Course Report for Intelligent Optimization Algorithms

Fan JIN

Abstract—This homework report has 3 parts. In the first part, simulated annealing (SA) and differential evolution (DE) are used to test the eggholder function. Greedy metasearch is also used to identify the parameter sensitivity of SA and DE. In the second part, SA is employed for combinatorial optimization, the TSP. In the thrid part, a work that improves the global convergence capability of the particle swarm optimization (PSO) is summarized. Two key improvements are formulated with reasons why they work, as long as the two experiments on the drilling path problem.

Index Terms—Intelligent optimization algorithm, simulated annealing, differential evolution, particle swarm

I. Question 1: Function Optimization

A. Target function: Eggholder

As intelligent optimization algorithms have emerged as a new paradigm of function optimization, many test functions have been proposed to serve as benchmarks [1]. In this report, the Eggholder function is chosen to test two intelligent optimization algorithms: the simulated annealing (SA) and the differential evolution (DE).

The Eggholder function is defined as

$$\begin{aligned} f(x,y) = \\ &- (y+47)\sin\sqrt{\left|\frac{x}{2} + (y+47)\right|} \\ &- x\sin\sqrt{|x-(y+47)|}, \end{aligned}$$

where $-512 < x, y < 512$. Its global minimum is known as $f(512, 404.2319) = -959.6407$. (Figure 1)

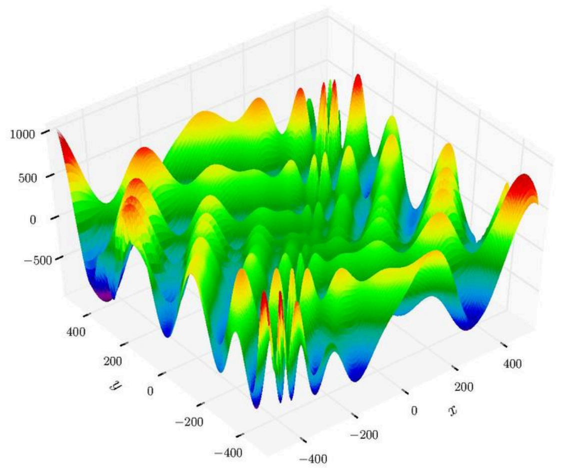

Fig. 1. Eggholder function [1]

Fan JIN 2015011506 jinf15@mails.tsinghua.edu.cn

Eggholder function is challenging bacause i) it has boundary constraints on both x and y, and ii) the global minimum is at the boundary of $x = 512$, and iii) there are dozens of local minimums throughout the space.

B. Implementation of SA and DE

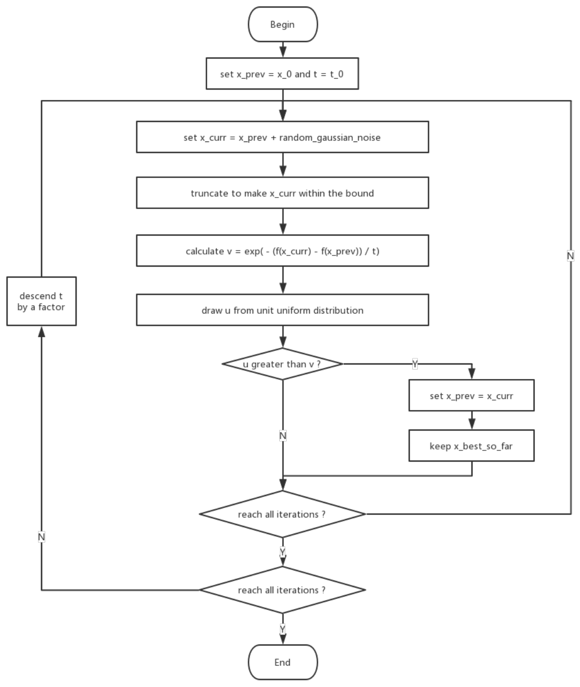
Begin
set x_prev = x_0 and t = t_0
set x_curr = x_prev + random_gaussian_noise
truncate to make x_curr within the bound
calculate v = exp(- (f(x_curr) - f(x_prev)) / t)
draw u from unit uniform distribution
u greater than v ?
Y
set x_prev = x_curr
keep x_best_so_far
N
reach all iterations ?
N
Y
reach all iterations ?
N
descend t by a factor
Y
End

Fig. 2. Flow chart of simulated annealing (SA)

To make full use of vectorized operations in MATLAB, we use "arrayfun" function to evaluate the target function in batches. The "encode" and "decode" functions are also written using vectorized operations to gain more performance. (Figure 2 and 3)

C. Experiments

1) The trivial sphere function: The sphere function is quite trivial. Defined as

$$f(x,y) = x^2 + y^2,$$

it is convex and has no local minimums but a global minimum.

We tested our SA and DE implementations on sphere function. The result shows that the algorithms we implemented give the global minimum with convergence as expected. (Figure 4 and 5)

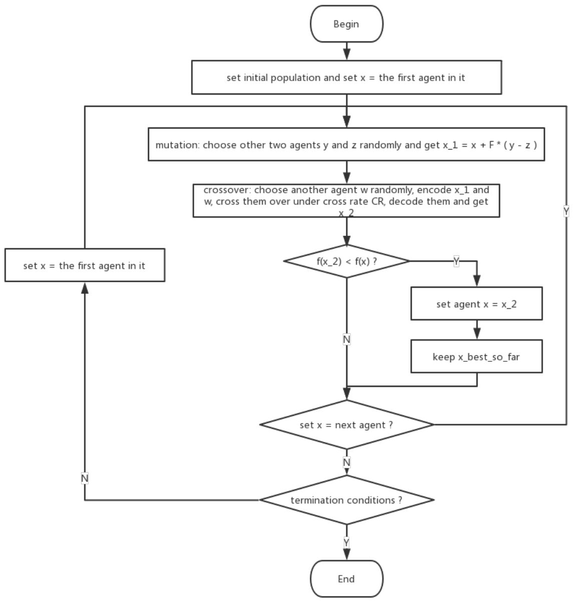
Begin
set initial population and set x = the first agent in it
mutation: choose other two agents y and z randomly and get x_1 = x + F * (y - z)
crossover: choose another agent w randomly, encode x_1 and w, cross them over under cross rate CR, decode them and get x_2
f(x_2) < f(x) ?
Y
set agent x = x_2
keep x_best_so_far
N
set x = next agent ?
Y
N
set x = the first agent in it
termination conditions ?
N
Y
End

Fig. 3. Flow chart of differential evolution (DE)

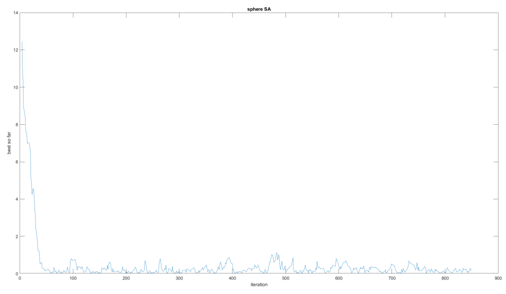

Fig. 4. Performance of SA on the sphere function

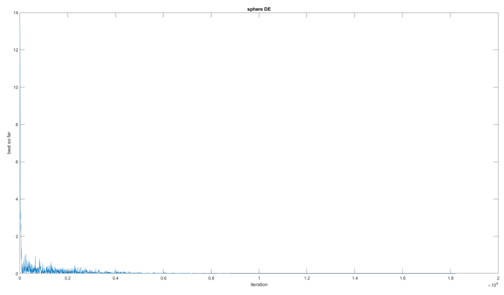

Fig. 5. Performance of DE on the sphere function

2) The eggholder function: Repeated for 20 times, the performance (best value searched and time cost) of each algorithm is recorded. (Table I and II) (Figure 6 and 7)

Alg.	best so far (min)	b.s.f. (mean)	b.s.f. (max)
SA	-959.6406	-872.9803	-555.5534
DE	-959.6407	-959.6404	-959.6386

TABLE I
Performance after 20th repitition

Alg.	time cost in sec (min)	t.c. (mean)	t.c. (max)
SA	0	0.0055	0.0469
DE	5.2656	6.6070	8.9688

TABLE II
Time cost after 20th repitition

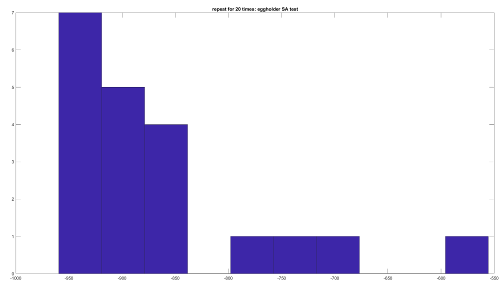

Fig. 6. Performance of SA after 20th repitition

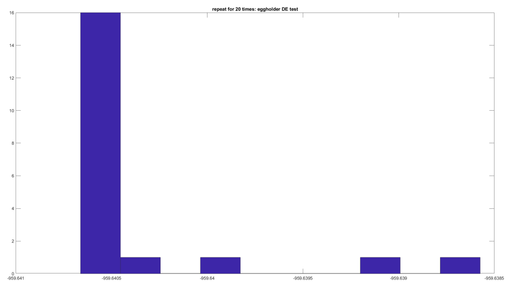

Fig. 7. Performance of DE after 20th repitition

- Performance: DE wins. DE always manages to find the global minimum at -959.64, while SA cannot. Although SA did find the global minimum occasionally, the average performance is -872, which is still far from the best value.
- Time cost: SA wins. SA has little time cost, consuming 47 milliseconds in the worst case. In contrast, it always takes seconds for DE to complete, because it is population-based rather than individual-based. Obviously, the time cost of DE is proportional to the population size. With a small population size, DE has a time cost affordable in certain circumstances.

3) Plots of best-so-fars: From Figure 8 and 9, we see that

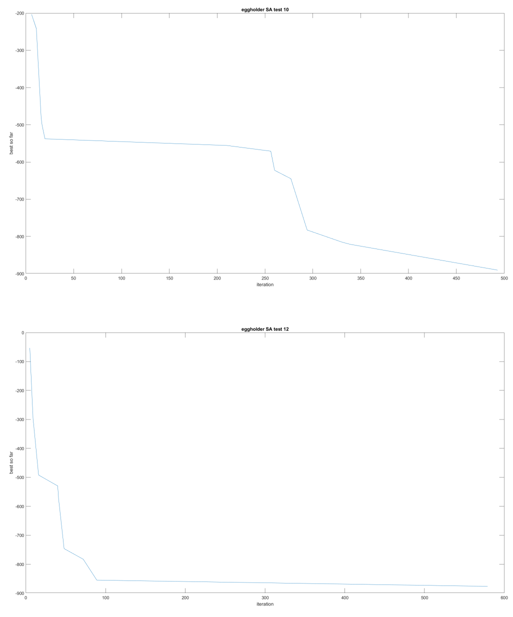

Fig. 8. 10th and 12th test of SA

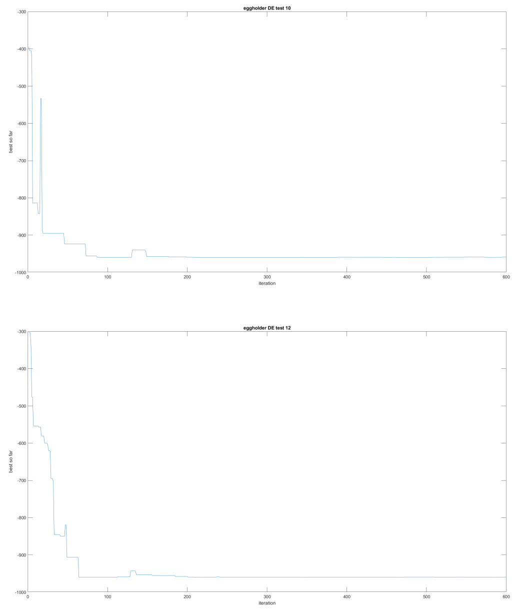

Fig. 9. 10th and 12th test of DE

- The number of iterations needed for SA varies a lot. For test 10, it took 400 epochs. For test 12, only 200 is enough.
- DE is much more stationary. Although it is not a monotonic descent, it converges very quickly.

D. Sensitivity Analysis

Hyper tuning could be one of the most tricky parts in an intelligent optimization algorithm. For the tests above, the following hyperparameters are used. (Table III and IV)

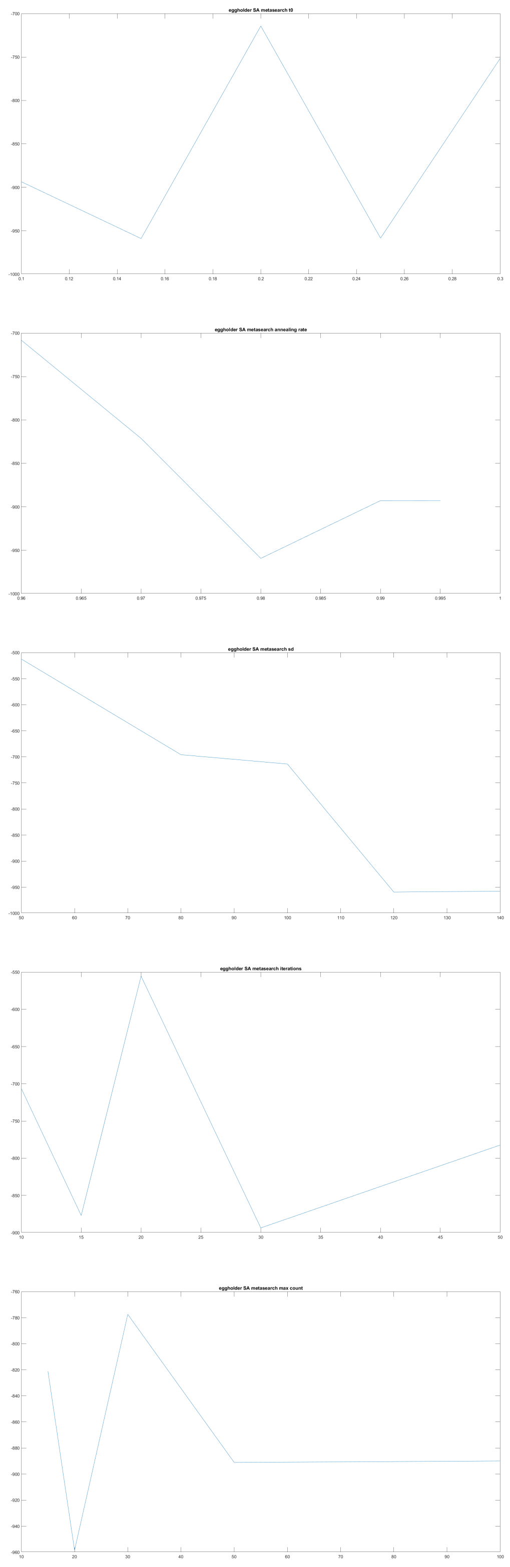

Fig. 10. Greedy sensitivity analysis for SA

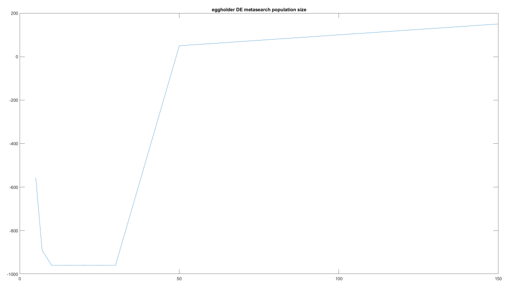

Fig. 11. Greedy sensitivity analysis for DE

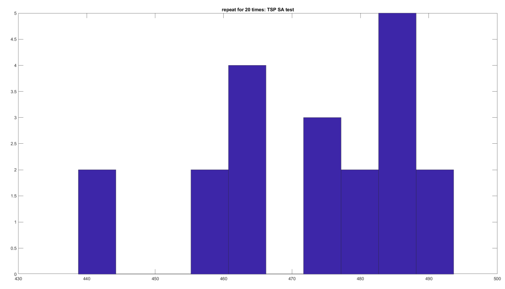

Fig. 12. Performance of SA after 20th repitition

From Figure 10, we find that SA is sensitive to hyperparameters, especially initial temperature and annealing rate. This can be easily interpreted: the temperature determines which side the balance to fall between exploitation and exploration. Luckily, the annealing rate we chose is appropriate among other four choices. Other hyperparametrs also count. For example, the standard deviation (std.) of the Gaussian proposal function also plays a role in the balance between exploitation and exploration.

From Figure 11, we see that DE depends on the population size. It works fine with a population size smaller than 40. With a size of 100, it needs more iterations to explore. If the iteration number is fixed, DE will suffer lack of exploration, which is also called "prematurity".

II. Question 2: Combinatorial Optimization

A. Benchmark Problem: 50-city TSP

The benchmark problem is a TSP of 50 cities. The best result known is 427.855 [2].

B. Experiment

Repeated for 20 times, the performance (best value searched and time cost) of each algorithm is recorded. (Table V and VI) (Figure 12)

Compared with the best result known, 427.855, the simulated annealing achieves similar performance in the best case. The worst case is also acceptable in most cases. The time cost, however, is very high because too many iterations are needed before it could converge.

Hyperparameter	Default value
initial temperature	0.2
annealing rate	0.98
std. for proposal func.	100
inner loop iterations	20
outer loop iterations	30

TABLE III
Default Hypers for SA

Hyperparameter	Default value
F (mutation)	0.5
cross rate	0.3
loop iterations	600

TABLE IV
Default Hypers for DE

C. Discussion

For the 10th experiment (Figure 13), we see that the simulated annealing converges slowly. At first, exploration outweights exploitation as the temperature is high and therefore worse results are more likely to be accepted. As the temperature decreases, exploitation is more and more preferred, making the process converge and halt.

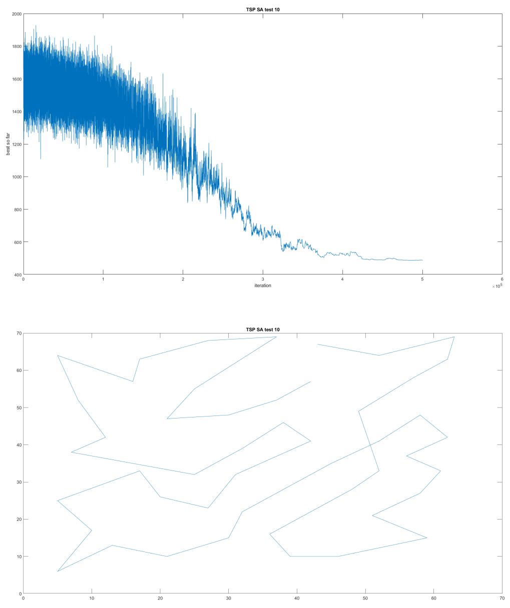

Fig. 13. Test case #10

Here are the final remarks:

Alg.	best so far (min)	b.s.f. (mean)	b.s.f. (max)
SA	438.7710	472.3798	493.6293

TABLE V
Performance after 20th repitition

Alg.	time cost in sec (min)	t.c. (mean)	t.c. (max)
SA	312.9063	341.9633	398.1719

TABLE VI
Time cost after 20th repitition

- Combinatorial optimization is usually more time-consuming than function optimization with the same size. This is partly due to the time cost of mutation and crossover, whereas the time cost of real number calculation is neglegible.
- Hyper tuning is the most tricky part in SA, particularly for the temperature. While an improper annealing rate would only slower the convergence process, an improper initial temperature could impede convergence from the very beginning. I tried different initial temperatures of a geometric series, ranging from 0.01, 0.1, 1, 10, 100, 1000, etc. After locating the optimal initial temperature t_0 between 0.1 and 1, I used binary search to find the optimal one. I finally chose $t_0 = 0.2$.
- To speed up the process and add more exploration to it, I modified SA a bit: In my implementation, the mutation can consist of several consecutive swaps, rather than only one swap. I tested it and only 400,000 iterations are needed with multiple swaps, whereas they are not enough with single swap.

III. Question 3: Paper Reading about Drilling Path Optimization

In a machining center, holes-machining is a typical operation. For instance, the production of PCB (printed circuit board) requires an intense procedure of drilling (Figure 14). The drilling process takes much time, as much as 70% of total operation time, due to slow motion of the cutting tools. [3] Therefore, it is worth it reducing that time by finding an optimal drilling path. Similar to the TSP, the drilling path problem is NP-hard, and therefore intelligent optimization algorithms are needed. In this section, a work using swarm particle algorithms [3] will be summarized.

Zhu's paper reviews related work, including the Hopfield algorithm, evolutionary ant colony algorithm, genetic algorithm, tabu search, simulated annealing, etc. It follows that the drilling path problem can be expressed as a single-objective TSP. Hence, the particle swarm optimization (PSO) can be used. However, standard PSO suffers in the drilling path problem for its slow convergence. In their work, they propose a mechanism that comes with global convergence capability. This is their major contribution in this paper.

A. Standard PSO

In the standard PSO, the position of a particle is corresponding to a solution of the problem. Here in the drilling path problem, the position is encoded into a discrete space. Similar to TSP, the encoded space is the permutations of an integer series indicating the order that the cutting tool should follow.

The PSO is a swarm intelligence algorithm inspired by bird swarm seeking food. A swarm of particles, like birds, are moving in the search space to find a optimal location, like the food. The fitness value in each location, defined

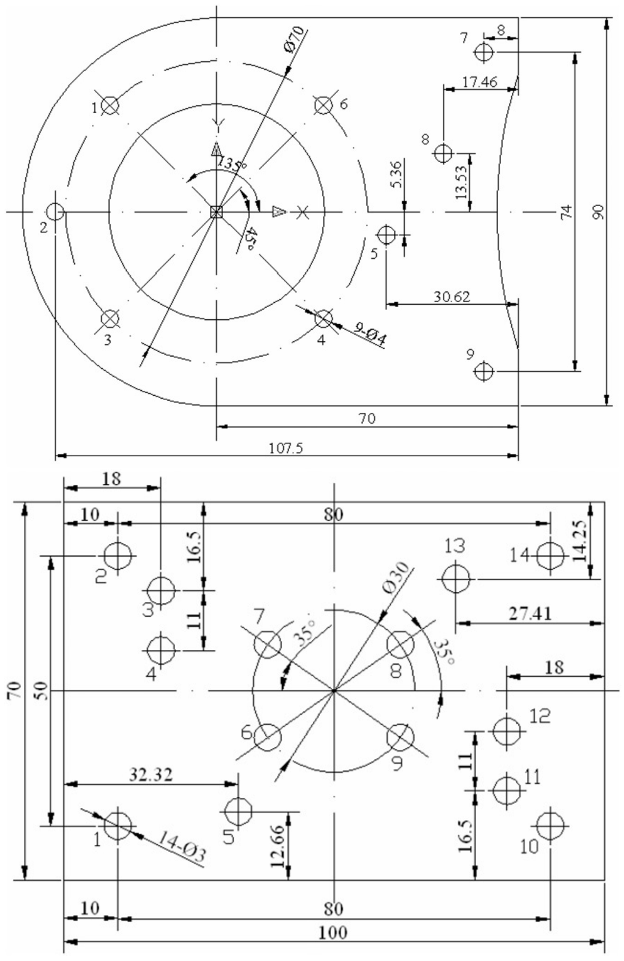

Fig. 14. Printed circuit boards with holes to drill [3]

by the target function value, indicates the "distance" of the current location. For the i-th particle at time $t+1$, it has a velocity of

$$v_i(t+1) = [c_1 \otimes (p_i(t) - x_i(t))] \oplus [c_2 \otimes (p_g(t) - x_i(t))],$$

where $x_i(t)$ is its current location at time t, $p_i(t)$ is the best location the particle has searched so far, and $p_g(t)$ is the best location the swarm has searched so far. The velocity is a linear combination, taking both individual and collective information into account. It follows that the location is updated according to the speed:

$$x_i(t+1) = x_i(t) + v_i(t+1).$$

The convergence of standard PSO has been studied by Solis and Wets (1981). Their work shows that the standard PSO is not guaranteed to be a local optimization solution, nor is the local convergence.

B. PSO with global convergence characteristics

In the topical paper [3], there are two key improvements:

1) Improvement 1: In standard PSO, the velocity of particle i at time $t+1$ should be

$$v_i(t+1) = [c_1 \otimes (p_i(t) - x_i(t))] \oplus [c_2 \otimes (p_g(t) - x_i(t))],$$

which does not depend on the previous speed $v_i(t)$.

In the topical paper, the term of previous speed and the speed memory remain, indicating that the capability of convergence on global solution is not affected.

$$v_i(t+1) = (\omega \otimes v_i(t)) \oplus [c_1 \otimes (p_i(t) - x_i(t))] \oplus [c_2 \otimes (p_g(t) - x_i(t))].$$

This is probably inspired by the momentum term, or inertia term, that has been in widespread use in other algorithms such as gradient descent. The inertia term helps make the velocity term smoother and increase the chance of getting rid of local minimum areas by exploration.

2) Improvement 2: When the swarm evolves to certain generation, there is at least one particle located in the best previous position of the swarm and this particle will stop evolving. Then, the particle is improved by the following method to reinforce the global convergence of the algorithm.

For the j-th particle at time step t, when the three positions of the particle $x_j(t)$, $p_j(t)$, $p_g(t)$ are superposing, the j-th particle will stop evolving. In order to improve the convergence of the algorithm, $p_g(t)$ is kept as the best previous position of the particle swarm, the j-th particle position $x_j(t+1)$ is generated randomly again in the search space S, then update $p_j(t+1)$, positions of other particles i can be obtained by using equations of standard PSO at time step $t+1$, then update $p_g(t+1)$.

3) Why these improvements work: With the improvements above, there is at least one particle j whose positions $x_j(t)$, $p_j(t)$ and $p_g(t)$ are superposing in certain evolutional generations. Thus at least one particle needs to be generated randomly again in the search space S, so the global convergence capability of the new algorithm is reinforced as a consequence.

C. Experiments and Conclusion

The authors tested the PSO with global convergence characteristics on two work-pieces (Figure 14). The hyperparameters c_1 (weight of individual memory term $p_i(t)$) and c_2 (weight of collective memory term $p_g(t)$) are chosen randomly between 0 and 1. The hyperparameter ω (weight of inertial term) is chosen from $\{0, 0.5, 1\}$. The performance of global convergence is defined as the search ratio, i.e.

$$\frac{\text{Total of search generation numbers}}{\text{the solution space}}.$$

Figure 15 demonstrates the result. The conclusions can be reached that [3]

- The global convergence PSO algorithm has fast convergence speed.
- The local optimization solutions are improved at the same time by the new PSO algorithm and the improved local optimization solutions are much nearer to the global optimization solution.
- The new global convergence PSO algorithm can obtain the known global optimization solution, but the basic PSO algorithm is not guaranteed to be convergent on the global optimization solution.

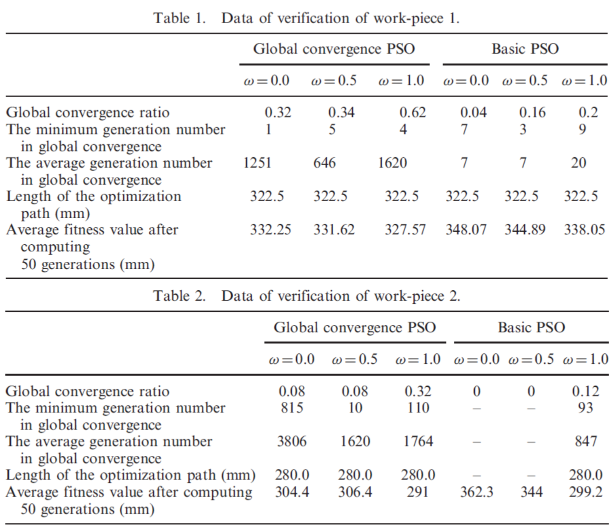

Table 1. Data of verification of work-piece 1.

	Global convergence PSO			Basic PSO		
	$\omega=0.0$	$\omega=0.5$	$\omega=1.0$	$\omega=0.0$	$\omega=0.5$	$\omega=1.0$
Global convergence ratio	0.32	0.34	0.62	0.04	0.16	0.2
The minimum generation number in global convergence	1	5	4	7	3	9
The average generation number in global convergence	1251	646	1620	7	7	20
Length of the optimization path (mm)	322.5	322.5	322.5	322.5	322.5	322.5
Average fitness value after computing 50 generations (mm)	332.25	331.62	327.57	348.07	344.89	338.05

Table 2. Data of verification of work-piece 2.

	Global convergence PSO			Basic PSO		
	$\omega=0.0$	$\omega=0.5$	$\omega=1.0$	$\omega=0.0$	$\omega=0.5$	$\omega=1.0$
Global convergence ratio	0.08	0.08	0.32	0	0	0.12
The minimum generation number in global convergence	815	10	110	–	–	93
The average generation number in global convergence	3806	1620	1764	–	–	847
Length of the optimization path (mm)	280.0	280.0	280.0	–	–	280.0
Average fitness value after computing 50 generations (mm)	304.4	306.4	291	362.3	344	299.2

Fig. 15. Experiments by [3]

The authors made final remarks on further work: how to improve the generating method of the stop evolution particle. In the topical paper, the generating method is a simple random generation with few exceptional cases of superposition. In future works, more generating methods may be studied, and this still remains an open question.

References

[1] Wikipedia contributors, "Test functions for optimization — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Test_functions_for_optimization&oldid=865176489, 2018, [Online; accessed 31-December-2018].

[2] L. Wang, Intelligent Optimization Algorithms, 1st ed. Tsinghua University Press, 10 2001.

[3] G.-Y. Zhu and W.-B. Zhang, "Drilling path optimization by the particle swarm optimization algorithm with global convergence characteristics," International Journal of Production Research, vol. 46, no. 8, pp. 2299–2311, 2008.