# A New Hybrid Cuckoo Search for the Resources-Constrained Project Scheduling Problem

Xin Shen , Xiaoxia Zhang , Ziqiao Yu

***Abstract*—Resource-constrained project scheduling problem (RCPSP) is a kind of representative practical engineering problem, the purpose is to schedule activities in the project through rational use of limited resources in the minimum time. This paper proposes an improved hybrid cuckoo algorithm (CS&SA) to solve the RCPSP problem. Firstly, the individual elements are randomly coded into priority vectors, and the population is decoded using serial scheduling to convert the individual into a set of task scheduling sequences. Secondly, the Levy flight is redesigned to change the algorithm from random walk to adaptive as the population fitness changes. Then, this article adds three neighborhood update techniques to meet the update requirements of the algorithm at different stages. Finally, in order to prevent the algorithm from falling into a local optimum, this paper introduces a simulated annealing strategy to allow the algorithm to accept some individuals with poor quality with a certain probability in each iteration. In the testing part, this paper firstly tests the effectiveness and optimization of three different-scale examples in the classic example library PSPLIB for RCPSP problems. Among them, the average error of the CS&SA algorithm in the small-scale J30 is 0.25%, and the average error in the medium-scale J60 is 11.21%, and the average error of the large-scale J120 is 19.83%. Then, by comparing other intelligent optimization algorithms, it is proved that NCS&SA is superior to other algorithms in optimization and accuracy.**

***Index Terms*—Cuckoo Search; Resources constrained project scheduling; Simulated annealing; combinatorial optimization.**

## I. Introduction

Scheduling problem has been a hot research topic in engineering problems in recent years. Although the scheduling problems are diverse, most of them can be regarded as variants of RCPSP [1], that is, an item with *N* activities and *K* renewable resources. Scheduling activities by allocating limited resources, and there is a priority relationship between activities. Under the circumstance of clear task priority and resource limit, the task can be reasonably scheduled in the minimum time. In terms of complexity, Blazewicz et al. [2] turned out that RCPSP is an NP-hard engineering scheduling problem. In recent years, RCPSP has been extensively studied, and has derived job shop scheduling and assembly line scheduling problems[3]. At the same time, RCPSP is widely used in engineering fields, such as medical research, software development and construction industries[4].

Because the RCPSP problem has a very extensive research and application, there are many solutions to solve this very complex problem. Kolisch and Hartmann [5] roughly divide them into two categories: precise algorithms and heuristic algorithms. Sprecher [6], Demeulemeester [7] and Brucker [8] have shown through a large number of experimental results that the traditional accurate algorithm can get the best results, but the calculation time is longer, and it can only solve small-scale problems of 60 activities at most. Therefore, in order to comply with actual engineering applications, heuristic algorithms are generally used to solve this problem.

Early heuristic algorithms mostly used methods based on priority rules, such as minimum relaxation (MSLK) [9], earliest start time (EST), shortest processing time (SPT) [10], etc. Many of these priority rules are developed from other scheduling problems. According to the priority rules, each activity in the construction plan is determined, and then the final construction plan is generated according to the scheduling. Regarding the scheduling generation scheme, this article will explain in detail later. Chen et al. [11] conducted an experimental analysis and summary of these different scheduling finite rules.

With the development of information technology and computational intelligence, traditional heuristic algorithms are no longer satisfied with the study of large-scale and complex scheduling problems. In order to obtain faster solution efficiency and higher quality solutions, intelligent optimization algorithms are widely used in RCPSP [12]. The intelligent optimization algorithm is an algorithm theory established by simulating biological behavior or physical movement laws in nature, adopting a suitable population coding scheme, and then searching and updating the population through a specific strategy, and finally finding the problem Optimal solution [13]. In RCPSP, there are two main ways to encode the population: active list (AL) and random key (RK). Commonly used search strategies include genetic algorithm (GA) [14], simulated annealing (SA) [15], particle swarm optimization (PSO) [16], etc. The meta-heuristic algorithm can not only solve the RCPSP with a large amount of data, but also search the solution space efficiently and find the global optimal solution.

Yang and Deb put forwarded to the cuckoo search algorithm (CS) in 2009, and proved the success rate of this

Manuscript received September 19, 2020, revised February 25, 2021. This work was supported by Project of Liaoning Xincheng Co., Ltd (Grant No. L20170989).

X. Shen is a Master Student of School of Computer Science and Software Engineering, University of Science and Technology LiaoNing, Anshan, 114051, China (e_mail: shenxin_4395@163.com).

X. X. Zhang is a Professor of School of Computer Science and Software Engineering, University of Science and Technology LiaoNing, Anshan,11041,China (corresponding author, phone:86-0412-5929812; e_mail: aszhangxx@163.com).

Z. Q. Yu is a Master Student of School of Computer Science and Software Engineering, University of Science and Technology LiaoNing, Anshan, 114051, China (e_mail: 1152899934@qq.com).

algorithm in global optimization. The proposal of CS has attracted the attention of experts and scholars in different fields. Ouaarab et al. [17] solved the traveling salesman problem through CS, which was the first time that CS algorithm was applied to a discrete optimization problem. In addition, studies have shown that the improved CS algorithm can solve the NP-Hard problem well.

Although CS can solve engineering optimization problems well, some shortcomings have been found in its application: the limitations of Lévy Flight and the weak local search ability, which affects the global optimization accuracy. Aiming at the deficiencies of CS, this paper uses the fitness value of the population to influence the search process of Lévy Flight, so that the search has adaptability. This paper also adds simulated annealing mechanism to improve the accuracy of the algorithm's global optimization, and make the algorithm more competitive in solving NP-Hard problems.

In this paper, we put forward an improved cuckoo algorithm CS&SA to solve the RCPSP. Firstly, the task priority of the population is encoded by random key method, and the individuals in the population are decoded into a feasible task scheduling scheme by serial scheduling. Second, redesign the probability density function that affects Lévy Flight search to make it adaptive as the population fitness changes. Then, according to the step length of adaptive Lévy Flight, this paper designs three different neighborhood update methods to meet the update requirements of the algorithm at different stages. Finally, the algorithm introduces a simulated annealing mechanism, so that the algorithm accepts some individuals with poor fitness values with a certain probability in each iteration, and expands the search range of the population. The rest of the paper is as follows: Section 2 introduces the RCPSP. Section 3 introduces CS and its principles. Section 4 introduces the CS&SA algorithm. Section 5 design experiment and analysis experimental results, Section 6 is the conclusion.

## II. Resource Constraint Project Scheduling Problem

RCPSP is to schedule tasks through rational use of resources to minimize the duration of the item. RCPSP has the following assumptions:

1. For logical constraints, only consider the situation when a task starts immediately after the previous task ends.
2. Every task cannot be interrupted.
3. Only consider the renewable resource limit.

Based on the above assumptions, we give the definition of RCPSP: there are $N$ activities in a limited set of project $J$, where each activity is represented as $J_i$ ($i$=0,1,2, …, $N$+1). $J_0$ and $J_{N+1}$ are two virtual activities in the project, representing the start and end of the item. There is a priority relationship between activities in the project, that is, no new tasks can be started before the scheduled high-priority activities are completed. Therefore, each activity $j$ ($1 \leq j \leq N$) has a period, and we use $d_j$ to represent it. The period of virtual activities is set to $d_0=d_{n+1}=0$.

Renewable resources used in activities are limited when they are mobilized, and are composed of a finite set $R$, where each resource is represented as $R_k$ ($k$=1,2,…,$K$). $R_{jk}$ represents the amount of resources required by the activity when it is scheduled. The amount of resources required for virtual activities is set to $r_{0k}=r_{(N+1)k}=0$.

Set $s_j$ ($j$=0,1, ..., $N$, $N$+1) is the start moment, where $s_0$ is the start moment of $J_0$. It represents the beginning of the project, Set $S_0=0$. $f_j$ is the end moment of the activities, and the relationship between $s_j$, $d_j$ and $f_j$ is expressed as:

$$s_j + d_j = f_j \tag{1}$$

It can be seen from (1) that the follow-up activities should start after the previous activities are completed and cannot be interrupted. The objective function is to find the minimum item duration.

In order to describe RCPSP more appropriately in mathematical language, we define the state of activity $j$ at time $t$ as a binary variable $x_{jt}$:

$$x_{jt} = \begin{cases} 1 & s \le t < f_i \\ 0 & otherwise \end{cases} \tag{2}$$

According to the assumptions we put forward and the mathematical variables defined, the mathematical model of RCPSP is expressed as:

$$Min: f_{n+1} = f_0 + f_1 + ... + f_n \tag{3}$$

$$f_0 = 0, d_0 = 0, d_{n+1} = 0 \tag{4}$$

$$\sum_{i=1}^{n} r_{ik} \cdot x_{jt} \le R_k \quad \forall k = 1, 2, ..., K \tag{5}$$

$$s_j - s_i \ge d_j \quad \forall (i, j) \in E \tag{6}$$

where (3) represents the objective function of RCPSP, (4) represents that virtual activities do not occupy any time and resources, and (5) represents the limit of resources, (6) expresses the time window for executing activities, that is, the priority relationship between activities.

In the execution of the project, each activity can be regarded as an activity-on-node (AON) graph [18]. The topology graph is composed of $J$ activity nodes and R renewable resources. The directed graph represents the priority relationship between them.

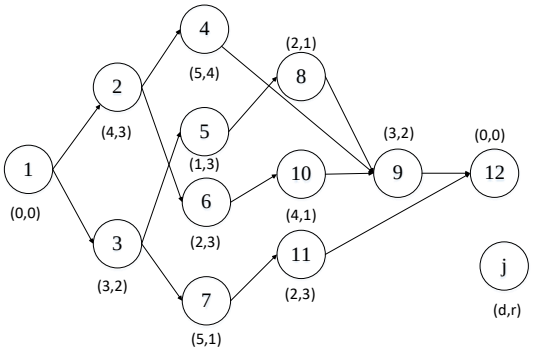


Fig.1.An example of a project

Fig.1 shows a simple example of a set of 12 activities in RCPSP (including two virtual activities). Each node represents an activity, the scheduling time and required resources of the activity are provided under the node. In this example, only one renewable resource. Fig.2 is the best completion time. Among them, the horizontal axis shows the scheduling time, and the vertical axis shows the

consumption of resources. Under the constraints of resources and time, a feasible completion time is 20.

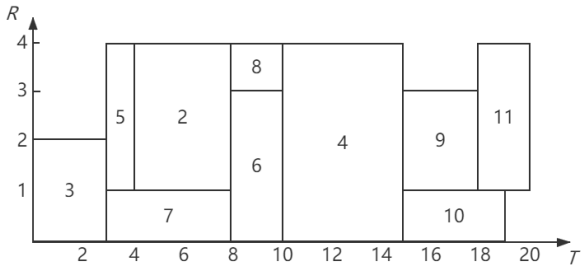


Fig.2. A feasible schedule

## III. Cuckoo Search

The idea of cuckoo algorithm comes from the special way of breeding cuckoos in nature: they lay eggs in other nests in order to breed offspring [19]. In order for their offspring to successfully hatch and survive, the eggs produced by the cuckoo will be as consistent as possible with the host's eggs in color and size. And if the host bird finds a foreign egg, they can choose to give up the cuckoo egg, or give up the nest, and randomly choose another place to build a new nest. Yang and Deb developed this swarm intelligence optimization algorithm based on this natural phenomenon and summarized three basic principles:

1. By default, the cuckoo only produces one egg and randomly selects a host bird nest.
2. The best nest and bird eggs are retained as the current optimal solution to the next generation.
3. Cuckoo's eggs will be found with a certain probability ($P_a$). When the cuckoo's eggs were found, the bird will randomly select a new nest.

The CS's characteristic is to use Lévy Flight search strategy [20] to optimize the search method of understanding space. When the algorithm moves in the solution space, a new solution of the algorithm is generated through (7).

$$x_i^{(t+1)} = x_i^t + \alpha \oplus Lévy(\lambda) \tag{7}$$

where the parameter $\alpha$ represents the step size control amount, and in most cases $\alpha$=1; Lévy (λ) represents the Lévy search path, and the value obeys the Lévy probability distribution:

$$Lévy(\lambda) = t^{-\lambda} \tag{8}$$

In Lévy Flight, it is very difficult to generate step length through algorithm. Now the most widely used method is known to simulate Lévy Flight distance and step length based on Mantegna algorithm:

$$s = \frac{u}{|v|^{-\beta}} \tag{9}$$

where $s$ is the search step length generated by Lévy Flight, $u$ and $v$ are two random variables that obey the normal distribution and affect the flight step length: $u \sim \left(0, \sigma_u^2\right)$ and $v \sim \left(0, \sigma_v^2\right)$.

Fig.3 is a simulation of the search path of 1000 Lévy Flight. It can be seen that Lévy Flight is a random walk process. After frequent small-distance flights, it will occasionally make large-scale jumps. Lévy Flight is introduced into the cuckoo algorithm. The search can be maximized in the unknown population range, and avoid falling into the local optimum.

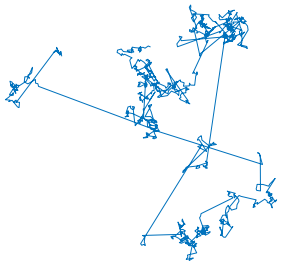

Fig.3. Lévy Flights trajectory under 1000 iterations

Algorithm: Cuckoo algorithm

Objective function $f(x), x = (x_1, ..., x_d)$;

Generate initial population of n host nest $s$, $x_i (i = 1, 2, ..., n)$;

**While** ($t <$ *MaxGenerations*) or (stop criterion) **do**
  Get a cuckoo randomly and Levy flight;
  Evaluate its quality/fitness $F_i$;
  Choose a nest among $n$ (says $j$) randomly;
  **If** $F_i > F_j$
    Replace $j$ by the new solution;
  **End if**
  A fraction ($P_a$) of worse nests are abandoned and new ones are built;
  Keep the best solutions (or nests with quality soultions);
  Rank the solutions find the current best;
**End while**
Postprocess result and visualization;

Fig.4. Pseudo-code of CS

As an emerging intelligent optimization algorithm, CS has the advantage of fewer parameters, strong global optimization capabilities and algorithm scalability. However, CS also has disadvantages such as fixed parameters and poor local optimization accuracy [21]. Therefore, finding suitable improvement strategies in solving RCPSP can better obtain the optimal solution.

Based on the principles defined above and the search policy pattern, The pseudo-code of CS is shown in Fig. 4.

## IV. The CS&SA For RCPSP

In this paper, we improve CS by analyzing its shortcomings. Firstly, the coding and decoding of the population were designed; secondly, Lévy flight was made self-adaptive; then three neighborhood update techniques were added; corresponding neighborhood update was selected at different stages of the algorithm according to the adaptive step size of Lévy flight; finally, the simulated annealing mechanism is introduced to enhance the ability of the algorithm to find the best solution.

### A. Encoding and Decoding

Designing populations and solutions is very important to solve RCPSP. Since various constraints are considered while coding the population, it is more efficient to perform subsequent operations after coding the population than directly operating it.

In CS, individuals in the population are represented by eggs and nests, expressed in the form of chromosomes. Each gene in the chromosome represents a task, and the index of the gene represents the scheduling order of the task. In the CS&SA proposed in this paper, assuming that the population scale is $N$ and activity scale is $M$, the population is expressed as: $P=\{P_1, P_2, \dots, P_i, \dots, P_N\}$.

Individuals in the population are a set of activity scheduling sequences:

$$\begin{Bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,j} & \cdots & p_{1,M} \\ p_{2,2} & p_{2,2} & \cdots & p_{2,j} & \cdots & p_{2,M} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ p_{i,1} & p_{i,2} & \cdots & p_{i,j} & \cdots & p_{i,M} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ p_{N,1} & p_{N,2} & \cdots & p_{N,j} & \cdots & p_{N,M} \end{Bmatrix}$$

The index of the individual is the task scheduling sequence, and $P_i$ is the $i^{th}$ chromosome in the population, which represents a set of feasible task scheduling schemes. $P_{ij}$ is the $j^{th}$ gene in the $i^{th}$ chromosome and represents the $j^{th}$ task number. Through algorithm iteration, the scheduling $P_i$ with the smallest project duration in the population is finally obtained as the best solution, which is recorded as $P_{best}$.

The solution of the algorithm is a set of activity scheduling sequences with the smallest completion time, and the index of the sequence is the execution order of the activities. Meanwhile, the coding and feasible solutions of individual populations correspond to each other.

When Kolisch and Hartmann solved the RCPSP, there were two main forms of representation of task scheduling, namely Random-Key (RK) and Activity-List (AL) [22]. AL is a list of $n$ activities directly generated, and the index of the list indicates the scheduling sequence of the activities. This paper uses RK to generate task priority vector to encode the population.

Encode the individuals in the population as a disordered random key vector with a uniform distribution and an interval of [0,1]. By sorting the random key vector, the relationship between the priority and the index is converted into the relationship between the task and the index, and the priority is converted into the task number to be scheduled.

For example, for a 1*5 chromosome, a random key vector (0.44,0.16,0.81,0.27,0.34) is generated by rand with an interval of [0,1], and then (4,1,5,2,3) is obtained by sorting. The time element indicates the sequence of activities, and the index indicates the corresponding activities. As shown in Fig.5.

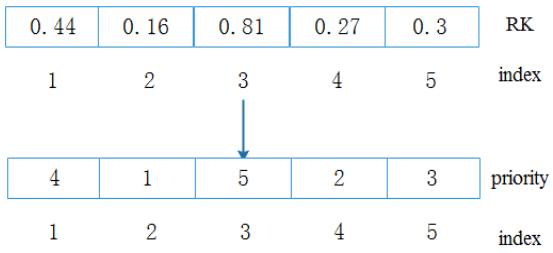


Fig.5. Generate random keys to sort

Then according to the labeling of the elements to the sequence of activities, the activities are rearranged, and finally the decoded chromosomes (2,4,5,1,3) are obtained. At this time, the element represents the corresponding activity, and the index represents the sequence of the activity, as shown in Fig.6.

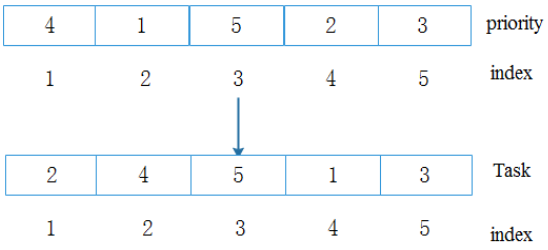


Fig.6. Random keys are converted to activities

### B. Serial Schedule Generation Scheme

In RCPSP, the schedule generation mechanism is the main way to decode the encoded population. The schedule generation mechanism allows tasks to be scheduled locally under various constraints, and gradually expands to generate a complete task scheduling plan globally. The scheduling generation mechanism is divided into serial schedule generation scheme (SSGS) and Parallel Schedule Generation Scheme (PSGS) [23]. In this paper, SSGS is used to decode and feasibly schedule tasks in the population.

In SSGS, suppose the number of tasks is $M$, the task scheduling moment is $T$, the list of tasks to be scheduled is $I$, the earliest start moment of activity $I_i$ is $ES_i$, the start scheduling moment is $S_i$, the end moment list is $F$, the total amount of resources at the current moment is $R_k(t)$, the set of scheduled tasks is $D$, and the priority matrix between tasks is $A$. SSGS is divided into three stages:

**Stage 1:** Add virtual activity $I_1$ to the scheduling sequence and set both the time demand and resource demand of $I_1$ to 0.

**Stage 2**: Iteratively select and schedule the remaining $M$-2 activities in the current individual. In each iteration, the limited set of activities to be scheduled $I$, the set $F$ with

limited activity end time, and the total amount of resources $R_k(t)$ at the current moment are first updated. Select the activity $I_i$ with high priority in the current to be scheduled set and find out all the previous activities of $I_i$. The maximum end time in the pre-order activity is taken as the earliest start moment $ES_i$ of $I_i$, and the moment closest to $ES_i$ is selected, and under the premise of meeting resource requirements, it is used as the start scheduling time of $I_i$, and the occupied resources are deducted. Calculate completion time based on the duration of $I_i$. Finally, the $I_i$ is added to the limited set of scheduling activities $D$ to complete the current scheduling.

**Stage 3**: At the current moment, only the active $I_M$ is not scheduled. Take the maximum completion time of all previous activities is taken as the start time of the active $I_M$ and added to the limited set of scheduled activities $D$.

**Function** SerialSGS ($I$, $A$)

$S(1)=0, F(1)=0, D=D(1)$;

**For** $i=2$ **to** $M-2$

  $I(i)=\text{find}(N)$;

  $ES(i)=\max\{F(pred(i))\}$;

  $S=\min\{t|t\geq ES_i, t\in F, r_{ik}\leq R_k\}$;

  $F(i)=S(i)+d(i)$;

  $D=D\cup I_i$;

**End for**

$F(M)=\max\{F\}$;

$D=D\cup I_M$;

Output: $D$

A sequence of activity to be scheduled $I$, activity priority matrix $A$,

Activity start time $S(i)$, activity end time $F(i)$, activity scheduling sequence $D$.

Fig.7. Pseudo-code of SSGS

When each task in the individual is scheduled in series, the time and resource constraints specified by the problem are considered, so there is no invalid scheduling scheme logically. Therefore, there is no need to verify the feasibility of the scheduling scheme after the scheduling, which improves the operating efficiency of the algorithm. Fig.7 shows the pseudo-code of SSGS

### C. Adaptive Lévy flight search and neighborhood update

In CS, the search and update of the population is realized based on Lévy Flight. Equation (7) determines the movement distance and direction of Lévy Flight, where $\alpha$ controls the movement distance of CS, and the search direction depends on (8) and (9). The two normal random variables $\mu$ and $\nu$ [24]. The fixed value of $\alpha$ and the randomness of $\mu$ and $\nu$ cause the algorithm to have great randomness in the direction and moving distance of Lévy Flight. A large step size is conducive to the global optimization of the algorithm in the early stage, but the search accuracy is low; small step size has It is helpful for the algorithm to improve the optimization accuracy in the later stage, but it will reduce the global convergence speed. In the Lévy Flight, this paper converts two random variables $\mu$ and $\nu$ that obey a normal distribution into variables that change adaptively as the population fitness changes:

$$\mu = Fitness(P_i) - \min(Fitness) \quad (10)$$

$$\nu = \max(Fitness) - \min(Fitness) \quad (11)$$

At the same time, the population renewal formula of cuckoo was redesigned:

$$s = \exp(-\frac{\mu}{\nu^{\beta^{-1}}}) \quad (12)$$

It can be seen from (10), (11) and (12) that when the algorithm begins to iterate, the gap between variables $\mu$ and $\nu$ is large, and the step size s is small, indicating that the current scheduling scheme has not converged to the optimum. When the algorithm is iterated to the later stage, the gap between $\mu$ and $\nu$ gradually decreases, and the step size s approaches 1. At this time, the algorithm changes from global search to local search.

When CS solves the continuous optimization problem, it updates every element in the population, which obviously does not conform to the update idea of the combinatorial optimization problem. Therefore, this paper designs three neighborhood update methods, and chooses different neighborhood update methods according to the calculated step length at different stages of the algorithm.

**Stage 1:** Neighborhood exchange. Neighborhood swap refers to the random selection of two elements in the neighborhood, exchange the location of two elements. The influence of neighborhood exchange is not very great, so it is fit for local optimization in the later stage. For example, the current activity scheduling sequence is $x$ = {1,2,3,4,5,6,7,8}, and two activities 3 and 7 are selected. The scheduling sequence after neighborhood exchange is $x$ = {1,2,7,4,5,6,3,8}. The operation process is shown in Fig.8.

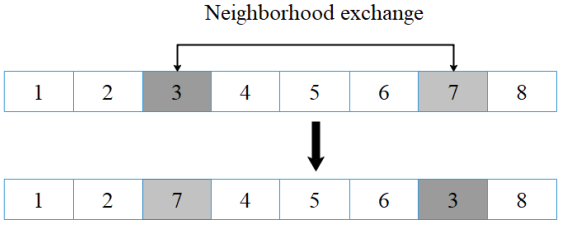


Fig.8. Neighborhood exchange

**Stage 2:** Neighborhood insert. Neighborhood insertion refers to the random selection of two elements in the neighborhood and the insertion of one element into the other. For example, the current task scheduling sequence is $x$ = {1,2,3,4,5,6,7,8}, two activities 3 and 7 are selected,

and the scheduling sequence after neighborhood insertion is $x$ = {1,2,4,5,6,7,3,8}. Unlike neighborhood exchange, when 3 is inserted into 7, all the other activities after 3 will move forward. Therefore, compared with neighborhood exchange, neighborhood insertion is more complex to neighborhood exchange, which is suitable for algorithm iteration to mid-term update. The operation process is shown in Fig.9.

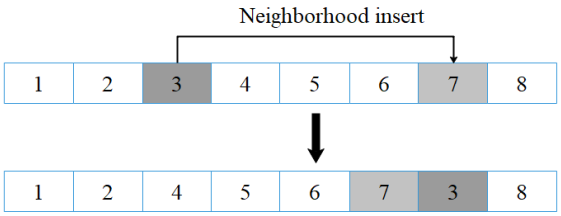


Fig.9. Neighborhood insert

**Stage 3:** 2-opt. 2-opt is a commonly used neighborhood update technology. In combinational optimization problems, 2-opt is often applied to such circular path optimization problems as travel agent problems. In this paper, the 2-opt technique is added to the RCPSP problem. Unlike the ring-shaped solution, the solution of the RCPSP problem is in chain shape, and the last element in the neighborhood is not related to the first element. 2-opt is a more complex neighborhood update mode than neighborhood exchange and neighborhood insertion, which is suitable for the early global optimization update in the algorithm. For example, the current task scheduling sequence is $x$ = {1,2,7,4,5,6,3,8}, select two activities 3 and 7, and reverse the order of all tasks between the two activities, leaving other activities unchanged. The task scheduling sequence after the execution of 2-opt is $x$ = {1,2,7,6,5,4,3,8}, as shown in Fig.10.

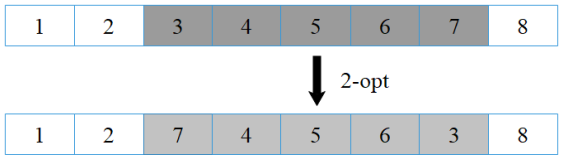


Fig.10. 2-opt

After updating the neighborhood, pay attention to whether the changed individual meets the timing constraints and resource constraints, and then reallocate resources for the new individual.

In order to make the algorithm select the appropriate neighborhood renewal strategy at different stages, the step size $s$ of cuckoo after Lévy flight is divided into three parts, each of which corresponds to a different neighborhood renewal strategy: when $s \in (0,0.3]$, the stage is neighborhood exchange, when $s \in (0.3,0.6]$, the stage is neighborhood insert, when $s \in (0.6,1]$, the stage is 2-opt.

For the updated population, a new scheduling plan shall be constructed through SGS again. The complete operation is the pseudo-code shown in Fig.11.

**Function** Levy ($P$, *Fitness*, $\beta$)
$P_i$ = rand ($P$);
$\mu$ = *Fitness* ($P_i$) – min (*Fitness*);
$\nu$ = max (*Fitness*) – min (*Fitness*);
**if** $\nu$ == 0
  *stepSize* = 0;
**end if**
*stepSize* = exp (-$\mu$./ ( $\nu$.^ (1 / $\beta$ )));
$P_{ij}$ = rand ($P_i$);
$P_{ik}$ = rand (max (*pred* ($P_{ij}$)), min (*sucs* ($P_{ij}$)));
**if** *stepSize* >= 0 && *stepSize* < 0.3
  $newP_i$ = 2-opt ($P_i$, $P_{ij}$, $P_{ik}$);
**else if** *stepSize* >= 0.3 && *stepSize* <0.6
  $newP_i$= insert ($P_i$, $P_{ij}$, $P_{ik}$);
**else**
  $newP_i$ = swap ($P_i$, $P_{ij}$, $P_{ik}$);
**end if**
$newP_i$ = serialSGS ($newP_i$, $A$);
**Output**: $newP_i$
The population $P$, population fitness *Fitness*

Fig.11. Pesudo-code of Lévy Fights

## D. SA For Updating Mechanism

The simulated annealing algorithm is an intelligent optimization algorithm proposed by Metropolis et al [25]. based on the annealing phenomenon of crystals. If the temperature of a high-temperature crystal drops too fast, it is easy to form a higher-energy amorphous, that is, the algorithm falls into a local optimum. At this time, add a little temperature to the amorphous, let the amorphous cool again, and finally become an energy-stable crystal structure, corresponding to the algorithm jumping out of the local optimum, the annealing process is successful, this phenomenon conforms to the law of conservation of energy.

SA accepts the update of the solution based on probability. First, an initial temperature $T_0$ is selected. In order to obtain the optimal solution, $T_0$ is usually set very high, and an initial solution ($P_1$) is randomly selected. Then, a new solution $P_2$ is produced according to the update rule of the individual, and the fitness of the two solutions are compared. If the fitness of $P_2$ is better than the fitness of $P_1$, the new solution is accepted. Otherwise, using the standard proposed by Boltzman's probability to propose a metropolis, if the difference ($\Delta E$) between the evaluation index of $P_1$ and the evaluation index of $P_2$ is equal to or greater than 0, one will be generated between [0,1] Obey a uniformly distributed random number $\delta$ seed, if it satisfies

(13), then accept the new solution

$$\delta \le e^{(-\Delta E/T)} \tag{13}$$

At the current temperature, perform all the algorithm operations. The termination condition is to perform all iterations at this temperature, then lower the temperature according to the temperature update rule, and proceed with the iteration operation of the next temperature. The final temperature drops to the preset minimum temperature and then stops. The temperature update rule is (14).

$$T(t) = T_0 \cdot \exp(-r \cdot t) \tag{14}$$

where $r$ is the cooling rate and $T_0$ is the initial temperature. To ensure that the global optimum can be searched, the value of $T_0$ should be set large enough.

Since that Lévy Flight generates new solutions through neighborhood movement, if the evaluation index of the new solution is not as good as that of the old solution, cuckoo will not accept the new solution, which will lead to the possibility of the algorithm falling into local optimal. Therefore, after CS&SA produces a new solution, a simulated annealing mechanism is added to intervene in the update of the solution. SA selected whether to accept the new solution or not according to the acceptance probability, and this mechanism would help the algorithm fall into the local optimal. A new solution will have a higher acceptance probability at high temperatures, even if the evaluation index of the new solution is not as good as the original solution. With the decrease of temperature, the acceptance probability decreases gradually, which means that the algorithm converges from global search to local search. At the same time, the addition of simulated annealing mechanism will increase the iteration times of the algorithm, enhance the searching ability of Lévy Flight, and ensure the quality of understanding.

After generating a new solution ($NewP_i$) through Levy flight, calculate its evaluation index. Randomly select a solution ($P_j$) for comparison. If the fitness of $P_j$ is less than the fitness of $NewP_i$, then as long as (9) is satisfied, the new solution will be accepted. When the fitness of the two solutions are the same, due to the resource constraints of RCPSP, we choose a solution with low resource utilization. The calculation formula for resource utilization is:

$$\rho = \frac{r_{ik}}{R_k} \tag{15}$$

Where $r_{ik}$ is the number of resources occupied by the activity, and $R_k$ is the total amount of resources.

In this way, the population will jump out of the current search space if it accepts the poor quality solution at the current temperature, and avoid the algorithm falling into the local optimal. Then proceed to other operations. The update pseudocode for the population is shown in Fig.12.

```
Function SA (P, T0, Ts, r, K)
Pi = rand (P);
while (T0 > Ts)
    for k = 1: K
        NewPi = Levyflights (Pi);
        Pj = rand (P);
        if Fitness (NewPi) < Fitness (Pj)
            Pj = NewPi;
        else if Fitness (NewPi) == Fitness (Pj)
                if ρ (NewPi) < ρ (Pj)
                    Pj = NewPi;
                end if
        else
            p = exp (Fitness (Pj) – Fitness (NewPi)) / T0;
            if rand () <= p
                Pj = newPi;
            end if
        end if
    end for
    T0 = T0 * r;
end while
Output: newPi
```

The population $P$, minimum temperature $T_s$, Maximum temperature $T_0$, annealing rate $r$.

Fig.12. Pesudo-code of SA

### E. PROCESS OF CS&SA FOR RCPSP

In view of the complexity of RCPSP, this paper firstly redesigned the encoding and decoding mode of CS&SA, coded the priority of each activity of individuals through RK, and used SSGS to decode the population. Secondly, the updating formulas that affect the distance and direction of Lévy flight are redesigned to make the algorithm self-adaptive in searching the population.

Then, three neighborhood updating strategies are added to meet the updating requirements of the algorithm at different stages, which can enhance the local capability. Finally, the algorithm is mixed with simulated annealing mechanism to interfere with the updating of individuals, so that the algorithm can accept individuals with different fitness values with a certain probability in each iteration, which can improve the global optimization ability. In order to increase the diversity of the population, the algorithm will discard the poor quality individuals with a certain probability $P_a$ and replace them with new individuals. The algorithm pseudo-code is shown in Fig.13.

**Algorithm** CS & SA ()

Initialize $N$, $P_a$, $Iter$;
$P$ = rand (0,1);
**for** $i$ = 1 : $N$
  $P_i$= Serials ($P_i$, $A$);
**end for**
**while** ($iter$ < $Iter$)
  Calculate population fitness;
  $NewP$ = SA ($P$, $T_0$, $T_s$, $r$, $K$);
  $r$ = rand (0,1);
  **if** $r > P_a$
    $NewP_w$ = rand (0, 1);
    $NewP_w$ = SerialSGS ($NewP_w$, $A$);
    $P_w = NewP_w$;
  **end if**
  Record the global optimal solution $P_{best}$;
  $iter = iter$ +1;
**end while**

**Output**: $P_{best}$;

Population size $N$, find the probability $P_a$,
Maximum iteration number $Iter$, Activity priority matrix $A$

Fig.13. Pesudo-code of CS&SA

## V. Analysis and Discussion of Experimental Results

### A. Benchmark Instance

In the choice of experimental data, this paper selects the standard instance library PSPLIB [26]. The PSPLIB is the RCPSP classic instance data, comprising a total of 2040 projects in four groups of 30,60,90 and 120. At present, the PSPLIB only provides the optimal solution for the J30 instance through the precise algorithm. Because RCPSP is a highly complex engineer problem, the optimal solution after J60 is still unknown, but the lower bound of the solution is given.

### B. Parameter settings

The algorithm proposed in this paper has seven parameters that can be set, among which three parameters are from CS: Population size ($N$), discovery probability ($P_a$), total number of iterations ($Iter$). Three parameters are derived from SA: The initial temperature ($T_0$), termination of the temperature ($T_s$), annealing rate ($r$). After the initial test, the specific values in table I are determined. The test shows that the efficiency and optimization of the algorithm are better when the current value is set.

TABLE I
ALGORITHM PARAMETERS

| Parameter | Value |
|---|---|
| Population size ($N$) | 25 |
| Discovery probability ($P_a$) | 0.25 |
| Total number of iterations ($Iter$) | 1000 |
| Initial temperature ($T_0$) | $e^{10}$ |
| Termination of the temperature ($T_s$) | 1 |
| Annealing rate ($r$) | 0.997 |

### C. Performance Evaluation

1) Convergence analysis of CS&SA

First of all, this paper selects a set of data of different sizes of J30, J60 and J120, and tests the effectiveness and convergence of CS&SA algorithm under 200 iterations.

Small-scale data analysis. Select the J301_8 instance in the PSPLIB. J301_8 is a total of 32 tasks, including two virtual tasks. There are four types of renewable resources, among which resource $R_1$ is 12, resource $R_2$ is 14, resource $R_3$ is 12 and resource $R_4$ is 10. The optimal value of the example is 53 by the precise algorithm. The convergence of CS and CS&SA after running for 200 times is shown in Fig.14.

Medium scale data analysis. Select the J601_2 instance in the PSPLIB. J601_2 has 62 tasks, including two virtual tasks. There are four types of renewable resources, among which resource $R_1$ is 13, resource $R_2$ is 15, resource $R_3$ is 14 and resource $R_4$ is 14. The lower bound of the optimal value is 68. The convergence of basic CS and CS&SA after running for 200 times is shown in Fig.15.

Large-scale instance testing. Select the J1201_1 instance in the PSPLIB. J1201_1 is a total of 122 tasks, including two virtual tasks. There are four types of renewable resources, in which resource $R_1$ is 14, resource $R_2$ is 12, resource $R_3$ is 13, and resource $R_4$ is 9. The lower bound of the optimal value is 104. The convergence of basic CS and CS&SA after running for 200 times is shown in Fig.16.

Analysis Fig.14-Fig.16. For the convergence of the algorithm, in the small-scale data J301_8, CS converges to the optimal value in generation 50. Due to the addition of simulated annealing strategy to CS&SA, after the algorithm has converged to 50 generations, the population accepts individuals with poor fitness values, which leads to a jump. In the medium-scale data J601_2 and large-scale data J1201_1, the number of times of simulated annealing strategy is increased, and the convergence curve starts to jump frequently due to the increasing data size. As for the convergence accuracy of the algorithm, because the algorithm has changed the self-adaptability of Lévy Flights, the convergence accuracy of CS&SA in different scales is better than CS, proving that CS&SA can find relatively good solutions in scheduling problems in different scales, and the convergence and optimization accuracy are better than CS.

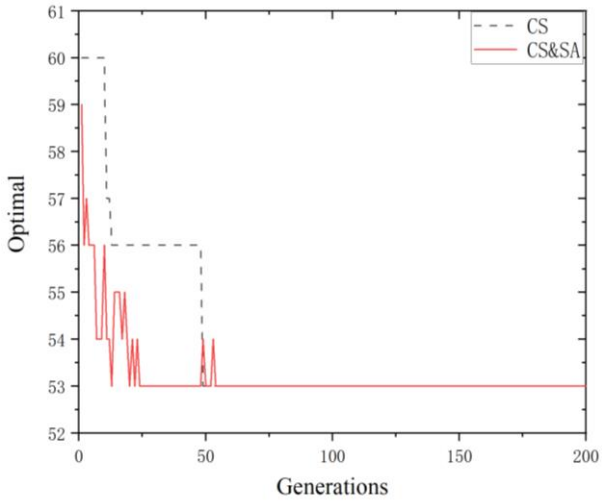


Fig.14. J301_8 optimal convergence curve

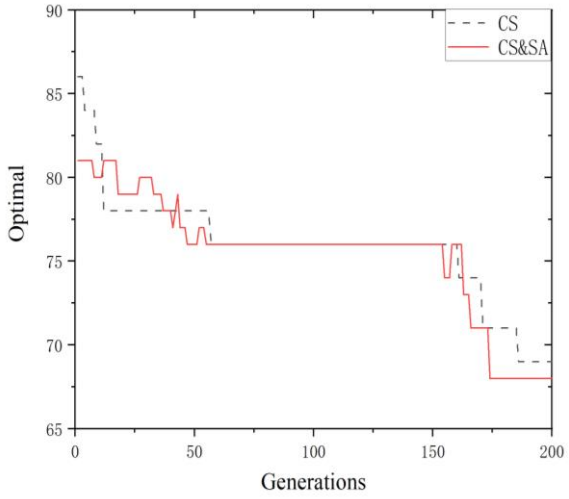


Fig.15. J601_2 optimal convergence curve

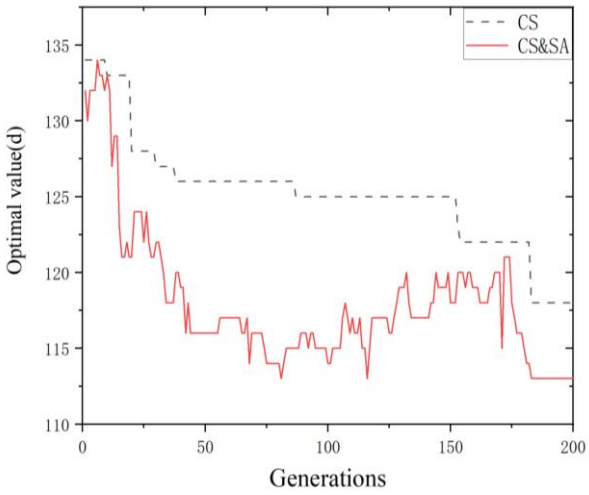


Fig.16. J1201_1 optimal convergence curve

2) Algorithm optimization analysis

In this paper, 20 sets of data of different sizes of J30, J60 and J120 were randomly selected and run independently for 50 times under 500 iterations, and the deviation of the algorithm from the best solution was calculated by (16) [27]:.

$$Dev(i) = \frac{(solution - opt)}{opt} .100 \quad (16)$$

where *Solution* is the average of the algorithm. *opt* is the optimal solution given by PSPLIB. Since J60, J90, and J120 do not provide the best solution, we use the lower bound provided by PSPLIB as a reference for the optimal value.

According to Table II- IV, for the convergence of the optimal solution (lower bound), the two algorithms can converge to the optimal value in the J30 small-scale data, while in the J60 medium-scale data and J120 large-scale data, only some instances of the two algorithms converge to the lower bound due to the complexity of the RCPSP. But CS&SA is closer to the lower bound than CS.

We compare the average deviation in Table II- IV, as shown in Fig.17-Fig.19. In the small-scale data of J30, the average deviation of CS is about 0.5%, and that of CS&SA is about 0.3%. In the J60 scale data, the average deviation of CS is about 13%, and that of CS&SA is about 12%. In the J120 large-scale data, the average deviation of CS is around 21%, and the average deviation of CS&SA is around 20%. It can be seen that the average error of CS&SA in solving scheduling problems of different scales is smaller than CS, which proves that CS&SA has better optimization performance than CS.

Table II
Average Deviation of J30 under 500 iterations

| Num | Data | Opt | Value | | Average | | Dev% | |
|---|---|---|---|---|---|---|---|---|
| | | | CS | CS&SA | CS | CS&SA | CS | CS&SA |
| 1 | 301-9 | 49 | 49 | 49 | 49.25 | 76.59 | 13.23 | 12.64 |
| 2 | 305-3 | 76 | 76 | 76 | 76.40/ | 80.71 | 13.76 | 12.10 |
| 3 | 305-7 | 76 | 76 | 76 | 76.41 | 119.02 | 14.48 | 12.29 |
| 4 | 306-2 | 51 | 51 | 51 | 51.24 | 82.92 | 13.24 | 12.64 |
| 5 | 306-8 | 39 | 39 | 39 | 39.21 | 93.18 | 13.56 | 12.27 |
| 6 | 309-2 | 92 | 92 | 92 | 92.48 | 75.81 | 14.04 | 13.15 |
| 7 | 309-5 | 70 | 70 | 70 | 70.32 | 92.12 | 13.44 | 12.35 |
| 8 | 3010-6 | 44 | 44 | 44 | 44.18 | 69.63 | 13.88 | 12.31 |
| 9 | 3013-4 | 72 | 72 | 72 | 72.36 | 68.46 | 13.62 | 12.24 |
| 10 | 3014-3 | 58 | 58 | 58 | 58.31 | 95.61 | 12.68 | 11.18 |

Table III
Average Deviation of J60 under 500 iterations

| Num | Data | Opt | Value | | Average | | Dev% | |
|---|---|---|---|---|---|---|---|---|
| | | | CS | CS&SA | CS | CS&SA | CS | CS&SA |
| 1 | 601-2 | 68 | 70 | 68 | 76.99 | 76.59 | 13.23 | 12.64 |
| 2 | 601-7 | 72 | 80 | 72 | 81.90 | 80.71 | 13.76 | 12.10 |
| 3 | 605-2 | 106 | 115 | 106 | 121.34 | 119.02 | 14.48 | 12.29 |
| 4 | 605-6 | 74 | 79 | 76 | 83.97 | 82.92 | 13.24 | 12.64 |
| 5 | 605-9 | 83 | 90 | 84 | 93.25 | 93.18 | 13.56 | 12.27 |
| 6 | 606-4 | 67 | 70 | 67 | 76.40 | 75.81 | 14.04 | 13.15 |
| 7 | 609-1 | 82 | 88 | 85 | 93.02 | 92.12 | 13.44 | 12.35 |
| 8 | 6010-2 | 62 | 66 | 62 | 70.60 | 69.63 | 13.88 | 12.31 |
| 9 | 6014-1 | 61 | 65 | 63 | 69.30 | 68.46 | 13.62 | 12.24 |
| 10 | 6017-1 | 86 | 89 | 86 | 96.60 | 95.61 | 12.68 | 11.18 |

Table IV
Average Deviation of J120 under 500 iterations

| Num | Data | Opt | Value | | Average | | Dev% | |
|---|---|---|---|---|---|---|---|---|
| | | | CS | CS&SA | CS | CS&SA | CS | CS&SA |
| 1 | 1201-3 | 125 | 135 | 132 | 152.43 | 151.07 | 21.95 | 20.86 |
| 2 | 1201-5 | 112 | 121 | 119 | 135.81 | 134.87 | 21.62 | 20.42 |
| 3 | 1201-10 | 108 | 120 | 118 | 131.47 | 129.99 | 21.74 | 20.37 |
| 4 | 1202-1 | 87 | 99 | 96 | 106.40 | 105.46 | 22.31 | 21.22 |
| 5 | 1202-5 | 103 | 113 | 111 | 124.87 | 123.73 | 21.24 | 20.13 |
| 6 | 1205-8 | 78 | 89 | 86 | 94.89 | 93.76 | 21.66 | 20.21 |
| 7 | 1205-10 | 92 | 104 | 102 | 111.46 | 110.53 | 21.16 | 20.15 |
| 8 | 1206-6 | 140 | 155 | 151 | 171.06 | 169.66 | 22.19 | 21.19 |
| 9 | 1208-5 | 99 | 112 | 108 | 120.05 | 119.08 | 21.27 | 20.29 |
| 10 | 1209-7 | 80 | 102 | 100 | 97.08 | 96.08 | 21.35 | 20.11 |

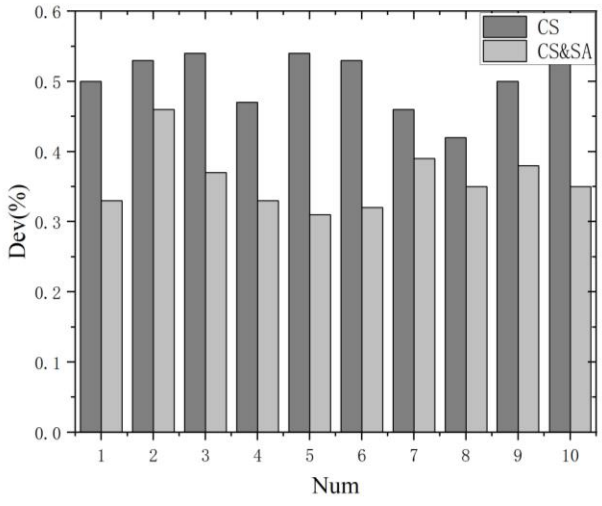


Fig.17. Mean deviation comparison of J30 for CS and CS&SA

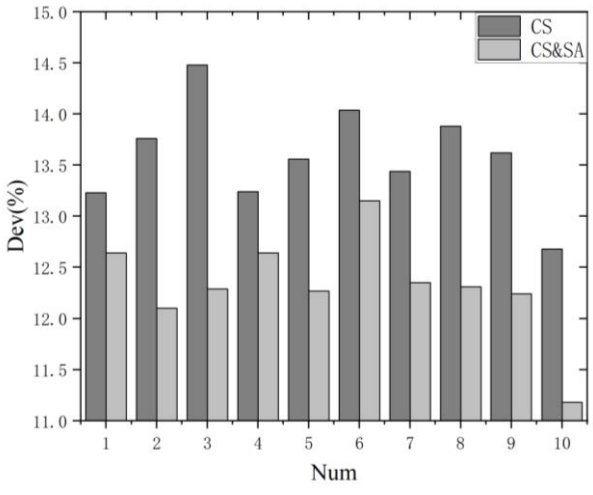


Fig.18. Mean deviation comparison of J60 for CS and CS&SA

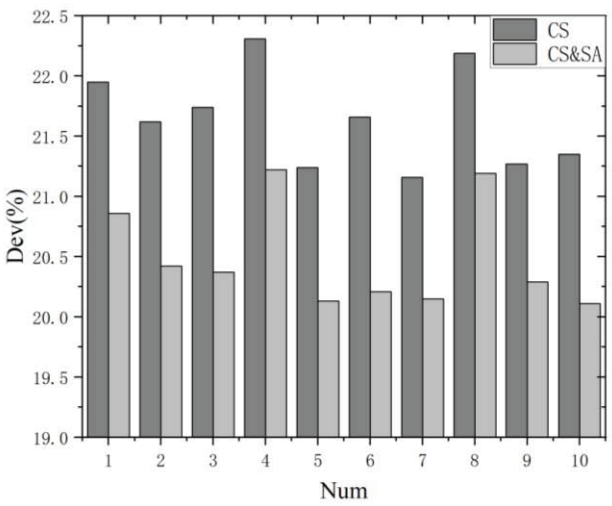


Fig.19. Mean deviation comparison of J120 for CS and CS&SA

3) Compare with other algorithms

We analyze the deviation of J30, J60 and J120 data by (17). Through this formula, we can clearly see the correctness of the algorithm in the whole data set.

$$Dev_{avg\%} = \frac{\sum_i |Dev_i|}{Instance} \cdot 100 \tag{17}$$

where the numerator of the fraction is to use (16) to analyze a single instance in each group of data and to sum it. *Instance* is the total number of instances per set of data.

TABLE V
PERFORMANCE COMPARISON WITH OTHER ALGORITHMS FOR J30

| Algorithm | $Dev_{avg}$(%) | | |
|---|---|---|---|
| | J30 | J60 | J120 |
| CS&SA | 0.25 | 11.21 | 19.83 |
| CS | 0.42 | 12.53 | 20.87 |
| ACOSS[28] | 0.14 | 11.72 | 35.19 |
| GA[29] | 0.34 | 12.21 | 35.39 |
| SA[30] | 0.38 | 12.75 | 42.81 |
| SS-FBI[31] | 0.27 | 11.73 | 35.22 |
| TS[32] | 0.46 | 12.97 | 40.86 |

Table V compares the average deviation with other algorithms under 1000 iterations. It can be seen that under the same iteration times, the average deviation of CS&SA in J30 small-scale data scheduling instances is 0.25%, which is better than most optimization intelligent optimization algorithms. The average deviation of the average data scheduling instance in J60 is 11.21%, which is slightly better than other intelligent optimization algorithms. The average deviation of J120 large-scale data scheduling instance is 19.83%, which is significantly better than other optimized intelligent algorithms. Because Lévy flights adaptive mechanism is improved in the algorithm, the local optimization ability of the algorithm is enhanced. The simulated annealing strategy added at the same time enhances the global convergence, so the optimization accuracy of the algorithm is significantly improved. J30 and J60 are relatively small in scale, and the improvement of the optimization efficiency is not obvious. However, in the J120 large-scale data set, the optimization efficiency is obviously superior to other optimization intelligent algorithms. It is proved that the improvement of CS&SA in RCPSP is feasible and the performance of the algorithm is superior.

## VI. CONCLUSIONS

This paper presents a CS&SA algorithm and solves the RCPSP. CS suitable for solving continuous problems is applied to the field of combinatorial optimization problems by redefining population and objective function. By changing the flight factors of Lévy flight, the algorithm can be adaptive in neighborhood updating. At the same time, the algorithm adds a simulated annealing mechanism in the population update stage to intervene in the update of the solution, so that the algorithm accepts some solutions with poor quality and avoids the algorithm from falling into a local optimum.

The algorithm uses a classical test example PSPLIB to test the efficiency of the algorithm. A single example of the algorithm is tested to verify the effectiveness of CS&SA. The optimization rate of the algorithm is verified by comparing other algorithms with the same iteration times. The test results show that the algorithm is significantly better than other algorithms in terms of efficiency and quality.

CS&SA is an effective and practical optimization algorithm, and it has high expansibility. In the future work, we will be dedicated to continue improving and optimizing the algorithm and apply it to solve more complex engineering problems.

## REFERENCES

[1] K. Bibiks, Y. F. Hu, J. P. Li, P. Prashant, and A. Smith, "Improved discrete cuckoo search for the resource-constrained project scheduling problem," *Applied Soft Computing*, vol. 69, pp. 493-503, 2018.

[2] J. Blazewicz, J. K. Lenstra, and A. H. G Rinnooy Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete applied mathematics,* vol. 5, no. 1, pp. 11-24, 1983.

[3] Georgios M. Kopanos, Thomas S. Kyriakidis, Michael C. Georgiadis, "New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems," *Computers and Chemical Engineering,* vol. 68, pp. 96-106, 2014.

[4] Z. Chu, Z. Xu, H. Li, "New heuristics for the RCPSP with multiple overlapping modes," *Computer and Industrial Engineering,* vol. 131, pp. 146-156, 2019.

[5] S. Hartmann and R. Kolisch, "Experimental evaluation of state -of-the-art heuristics for the resource-constrained project schedu -ling problem," *European Journal of Operational Research,* vol. 127, no. 2, pp. 394-407, 2000.

[6] A. Sprecher, "Scheduling resource-constrained projects competit -ively at modest memory requirements," *Management Science,* vol. 46, no. 5, pp. 710-723, 2000.

[7] E. Demeulemeester and W. Herroelen,"A branch-and-bound pr-ocedure for the multiple resource-constrained project scheduling problem," *Management science,* vol. 38, no. 12, pp. 1803-181 8, 1992.

[8] P. Brucker, S. Knust, A. Schoo, and O. Thiele, "A branch and bound algorithm for the resource-constrained project scheduling problem," *European journal of operational research,* vol. 107, no. 2, pp. 272-288, 1998.

[9] H. Chen, G. Ding, J. Zhang, S. Qin, "Research on priority rules for the stochastic resource constrained multi-project scheduling problem with new project arrival," *Computers and Industrial Engineering,* vol. 137, p. 106060, 2019.

[10] R. A.-V. Olaguíbel and J. M. T. Goerlich, "Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis," in *Advances in project scheduling*: Elsevier, pp. 113-134,

1989.
[11] Z. Chen, E. Demeulemeester, S. Bai, and Y. Guo, "Efficient priority rules for the stochastic resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 270, no. 3, pp. 957-967, 2018.
[12] J. Rezaeian, F. Soleimani, S. Mohaselafshary, and A. Arab, "Using a meta-heuristic algorithm for solving the multi-mode resource-constrained project scheduling problem," *International Journal of Operational Research*, vol. 24, no. 1, pp. 1-16, 2015.
[13] B. Roy and A. K. Sen, "Meta-heuristic techniques to solve resource-constrained project scheduling problem," in *International conference on innovative computing and communications*, 2019, pp. 93-99: Springer.
[14] J. Liu, Y. Liu, Y. Shi, and J. Li, "Solving Resource-Constrained Project Scheduling Problem via Genetic Algorithm," *Journal of Computing in Civil Engineering*, vol. 34, no. 2, p. 04019055, 2020.
[15] S. Ghafoori and M. R. Taghizadeh Yazdi, "Proposing a Multi-Objective Mathematical Model for RCPSP and Solving It with Firefly and Simulated Annealing algorithms," *Modern Researches in Decision Making*, vol. 1, no. 4, pp. 117-142, 2017.
[16] N. Kumar and D. P. Vidyarthi, "A model for resource-constrained project scheduling using adaptive PSO," *Soft Computing*, vol. 20, no. 4, pp. 1565-1580, 2016.
[17] A. Ouaarab, B. Ahiod, X-S. Yang, Neural Computing, and Ap -plications, "Discrete cuckoo search algorithm for the travelling salesman problem," *J.*, vol. 24, no. 7-8, pp. 1659-1669, 2014.
[18] Y. Zhou and Y. Chen, "Business process assignment optimization," in *IEEE International Conference on Systems, Man and Cybernetics*, 2002, vol. 3, p. 6 pp. vol. 3: IEEE.
[19] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights," in *2009 World congress on nature & biologically inspired computing (NaBIC)*, 2009, pp. 210-214: IEEE.
[20] Ilya Pavlyukevich, "Lévy flights, non-local search and simulated annealing," *Journal of Computational Physics*, vol. 226, no. 2, pp. 1830-1844, 2007.
[21] G. Viswanathan, E. Raposo, and M. Da Luz "Lévy flights and superdiffusion in the context of biological encounters and random searches," *Physics of Life Reviews*, vol. 5, no. 3, pp. 133-150, 2008.
[22] J. Weglarz, "Handbook on recent advances in project scheduling," ed: Kluwer, Amsterdam, 1998.
[23] M. R. Sierra, C. Mencía, and R. Varela "New schedule generation schemes for the job-shop problem with operators," *Journal of Intelligent Manufacturing*, vol. 26, no. 3, pp. 511-525, 2015.
[24] R. N. Mantegna, "Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes," *Physical Review E*, vol. 49, no. 5, p. 4677, 1994.
[25] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth and A. H. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087-1092, 1953.
[26] A. Sprecher, R. Kolisch, and A. Drexl "Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 80, no. 1, pp. 94-102, 1995.
[27] S. U. Kadam and S. U. Mane, "A genetic-local search algorithm approach for resource constrained project scheduling problem," in *2015 International Conference on Computing Communication Control and Automation*, 2015, pp. 841-846: IEEE.
[28] Chen W, Shi Y, Teng H, et al, "An efficient hybrid algorithm for resource-constrained project scheduling," *Information Sciences*, vol. 180, no. 6, pp. 1031-1039, 2010.
[29] Valls V, Ballestin F, Quintanilla S. "Justification and RCPSP: A technique that pays," *European Journal of Operational Research*, vol. 165, no. 2, pp. 375-386, 2005.
[30] Bouleimen H Lecocq. "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *European Journal of Operational Research*, vol. 149, no. 2, pp. 268-283, 2003.
[31] Debels D, De Reyck B, Leus R, et al. "A hybrid scatter search/electromagnetism meta-heuristic for project scheduling," *European Journal of Operational Research*, vol. 169, no. 2, pp. 638-653, 2006.
[32] Nonobe K, Ibaraki T. "Formulation and tabu search algorithm for the resource constrained project scheduling problem," in *Essays and surveys in metaheuristics*: Springer, 2002, pp. 557-588.