1) \bar{d}} \quad (34)$$

In the equation, d represents the distance between extreme solutions on the Pareto approximate solution set, and d_i represents the distance between boundary solutions of each objective. N is the number of individuals in the approximate solution set. \bar{d} is the average value of all d_i distances.

5.2 Test Problem

Currently, there is no available open-source benchmark specifically designed for DHPFS-VPS. In this study, Reference [26] served as the basis, and a total of 20 different problems are generated using the Taillard benchmark. These problems are named following the n-m-F notation. The specific parameters are presented in Table 3, where $n = \{20, 50, 100, 200\}$, $F = \{2, 3\}$, $m = \{5, 10, 20\}$, and $V_v = \{1, 1.3, 1.55, 1.75, 2.1\}$. Each operation's standard processing time follows a uniform distribution $U(1, 99)$. In heterogeneous factories, each operation has different processing times in different factories. The processing power is set to 2.0 kWh, with the idle power is set to 1.0 kWh. The CPU stopping time is set to $0.5 \times n$ s. All codes are programmed using MATLAB 2019b, and the software is run on a system with an Intel(R) Core(TM) i9-9900K CPU @ 3.60 GHz and 64 GB RAM, operating on a Windows 10 system. Each problem is independently run 20 times to confirm the reliability of the results.

Table 3: Data set distribution

Input variables	Value
Number of jobs (n)	20, 50, 100, 200
Number of factors (F)	2, 3
Number of machines (m)	5, 10, 20
Production speed (V_v)	1, 1.3, 1.55, 1.75, 2.1
Standard processing time (t_{ij})	$\sim DU(1,99)$

5.3 Parameter Settings

The performance of an algorithm is influenced by multiple factors, with parameters being particularly significant. Therefore, it is crucial to determine the optimal parameters for algorithmic applications. In DMSSA, there are three parameters: the population size (ps), the learning rate (α), and the decay rate (γ). This study adopts the Taguchi method of design of experiment (DOE) [47] to find the optimal parameters by setting parameter levels. Based on mainstream parameter settings, this study sets the parameter levels as follows: $ps = 50, 100, 150$. $\gamma = 0.7, 0.8, 0.9$. $\alpha = 0.1, 0.2, 0.3$.

A calibration experiment is conducted using an orthogonal array $L9(3^3)$. Each parameter is independently run 30 times on the 20_5_2 problem, and the average HV are calculated. As shown in Fig. 10, we evaluate the impact of the three parameters on the HV value. Considering the comprehensive influence, the optimal configuration of the parameters is $ps = 100$, $\gamma = 0.9$, $\alpha = 0.2$.

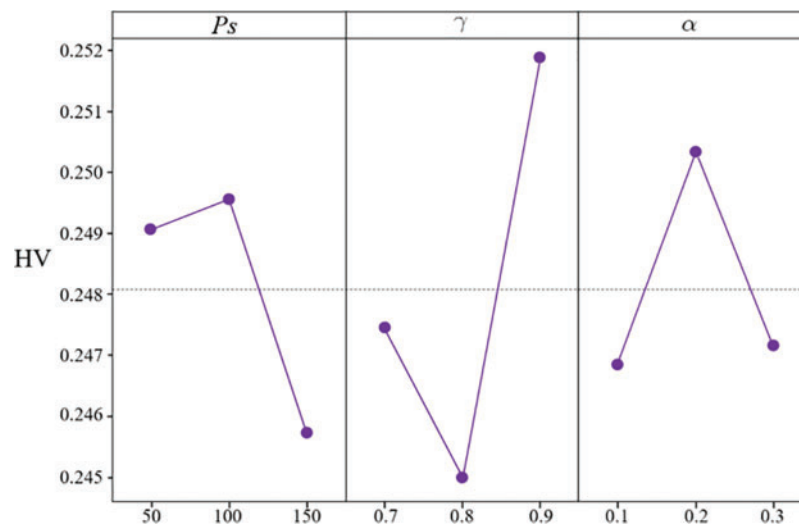


Figure 10: Main effects plot of HV

5.4 Effectiveness of Hybrid Initialization

To demonstrate the competitiveness of the population generated through hybrid initialization, 100 individuals are generated using both hybrid initialization and random initialization on the 20_10_2 problem. Fig. 11 displays the box plots of the individuals generated by both methods for the two objectives. After comparing and analyzing the individuals generated by the two different initialization methods on the two objectives, the results indicate that the population generated by hybrid initialization has higher quality. Specifically, on makespan and TEC, the population generated by hybrid initialization exhibits a broader distribution, enabling better exploration and utilization of the objective space, as seen in Fig. 11a,b. In contrast, the individuals generated by random initialization are relatively concentrated, constraining the population's evolutionary development space. These results suggest that, in the selection of the initial population, hybrid initialization is a more effective and appropriate strategy.

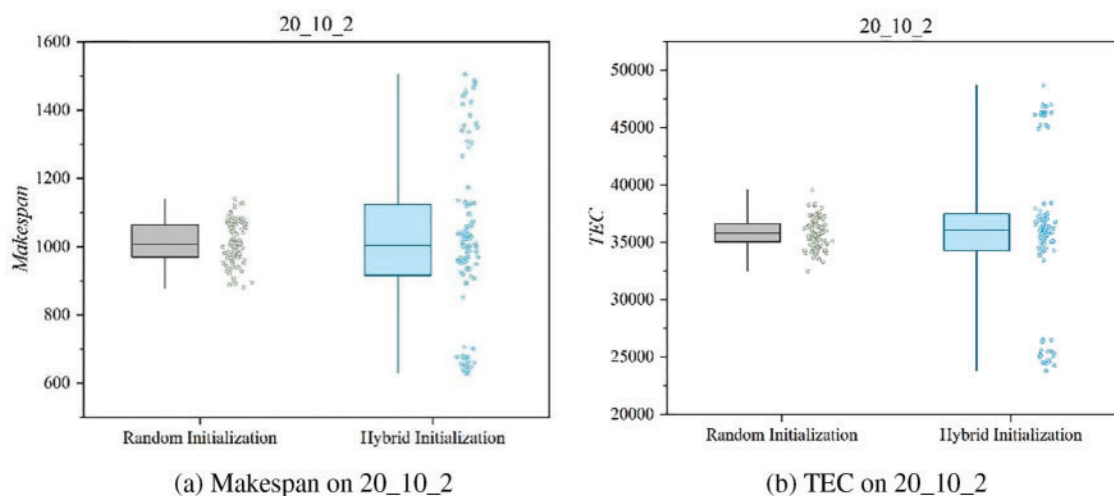


Figure 11: Population boxplot generated by two methods

5.5 Effectiveness of Local Search Strategy

To validate the effectiveness of the local search strategy, DMSSA is compared with DMSSA without local search (referred to as DMSSA_nols). For each problem, the average results of HV, IGD, and Spread are shown in Table 4, with bold values indicating the optimal results. DMSSA achieves 15 HV values, 16 Spread values, and all IGD values outperform DMSSA_nols. This indicates that DMSSA discovers a larger set of non-dominated solutions, and the convergence and distribution of the solutions are enhanced. In conclusion, the local search strategy maximizes the utilization of problem attributes, enabling flexible exploration of potential solutions.

Table 4: Data for DMSSA and DMSSA_nols

Problem	DMSSA_nols			DMSSA		
	HV	IGD	Spread	HV	IGD	Spread
20_5_2	0.2135	0.0645	0.8823	0.2485	0.0620	0.7918
20_10_2	0.2235	0.8099	0.9003	0.2315	0.7476	0.9032
20_20_2	0.2218	1.0033	0.8792	0.2253	0.9512	0.8841
50_5_2	0.2048	0.0850	0.8459	0.1993	0.0690	0.8738
50_10_2	0.2171	0.5769	0.9253	0.2285	0.5566	0.9122
50_20_2	0.2269	0.9002	0.9572	0.2286	0.8318	0.8987
100_10_2	0.2179	0.3838	0.9318	0.2278	0.3726	0.9099
100_20_2	0.2273	0.6982	0.9749	0.2262	0.6857	0.9311
200_10_2	0.2182	0.2080	0.9066	0.2233	0.1058	0.8933
200_20_2	0.2227	0.4286	0.9905	0.2228	0.4184	0.9427
20_10_3	0.2275	0.7851	0.9199	0.2249	0.7192	0.8962
20_20_3	0.2233	23.9011	0.9289	0.2275	23.5078	0.9462
50_5_3	0.2081	0.1521	0.9397	0.2091	0.1460	0.8874

(Continued)

Table 4 (continued)

Problem	DMSSA_nols			DMSSA		
	HV	IGD	Spread	HV	IGD	Spread
50_10_3	0.2230	0.9084	0.9440	0.2357	0.8896	0.9370
50_20_3	0.2151	115.6593	0.9429	0.2184	113.9682	0.9391
100_5_3	0.2093	0.1443	0.9396	0.2076	0.1400	0.9229
100_10_3	0.2283	0.4590	0.9716	0.2350	0.4509	0.9293
100_20_3	0.2283	0.9482	0.9351	0.2281	0.9425	0.9396
200_10_3	0.2224	0.1122	0.9479	0.2367	0.1015	0.8982
200_20_3	0.2233	0.5456	0.9566	0.2194	0.4601	0.9339

5.6 Effectiveness of Dynamic Predator Strategy Based on Q-Learning

To demonstrate the effectiveness of the dynamic predator strategy based on Q-learning, Fig. 12 illustrates the convergence curves of CV and DV on the 20_5_2 problem. In the figure, each trough-to-peak cycle can be thought of as one period. Initially, multiple strategies collaborate during the iteration, leading to a rapid convergence of the population and a significant decrease in CV. During the search process, Q-learning continuously selects optimal parameters to increase diversity, leading to continuous fluctuations in DV. In the optimization process, the algorithm often gets trapped in local optima, but Q-learning is able to continuously select optimal parameters to aid its getting away from local optima. In later iterations, DV shows an upward trend of fluctuation, indicating an increasing population diversity. Table 5 provides the Q-table for the last iteration. In the early iterations, a larger P_{dp} is needed to increase diversity. During the iteration, to hasten the algorithm's convergence, it is necessary to continuously decrease P_{dp} to guide individuals towards better individuals. However, blindly approaching the top individuals can cause the algorithm to get caught in local optima, and in such cases, it is necessary to increase P_{dp} to enhance population diversity. The data in Table 5 indicates that in the last iteration, $P_{dp} = 0.3$, which means Q-learning can continuously guide individuals towards excellent individuals in the later stages. In summary, the predator adaptation strategy based on Q-learning developed in this study can dynamically select optimal parameters at each state.

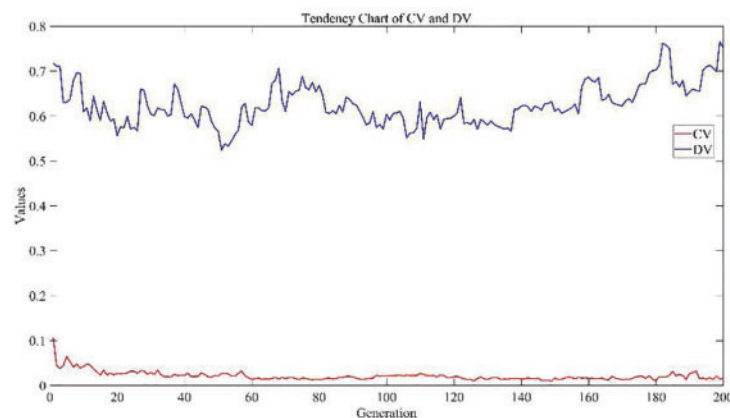
**Figure 12:** Change tendency graph of convergence and diversity

Table 5: The Q-table for the last iteration

	Pdp = 0.3	Pdp = 0.5	Pdp = 0.7
State1 ($\Delta CV > 0, \Delta DV > 0$)	1.53	3.62	13.67
State2 ($\Delta CV > 0, \Delta DV \leq 0$)	1.41	1.08	15.86
State3 ($\Delta CV \leq 0, \Delta DV > 0$)	15.96	0	2.21
State4 ($\Delta CV \leq 0, \Delta DV \leq 0$)	5.04	0	11.39

5.7 Comparison with Other Algorithms

To further demonstrate the outperformance of DMSSA, it is compared with classical discrete multi-objective evolutionary algorithms (MOEAs): (1) General MOEAs: NSGA-II [46], MOEA/D [48], PESA2 [49]. (2) Famous MOEAs proposed recently: AR-MOEA [50], TSNSGA-II [51]. (3) The algorithm specifically designed for DHPFSP-VPS, BRCE [26]. All algorithms have a population size (ps) of 100. The mutation probability for NSGA-II, MOEA/D, PESA2, TSNSGA-II and BRCE is set to 0.2, and the crossover probability is set to 1.0. The number of neighborhoods for MOEA/D and AR-MOEA is 10. The experiments are conducted independently, with each algorithm running 20 times on the same experimental platform. All algorithms terminate when the CPU time reaches $0.5 * n$ s.

Tables 6–8 summarize the average HV, IGD, and Spread values obtained by all algorithms on the 20 problems. The bold values indicate the best results. As shown in Tables 5 and 6, DMSSA outperforms all other algorithms significantly, achieving better HV and IGD values, indicating superior overall performance and diversity. However, DMSSA performs relatively worse than BRCE in terms of HV on 4 problems and IGD on 6 problems. This is because BRCE utilizes an excellent dual-population cooperation strategy, resulting in better solution sets in a few problems. Overall, this outcome is deemed acceptable. Regarding the Spread metric presented in Table 8, DMSSA achieves the highest number of optimal solutions, ranking first, followed by NSGA-II, suggesting that the Pareto solutions obtained by DMSSA exhibit the best distribution. Table 9 presents the Friedman rank test results for all MOEAs with a significance level of 0.05. DMSSA ranks first in all metrics, and the p -values are all below 0.05, signifying the significant superiority of DMSSA over all compared algorithms.

Table 6: HV of DMSSA and other MOEAs

Problem	HV						
	MOEA/D	NSGA-II	SPEA2	AR-MOEA	TSNSGA-II	BRCE	DMSSA
20_5_2	0.0857	0.0793	0.0729	0.0085	0.1017	0.2489	0.2383
20_10_2	0.0501	0.0685	0.0375	0.0114	0.0768	0.2331	0.2385
20_20_2	0.0403	0.0595	0.0257	0.0223	0.0509	0.2199	0.2269
50_5_2	0.0435	0.0407	0.0441	0.0660	0.1825	0.2076	0.2092
50_10_2	0.0318	0.0382	0.0238	0.0530	0.0441	0.2304	0.2360
50_20_2	0.0256	0.0379	0.0177	0.0318	0.0568	0.2169	0.2276
100_10_2	0.0243	0.0404	0.0235	0.0266	0.0301	0.2316	0.2189

(Continued)

Table 6 (continued)

Problem	HV						
	MOEA/D	NSGA-II	SPEA2	AR-MOEA	TSNSGA-II	BRCE	DMSSA
100_20_2	0.0210	0.0440	0.0126	0.0195	0.0396	0.2109	0.2186
200_10_2	0.0183	0.0184	0.0124	0.0231	0.0222	0.2273	0.2281
200_20_2	0.0167	0.0287	0.0130	0.0192	0.0470	0.2108	0.2236
20_10_3	0.0581	0.0635	0.0565	0.0778	0.0876	0.2272	0.2300
20_20_3	0.0380	0.0602	0.0333	0.1775	0.1751	0.2291	0.2258
50_5_3	0.0438	0.0412	0.0415	0.8599	0.8512	0.2058	0.2070
50_10_3	0.0314	0.0444	0.0253	0.1651	0.7672	0.2193	0.2284
50_20_3	0.0292	0.0470	0.0189	0.5965	0.6071	0.2223	0.2171
100_5_3	0.0300	0.0290	0.0314	0.7289	0.7170	0.2175	0.2224
100_10_3	0.0208	0.0332	0.0241	0.0983	0.1896	0.2259	0.2265
100_20_3	0.0217	0.0388	0.0128	0.3420	0.3934	0.2123	0.2213
200_10_3	0.0175	0.0167	0.0141	0.2092	0.2155	0.2206	0.2275
200_20_3	0.0165	0.0318	0.0122	0.0512	0.1083	0.2012	0.2217

Table 7: IGD of DMSSA and other MOEAs

Problem	IGD						
	MOEA/D	NSGA-II	SPEA2	AR-MOEA	TSNSGA-II	BRCE	DMSSA
20_5_2	0.2553	0.1450	0.1347	0.4209	0.1316	0.0366	0.0835
20_10_2	2.3630	1.5430	1.5375	0.6033	1.6697	0.8365	0.8240
20_20_2	2.7245	1.6589	1.6907	0.7387	1.8082	0.8377	0.8374
50_5_2	0.3059	0.2058	0.2017	0.4777	0.2049	0.0610	0.1085
50_10_2	2.2326	1.5291	1.4905	0.4852	1.6623	0.6025	0.5260
50_20_2	3.4822	2.7161	2.8482	1.2311	2.5595	0.8858	0.8536
100_10_2	1.0397	0.7721	0.8263	0.5343	0.7911	0.4923	0.4630
100_20_2	1.6208	1.3932	1.5023	0.8070	1.5042	0.8329	0.7630
200_10_2	0.2611	0.2403	0.2401	0.4411	0.2281	0.0987	0.1169
200_20_2	1.6605	1.4719	1.6559	0.9248	1.5384	0.6250	0.5363
20_10_3	2.7987	1.5957	1.6116	0.3349	1.6816	0.7560	0.7547
20_20_3	31.1316	27.9643	29.9063	39.9080	26.5991	23.7283	23.4942
50_5_3	0.4645	0.2156	0.2095	0.5399	0.2189	0.0893	0.2124
50_10_3	4.1910	2.7143	2.7800	0.8638	2.7907	0.8711	0.8024
50_20_3	111.8082	96.7437	105.8329	132.2805	90.0112	82.4946	82.0380
100_5_3	0.4210	0.2221	0.2156	0.4800	0.2114	0.0905	0.2016
100_10_3	1.7394	1.1709	1.2389	0.4612	1.3224	0.4609	0.4119
100_20_3	3.1533	2.3935	2.8024	1.4782	2.3634	0.9448	0.9436
200_10_3	0.3485	0.3368	0.3343	0.4091	0.3252	0.0809	0.1167
200_20_3	2.6107	2.0960	2.3790	1.5708	2.1225	0.8331	0.8013

Table 8: Spread(min) of DMSSA and other MOEAs

Problem	Spread						
	MOEA/D	NSGA-II	SPEA2	AR-MOEA	TSNSGA-II	BRCE	DMSSA
20_5_2	1.0223	0.9613	0.9816	1.2346	0.9771	0.8101	0.8626
20_10_2	1.0077	0.9779	0.9862	1.0018	0.9968	0.9530	0.9372
20_20_2	1.0067	0.9852	0.9938	1.0073	0.9898	0.9528	0.9259
50_5_2	1.0719	0.9877	0.9869	1.3622	0.9937	0.9656	0.9601
50_10_2	1.0153	0.9937	0.9964	1.0764	1.0003	1.0439	1.0134
50_20_2	1.0017	0.9998	0.9958	1.0410	0.9966	1.0661	0.9956
100_10_2	1.0021	0.9933	0.9966	1.0378	0.9989	1.0598	1.0098
100_20_2	1.0006	0.9949	0.9991	1.0441	0.9948	1.0663	1.0146
200_10_2	1.0070	0.9963	0.9998	1.0094	0.9989	1.2056	1.0827
200_20_2	1.0005	1.0003	1.0431	0.9996	1.0000	1.0615	0.9993
20_10_3	1.0177	0.9977	0.9867	1.0458	1.3962	0.9572	0.9297
20_20_3	1.0092	0.9860	0.9930	1.0366	1.0737	1.0070	0.9409
50_5_3	1.0197	0.9986	0.9959	1.1864	1.2111	0.9744	0.9764
50_10_3	1.0095	0.9994	0.9991	1.0432	1.0688	1.0300	0.9954
50_20_3	1.0044	0.9991	0.9989	1.0499	1.0377	0.9980	0.9753
100_5_3	1.0269	0.9933	1.0073	1.1297	1.0924	1.0374	0.9651
100_10_3	1.0008	1.0019	0.9973	1.0555	1.0276	1.0495	1.0114
100_20_3	1.0017	0.9984	1.0032	1.0694	1.0116	1.0667	0.9936
200_10_3	1.0029	1.0008	0.9983	1.7114	1.0242	1.2155	1.0527
200_20_3	1.0003	0.9996	1.0028	1.0055	1.0046	1.0490	1.0142

Table 9: Frideman ranking of all algorithms

MOEAs	HV		IGD		Spread	
	Rank	P	Rank	P	Rank	P
MOEA/D	5.75	2.35E-16	6.42	1.77E-16	4.55	1.39E-08
NSGA-II	6.35		4.12		2.70	
SPEA2	5.10		4.42		2.83	
AR-MOEA	3.95		4.69		6.20	
TSNSGA-II	3.10		4.46		4.60	
BRCE	2.30		2.31		4.60	
DMSSA	1.45		1.58		2.53	

In order to demonstrate the energy-efficient effect of DMSSA more intuitively, Fig. 13 provides a graphical representation of the PF. Two problems with varying numbers of factories are chosen to verify the efficacy of DMSSA. The figure illustrates that DMSSA can generate solutions with a

uniform distribution, explore a wider range of solutions, and discover a greater number of solutions on both ends. Fig. 13 presents the set of nondominated solutions found by all algorithms running 20 times on the 20_5_2 instance and 200_10_3 instances, which strongly illustrates that the non-dominated solution set found by DMSSA can better explore the frontier, and the number and distribution of solutions are better. The lowest energy consumption of DMSSA finding solutions (829, 9544.37) is the most energy-efficient, which is 1.37% less than the lowest energy consumption of BRCE (784, 9677.23), which is the second most energy efficient, and 2.22% less than the lowest energy consumption of MOEA/D (846, 9761.43), which is the third most energy efficient. Around the completion time 3500 In Fig. 13, the points found by DMSSA are (3496.09, 239556), the points found by SPEA2 are (3504.46, 245153), and the points found by BRCE are (3495.55, 249051). It can be seen that DMSSA saves 2.28% and 3.81% energy than SPEA2 and BRCE respectively when the completion time is close. Through the analysis of the time complexity, DMSSA is comparable to other commonly used metaheuristic algorithms. Moreover, compared to other algorithms, DMSSA can always find scheduling schemes with even lower TEC. The majority of solutions discovered by DMSSA exhibit higher efficiency and superiority compared to those found by other algorithms, providing further evidence of the effectiveness of the speed control energy-efficient strategy.

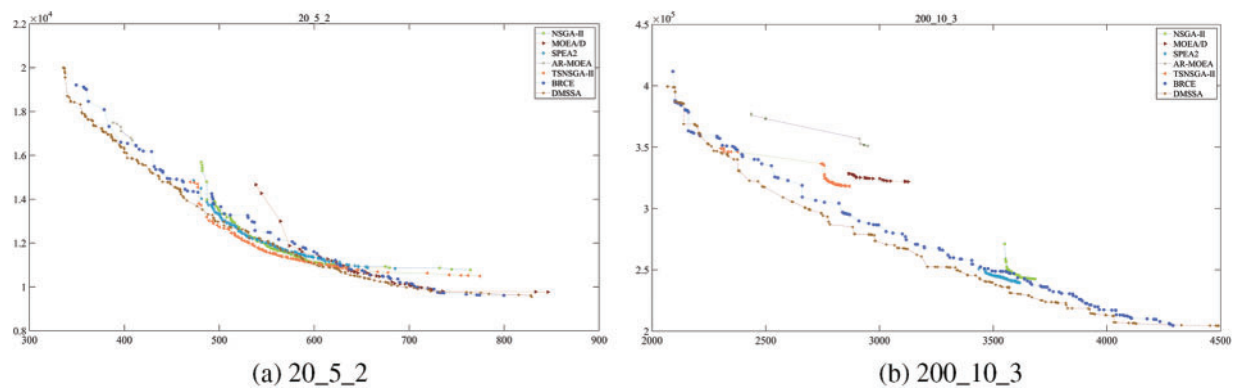


Figure 13: Pareto Frontier obtained by different algorithms

The excellent results of DMSSA are due to the synergy among all the strategies employed. The initial solutions' quality is improved by the effective hybrid initialization method, as detailed in Section 5.3. Furthermore, the improved population ranking system and position update mechanism enable more efficient solution space exploration. Experimental outcomes, as discussed in Section 5.4, have demonstrated that the local search strategy substantially enhances the algorithm's overall performance. The dynamic predator strategy based on Q-learning effectively improves the distribution and convergence of solutions, as described in Section 5.5. The comparative results with other algorithms indicate that DMSSA exhibits outstanding performance, and the proposed speed control energy-efficient strategy effectively reduces TEC.

In conclusion, the experimental dates validate that DMSSA outperforms other MOEAs in all cases. The superiority of DMSSA is statistically significant across all scenarios. Therefore, DMSSA is an effective method for addressing the energy-efficient DHPFSP-VPS problem.

6 Conclusion

This paper proposes a discrete multi-objective squirrel search algorithm aimed at solving the distributed heterogeneous permutation flowshop problem with variable processing speed (DHPFSP-VPS). Firstly, four initialization methods are designed based on the two optimization objectives. Secondly, six local search strategies are developed based on the characteristics of the three-stage encoding. Concurrently, improvements are made to the population ranking system and position update mechanism of SSA to better suit the characteristics of DHPFSP-VPS and enhance its search process. Furthermore, the predator strategy based on Q-learning dynamically selects optimal parameters to balance global exploration and local exploitation, thereby improving the convergence and distribution of the Pareto solution set. Finally, two speed control energy-efficient strategies are designed to reduce idle time and decrease TEC. Experimental results on 20 problems validate the effectiveness of each strategy. Comparative experiments with other algorithms indicate the outstanding performance of the proposed DMSSA in solving DHPFSP-VPS.

However, there are several aspects of work in this field that deserve further research. It would be significant to apply DMSSA to practical industrial difficulties based on the real-world manufacturing context. Moreover, in terms of the problem model, it is also possible to consider more factors, such as processing delays and machine malfunctions. Furthermore, dynamic scheduling scenarios should be considered to effectively accommodate the dynamic nature of real-world production.

Acknowledgement: Not applicable.

Funding Statement: This work was in part supported by the Key Research and Development Project of Hubei Province (Nos. 2020BAB114 and 2023BAB094).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Liang Zeng; data collection: Ziyang Ding; analysis and interpretation of results: Shanshan Wang; draft manuscript preparation and manuscript proofreading: Junyang Shi. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Data is available on request from the authors. The data that support the findings of this study are available from the corresponding author, Shanshan Wang, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of 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.

References

- [1] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Nav. Res. Logist. Q.*, vol. 1, no. 1, pp. 61–68, 1954. doi: [10.1002/NAV.3800010110](https://doi.org/10.1002/NAV.3800010110).
- [2] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976. doi: [10.1287/moor.1.2.117](https://doi.org/10.1287/moor.1.2.117).
- [3] C. N. Wang, G. Andrew Porter, C. C. Huang, V. Tinh Nguyen, and S. Tam Husain, "Flow-shop scheduling with transportation capacity and time consideration," *Comput. Mater. Contin.*, vol. 70, no. 2, pp. 3031–3048, 2022. doi: [10.32604/cmc.2022.020222](https://doi.org/10.32604/cmc.2022.020222).

- [4] F. T. S. Chan, S. H. Chung, and P. L. Y. Chan, "An adaptive genetic algorithm with dominated genes for distributed scheduling problems," *Expert Syst. Appl.*, vol. 29, pp. 364–371, 2005. doi: [10.1016/j.eswa.2005.04.009](https://doi.org/10.1016/j.eswa.2005.04.009).
- [5] A. Toptal and I. Sabuncuoglu, "Distributed scheduling: A review of concepts and applications," *Int. J. Prod. Res.*, vol. 48, no. 18, pp. 5235–5262, Sep. 2010. doi: [10.1080/00207540903121065](https://doi.org/10.1080/00207540903121065).
- [6] B. Naderi and R. Ruiz, "The distributed permutation flowshop scheduling problem," *Comput. Oper. Res.*, vol. 37, pp. 754–768, 2010. doi: [10.1016/j.cor.2009.06.019](https://doi.org/10.1016/j.cor.2009.06.019).
- [7] J. Gao and R. Chen, "A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem," *Int. J. Comput. Intell. Syst.*, vol. 4, no. 4, pp. 497–508, 2011. doi: [10.1080/18756891.2011.9727808](https://doi.org/10.1080/18756891.2011.9727808).
- [8] J. Gao, R. Chen, and W. Deng, "An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem," *Int. J. Prod. Res.*, vol. 51, no. 3, pp. 641–651, 2013. doi: [10.1080/00207543.2011.644819](https://doi.org/10.1080/00207543.2011.644819).
- [9] A. P. Rifai, H. T. Nguyen, and S. Z. M. Dawal, "Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling," *Appl. Soft Comput.*, vol. 40, pp. 42–57, 2016. doi: [10.1016/j.asoc.2015.11.034](https://doi.org/10.1016/j.asoc.2015.11.034).
- [10] S. W. Lin, K. C. Ying, and C. Y. Huang, "Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm," *Int. J. Prod. Res.*, vol. 51, no. 16, pp. 5029–5038, 2013. doi: [10.1080/00207543.2013.790571](https://doi.org/10.1080/00207543.2013.790571).
- [11] W. Shao, D. Pi, and Z. Shao, "A pareto-based estimation of distribution algorithm for solving multiobjective distributed noawait flow-shop scheduling problem with sequence-dependent setup time," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 3, pp. 1344–1360, 2019. doi: [10.1109/TASE.2018.2886303](https://doi.org/10.1109/TASE.2018.2886303).
- [12] C. Lu, Y. Huang, L. Meng, L. Gao, B. Zhang and J. Zhou, "A pareto-based collaborative multi-objective optimization algorithm for energy-efficient scheduling of distributed permutation flowshop with limited buffers," *Robot. Comput. Integr. Manuf.*, vol. 74, 2022, Art. no. 102277. doi: [10.1016/j.rcim.2021.102277](https://doi.org/10.1016/j.rcim.2021.102277).
- [13] F. Zhao, D. Shao, L. Wang, T. Xu, and N. Zhu, "Jonrinaldi, an effective water wave optimization algorithm with problemspecific knowledge for the distributed assembly blocking flow-shop scheduling problem," *Knowl.-Based Syst.*, vol. 243, 2022, Art. no. 108471. doi: [10.1016/j.knosys.2022.108471](https://doi.org/10.1016/j.knosys.2022.108471).
- [14] R. Ruiz, Q. K. Pan, and B. Naderi, "Iterated Greedy methods for the distributed permutation flowshop scheduling problem," *Omega*, vol. 83, pp. 213–222, 2019. doi: [10.1016/j.omega.2018.03.004](https://doi.org/10.1016/j.omega.2018.03.004).
- [15] H. Bargaoui, O. Belkahla, and K. Ghédira, "A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion," *Comput. Ind. Eng.*, vol. 111, pp. 239–250, 2017. doi: [10.1016/j.cie.2017.07.020](https://doi.org/10.1016/j.cie.2017.07.020).
- [16] J. Li, S. Bai, P. Duan, H. Sang, Y. Han and Z. Zheng, "An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system," *Int. J. Prod. Res.*, vol. 57, no. 22, pp. 6922–6942, 2019. doi: [10.1080/00207543.2019.1571687](https://doi.org/10.1080/00207543.2019.1571687).
- [17] J. Lin, Z. J. Wang, and X. Li, "A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem," *Swarm Evol. Comput.*, vol. 36, pp. 124–135, 2017. doi: [10.1016/j.swevo.2017.04.007](https://doi.org/10.1016/j.swevo.2017.04.007).
- [18] M. Akbar and T. Irohara, "Scheduling for sustainable manufacturing: A review," *J. Clean. Prod.*, vol. 205, pp. 866–883, 2018. doi: [10.1016/j.jclepro.2018.09.100](https://doi.org/10.1016/j.jclepro.2018.09.100).
- [19] IEA, "Worldwide trends in energy use and efficiency: Key insights from IEA indicator analysis," 2008. Accessed: Aug. 11, 2024. [Online]. Available: http://sa.indiaenvironmentportal.org.in/files/Indicators_2008.pdf
- [20] J. Wang and L. Wang, "A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 50, no. 5, pp. 1805–1819, 2018. doi: [10.1109/TSMC.2017.2788879](https://doi.org/10.1109/TSMC.2017.2788879).
- [21] J. Wang and L. Wang, "A cooperative memetic algorithm with feedback for the energy-aware distributed flow-shops with flexible assembly scheduling," *Comput. Ind. Eng.*, vol. 168, 2022, Art. no. 108126. doi: [10.1016/j.cie.2022.108126](https://doi.org/10.1016/j.cie.2022.108126).

- [22] Y. Li, Q. Pan, K. Gao, M. F. Tasgetiren, B. Zhang and J. Li, “A green scheduling algorithm for the distributed flowshop problem,” *Appl. Soft Comput.*, vol. 109, 2021, Art. no. 107526. doi: [10.1016/j.asoc.2021.107526](https://doi.org/10.1016/j.asoc.2021.107526).
- [23] F. Zhao, R. Ma, and L. Wang, “A self-learning discrete jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system,” *IEEE Trans. Cybern.*, vol. 52, no. 12, pp. 12675–12686, Dec. 2022, doi: [10.1109/TCYB.2021.3086181](https://doi.org/10.1109/TCYB.2021.3086181).
- [24] A. M. Fathollahi-Fard, L. Woodward, and O. Akhrif, “Sustainable distributed permutation flow-shop scheduling model based on a triple bottom line concept,” *J. Ind. Inf. Integr.*, vol. 24, 2021, Art. no. 100233. doi: [10.1016/j.jii.2021.100233](https://doi.org/10.1016/j.jii.2021.100233).
- [25] J. Jiang, Y. An, Y. Dong, J. Hu, Y. Li and Z. Zhao, “Integrated optimization of non-permutation flow shop scheduling and maintenance planning with variable processing speed,” *Reliab. Eng. Syst. Safety*, vol. 234, 2023, Art. no. 109143. doi: [10.1016/j.ress.2023.109143](https://doi.org/10.1016/j.ress.2023.109143).
- [26] K. Huang, R. Li, W. Gong, R. Wang, and H. Wei, “BRCE: bi-roles co-evolution for energy-efficient distributed heterogeneous permutation flow shop scheduling with flexible machine speed,” *Complex Intelli. Syst.*, vol. 9, pp. 4805–4816, 2023. doi: [10.1007/s40747-023-00984-x](https://doi.org/10.1007/s40747-023-00984-x).
- [27] G. Wang, L. Gao, X. Li, P. Li, and M. F. Tasgetiren, “Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm,” *Swarm Evol. Comput.*, vol. 57, 2020, Art. no. 100716. doi: [10.1016/j.swevo.2020.100716](https://doi.org/10.1016/j.swevo.2020.100716).
- [28] P. Perez-Gonzalez and J. M. Framinan, “A review and classification on distributed permutation flowshop scheduling problems,” *Eur. J. Oper. Res.*, vol. 312, no. 1, pp. 1–21, 2023. doi: [10.1016/j.ejor.2023.02.001](https://doi.org/10.1016/j.ejor.2023.02.001).
- [29] M. Jain, V. Singh, and A. Rani, “A novel nature-inspired algorithm for optimization: Squirrel search algorithm,” *Swarm Evol. Comput.*, vol. 44, pp. 148–175, 2019. doi: [10.1016/j.swevo.2018.02.013](https://doi.org/10.1016/j.swevo.2018.02.013).
- [30] T. Zheng and W. Luo, “An improved squirrel search algorithm for optimization,” *Complexity*, vol. 2019, no. 1, 2019, Art. no. 153. doi: [10.1155/2019/6291968](https://doi.org/10.1155/2019/6291968).
- [31] P. Wang, Y. Kong, X. He, M. Zhang, and X. Tan, “An improved squirrel search algorithm for maximum likelihood DOA estimation and application for MEMS vector hydrophone array,” *IEEE Access*, vol. 7, pp. 118343–118358, 2019. doi: [10.1109/ACCESS.2019.2936823](https://doi.org/10.1109/ACCESS.2019.2936823).
- [32] E. S. M. El-Kenawy *et al.*, “Advanced meta-heuristics, convolutional neural networks, and feature selectors for efficient COVID-19 X-ray chest image classification,” *IEEE Access*, vol. 9, pp. 36019–36037, 2021. doi: [10.1109/ACCESS.2021.3061058](https://doi.org/10.1109/ACCESS.2021.3061058).
- [33] V. P. Sakthivel and P. D. Sathya, “Multi-area economic environmental dispatch using multi-objective squirrel search algorithm,” *Evol. Syst.*, vol. 13, no. 2, pp. 183–199, 2022. doi: [10.1007/s12530-021-09366-5](https://doi.org/10.1007/s12530-021-09366-5).
- [34] Y. Wang and J. Han, “A FJSSP method based on dynamic multi-objective squirrel search algorithm,” *Int. J. Antennas Propag.*, vol. 2021, pp. 1–19, 2021. doi: [10.1155/2021/6062689](https://doi.org/10.1155/2021/6062689).
- [35] B. Jaishankar, S. Vishwakarma, P. Mohan, A. K. S. Pundir, I. Patel and N. Arulkumar, “Blockchain for securing healthcare data using squirrel search optimization algorithm,” *Intell. Autom. Soft Comput.*, vol. 32, no. 3, pp. 1815–1829, 2022. doi: [10.32604/iasc.2022.021822](https://doi.org/10.32604/iasc.2022.021822).
- [36] D. Guha, P. K. Roy, and S. Banerjee, “Frequency control of a wind-diesel-generator hybrid system with squirrel search algorithm tuned robust cascade fractional order controller having disturbance observer integrated,” *Elect. Power Compon. Syst.*, vol. 50, no. 14–15, pp. 814–839, 2022. doi: [10.1080/15325008.2022.2141925](https://doi.org/10.1080/15325008.2022.2141925).
- [37] K. Ishwarya and A. A. Nithya, “Squirrel search optimization with deep convolutional neural network for human pose estimation,” *Comput. Mater. Contin.*, vol. 74, no. 3, pp. 6081–6099, 2023. doi: [10.32604/cmc.2023.034654](https://doi.org/10.32604/cmc.2023.034654).
- [38] D. Maden, E. Çelik, E. H. Houssein, and G. Sharma, “Squirrel search algorithm applied to effective estimation of solar PV model parameters: A real-world practice,” *Neural Comput. Appl.*, vol. 35, no. 18, pp. 13529–13546, 2023. doi: [10.1007/s00521-023-08451-x](https://doi.org/10.1007/s00521-023-08451-x).
- [39] M. Wu, D. Yang, Z. Yang, and Y. Guo, “Sparrow search algorithm for solving flexible jobshop scheduling problem,” in Y. Tan, Y. Shi (eds.), *Advances in Swarm Intelligence*, Cham: Springer, 2021, vol. 12689. doi: [10.1007/978-3-030-78743-1_13](https://doi.org/10.1007/978-3-030-78743-1_13).

- [40] Y. An, X. Chen, K. Gao, Y. Li, and L. Zhang, "Multiobjective flexible job-shop rescheduling with new job insertion and machine preventive maintenance," *IEEE Transact Cybern*, vol. 53, no. 5, pp. 3101–3113, May 2023. doi: [10.1109/TCYB.2022.3151855](https://doi.org/10.1109/TCYB.2022.3151855).
- [41] C. J. C. H. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, 1992. doi: [10.1023/A:1022676722315](https://doi.org/10.1023/A:1022676722315).
- [42] X. Zhang, K. Zhao, L. Wang, Y. Wang, and Y. Niu, "An improved squirrel search algorithm with reproductive behavior," *IEEE Access*, vol. 8, pp. 101118–101132, 2020. doi: [10.1109/ACCESS.2020.2998324](https://doi.org/10.1109/ACCESS.2020.2998324).
- [43] R. Li, W. Gong, and C. Lu, "A reinforcement learning based RMOEA/D for bi-objective fuzzy flexible job shop scheduling," *Expert Syst. Appl.*, vol. 203, 2022, Art. no. 117380. doi: [10.1016/j.eswa.2022.117380](https://doi.org/10.1016/j.eswa.2022.117380).
- [44] L. While, P. Hingston, L. Barone, and S. Huband, "A faster algorithm for calculating hypervolume," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 29–38, 2006. doi: [10.1109/TEVC.2005.851275](https://doi.org/10.1109/TEVC.2005.851275).
- [45] C. H. R. Jethmalani, S. P. Simon, K. Sundareswaran, P. S. R. Nayak, and N. P. Padhy, "Auxiliary hybrid PSO-BPNN-based transmission system loss estimation in generation scheduling," *IEEE Trans. Ind. Inform.*, vol. 13, pp. 1692–1703, 2017. doi: [10.1109/TII.2016.2614659](https://doi.org/10.1109/TII.2016.2614659).
- [46] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002. doi: [10.1109/4235.996017](https://doi.org/10.1109/4235.996017).
- [47] R. C. Van, "Design of experiments using the taguchi approach: 16 steps to product and process improvement," *Technometrics*, vol. 44, no. 3, p. 289, 2002. doi: [10.1198/004017002320256440](https://doi.org/10.1198/004017002320256440).
- [48] Q. Zhang and L. Hui, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, 2007. doi: [10.1109/TEVC.2007.892759](https://doi.org/10.1109/TEVC.2007.892759).
- [49] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," in *TIK Report*, ETH Zurich, Computer Engineering and Networks Laboratory, 2001, vol. 103. doi: [10.3929/ethz-a-004284029](https://doi.org/10.3929/ethz-a-004284029).
- [50] B. Li, K. Tang, J. Li, and X. Yao, "Stochastic ranking algorithm for many-objective optimization based on multiple indicators," *IEEE Trans. Evol. Comput.*, vol. 20, no. 6, pp. 924–938, 2016. doi: [10.1109/TEVC.2016.2549267](https://doi.org/10.1109/TEVC.2016.2549267).
- [51] F. Ming, W. Gong, and L. Wang, "A two-stage evolutionary algorithm with balanced convergence and diversity for many-objective optimization," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 52, no. 10, pp. 6222–6234, 2022. doi: [10.1109/TSMC.2022.3143657](https://doi.org/10.1109/TSMC.2022.3143657).