A Novel Multi-objective Lion Swarm Intelligent Optimization Based on Cloud Model for Manufacturing Scheduling Problem

Shengwei Dong and Mingyan Jiang
School of Information Science and Engineering, Shandong University, Qingdao, China
Email: jiangmingyan@sdu.edu.cn

Abstract. With the continuous upgrading of industrial manufacturing, various artificial intelligence technologies have gradually been applied to the field of industrial production, including swarm intelligence optimization algorithms. Aiming at the flow shop scheduling problem (FSP) and job shop scheduling problem (JSP) in industrial production, which are NP-hard problems, we use a multi-objective optimization method to solve them. We proposed a novel multi-objective optimization named multi-objective lion swarm optimization based on cloud model mutation (CMOLSO). This new optimization algorithm, which is based on the Lion Swarm Optimization (LSO), introduces the concept of cloud model and cloud generator algorithm. The introduction of the cloud model mechanism can expand the search range of CMOLSO in high-dimensional multi-objective problems, make it avoid falling into local extremes, and improve its optimization accuracy. Compared with the traditional multi-objective optimization, the new algorithm CMOLSO achieves better performance, and it can effectively solve the scheduling problems in practice.

Keywords. Multi-objective optimization; lion swarm optimization; cloud model; flow-shop scheduling problem; job-shop scheduling problem.

1. Introduction
The study of the scheduling problem mainly concentrates on the allocation of resources. For different tasks, it formulates corresponding optimization goals, and finally finds the optimal or approximately optimal solution. It requires a scientific and reasonable allocation of limited human resources, material resources and time under the premise of ensuring a certain optimal performance, so that the production process can run efficiently and effectively [1].

Scheduling problems widely exist in manufacturing enterprises. A well-designed production scheduling plan can help improve production efficiency and help enterprises occupy an active position in competitive market [2]. The job shop scheduling problem (JSP) and its simplified model -- flow shop scheduling problem (FSP) are all hot topics in combinatorial optimization. The JSP has been proved to be a NP-hard problem. Its optimal solution of a JSP is difficult to find and takes much time and computing resources with traversal method [3].

At present, based on FSP and JSP, researchers have further proposed various scheduling optimization models for specific fields and problems, including flexible job shop scheduling [4], no-wait flow shop scheduling [5], rescheduling method for new job insertion [6], energy-efficient flow shop scheduling [7], real-time production scheduling [8], and the multi-objective and multi-constrained scheduling model [9].
There are two common means. One is to find the optimal solution by using the mathematical methods, such as integer programming, dynamic programming, backtracking pruning algorithm and branch and bound algorithm, etc. Another method is to obtain approximate solutions by using heuristic swarm intelligent optimization algorithms such as artificial fish swarm algorithm [10], artificial bee colony algorithm [11] and particle swarm algorithm, etc. The traditional mathematical methods have faced great difficulties in solving large-scale scheduling problems due to some reasons such as high computational complexity and low robustness. They all have high requirements on the constraints of the scheduling problems, and sometimes are difficult to obtain a satisfactory solution in the actual production.

Swarm intelligence optimization is a meta-heuristic algorithm. Compared with common mathematical methods, swarm intelligence algorithms have many advantages. They have better intelligence, parallelism, robustness, and good adaptability, strong global search capabilities and are simple and efficient [12].

The research on multi-objective optimization has important practical significance. Multi-objective optimization problems are very common in practice. In addition to the FSP and JSP, many other issues are all potential applications of multi-objective optimization.

Researchers have proposed a series of multi-objective optimization algorithms. In recent years, apart from the traditional classic algorithms such as NSGA and MOPSO, some new multi-objective optimizations have also been continuously proposed. Hui Wang proposed the hybrid multi-objective firefly algorithm (HMOFA) [13] based on firefly algorithm. Babalik proposed the multi-objective artificial algae algorithm (MOAA) [14]. Lai proposed the multi-objective artificial sheep algorithm (MOASA) [15]. Zarier proposed the Multi-Objective Grasshopper Optimization Algorithm (MOGOA) [16] by modified Grasshoppe rOptimization Algorithm (GOA). Mohamad proposed a modified MOPSO suitable for dynamic boundary search [17]. Mirjalili proposed the Dragonfly Algorithm (DA) [18] and the Salp Swarm Algorithm (SSA) [19].

In actual application, Zhang and Gan applied MOPSO on the optimization model of independent micro-grid aimed at economy and environmental protection [20]. Yang and Liu used a hybrid multi-objective gray wolf optimization algorithm to solve blocking FSP [21]. Piroozfard used an improved multi-objective genetic algorithm to solve the flexible JSP [22].

The paper describes the PFSP and its mathematical model in section 2. Then we introduce the JSP and its mathematical model in section 3. In section 4, we will introduce several problems about multi-objective optimization and propose the CMOLSO. In section 5, we will discuss the performance of the new algorithm CMOLSO and its application in solving FSP and JSP. We establish a multi-objective optimization mathematical model for them, and successfully solves them using the multi-objective lion swarm optimization.

2. Mathematical Model for Permutation FSP
FSP studies the processing of $n$ jobs which have to be performed on $m$ machines. Permutation flow shop scheduling problem (PFSP) is a special FSP which is most studied. PFSP stipulates that the processing sequence on $m$ machines of every job is the same, and the processing sequence of the jobs on each machine is also the same.

2.1. Description for PFSP
PFSP is formulated as follows: Let us have $n$ jobs and $m$ machines for processing. Let $\pi^i (i=1,2,...)$ presents one of jobs. $\pi^i = \{\pi_1, \pi_2, ..., \pi_n\}$ is an order of all jobs. $\Pi$ is an aggregate of all orders. $\pi^i \in \Pi$ $\Pi$ is the full permutation of $n$ jobs. $p(\pi_i, k)$ is the processing time of the job $\pi_i$ processed on the machine with the number $k$. $C(\pi_i, k)$ represents the finish time of job $\pi_i$ processed on machine $k$.

To ensure generality, each job is moved from machine 1 to $m$. Then we can deduce that:

$$C(\pi_i, 1) = p(\pi_i, 1)$$ (1)
C(\pi_i, 1) = C(\pi_{i-1}, 1) + p(\pi_i, 1), \quad i = 2, \ldots, n \tag{2}

C(\pi_i, k) = C(\pi_i, k - 1) + p(\pi_i, k), \quad k = 2, \ldots, m \tag{3}

C(\pi_i, k) = \max \{C(\pi_{i-1}, k), C(\pi_i, k - 1)\} + p(\pi_i, k), \quad i = 2, \ldots, n; k = 2, \ldots, m \tag{4}

We choose two scheduling objective functions. They are the max finish time and max delay time. The max finish time which is also called makespan is:

\[ C_{\text{max}}(\pi) = \max_{i=1, \ldots, n} C(\pi_i, m) \tag{5} \]

Its implication is the finish time of the last job processed.

For a job \( \pi_i \), the ideal processing is that it starts from the first machine and has no stopping time until the processing is completed on the last machine. In this case, the delay time of the job \( \pi^i \) is zero. Define the processing finish time of the job \( \pi^i \) in the ideal state as the expected processing completion time \( e(\pi_i) \):

\[ e(\pi_i) = C(\pi_i, 1) + \sum_{k=2}^{m} p(\pi_i, k) \tag{6} \]

Then, the delay time of the job \( \pi^i \) in the actual processing is:

\[ D(\pi_i) = C(\pi_i, m) - e(\pi_i) \tag{7} \]

The objective function \( D_{\text{max}}(\pi) \) named maximum delay time of PFSP is the maximum value among the delay times of all jobs.

\[ D_{\text{max}}(\pi) = \max_{i=1, \ldots, n} D(\pi_i) \tag{8} \]

Let \( C_{\text{max}}(\pi) \) be the objective function \( f_1 \) and \( D_{\text{max}}(\pi) \) be the objective function \( f_2 \). The optimization of PFSP can be described as determining one or more non-inferior solutions to minimize the multi-objective function:

\[ \min \{f_1(\pi'), f_2(\pi')\}, \quad \pi' \in \Pi \tag{9} \]

2.2. Solution Expression and LOV Rule

The individuals in the standard swarm intelligence optimization algorithm are usually represented by strings of real numbers. Therefore, we use the largest-order-value (LOV) rule to transform the real number string lion individual \( X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,n}\} \) into discrete processing job order \( \pi' = \{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,n}\} \).

**Algorithm 1. LOV algorithm**

Step1: \( X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,n}\} \) are labeled in descending order by the values \( x_{i,j} (j = 1, 2, \ldots, n) \) to get the intermediate sequence \( \varphi_i = \{\varphi_{i,1}, \varphi_{i,2}, \ldots, \varphi_{i,n}\} \). The labels should use integers from 1 to \( n \).

Step2: According to the formula \( \pi_{i, \varphi_j} = k \) \( k = 1, 2, \ldots, n \) to get final processing order \( \pi' \).
Table 1 takes individual \( X_i = \{1.36 \ 3.85 \ 2.55 \ 0.63 \ 2.68 \ 0.82\} \) as an example to show the rules of LOV conversion.

**Table 1. Example of conversion based on LOV.**

| Dimension | 1   | 2   | 3   | 4   | 5   | 6   |
|-----------|-----|-----|-----|-----|-----|-----|
| \( x_i \) | 1.36| 3.85| 2.55| 0.63| 2.68| 0.82|
| \( \phi_i \) | 4   | 1   | 3   | 6   | 2   | 5   |
| \( \pi'^{i} \) | 2   | 5   | 3   | 1   | 6   | 4   |

3. Mathematical Model for JSP

In JSP, the machine sequence of each job in its processing can be different, and the jobs sequence on different machines can also be different. It can be described as follows: In a processing system, there are \( n \) jobs and \( m \) processing machines. Each job needs to be processed by all machines without duplication. The machine sequence of every job and its processing time on each machine are known. The buffer capacity between machines is not limited [23].

3.1. Solution Expression and SOV Rule

Because the processing orders of the jobs are different, the solution representing the population individual needs to contain more information. And its coding is more complicated than PFSP. We use the SOV (smallest-order-value) rule to transform the individual \( X_i \) into discrete processing order \( \pi'^{i} \).

Define a population individual as a real number string of \( m \times n \) length.

**Algorithm 2. SOV algorithm**

Step1: \( X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,mn}\} \) are labeled in ascending order by the values \( x_{i,j} \) \((j = 1, 2, \ldots, m \times n)\) to get the intermediate sequence \( \phi_i = \{\phi_{i,1}, \phi_{i,2}, \ldots, \phi_{i,mn}\} \). The labels should use integers from 1 to \( m \times n \).

Step2: According to the formula \( \pi_{i,k} = \left\lceil\frac{(\phi_{i,k} - 1) m^{-1}}{n}\right\rceil + 1 \) \((k = 1, 2, \ldots, n \times m)\) to get final processing order \( \pi'^{i} \).

Table 2 takes individual \( X_i = \{1.36 \ 3.85 \ 2.55 \ 0.63 \ 2.68 \ 0.82\} \) as an example:

**Table 2. Example of conversion based on SOV.**

| Dimension | 1   | 2   | 3   | 4   | 5   | 6   |
|-----------|-----|-----|-----|-----|-----|-----|
| \( x_i \) | 1.36| 3.85| 2.55| 0.63| 2.68| 0.82|
| \( \phi_i \) | 3   | 6   | 4   | 1   | 5   | 2   |
| \( \pi'^{i} \) | 2   | 3   | 2   | 1   | 3   | 1   |

3.2. Semi-active Decoding

Define the machine sequence matrix as \( M \) and the processing time matrix as \( T \), both of which are \( m \times n \) dimensional matrices. Among them, the element \( M_{i,j} \) represents the machine number of the \( j \)-th process for the job with number \( i \), and the element \( T_{i,j} \) is the processing time required for the \( j \)-th process of the job \( i \). \( C(\pi, k) \) is the finish time when the job \( \pi_i \) is processed completely on machine \( k \). Now there is a job order \( \pi'^{i} = \{\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,mn}\} \). The steps of semi-active decoding are as follows.
Algorithm 3. Semi-active Decoding

Step1 Define variable vector \( k(\omega) = 1 \), \( \omega = 1, 2, ..., n \); \( g(\omega) = 1 \), \( \omega = 1, 2, ..., m \)

\( j = 1 \)

Step2 Get the current process \( k(\pi_j) \) of the current processed job \( \pi_j \) and the corresponding machine number \( M_{s_j, k(\pi_j)} \).

Step3 Compare the finish time \( t_1 \) of the previous process \( k(\pi_j) - 1 \) of the job \( \pi_j \) with finish time \( t_2 \) of the previous mission \( g(M_{s_j, k(\pi_j)}) - 1 \) of the machine \( M_{s_j, k(\pi_j)} \).

Step4 \( C(\pi_j, M_{s_j, k(\pi_j)}) = \max(t_1, t_2) + T(\pi_j, k(\pi_j)) \)

Step5 let \( k(\pi_j) = k(\pi_j) + 1 \), \( g(M_{s_j, k(\pi_j)}) = g(M_{s_j, k(\pi_j)}) + 1 \)

Step6 let \( j = j + 1 \). if \( j \leq n \times m \), jump to step 2. Otherwise, finish algorithm.

4. Multi-objective Lion Swarm Optimization Based on Cloud Model Mutation Mechanism

4.1. Basic Lion Swarm Optimization

The Lion Swarm Optimization (LSO) has good co-evolution capabilities [24] and the evolution mechanism is more complex and diversified. So it has good optimization capabilities.

LSO divides the lion swarm into three parts, which are the king of lions, female lions and young lions. The lion king follows the global optimal position and leads the movement and optimization of the entire lion swarm. Its location update method is [25]:

\[
x^{k+1}_i = g^k \left( 1 + \gamma \left\| p^k_i - g^k \right\| \right)
\]

The female lions cooperate with each other to track and hunt, and they follow

\[
x^{k+1}_i = \frac{p^k_i + p^k_{\alpha}}{2} (1 + \alpha_i \gamma)
\]

to move. The young lions learn from the lion king and female lions, and is responsible for global walking exploration to prevent the population from falling into a local optimum. There are three methods for a young lion to update its position.

\[
x^{k+1}_i = \begin{cases} 
\frac{g^k + p^k_{\alpha}}{2} (1 + \alpha_i \gamma), & q \leq \frac{1}{3} \\
\frac{p^k_{\alpha} + p^k_{\gamma}}{2} (1 + \alpha_i \gamma), & \frac{1}{3} \leq q \leq \frac{2}{3} \\
\frac{g^k + p^k_{\gamma}}{2} (1 + \alpha_i \gamma), & \frac{2}{3} \leq q < 1 
\end{cases}
\]

In above formulas, \( p^k_{\gamma} \) is the historically optimum position of the lion \( x_i \) in \( k \)-th generation, \( g^k \) is the optimum position. \( p^k_{\alpha} \) is the historically optimum position of young lions in \( k \)-th generation.

\[
\overline{g}^k = L_{\text{mean}} + H_{\text{mean}} - g^k
\]

\( L_{\text{mean}} \) and \( H_{\text{mean}} \) are the mean of minimums and the mean of maximums of the ranges of all dimensions. \( \gamma \) is a random number obeying normal distribution \( N(0, 1) \). \( \alpha_j \) and \( \alpha_c \) are perturbation factors. They are factors that dynamically decrease with the number of iterations.
4.2. Multi-objective Lion Swarm Optimization

In order to adapt to some unique problems in multi-objective optimization, it needs some certain conversion strategies from LSO to multi-objective lion swarm optimization (MOLSO) algorithm.

4.2.1. Evaluation Criteria for Good Solution. In the MOLSO, each individual $x_i$ corresponds to multiple objective function values. We consider the Pareto solution or non-dominated solution as a better solution. Also we consider an individual which has the Pareto solution or non-dominated solution as the better one in two individuals.

Supposing a minimum multi-objective optimization function contains $m$ objective functions.

$$\min F(x) = (f_1(x), f_2(x), \ldots, f_m(x))^T, x \in \Omega$$

(14)

$F(x): \Omega \rightarrow \mathbb{R}^m$ represents the mapping from the domain $\Omega$ to the $m$-dimensional target space $\mathbb{R}^m$.

For two different vectors $x_1, x_2 \in \Omega$, there are $F(x_1), F(x_2)$. If the following two conditions are met at the same time,

$$\begin{align*}
f_1(x_1) & \leq f_1(x_2) \\
f_2(x_1) & \leq f_2(x_2) \\
& \vdots \\
f_m(x_1) & \leq f_m(x_2)
\end{align*}$$

(15)

$$\exists i = 1, 2, \ldots, m \quad f_i(x_1) < f_i(x_2)$$

(16)

Then we think $x_1$ dominate $x_2$. Otherwise not. If $x_1$ dominate $x_2$, $x_1$ is better than $x_2$, and $x_1$ is the Pareto solution. On the contrary, if $x_2$ dominate $x_1$, $x_2$ is better than $x_1$, and $x_2$ is the Pareto solution. If $x_1$ can not dominate $x_2$, and $x_2$ can not dominate $x_1$, then $x_1$ and $x_2$ are all Pareto solutions. Usually in the iterative update of the population, we will use the Pareto solution instead of the original solution as the optimal solution. If there is no dominance relationship between the new solution and the original solution, that is, they are both non-inferior solutions, then we can choose a solution randomly or keep the original solution as needed.

4.2.2. Solution Set of Multi-objective Optimization. $X^n$ is the solution set of $F(x)$. $x_i$ is a solution of the set. If there is no other solution in the set $X^n$ that can dominate $x_i$, $x_i$ is called the Pareto optimal solution. All Pareto optimal solution in $X^n$ make up a subset called Pareto optimal solution set, also called Pareto front.

4.2.3. External File Mechanism. The optimization result of multi-objective function is an optimal solution set. To ensure that the good solutions found in the previous optimization process are not lost during the iteration process, a limited size external file has been created before the optimization begins to save these good solutions. The new Pareto optimal solution is added to the external file after each population update.

In order to ensure that the calculation is not too redundant, external files are usually set to a limited size and an elimination mechanism is established. This paper uses crowding degree as the elimination mechanism. We consider all the solutions in the external file as a solution set, and calculate the congestion value according to the Euclidean distance from each solution to the nearest several solutions, and sort according to the congestion value. The smaller the sequence value of a solution, the sparser the solutions around it, that is, the less crowded.
If the solution number exceeds the limit size $K$, the solutions in the upper part of the sequence are retained, and the redundant parts in the last part of the sequence are deleted. At the same time, we choose the solution with the highest sequence in the crowding degree ranking as the global optimal information to guide the direction of population optimization. In this way, it is possible to avoid the situation where the population individuals are too concentrated in a certain local area, but some parts of the Pareto front are not sufficiently searched, so that the distribution of the solution is more uniform.

4.3. Cloud Model
Cloud model theory [26] is a model of mutual mapping and transformation between a qualitative concept described in natural language and a quantitative concept described in numerical language [26]. It can synthetically describe randomness, fuzziness and their relationship [27]. The cloud droplet group generated by the cloud model obeys a certain regular random distribution, which embodies the basic principle of natural species evolution. Therefore, people in the field of evolutionary computing have also begun to pay attention to cloud models [28].

Let us give the definition. Suppose $\Omega$ is a quantitative domain that is accurately represented by numerical values, $C$ is a qualitative fuzzy concept on $\Omega$. For any quantitative value $x \in \Omega$, it is a random realization of the corresponding qualitative concept $C$, and $x$ has a stable accuracy $\eta(x)$ for $C$. Then the distribution of $x$ on $\Omega$ is called cloud, and it is recorded as cloud $C(x)$. Each $x$ is called a cloud drop. Use mathematical language to describe as [29]:

$$\eta(x) : \Omega \rightarrow [0,1], \quad x \in \Omega, \quad x \rightarrow \eta(x)$$ (17)

Expectation $Ex$, entropy $En$, and hyper entropy $He$ are three parameters of the cloud model: $Ex$ represents the central of cloud drop distribution [30]. $En$ represents the randomness of cloud drops. The smaller the $En$, the smaller the range of cloud drops. $He$ is the uncertainty of entropy $En$, that is, the entropy of entropy. The smaller the $He$ is, the thinner the cloud thickness will be.

The generation and update of cloud drops are completed by the forward cloud generator. The cloud droplets can satisfy the normal cloud distribution when the digital features of cloud $Ex$, $En$ and $He$ was known.

Figure 1. Cloud model diagram with $Ex=0$, $En=2$, $He=0.2$. The horizontal axis represents the independent variable in the universe of discourse, and the vertical axis represents the degree of certainty or membership.
Algorithm 4. One-dimensional Cloud Generator

Step 1: Create a normal random number $E_n'$ with the expectation $E_n$ and the standard deviation $H_e$ according to the numerical characteristics of the cloud $(E_x, E_n, H_e)$. The calculation formula is $E_n' = \text{normend}(E_n, H_e)$. normend is the normal random number generating function.

Step 2: Using the following calculation formula, generate a normal distributed random number $x$ with the expectation $E_x$ and the standard deviation $E_n'$. $x$ is an offspring cloud drop. $x = \text{normend}(E_x, E_n')$.

Step 3: Calculate the accuracy $\mu(x)$ of the qualitative concept $C$. $\mu(x) = e^{\frac{(x-E_x)^2}{2(E_n')^2}}$

Step 4: Jump to step 1 until the generated cloud droplets meet the number of demanded.

For the cloud model $C(x)$, the most representative numerical feature $E_x$ can be used as an excellent feature inherited from the previous generation and inherited by the offspring. Entropy $E_n$ determines the range of the offspring update. $H_e$ guarantees the stability of the offspring update. If we need to fast local refinement and improve the search accuracy, we can reduce $E_n$ and $H_e$ to reduce the value range of the offspring and increase the stability. If we need to expand the search range of the offspring and jump out of the local optimal position, then we can increase the value of $E_n$ and $H_e$ so that it can increase the diversity of individuals, and expand the search range of the algorithm.

4.4. Multi-objective Lion Swarm Optimization Based on Cloud Model Mutation

Cloud model theory can balance the difference between the global search ability and the local search ability of the lion swarm optimization in different periods. Therefore, the mutation strategy based on cloud model can be introduced into the multi-objective lion swarm algorithm. In the iterative process, we set a critical condition for individual mutation. When this condition is reached, the mutation operation will be performed on the individual $X_i$ according to the method of cloud model generator. If the mutation condition is not reached, the normal iterative process continues. The mutation condition of this strategy is set as:

$$t > k \times T \mod(t, i) = 0$$

(18)

In the above formula, $t$ is the current iteration number, and $T$ is the max iteration number. The scale coefficient $k$ and the interval coefficient $i$ are given in advance. When the evolutionary number reaches a certain proportion of the maximum number of iterations, it can be considered that the population individuals have gradually approached the best position that they can find after sufficient iterations, and mutation evolution operations can be performed. A mutation process for the population makes the population individuals expand the search range according to a specific pattern. If this mutation is too frequent, then the population individuals cannot fully search in the new area. If the number of mutations is too small, it cannot be fully utilized. According to actual tests, the iterations of mutation interval shall be 5 times to 10 times, which has the best effect on improving the performance of the algorithm. Setting the interval coefficient helps to maintain the continuity of the individual’s movement, so that the individual has sufficient time to search in a new location after the mutation operation, instead of “jumping” random optimization.

The mutation of individual lions is completed by the one-dimensional forward normal cloud generator. For individual $X_i$, there is an individual historical optimal value $p_{best_i}$, then set $E_x$, $E_n$ and $H_e$ as

$$E_x = p_{best_i}, E_n = \alpha_j, H_e = \frac{E_n}{10}$$

(19)
\[ \alpha_f = \text{step}(T - t)^{-1} \] (20)

\( En \) is a dynamically decreasing value. The higher the entropy represented by \( En \), the greater the width of the cloud model, and the larger the range of particle optimization. In the early stage of the iteration, a larger optimization range is conducive to the global search. And it can help jumping out of the local optimum. In the later stage of the iteration, a smaller optimization range is helpful for precise local optimization.

New lion individuals will be generated, according to the cloud generator in Algorithm 4. Algorithm 5 is the detailed steps of CMOLSO.

**Algorithm 5.** Multi-objective Lion Swarm Optimization based on Cloud Model Mutation (CMOLSO)

- **Step1:** Initialize the population, determine population number \( N \) and the individual dimension \( D \).
- **Step2:** Initialize the individual historical optimal values. Establish the external file according to the fitness values of the initial population. The global optimum is randomly selected at a certain proportion in the front part of the external file sequence.
- **Step3:** Update the population. The number of iterations \( t = t + 1 \)
- **Step4:** According to new fitness values and the Pareto domination principle, update the individual history optimal. If they are all non-dominated solutions, randomly choose one. Update and reorder external file, and then update the global optimal based on the external file.
- **Step5:** If the mutation condition was meeting, Perform cloud model mutation operations on population individuals. Otherwise, Skip this step.
- **Step6:** If the iteration end condition was met, stop the iteration. Otherwise, skip to step3.
- **Step7:** Output the final external file and performance parameters.

5. Simulation and Analysis

5.1. Multi-objective Test Functions

We use three multi-objective functions for testing: ZDT1 is an optimization function with a continuous concave Pareto front, ZDT2 has a continuous convex Pareto front, and ZDT3 has a segmented discontinuous Pareto front.

5.2. Multi-objective Lion Swarm Optimization Performance Test

There are two main indicators of performance evaluation in multi-objective optimization, generational distance (GD) and diversity indicator \( \Delta \). GD is used to measure how close the Pareto solution obtained by the algorithm is to the true Pareto front of the problem. The smaller GD indicates the better algorithm performance. The diversity indicator \( \Delta \) is used to measure the uniformity of the solution distribution. The smaller the \( \Delta \), the more uniform the solution distribution of the solution set. If all Pareto solutions are completely evenly distributed on the Pareto front, the \( \Delta \) is zero. Table 3 lists the performance data of MOPSO, NSGA-2[31], MOLSO and CMOLSO. The population sizes of algorithms are all 40 and the numbers of population iterations of algorithms are all 200.

**Table 3.** Performance data of MOPSO, NSGA-2, MOLSO and CMOLSO in low dimension.

| Function | Indicators | MOPSO       | NSGA-2      | MOLSO       | CMOLSO      |
|----------|------------|-------------|-------------|-------------|-------------|
| ZDT1     | GD         | 2.01E-04    | 1.40E-04    | 1.12E-04    | 1.30E-04    |
|          | \( \Delta \) | 0.7355      | 0.7048      | **0.3502**  | 0.3504      |
| ZDT2     | GD         | 2.06E-04    | **6.77E-05** | 1.85E-04    | 1.09E-04    |
|          | \( \Delta \) | 0.6999      | 0.6671      | 0.3761      | **0.3604**  |
| ZDT3     | GD         | 3.24E-04    | 2.92E-04    | 2.34E-04    | **2.22E-04** |
|          | \( \Delta \) | 0.8313      | **0.5754**  | 0.7845      | 0.8455      |
Table 3 is the performance data of the four algorithms measured in low dimension (dimension $D=2$). It can be seen that MOLSO and CMOLSO achieve better results than MOPSO and NSGA-2 on most data, indicating that they have excellent optimization performance.

Also, we know that under low-dimensional conditions, the performance of CMOLSO and ordinary MOLSO can not clearly reflect the effect of cloud model mutation mechanism. However, under high-dimensional conditions, the performance indicators of CMOLSO have been significantly improved compared to MOLSO. Table 4 shows the data of different algorithms under the condition of 30-dimension. By comparing MOLSO and CMOLSO, it can be seen that on the continuous Pareto front function $ZDT1$ and the segmented discontinuous Pareto front function $ZDT3$, the introduction of the cloud model mutation mechanism improves the optimization performance of the algorithm under high-dimensional conditions, and improves the success rate of optimization.

Table 4. Performance data of MOPSO, NSGA-2, MOLSO and CMOLSO under 30-dimension.

| Function | Indicator | MOPSO    | NSGA-2   | MOLSO    | CMOLSO   |
|----------|-----------|----------|----------|----------|----------|
| ZDT1     | GD        | -        | 5.8615E-4| 1.90E-03 | 4.6325E-04|
|          | $\Delta$  | -        | 0.6253   | 0.6205   | 0.3936   |
|          | Success rate | 0       | 100%     | 60%      | 90%      |
| ZDT3     | GD        | -        | 7.8504E-4| 8.5531E-04| 5.0021E-04|
|          | $\Delta$  | -        | 0.5562   | 1.0497   | 1.0281   |
|          | Success rate | 0       | 80%      | 60%      | 80%      |

Let us analysis the experiment results. Regarding the CMOLSO, we can mainly get two conclusions. First, this algorithm has better performance than traditional algorithms. The second one is the introduction of cloud model theory does improve performance under higher dimension.

For the first conclusion above, it mainly relies on the lion swarm optimization to be more sophisticated and diversified than NSGA-2 and MOPSO in the design of the behavior mechanism. In addition to the common mechanism designed for multi-objective search, NSGA-2 has a single population update method, and each individual behavior is basically the same. And the algorithm MOPSO is the same. But the behavior mode of CMOLSO is more diverse, and information exchange exists within the population. So its search and optimization capabilities are better.

For the second conclusion, the role of cloud model theory is to guide the population to mutate in a specific pattern. The cloud model has two Gaussian random processes due to the concept of hyper entropy. Because the distribution of cloud droplets has “thickness”, it has a wider sample distribution than ordinary Gaussian random or uniform random. Cloud model mutation can help individuals jump out of a certain local area or local extreme value position, thereby expanding the search range. We have known that multi-objective optimization algorithms have unsatisfactory optimization results under high dimension. The optimization results might not close enough to the real Pareto front and the solutions are too concentrated in the local area of the Pareto front. The cloud model mutation mechanism helps the lion individuals jump out of the local optimum and expands the search range. So it can enhance the population search ability, and improve the optimization performance.

In summary, CMOLSO is a good multi-objective intelligent optimization. Compared with traditional optimization algorithms, its population is divided into three parts to take on different functions. And these parts have good co-evolution capabilities. Its evolution mechanism is more complex and diversified, so it has good optimization and adaptive capabilities. Furthermore, it improves the performance of the algorithm through the innovation of the cloud model mechanism. These are the main advantages of CMOLSO compared with other algorithms. This algorithm provides a new idea and method for us to solve complex optimization problems.
5.3. Simulation of PFSP
The PFSP optimization function contains two optimization goals. They are minimizing the max finish time $f_1$ and minimizing the max delay time $f_2$. The simulation data is from the Car3 calculation example in the Car series standard calculation examples. It contains the processing time of 12 jobs processed on 5 machines and is a $5 \times 12$ scale model. Table 5 lists the data about processing time in order which is required on machine 1 to machine 5 for a job in each row.

| Jobs | Process times from machine 1 to machine 5 |
|------|------------------------------------------|
| J1   | 456 537 123 214 234                     |
| J2   | 789 854 225 528 123                     |
| J3   | 876 632 588 896 456                     |
| J4   | 543 145 669 325 789                     |
| J5   | 210 785 966 147 876                     |
| J6   | 123 214 332 856 543                     |
| J7   | 456 752 144 321 210                     |
| J8   | 789 143 755 427 123                     |
| J9   | 876 698 322 546 456                     |
| J10  | 543 532 100 321 789                     |
| J11  | 210 145 114 401 876                     |
| J12  | 124 247 753 214 543                     |

Figure 2 is the Pareto curve obtained by connecting 10 Pareto solutions. Figure 3 is a Gantt chart of one of the ten solutions obtained by the CMOLSO algorithm.

**Figure 2.** The Pareto front diagram of the optimization result of the calculation example Car3. The horizontal axis represents $f_1$; the vertical axis is 10 times the value of $f_2$. 
5.4. Simulation of JSP

Similarly, the JSP model optimization function contains two same optimization goals as PFSP. The simulation data is from the ft06 calculation example in the ft series standard calculation examples. It contains the procedures of 6 jobs on 6 processing machines and their corresponding machine numbers and processing time. It is a $6 \times 6$ scale model.

Table 6 lists the machine numbers corresponding to each process of the jobs. From the table we can know which machine should be used to process a certain process $J_{ik}$ of a certain job. Table 7 lists the processing time required for each process of the jobs. From table 6 we can know the length of time consumed by the $J_{ik}$ process of a certain job.

Table 6. Correspondence between job process and machines of JSP model.

| Jobs | Process/machine number |
|------|------------------------|
| J1   | J_{11}/3               |
| J1   | J_{12}/1               |
| J1   | J_{13}/2               |
| J1   | J_{14}/4               |
| J1   | J_{15}/6               |
| J1   | J_{16}/5               |
| J2   | J_{21}/2               |
| J2   | J_{22}/3               |
| J2   | J_{23}/5               |
| J2   | J_{24}/6               |
| J2   | J_{25}/1               |
| J2   | J_{26}/4               |
| J3   | J_{31}/3               |
| J3   | J_{32}/4               |
| J3   | J_{33}/6               |
| J3   | J_{34}/1               |
| J3   | J_{35}/2               |
| J3   | J_{36}/5               |
| J4   | J_{41}/2               |
| J4   | J_{42}/1               |
| J4   | J_{43}/3               |
| J4   | J_{44}/4               |
| J4   | J_{45}/5               |
| J4   | J_{46}/6               |
| J5   | J_{51}/3               |
| J5   | J_{52}/2               |
| J5   | J_{53}/5               |
| J5   | J_{54}/6               |
| J5   | J_{55}/1               |
| J5   | J_{56}/4               |
| J6   | J_{61}/2               |
| J6   | J_{62}/4               |
| J6   | J_{63}/6               |
| J6   | J_{64}/1               |
| J6   | J_{65}/5               |
| J6   | J_{66}/3               |

Table 7. Correspondence between job process and processing time of JSP model.

| Jobs | Process/Processing time |
|------|-------------------------|
| J1   | J_{11}/1               |
| J1   | J_{12}/3               |
| J1   | J_{13}/6               |
| J1   | J_{14}/7               |
| J1   | J_{15}/3               |
| J1   | J_{16}/6               |
| J2   | J_{21}/8               |
| J2   | J_{22}/5               |
| J2   | J_{23}/10              |
| J2   | J_{24}/10              |
| J2   | J_{25}/10              |
| J2   | J_{26}/4               |
| J3   | J_{31}/5               |
| J3   | J_{32}/4               |
| J3   | J_{33}/8               |
| J3   | J_{34}/9               |
| J3   | J_{35}/1               |
| J3   | J_{36}/7               |
| J4   | J_{41}/5               |
| J4   | J_{42}/5               |
| J4   | J_{43}/3               |
| J4   | J_{44}/8               |
| J4   | J_{45}/9               |
| J5   | J_{51}/9               |
| J5   | J_{52}/3               |
| J5   | J_{53}/5               |
| J5   | J_{54}/4               |
| J5   | J_{55}/3               |
| J5   | J_{56}/1               |
| J6   | J_{61}/3               |
| J6   | J_{62}/3               |
| J6   | J_{63}/9               |
| J6   | J_{64}/10              |
| J6   | J_{65}/4               |
| J6   | J_{66}/1               |

Table 8 lists the two objective function values of some solutions obtained from CMOLSO. These solutions are all Pareto solutions and they obey the Pareto criterion.
Table 8. objective function values of part of Pareto solutions.

| Solutions | $f_1 : C_{\text{max}}$ | $f_2 : T_{\text{max}}$ |
|-----------|------------------------|------------------------|
| 1         | 59                     | 33                     |
| 2         | 75                     | 16                     |
| 3         | 67                     | 17                     |
| 4         | 60                     | 24                     |
| 5         | 62                     | 19                     |
| 6         | 61                     | 22                     |

Figure 4 is one solution Gantt chart of JSP. The numbers of jobs, from 1 to 6, and their processing time are marked in the rectangles. The figure visually shows the situation of the jobs processed at different times on each machine.

Figure 4. Gantt chart of one of the optimization results of calculation example $ft06$.

From the simulation experiments, we know the algorithm can calculate multiple Pareto solutions we need. This shows the algorithm is feasible and practical in solving FSP and JSP problems.

6. Conclusion
Both the FSP and JSP are more complex combinatorial optimization problems. On the basis of previous work, we established a multi-objective optimization model and determined the two optimization goals of minimizing the maximum completion time and minimizing the maximum delay time as the multi-objective optimization function. In order to solve this multi-objective optimization problem, based on the original single-objective lion swarm optimization, we further proposed the multi-objective lion swarm optimization through a certain transformation mechanism. At the same time, the cloud model mutation mechanism is adopted into the multi-objective lion swarm optimization, which further improves the performance of the algorithm, to a certain extent, prevents the population from falling into local optimal and local excessive concentration, and expands the optimization range. In conclusion, the algorithm has diversity of behavior good cooperation within a population and has better adaptability in high dimension. It has greater search and exploration capabilities and achieves ideal effect in comparison with other optimization algorithms according to the experimental results. Using the CMOLSO, we can give a variety of solutions to the FSP and JSP, and then provide a reference plan for the actual industrial production and processing.
Acknowledgments
This study was financially supported by The Shandong Province Science Foundation of China (Grant No. ZR2020MF153) and the Key Innovation Project of Shandong Province (Grant No. 2019JZZY010111).

References
[1] Wu Z, Sun S and Yu S 2020 Optimizing makespan and stability risks in job shop scheduling Computers & Operations Research 122 104963.
[2] Zhang L and Wang L 2020 Multi-objective optimization for flexible job-shop scheduling based on energy consumption Modern Electronic Technology 43 558 (07) 126-130.
[3] Liu H, Zeng H, Zhou W, et al. 2019 Optimization of job shop scheduling based on improved particle swarm optimization algorithm Journal of Shandong University (Engineering Science) 49 (1) 57-62.
[4] Gao K, Cao Z, Zhang L, et al. 2019 A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems IEEE/CAA J. Automat. Sinica 6 (4) 904-916.
[5] Shao W, Pi D and Shao Z 2019 A Pareto-based estimation of distribution algorithm for solving multi-objective distributed no-wait flow-shop scheduling problem with sequence-dependent setup time IEEE Trans. on Automation Science and Engineering 16 (3) 1344-1360.
[6] Gao K, Yang F, Zhou M C, et al. 2019 Flexible job-shop rescheduling for new job insertion by using discrete JAYA algorithm IEEE Trans. on Cybernetics 49 (5) 1944-1955.
[7] Oztop H, Tasgetiren M F, Eliyi D T, et al. 2020 An energy-efficient permutation flowshop scheduling problem Expert Syst. Appl. 150 113279.
[8] Ghaleb M, Zolfagharinia H and Taghipour S 2020 Real-time production scheduling in the Industry-4.0 context: Addressing uncertainties in job arrivals and machine breakdown Comput. Oper. Res. 123 105031.
[9] Qiu H, Fan P, Tang H, et al. 2018 An effective hybrid discrete grey wolf optimizer for the casting production scheduling problem with multi-objective and multi-constraint Comput. Ind. Eng. 128 458-476.
[10] Jiang M and Yuan D 2012 Artificial Fish Swarm Algorithm and Its Applications (Science Press).
[11] Jiang M and Yuan D 2016 Artificial Bee Colony Algorithm and Its Applications (Science Press).
[12] Zarrouk R, Bennour J E and Jemai A 2019 A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem Swarm Intelligence 13 (2) 145-168.
[13] Wang H, Wang W, Cui L, et al. 2018 A hybrid multi-objective firefly algorithm for big data optimization Appl. Soft Comput. 69 806-815.
[14] Babalik A, Ozkis A, Uymaz S A, et al. 2018 A multi-objective artificial algae algorithm Appl. Soft Comput. 68 377-395.
[15] Lai X, Li C, Zhang N, et al. 2019 A multi-objective artificial sheep algorithm Neural Comput. Appl. 31 4049-4083.
[16] Tharwat A, Houssein E H, Ahmed M M, et al. 2018 MOGOA algorithm for constrained and unconstrained multi-objective optimization problems Appl. Intelligence 48 2268-2283.
[17] Zain M, Kanesan J, Chuah J H, et al. 2018 A multi-objective particle swarm optimization algorithm based on dynamic boundary search for constrained optimization Appl. Soft Comput. 70 680-700.
[18] Mirjalili S 2016 Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems Neural Comput. Appl. 27 (4) 1053-1073.
[19] Mirjalili S, Gandomi A H, Mirjalili S E, et al. 2017 SALP swarm algorithm: A bio-inspired optimizer for engineering design problems Adv. Eng. Softw. 114 163-191.
Zhang J and Gan Y 2018 Optimization of multi-objective micro-grid based on improved particle swarm optimization algorithm AMME 2018.

Yang Z and Liu C 2018 A hybrid multi-objective gray wolf optimization algorithm for a fuzzy blocking flow shop scheduling problem Adv. Mech. Eng. 10 (3) 1-13.

Piroozfarda H, Wong K Y and Wong W P 2018 Minimizing total carbon footprint and total late work criterion in flexible job shop scheduling by using an improved multi-objective genetic algorithm Resour. Conserv. Recy. 128 267-283.

Jiao S, Pan Z, Chen Y, et al. 2020 Cloud annealing: a novel simulated annealing algorithm based on cloud model ICE Trans. on Information and Systems E103.D (1) 85-92.

Maziar Y and Fariborz J 2016 Lion optimization algorithm (LOA): A nature-inspired metaheuristic algorithm J. Computational Design and Engineering 3 (1) 24-36.

Liu S, Yang Y and Zhou Y 2018 A swarm intelligence algorithm -- Lion swarm algorithm Pattern Recognition and Artificial Intelligence 31 179 (05) 45-55.

Shi W and Yan X 2019 Improved particle swarm PMSM parameter identification algorithm based on cloud model Journal of Dalian Jiaotong University 40 (001) 113-119.

Guan X, Huangpu X, Li X, et al. 2016 Adaptive mutation quantum particle swarm optimization based on normal cloud model Electronic Design Engineering 24 (08) 64-67.

Zhang Y and Shao S 2011 A cloud mutation particle swarm optimization based on cloud model Pattern Recognition and Artificial Intelligence 24 (01) 90-96.

Wang B 2017 Improvement of Particle Swarm Optimization and Its Application in PID Parameter Tuning (Changsha: Hunan University).

Bi K and Qiu T 2019 Novel Naphtha molecular reconstruction process using a self-adaptive cloud model and hybrid genetic algorithm-particle swarm optimization algorithm Industrial & Engineering Chemistry Research 58 (36) 16753-16760.

Alikar N, Mousavi S M, Ghazilla R A R, et al. 2017 Application of the NSGA-II algorithm to a multi-period inventory-redundancy allocation problem in a series-parallel system Reliability Engineering System Safety 160 1-10.