An Improved Cuckoo Search Algorithm for the Dual Resource Constrained Flexible Job Shop Scheduling Problem (postprint)

Authors: Luo Haojia, Pan Dazhi

Date: 2022-04-07T15:01:57Z

Abstract

For the Dual-Resource Constrained Flexible Job Shop Scheduling Problem (DRCFJSP), a Cuckoo Algorithm with an improved decoding scheme is proposed to optimize the makespan. Since DRCFJSP requires consideration of both machine allocation and worker processing status, the traditional decoding method is improved to avoid conflicts between machine and worker processing times, while simultaneously utilizing the idle time of machines and workers during decoding. Under the core framework of the Cuckoo Algorithm, the cuckoo population is randomly divided into three subgroups, each of which independently conducts optimization using different Lévy flight strategies, and information exchange among subgroups is realized through a differential operator, thereby enhancing the global search capability of the algorithm and balancing its local search capability. Finally, experimental simulations are conducted through benchmark test cases and compared with other algorithms, which verifies the effectiveness and superiority of the improved Cuckoo Algorithm and the improved decoding method.

Full Text

Preamble

Vol. 39 No. 8
Application Research of Computers
ChinaXiv Cooperative Journal

Improved Cuckoo Algorithm for Flexible Job Shop Scheduling with Dual Resource Constraints

Luo Haojia , Pan Dazhi , †
(a. School of Mathematics & Information; b. Institute of Computing Methods

& Applications, China West Normal University, Nanchong, Sichuan 637009, China)

Abstract: This paper addresses the flexible job shop scheduling problem with dual resource constraints (DRCFJSP) and designs a cuckoo algorithm with an improved decoding scheme to minimize the maximum completion time. Since DRCFJSP requires consideration of both machine allocation and worker processing conditions, we improve the traditional decoding method to avoid conflicts in processing time between machines and workers while maximizing the utilization of idle time for both resources. Within the core framework of the cuckoo algorithm, the population is randomly divided into three subpopulations, each employing different Lévy flight strategies for independent optimization. Information exchange between subpopulations is achieved through a differential operator, which enhances global search capability while balancing local search ability. Experimental simulations using benchmark test cases and comparisons with other algorithms verify the effectiveness and superiority of the improved cuckoo algorithm and decoding method.

Keywords: flexible job shop scheduling; dual resource constraints; cuckoo algorithm; improved decoding method

0 Introduction

Scheduling plays a vital role in manufacturing and service industries. As a typical scheduling problem, the job shop scheduling problem has received extensive attention in the field of manufacturing systems. Recent years have witnessed numerous extended studies based on job shop scheduling, including fuzzy processing time scheduling [1,2], blocking problems [3~5], and multi-objective scheduling [6,7], all representing practical applications. The Dual Resource Constrained Flexible Job Shop Scheduling Problem (DRCFJSP) extends the Flexible Job Shop Scheduling Problem (FJSP) by adding worker constraints, requiring simultaneous consideration of worker and machine processing conditions. Consequently, DRCFJSP more closely reflects real production scenarios.

In recent years, several metaheuristic algorithms have been applied to DRCFJSP. Li et al. [8] proposed a branch-population genetic algorithm featuring a roulette wheel selection operator based on sector segmentation and an elite evolutionary operator, which effectively reduced computational complexity and avoided premature convergence. Zhang et al. [9] introduced a novel hybrid discrete particle swarm optimization algorithm for dual-resource constrained job shop scheduling with resource flexibility, incorporating an improved simulated annealing algorithm with variable neighborhood structure to enhance local search capability. Lei et al. [10] developed an effective variable neighborhood search that sequentially executed two neighborhood search procedures to generate new solutions for two subproblems, thereby strengthening the algorithm' s search ability. Zheng et al. [11] presented a knowledge-guided fruit

fly optimization algorithm with a new encoding scheme for DRCFJSP to minimize makespan, combining knowledge-guided search and smell-based search to balance global exploration and local exploitation.

The Cuckoo Search (CS) algorithm [12], proposed by Cambridge scholars Yang and Deb in 2009, simulates the brood parasitism behavior of cuckoos. Due to its minimal parameter requirements and strong global search capabilities, CS has been widely applied to continuous optimization, scheduling, and engineering optimization problems. Ouaar et al. [13] discretized the cuckoo algorithm without changing its framework to solve job shop scheduling problems. ALAA et al. [14] improved the Lévy flight in CS to intensify search around the population' s best individuals and applied it to flexible job shop scheduling. Majumdera et al. [15] proposed a Hybrid Discrete Cuckoo Search (HDCS) algorithm for parallel batch processing machines with unequal job ready times, combining CS with variable neighborhood search and improving the Lévy flight mechanism. Tang et al. [16] established a distributed flexible job shop scheduling model minimizing maximum completion time under realistic production conditions, improving CS encoding and initialization strategies to enhance initial solution quality while modifying search operations to accelerate convergence without compromising solution quality. Luo et al. [17] discretized the cuckoo algorithm and improved its decoding method for hybrid flow shop scheduling problems.

Current research on DRCFJSP remains limited, yet worker collaboration is indispensable in most factory operations. Therefore, this paper adds worker constraints to the FJSP problem, modeling workers operating machines to process jobs. We incorporate worker information into the encoding scheme and divide the population into three subpopulations with adaptive step size adjustment to improve population diversity. Additionally, we improve the decoding algorithm by integrating worker information and fully utilizing idle time to reduce final completion time. Simulation experiments using standard benchmark test sets and comparisons with other algorithms demonstrate the effectiveness and stability of the improved CS algorithm for DRCFJSP.

---

1 Dual Resource Constrained Flexible Job Shop Scheduling Model

The Dual Resource Constrained Flexible Job Shop Scheduling Problem (DRCFJSP) is described as follows: There are n jobs $\{J_1, J_2, ..., J_n\}$ to be processed on m machines $\{M_1, M_2, ..., M_m\}$. Each job J_i $(i = 1, 2, ..., n)$ has n_i operations $O_{i,1}, O_{i,2}, ..., O_{i,n_i}$. The processing time for each operation depends on both machine and worker assignments, with different combinations yielding different processing times.

The scheduling objective is to determine the processing sequence of all jobs and assign workers and machines to each operation to minimize the makespan

(maximum completion time). Symbol definitions for DRCFJSP are provided in Table 1.

Table 1 Symbol Variable Definition

Symbol	Definition
W	Total set of workers
M	Total set of machines
J	Total set of jobs
$O_{i,j}$	The j-th operation of job J_i
$SE_{i,j}$	Earliest start time of operation $O_{i,j}$
$CE_{i,j}$	Earliest completion time of operation $O_{i,j}$
$SL_{i,j}$	Latest start time of operation $O_{i,j}$
$CL_{i,j}$	Latest completion time of operation $O_{i,j}$
S_k	Start processing time of machine M_k
F_k	End processing time of machine M_k
PW_v	Preceding operation processed by the same worker v; if $O_{i,j}$ is the first operation, then $PW_v = \emptyset$
SW_v	Succeeding operation processed by the same worker v; if $O_{i,j}$ is the last operation, then $SW_v = \emptyset$
$T_{i,j,k,w}$	Processing time of operation $O_{i,j}$ on machine M_k by worker w
L	A sufficiently large positive number

Decision variables are defined as follows:

$$x_{i,j,k,w} = \begin{cases} 1 & \text{if operation } O_{i,j} \text{ is processed on machine } M_k \text{ by worker } w \\ 0 & \text{otherwise} \end{cases}$$

$$y_{g,h,i,j,k} = \begin{cases} 1 & \text{if operations } O_{g,h} \text{ and } O_{i,j} \text{ are both processed on machine } M_k \\ & \text{and } O_{g,h} \text{ precedes } O_{i,j} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{g,h,i,j,w} = \begin{cases} 1 & \text{if operations } O_{g,h} \text{ and } O_{i,j} \text{ are both processed by worker } w \\ & \text{and } O_{g,h} \text{ precedes } O_{i,j} \\ 0 & \text{otherwise} \end{cases}$$

Constraints:

Equation (1) ensures each operation is processed by exactly one worker on one machine:

$$\sum_{k\in M}\sum_{w\in W} x_{i,j,k,w} = 1, \quad \forall i \in J, \forall j \in J_i$$

Equation (2) defines the earliest completion time:

$$CE_{i,j} = SE_{i,j} + \sum_{k\in M}\sum_{w\in W} T_{i,j,k,w}x_{i,j,k,w}, \quad \forall i \in J, \forall j \in J_i$$

Equation (3) defines the latest completion time:

$$CL_{i,j} = SL_{i,j} + \sum_{k\in M}\sum_{w\in W} T_{i,j,k,w}x_{i,j,k,w}, \quad \forall i \in J, \forall j \in J_i$$

Equation (4) enforces precedence constraints for operations within the same job:

$$CE_{i,j} - SE_{i,j+1} \leq 0, \quad \forall i \in J, \forall j \in J_i - \{1, 2, ..., n_i - 1\}$$

Equation (5) ensures each machine processes at most one operation at any time:

$$CE_{g,h} - SE_{i,j} + L(1 - y_{g,h,i,j,k}) \geq \sum_{w\in W} T_{i,j,k,w}x_{i,j,k,w}, \quad \forall g, i \in J, \forall h \in J_g, \forall j \in J_i, \forall k \in M$$

Equation (6) ensures each worker processes at most one operation at any time:

$$CE_{g,h} - SE_{i,j} + L(1 - z_{g,h,i,j,w}) \geq \sum_{k\in M} T_{i,j,k,w}x_{i,j,k,w}, \quad \forall g, i \in J, \forall h \in J_g, \forall j \in J_i, \forall w \in W$$

Equation (7) states all machines become available starting from time 0:

$$S_k \geq 0, \quad \forall k \in M$$

Equation (8) ensures a machine must be idle when processing an operation:

$$S_k - SE_{i,j} + L(1 - x_{i,j,k,w}) \geq 0, \quad \forall i \in J, \forall j \in J_i, \forall k \in M, \forall w \in W$$

Equation (9) states machines cannot stop until processing is completed:

$$C_{i,j,k,w} \cdot x_{i,j,k,w} \leq F_k, \quad \forall i \in J, \forall j \in J_i, \forall k \in M, \forall w \in W$$

Equation (10) calculates the earliest start time of an operation:

$$SE_{i,j} = \max(CE_{i,j-1}, S_k, SW_v), \quad \forall i \in J, \forall j \in J_i, \forall k \in M, \forall w \in W$$

Equation (11) calculates the latest completion time of an operation:

$$CL_{i,j} = \max(SL_{i,j-1}, EM_k, EW_v), \quad \forall i \in J, \forall j \in J_i, \forall k \in M, \forall w \in W$$

Table 2 presents a DRCFJSP instance consisting of three jobs, three machines, and two workers. The table shows the processing time required for each operation on a given machine by a specific worker, where "-" indicates that the corresponding worker cannot operate that machine for the operation. For example, operation $O_{1,1}$ cannot be processed by worker W_1 on machine M_1, but can be processed by worker W_2 on M_1 with a processing time of 1.

2.1 Cuckoo Algorithm

The cuckoo algorithm is a novel heuristic search algorithm based on two update strategies: (1) searching for host nests through Lévy flight mechanisms, and (2) replacing discovered nests through preference-based random walk. In the cuckoo algorithm, each nest represents a feasible solution. Under ideal conditions, the position update formula for a cuckoo is:

$$X_i^{t+1} = X_i^t + \alpha \oplus \text{Lévy}(\beta)$$

where X_i^t represents the position of the i-th nest at generation t, α is the step size scaling factor, and $\oplus$ denotes point-to-point multiplication. The random search path Lévy(β) follows a Lévy distribution.

For computational convenience, literature [12] uses Equation (13) to generate Lévy random numbers:

$$\text{Lévy}(\beta) = \frac{\phi \times \mu}{|\nu|^{1/\beta}}$$

where μ and ν follow standard normal distributions, and ϕ is calculated as:

$$\phi = \left(\frac{\Gamma(1+\beta) \times \sin(\pi\beta/2)}{\Gamma((1+\beta)/2) \times \beta \times 2^{(\beta-1)/2}} \right)^{1/\beta}$$

The parameter β is typically set to 1.5. Combining these formulas yields the cuckoo position update equation:

$$X_i^{t+1} = X_i^t + \alpha \times \phi \times \mu \times (X_i^t - X_{\text{best}}^t)$$

After discarding partial solutions with discovery probability P_a, preference-based random walk generates an equal number of new solutions:

$$X_i^{t+1} = X_i^t + \gamma \times (X_{r1}^t - X_{r2}^t)$$

where γ is a scaling factor following a uniform distribution, and X_{r1}^t, X_{r2}^t represent two randomly selected nests from generation t.

2.2 Algorithm Encoding

Since the standard cuckoo algorithm solves continuous optimization problems, it cannot be directly applied to job shop scheduling. This paper introduces the Smallest Position Value (SPV) rule to establish a mapping between individuals and actual schedules. As shown in Figure 1, SPV sorts the original vector in ascending order to obtain a new vector; the positions of components in the original vector corresponding to the sorted sequence form the integer encoding sequence.

During encoding, we first randomly generate an Operation Sequence (OS), then allocate machines and workers based on this sequence. The complete encoding comprises three layers: the first layer is the Operation Sequence (OS), the second is the Machine Assignment sequence (MA), and the third is the Worker Assignment sequence (WA). This encoding, derived from the Table 1 instance, represents a schedule where three jobs are processed by two workers on three machines. Job 1 contains 2 operations, Job 2 contains 2 operations, and Job 3 contains 3 operations, with the processing order: $O_{2,1} \rightarrow O_{2,2} \rightarrow O_{3,1} \rightarrow O_{3,2} \rightarrow O_{3,3} \rightarrow O_{1,1} \rightarrow O_{1,2}$. From the complete sequence, operation $O_{3,1}$ is processed by worker 2 on machine 3, operation $O_{2,1}$ by worker 2 on machine 3, and operation $O_{3,2}$ by worker 1 on machine 2, with remaining operations following similarly.

2.3 Improved Decoding Algorithm

Unlike single-resource constrained FJSP, DRCFJSP is constrained not only by the completion time of preceding operations and the earliest available machine time but also by the earliest available worker time. This paper improves the insertion-based decoding scheme to enable proactive decoding for both machines and workers simultaneously. The core idea is to effectively utilize idle time generated during machine and worker processing by proactively scheduling them into appropriate idle intervals, thereby reducing overall makespan.

Let $ST_{i,j}$ denote the start time of operation $O_{i,j}$, $ET_{i,j}$ its completion time, $ET_{i,j-1}$ the completion time of its immediate predecessor, s_k the start time of machine M_k, and $T_{i,j,k,w}$ the processing time required for operation $O_{i,j}$ on machine M_k by worker w. During decoding, we must first check the availability of the selected machine and worker, then determine whether idle time exists and satisfies insertion conditions. When both machine and worker have idle time, three possible scenarios may occur:

Case 1: As shown in Figure 2, the idle times of machine and worker do not overlap, making insertion impossible.

Case 2: As shown in Figure 3, while machine and worker have overlapping idle intervals, the overlapping duration T_a is less than the required processing time $T_{i,j,k,w}$, preventing insertion.

Case 3: As shown in Figure 4, machine and worker have overlapping idle intervals with duration $T_a \geq T_{i,j,k,w}$, allowing insertion.

Based on all possible decoding scenarios, Algorithm 1 presents the pseudocode for the improved decoding method.

Algorithm 1: Pseudocode for Improved Decoding Method

Input: Total number of operations TP, machine and worker processing capability information
Output: Operation start times $ST_{i,j}$ and completion times $ET_{i,j}$, machine processing timeline $[SM_k, EM_k]$, worker processing timeline $[SW_s, EW_s]$

1. For each operation $O_{i,j}$ (where $tp = 1$ to TP):
2. Initialize insertion flag: $flag = 0$
3. Obtain available machines and workers k, w and processing time $T_{i,j,k,w}$
4. Retrieve machine timeline $[SM_k, EM_k]$ and worker timeline $[SW_s, EW_s]$, along with corresponding idle intervals $[ASM_k, AEM_k]$ and $[ASW_s, AEW_s]$
5. If $j = 1$ (first operation of job):
6. `If both machine and worker have not started processing:`
7. `  Set $ST_{i,j} = \max(ASM_k, ASW_s)$`
8. `  $ET_{i,j} = ST_{i,j} + T_{i,j,k,w}$`
9. `  Set $flag = 1$`
10. Else:
11. `Check insertion conditions for idle intervals`
12. `If machine and worker both have idle time:`
13. `  For each idle interval:`
14. `    If overlapping duration $T_a \geq T_{i,j,k,w}$:`
15. `      Set $ST_{i,j} = \max(ASM_k, ASW_s)$`

16. `$ET_{i,j} = ST_{i,j} + T_{i,j,k,w}$`
17. `Set $flag = 1$; break`
18. End If
19. Else:
20. $ET_{i,j-1}$ = completion time of preceding operation
21. If machine has idle time but worker does not:
22. `Check insertion conditions`
23. `If $ET_{i,j-1} \leq AEM_k$ and $T_a \geq T_{i,j,k,w}$:`
24. `Set $ST_{i,j} = \max(ET_{i,j-1}, ASM_k)$`
25. `$ET_{i,j} = ST_{i,j} + T_{i,j,k,w}$`
26. `Set $flag = 1$`
27. Else If worker has idle time but machine does not:
28. `Similar checking process`
29. Else If both have idle time:
30. `Check overlapping intervals for insertion feasibility`
31. If $flag = 0$ (no insertion possible):
32. Set $ST_{i,j} = \max(ET_{i,j-1}, EM_k, EW_s)$
33. $ET_{i,j} = ST_{i,j} + T_{i,j,k,w}$
34. Update operation's earliest completion time and machine/worker timelines
35. End For

2.4.1 Improved Lévy Flight

In standard cuckoo search, longer Lévy flight steps yield lower search precision suitable for global exploration, while shorter steps provide higher precision and

stronger local search capability. Since the standard Lévy flight uses a fixed step size factor lacking adaptivity, it may result in slow search speed and susceptibility to local optima. To address this, we employ two methods to improve the step size factor by randomly dividing the cuckoo population into three subpopulations. Each subpopulation independently updates using either fixed-step Lévy flight or two improved adaptive-step Lévy flight strategies. The step size factor automatically adjusts based on the population' s search status at different stages, balancing global optimization speed and search precision.

The first improvement method, shown in Equation (18), adaptively adjusts the step size factor α based on the current nest' s position relative to the global best nest, enabling more refined search near the optimal solution to help locate the global optimum:

$$\alpha = \alpha_0 \times |X_i^t - X_{\text{best}}^t|$$

where α_0 is typically set to 0.01, X_i^t represents the position of the i-th nest at generation t, and X_{best}^t is the current global best nest position.

The second improvement method, shown in Equation (19), adopts the approach from literature [18] and controls the step size factor α through algorithm iteration count. During early iterations, α is large, providing extensive search range and reducing the risk of local optima. In later stages, α adaptively decreases, improving search precision for better optimal solution identification:

$$\alpha = \alpha_{\max} \times \cos\left(\frac{t}{t_{\max}} \times \gamma\right)$$

where t is the current iteration number, $t_{\max}$ is the total iteration count, and γ is a random step factor in $[0.05, 0.05]$.

2.4.2 Information Exchange

Since the three subpopulations conduct optimization independently, information exchange between them every k generations helps improve efficiency and maintain population diversity. The exchange principle involves communicating suboptimal individuals with the best individuals to guide weaker individuals toward the global optimum. We introduce the DE/best/1 mutation strategy from differential evolution algorithms for subpopulation differentiation, calculated as:

$$V_{i,g} = X_{\text{best},g} + F \times (X_{r1,g} - X_{r2,g})$$

where $X_{\text{best},g}$ represents the current global best individual, and $X_{r1,g}$, $X_{r2,g}$ denote two relatively inferior individuals in subpopulation g.

The improved cuckoo algorithm is presented in Algorithm 2.

Algorithm 2: Pseudocode for Improved Cuckoo Algorithm

Input: Iteration count $iter$, number of nests n, maximum iterations $iter_{\max}$, discovery probability P_a
Output: Current best nest information

1. For $iter = 1$ to $iter_{\max}$:
2. For each subpopulation (1 to 3):
3. `Update subpopulation using Equations (12), (18), and (19)`
4. `Each subpopulation updates via Lévy flight with different step factors`
5. `Evaluate nests and discard inferior ones with probability $P_a$, generating new nests v`
6. End For
7. Evaluate current generation and update global best nest
8. If $iter \mod k = 0$:
9. `Apply differential operator (Equation 20) for subpopulation information exchange`
10. Decode using improved decoding algorithm and update global best
11. End For

2.5 Algorithm Flow

Step 1: Initialize parameters: cuckoo population size n, maximum iterations $iter_{\max}$, discovery probability P_a.

Step 2: Initialize population. Convert nest information into feasible scheduling sequences using mapping rules. Apply improved decoding to obtain makespan, retain the current best nest, and randomly divide the population into 3 subpopulations.

Step 3: Update each subpopulation using standard Lévy flight and two improved Lévy flight formulas. Decode to obtain makespan and update the current best nest. Perform preference-based random walk via Equation (17) to eliminate poor solutions and generate equal numbers of new solutions.

Step 4: Every k iterations, introduce the differential operator for information exchange among the three subpopulations. Decode using the improved decoding algorithm and update the current best nest.

Step 5: Output the global optimal solution when maximum iterations are reached; otherwise, return to Step 3.

3.1 Experimental Environment and Test Instances

This study employs Matlab R2018a programming on an Intel(R) Xeon(R) CPU E5-@3.30GHz with 8GB RAM. Given limited research on DRCFJSP and the lack of standard benchmarks for comparative evaluation, we select the MK1-MK10 instances from the Brandimarte [19] benchmark set to test algorithm performance. Worker allocation follows the simulation method from literature [11], with specific worker-machine assignments detailed in Table 3.

Table 3 Worker and Machine Allocation

Instance	Allocation
MK1-2	$\{M_1, M_3, M_5\}, \{M_2, M_4, M_5\}$
MK3-4	$\{M_1, M_2, M_4\}, \{M_3, M_5\}$
MK6,10	$\{M_1, M_3, M_4, M_6\}, \{M_2, M_4, M_5\}$
MK7,11	$\{M_2, M_3, M_4\}, \{M_1, M_5, M_6, M_7\}$
MK8,9,12,13	$\{M_1, M_5, M_6, M_7, M_8\}, \{M_6, M_5, M_8\}$
MK5	$\{M_1, M_3, M_4, M_5\}, \{M_2, M_4\}$
MK14	$\{M_1, M_2, M_3\}, \{M_2, M_4\}$

3.2 Results and Analysis

Since algorithm parameters significantly impact performance and runtime, we employ the orthogonal experimental method from literature [20] for parameter tuning. Figure 5 shows the average trend over 10 runs of our algorithm solving MK01 under different parameter settings.

Figure 5(a) illustrates the impact of maximum iterations $iter_{\max}$. Small iteration limits may force premature termination before convergence, degrading solution quality. As iterations increase, solution quality improves, but beyond a certain point, further increases do not significantly improve optimal solution frequency while raising computational complexity. Therefore, we set $iter_{\max} = 200$ to ensure efficiency. Figure 5(b) shows that a population size around 50 yields excellent solutions, so we set $n = 50$. Figure 5(c) displays the effect of discovery probability P_a, with optimal solutions obtained around $P_a = 0.25$. Larger values gradually degrade solution quality, leading to our final setting of $P_a = 0.25$.

To validate our algorithm' s effectiveness, we compare it with alternative algorithms. Each instance runs 20 consecutive times, with results recorded and analyzed in Table 4. Here, VNS [10], $KGFOA$ [11], and $MBSA$ [21] represent optimal values from literature for DRCFJSP; ICS denotes the improved cuckoo algorithm with standard decoding; $CSND$ represents standard cuckoo algorithm with improved decoding; and $ICSND$ is our proposed improved cuckoo algorithm with improved decoding.

Table 4 reveals that with only algorithmic improvements, good results are easily obtained for small-scale problems. However, as problem scale increases, idle times for machines and workers grow, and standard decoding cannot effectively utilize these intervals, making algorithmic improvements alone insufficient. $CSND$ benefits from cuckoo algorithm's inherent superiority in balancing global and local search, with improved decoding further reducing makespan and enhancing efficiency. Our $ICSND$ not only improves the cuckoo algorithm to strengthen search capability but also adjusts decoding to effectively utilize idle time, substantially shortening makespan. Compared with other algorithms, $ICSND$ consistently produces superior solutions across most instances.

Figures 6 and 7 present Gantt charts for MK1 and MK2 solutions obtained by our algorithm, where the x-axis represents processing time, y-axis shows machines, and each box contains job number and worker number.

To better evaluate performance, we introduce the Mean Relative Percentage Deviation ($MRPD$) and Average Relative Percentage Deviation ($ARPD$) metrics from literature [10]:

$$MRPD = \frac{C_{\max}^{\text{best}} - C_{\max}^{\text{low}}}{C_{\max}^{\text{low}}} \times 100$$

$$ARPD = \frac{1}{s} \sum_{i=1}^{s} \frac{C_{\max}^{\text{best}} - C_{\max}^{\text{low}}}{C_{\max}^{\text{low}}} \times 100$$

where $C_{\max}^{\text{best}}$ is each algorithm' s best solution and $C_{\max}^{\text{low}}$ is our algorithm' s best solution.

Table 5 demonstrates that algorithms with improved decoding outperform standard decoding versions. $ICSND$ surpasses $CSND$ in both solution quality and best solution attainment, while the improved cuckoo algorithm overall outperforms standard CS, exhibiting stronger search capability. Table 6 compares $ICSND$ with three other algorithms, showing $ICSND$ achieves minimum percentage deviation of 0 for all instances with smaller average deviations, indicating more stable and higher-quality solutions. Other algorithms generally exhibit poorer solution quality and larger average deviations. Thus, $ICSND$ demonstrates superior solving capability, stability, and solution quality for DRCFJSP.

4 Conclusion

This paper improves the cuckoo algorithm while maintaining its core framework by dividing the population into three subpopulations with different Lévy flight strategies and introducing a differential operator for inter-subpopulation information exchange, significantly enhancing search capability. An improved decoding scheme is designed to avoid processing time conflicts between machines and workers while proactively identifying insertable idle intervals, substantially reducing overall makespan. Benchmark testing and comparative analysis validate the effectiveness and stability of the improved cuckoo algorithm and decoding method. Future research will incorporate dynamic production factors such as machine failures, maintenance, urgent order insertion, and order cancellation.

References

[1] Chen Kejia, Duan Ruiming, Liu Biyu, et al. A multi-objective model for fuzzy replacement flow shop scheduling [J]. Operations Research and Management, 2021, 30 (08): 28-36.

[2] Lin J. Backtracking search based hyper-heuristic for the flexible job-shop scheduling problem with fuzzy processing time [J]. Engineering Applications of Artificial Intelligence, 2019, 77: 186-196.

[3] Xuan Hua, Wang Jing, Li Bing, et al. Research on optimal scheduling of blocked mixed flow shop [J]. Control Engineering, 2020, 27 (08): 1346-1350.

[4] Shao Z, Shao W, Pi D. Effective constructive heuristic and iterated greedy algorithm for distributed mixed blocking permutation flow-shop scheduling problem [J]. Knowledge-Based Systems, 2021, 221 (5): 107960.

[5] Xuan Hua, Wang Jing, Zhang Huixian, et al. Hybrid Genetic Algorithm for HFSP with Machine Availability Constraints [J]. Computer Applications and Software, 2021, 38 (06): 176-181.

[6] Zhang Hongliang, Ding Renman, Xu Gongjie. Multi-objective Flexible Job Shop Energy Saving Scheduling Considering Interval Working Hours [J/OL]. Journal of System Simulation: 1-13 (2021-08-17) [2021-09-10]. https://doi.org/10.16182/j.issn1004731x.joss.21-0395.

[7] Wang H, Sheng B, Lu Q, et al. A novel multi-objective optimization algorithm for the integrated scheduling of flexible job shops considering preventive maintenance activities and transportation processes [J]. Soft Computing, 2021, 25 (4): 1-27.

[8] Li J, Yuan H. A Hybrid Genetic Algorithm for Dual-Resource Constrained Job Shop Scheduling Problem [J]. Computers & Industrial Engineering, 2016, 102: 113-131.

[9] Jing Z, Wang W, Xu X. A hybrid discrete particle swarm optimization for dual-resource constrained job shop scheduling with resource flexibility [J]. Journal of Intelligent Manufacturing, 2017, 28 (8): 1961-1972.

[10] Lei D, Guo X. Variable neighbourhood search for dual-resource constrained flexible job shop scheduling [J]. International Journal of Production Research, 2014, 52 (9): 2519-2529.

[11] Zheng X L, Wang L. A knowledge-guided fruit fly optimization algorithm for dual resource constrained flexible job-shop scheduling problem [J]. International Journal of Production Research, 2016, 54 (18): 5450-5465.

[12] Yang X S, DEB S. Cuckoo search via Lévy flight [C]// Proceedings of World Congress on Nature & Biologically Inspired Computing. India: IEEE Publications, 2009: 210-214.

[13] Ouaarab A, Ahiod B, Yang X S. Discrete Cuckoo Search Applied to Job Shop Scheduling Problem [M]. Recent Advances in Swarm Intelligence and Evolutionary Computation. Springer International Publishing, 2015: 121-137.

[14] Alaa S, Alobaidi A. Two Improved Cuckoo Search Algorithms for Solving The Flexible Job-Shop Scheduling Problem [J]. International Journal on Perceptive and Cognitive Computing, 2016, 2 (2): 25-31.

[15] Majumdera A, Lahaa D, Suganthan P N. A hybrid cuckoo search algorithm in parallel batch processing machines with unequal job ready times [J]. Computers & Industrial Engineering, 2018, 124: 65-76.

[16] Tang Hongtao, Liu Jiayi. Improved Cuckoo Algorithm for Distributed Flexible Flow Shop Scheduling Problem Considering Transportation Time [J]. Operations Research and Management, 2021, 30 (11): 76-83.

[17] Luo Hanming, Luo Tianhong, Wu Xiaodong, et al. Discrete Cuckoo Algorithm for Solving Hybrid Flow Shop Scheduling Problem [J]. Computer Engineering and Applications, 2020, 56 (22): 264-271.

[18] Shi Wenzhang, Han Wei, Dai Ruiwen. Cuckoo algorithm under simulated annealing to solve job shop scheduling problems [J]. Computer Engineering and Applications, 2017, 53 (17): 249-253, 259.

[19] Brandimarte P. Routing and scheduling in a flexible job shop by tabu search [J]. Annals of Operations research, 1993, 41 (3): 157-183.

[20] Wang Wenyan, Xu Zhenhao, Gu Xingsheng. Discrete water wave optimization algorithm for batch flow scheduling problem in mixed flow workshop with batch processing [J]. Journal of East China University of Science and Technology: Natural Science, 2021, 47 (05): 598-608.

[21] Gnanavelbabu A, Caldeira R H, Vaidyanathan T. A simulation-based modified backtracking search algorithm for multi-objective stochastic flexible job shop scheduling problem with worker flexibility [J]. Applied Soft Computing, 2021, 2021 (113): 107960.