ISSN : 0974 - 7435 Volume 10 Issue 24

2014

BioTechnology

An Indian Journal

FULL PAPER

BTAIJ, 10(24), 2014 [14968-14980]

Multi-strategy multi-agent simulated annealing algorithm based on particle swarm optimization algorithm

Changying Wang*, Ming Lin, Yiwen Zhong
College of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou, 350002, (CHINA)
E-mail : cptone@163.com

ABSTRACT

Multi-agent simulated annealing (MSA) algorithm based on particle swarm optimization (PSO) is a population-based SA algorithm, which uses the velocity and position update equations of PSO algorithm for candidate solution generation. MSA algorithm can achieve significantly better intensification ability by taking advantage of the learning ability from PSO algorithm; meanwhile Metropolis acceptance criterion is efficient to keep MSA from local minima. Taking into account that different problems may require different parameters for MSA to achieve good performance, this paper proposes a multi-strategy MSA (MMSA) algorithm. In MMSA algorithm, three parameter control strategies, multiple perturbation equations, variant number of perturbed dimensions and declining population size, are used to enhance the performance of MSA algorithm. Simulation experiments were carried on 10 benchmark functions, and the results show that MMSA algorithm has good performance in terms of solution accuracy.

KEYWORDS

Multi-agent simulated annealing; Particle swarm optimization; Metropolis acceptance criterion.

INTRODUCTION

Simulated annealing (SA) algorithm, which was first independently presented for combinatorial optimization problems [1, 2], is a popular iterative meta-heuristic algorithm widely used to address discrete and continuous optimization problems. The key feature of SA algorithm lies in a means to escape from local optima by allowing hill-climbing moves to find a global optimum. The major shortage of SA is that it can be extremely slow and require significantly more processing time than other meta-heuristics. This is mainly because SA does not learn from its searching history intelligently when sampling to generate candidate solutions. Several attempts have been made to improve SA's performance either by reducing the annealing length or changing the generation and the acceptance mechanisms. However, these improved schemes, while run faster, do not generally inherit SA's capability of escaping from local minima [3]. Adding parallelism is another efficient way to reduce the processing time [4]. However, the implementation and efficiency of parallel SA algorithms are often problem-dependent. On the contrary, population-based meta-heuristics, such as genetic algorithm [5], differential evolution (DE) algorithm [6, 7], particle swarm optimization (PSO) algorithm [8-10], firefly algorithm [11] and cuckoo search [12, 13] etc., are equipped with intelligent learning mechanisms to guide their searching, so they can balance intensification and diversification more easily.

Inspired by the cooperative behaviours among individuals of population-based meta-heuristics, multi-agent SA (MSA) algorithm based on PSO algorithm [14] and MSA algorithm based on DE algorithm [15] are learning based MSA algorithms. MSA algorithms take advantage of both the learning ability from population-based meta-heuristics and the hill-climbing ability from SA. A population of agents runs SA algorithm collaboratively, and use the velocity and position update equations from PSO or mutation operator formulas from DE to generate candidate solutions. Those sampling schemes leverage the learning ability from PSO or DE algorithm; it can adjust the neighbourhood structure adaptively and exploit the promising search space more finely in the later stage, thus it can improve the convergence speed of MSA algorithms effectively. MSAs have been successfully applied for continuous space optimization problems, such as function optimization problems and protein structure prediction problem on AB model [16].

In this work we are motivated by the factor that different problems may require different parameters for MSA to achieve good performance. For example, for multi-dimensional problems, MSA perturbs one dimension only to produce a candidate solution. But for some problems, MSA may have better performance when perturbing all dimensions to produce a candidate solution. Aims to tackle those shortcomings, this paper propose a multi-strategy MSA (MMSA) algorithm. In MMSA algorithm, three parameter control strategies, multiple perturbation equations, variant number of perturbed dimensions and declining population size, are used to enhance the performance of MSA algorithm.

The remainder of this paper is organized as follows: Section 2 provides a short description of PSO algorithm and MSA algorithm. Section 3 presents our MMSA algorithm. Section 4 gives the experimental approach and results of experiments carried on 10 benchmark functions. Finally, section 5 summaries the study and possible future works.

PRELIMINARIES

Particle Swarm Optimization Algorithm

PSO algorithm is a stochastic population based optimization algorithm, first published by Kennedy and Eberhart in 1995. There are two variants of the PSO algorithm. One has a global neighborhood, and the other has a local neighborhood. In the global variant, each particle moves towards its best previous position and towards the best particle in the whole swarm. On the other hand, in the local variant, each particle moves towards its best previous position and towards the best particle in its specified neighborhood. Suppose that the search space is N-dimensional, then the i-th particle of the swarm can be represented by a N-dimensional vector, $\mathbf{X}_i = (x_{i1}, x_{i2}, ..., x_{iN})$. The velocity of this particle, which represents the position change of this particle, can be represented by another N-dimensional vector $\mathbf{V}_i = (v_{i1}, v_{i2}, ..., v_{iN})$. The best previously visited position of the i-th particle is denoted as $\mathbf{P}_i = (p_{i1}, p_{i2}, \ldots, p_{iN})$. For the global variant, the best previously visited position of the swarm is $\mathbf{G}_i = (g_{i1}, g_{i2}, ..., g_{iN})$, and let the superscripts denote the iteration number, then each particle is manipulated according to the following two equations [17]:

$$v_{ij}^{t+1} = wv_{ij}^{t} + c_1 r_1 (p_{ij}^{t} - x_{ij}^{t}) + c_2 r_2 (g_j^{t} - x_{ij}^{t}) \quad (1)$$

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1} \quad (2)$$

where j =1, 2, ..., N; i =1, 2, ..., M, and M is the size of the swarm; w is called inertia weight, which is used to control the impact of the previous history of velocities on the current velocity, thus to influence the trade-off between global and local exploration abilities of the particle [17]; c_1, c_2 are two positive constants, called cognitive and social parameter respectively; r_1, r_2 are random numbers, uniformly distributed in [0, 1]; and t =1, 2, ..., determines the iteration number. Eq. (1) is used to calculate the particle's new velocity according to its previous velocity and the distances of its current position from its own best previously visited position and the global best experience. Then the particle flies toward a new position according to Eq. (2).

Since its first publication, a large body of research has been done to study the applications of PSO, and to improve its performance. To improve PSO's performance, many studies concentrated on a better understanding of the PSO control parameters, such as w, c_1 and c_2. Those parameters control particle's learning from its previous velocity, its history best position and the best position of its neighbours. Using Eq. (1) as an example, w controls the learning from its previous velocity, c_1 controls the learning from itself and c_2 controls the learning from its neighbours. Different problems may require different velocity equation to achieve good performance for PSO algorithm, in section 3.1, we will compare two different velocity equation, one is the classical PSO with constriction factor (CPSO) [18], and the other is a newly proposed Median-oriented PSO (MPSO) [19].

Clerc [18] indicates that the use of a constriction factor may be necessary to insure convergence of the PSO algorithm. As a result of the constriction factor, the velocity equation is changed to Eq. (3), where constriction factor k is a function of c_1 and c_2 as reflected in Eq. (4).

$$v_{ij}^{t+1} = k(v_{ij}^t + c_1 r_1 (p_{ij}^t - x_{ij}^t) + c_2 r_2 (g_j^t - x_{ij}^t)) \tag{3}$$

$$k = 2 \Big/ \left| 2 - \varphi - \sqrt{\varphi^2 - 4\varphi} \right|, \text{where } \varphi = c_1 + c_2, \varphi > 4 \tag{4}$$

Beheshti et al. [19] propose the MPSO algorithm to carry out a global search over entire search space with accelerating convergence speed and avoiding local optima. The median position of particles and the worst and median fitness values of the swarm are incorporated in the standard PSO to achieve the mentioned goals. In MPSO, each particle updates its velocity based on Eq. (5):

$$v_{ij}^{t+1} = v_{ij}^t + M_{ij}^t \tag{5}$$

where M_{ij} represents the median-oriented acceleration which is defined as follows:

$$M_{ij}^t = a_i^t (r_1 (p_{ij}^t - p_{mj}^t - x_{ij}^t) + r_2 (g_j^t - p_{mj}^t - x_{ij}^t)) \tag{6}$$

where p_{mj} is the current median position of the swarm in the jth dimension. a_i is acceleration factor as:

$$A_i^t = \frac{fit_i^t - Maxfit^t}{Medfit^t - Maxfit^t} \tag{7}$$

$$a_i^t = A_i^t \Big/ \sum_{j=1}^{N} A_j^t \tag{8}$$

where fit_i represents the fitness value of the particle i, *Maxfit* and *Medfit* are the current maximum and median fitness values of swarm.

Multi-agent Simulated Annealing Algorithm

SA algorithm is commonly said to be the oldest among the meta-heuristics and surely one of the first algorithms that had an explicit strategy to avoid local minima. The origins of SA are in statistical mechanics (Metropolis algorithm) and it was first presented as a search algorithm for combinatorial optimization problems. The fundamental idea is to allow moves resulting in solutions of worse quality than the current solution in order to escape from local minima. The probability of performing such a move is decreased during the search through parameter temperature.

SA's performance depends mainly on the following four ''enough'': (a) the initial temperature is high enough, (b) the temperature is cooled slowly enough, (c) the parameter space is sampled often enough, (d) the stop temperature is low enough. These requirements make the SA converge very slowly in most cases [20]. In order to decrease its computing time, some researchers have studied Parallel SA algorithms based on population based intelligent algorithm, such as those using genetic operators to exchange information [21-23], and MSAs which use learning-based sampling to produce candidate solution [14-15, 24].

MSA algorithm based on PSO takes advantage of the learning ability of PSO and the hill-climbing ability of SA. A population of agents runs SA algorithm collaboratively. Like particles in PSO algorithm, each agent has a velocity, a position and limited memory. It can remember the best position it has ever visited. In order to generate a candidate solution, a random dimension is generated first. And then equation (1) and (2) are used to calculate the new velocity and candidate solution. Finally the Metropolis criterion is used to decide whether to accept the new solution.

Algorithm 1 is the pseudo code of MSA. Where *x*, *v*, and *pb* are all 2-dimensional arrays. The *x* represents positions of all agents, *x*[*i*] represents the position of *i*-th agent, and *x*[*i*][*j*] represents the *j*-th dimensional position of *i*-th agent. Similarly, the *v* represents velocities of all agents, *v*[*i*] represents the velocity of *i*-th agent, and *v*[*i*][*j*] represents the *j*-th dimensional velocity of *i*-th agent. The *pb* represents best previously visited positions of all agents, *pb*[*i*] represents the best previously visited position of *i*-th agent, and *pb*[*i*][*j*] represents the *j*-th dimensional value of best previously visited position of *i*-th agent. Variable *gb* is a 1-dimensional array, which represents the best previously visited position of all agents. In line 19, agent will change the direction of its velocity if the solution is not improved. We call this opposite velocity strategy, which can be viewed as an application of opposition-based learning.

Algorithm 1 MSA

1. Initialize parameter w, k, c_1, c_2, t_0 and a;
2. For each agent i from 1 to M
3. Generate position $x[i]$ and velocity $v[i]$ randomly;
4. $pb[i] = x[i]$;
5. End For
6. gb = findBest(x); //to find the best agent in population
7. $t = t_0$; //initial temperature
8. While (End condition is not met)
9. For each agent i from 1 to M
10. Repeat the following operation sampling times
11. Candidate solution $y = x[i]$;
12. Generate a dimension j randomly;
13. Calculate j-D velocity $v[i][j]$ using (3);
14. Calculate j-D position $y[j]$ using (2);
15. $e = f(y) - f(x[i])$;
16. If ($e < 0$ OR random() < exp($-e/t$)) Then
17. $x[i] = y$;
18. End if
19. If ($e >= 0$) then $v[i][j] = - v[i][j]$
20. End Repeat
21. If ($x[i]$ is better than $pb[i]$) Then $pb[i] = x[i]$;
22. If ($x[i]$ is better than gb) Then $gb = x[i]$;
23. $t = at$
24. End for
25. End while
26. Return gb

MULTI-STRATEGY MULTI-AGENT SIMULATED ANNEALING ALGORITHM BASED ON PSO ALGORITHM

Effect of Perturbation Equations

Because different PSO algorithms use different velocity and position equations to produce new velocity and position, we may use different equations to produce candidate solutions in MSA. In order to observe the effect of those equations (Because those equations are used to produce perturbation for current position, we call them perturbation equations hereafter.), we compare the performance of CPSO and MPSO on F3, F4, F9, and F10 functions (Those functions are listed in section 4.1). Results, which are recorded as function error value and reported in Table 1, show how the performance of PSO changes with different perturbation equations. MPSO has better performance on F3 and F4 functions, but CPSO has better performance on F9 and F10 functions. The iteration process of median solution of CPSO and MPSO on the 4 test functions are displayed in Fig. 1 to Fig.4 respectively. Those figures show clearly that different test function may require different perturbation equation. Taking into account this factor, we classify agents in MMSA as CPSO type or MPSO type. A CPSO type agent will use Eq. (3) to produce candidate solution, and a MPSO type agent will use Eq. (5) to produce candidate solution. In the initialization step, half of the agents will be set as CPSO type, and the others will be set as MPSO type.

Table 1 : Effect of perturbation equations

fun	CPSO			MPSO		
	Best	**Median**	**Ave**	**Best**	**Median**	**Ave**
F3	7.55E-15	1.78	1.79	4.00E-15	**4.00E-15**	**4.00E-15**
F4	0	3.45E-2	0.03	0	**0**	**0**
F9	1.57E-32	**2.03E-31**	**0.30**	0.66	1.12	1.07
f10	1.35E-32	**1.22E-31**	**0.48**	31.56	31.95	31.93

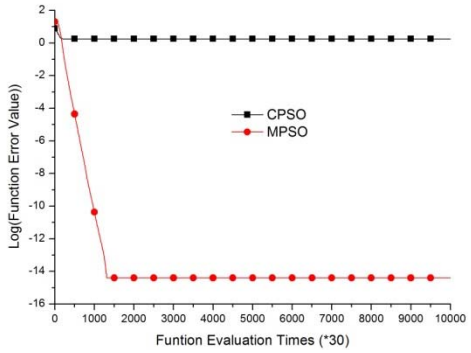

Figure 1 : Iteration process of median solution on F3 function

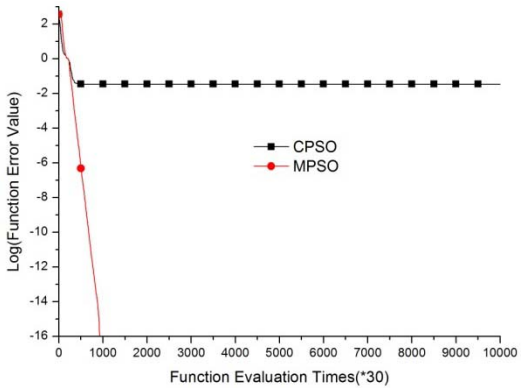

Figure 2 : Iteration process of median solution on F4 function

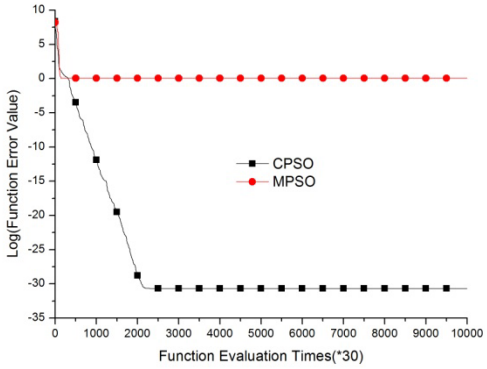

Figure 3 : Iteration process of median solution on F9 function

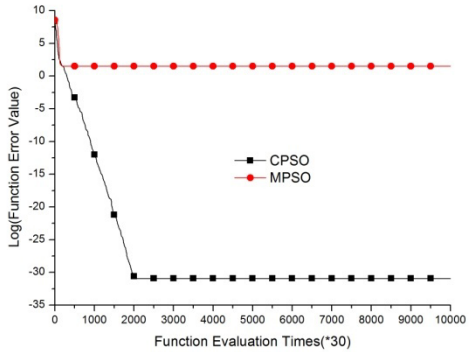

Figure 4 : Iteration process of median solution on F10 function

Effect of Number of Perturbed Dimensions

Performance of MSA may be affected by the number of perturbed dimensions (*PD*) used to generate candidate solution. MSA with low *PD* may search the solution space more finely, but it may have much slow convergence speed. Due to interfering among dimensions, MSA with large *PD* may not be able to search the solution space finely enough. To investigate the effect of variant *PD* on MSA, we experimented with different *PD* in [1, 5, 10, 15, 20, 25, 30] on function F1 and F6. Results, reported in Table 2, show how the performance of MSA changes with the *PD*. For F1 function, MSA has best performance when *PD* is 20. For F7 function MSA has best performance when *PD* is 5. Taking into account that different test function may have different optimal *PD*, we use a random strategy to generate *PD* from [1, *D*] as following pseudo-code.

Method 1 producePD( )
  PD = 1
  While (rand(0,1) > *p*) and (*PD* < *D*)
    PD = *PD* + 1
  End While
  Return *PD*

Parameter *p* is a used to control the distribution of *PD*.

Table 2 : Effect of number of perturbed dimensions

PD	F1			F7		
	Best	**Worst**	**Ave**	**Best**	**Worst**	**Ave**
1	4.77E-118	8.99E-109	7.53E-110	0.70	1.20	0.96
5	5.64E-199	8.19E-196	1.04E-196	0.40	0.90	**0.61**
10	1.68E-273	1.29E-269	1.38E-270	0.40	1.40	0.94
15	1.0E-323	7.07E-318	6.73E-319	0.80	2.40	1.41
20	0	0	**0**	1.20	3.30	1.97
25	0	4.0E-322	2.00E-323	1.80	4.30	2.66
30	2.79E-243	1.36E-207	5.52E-209	2.00	4.70	3.07

Sensitivities to Population Size

Performance of MSA is sensitive to the selected population size. This is easily conceivable because MSA employs a one-to-one perturbation strategy. Therefore, if a very large population size is selected, then MSA may have good diversification, but may exhaust the fitness evaluations very quickly without being able to locate the optimum. A small population size may have good intensification, but may make MSA trapped into local minimum occasionally. To investigate the sensitivity of MSA to variations of population size, we experimented with different population sizes from 10 to 100 at increments of 10 on function F1 and F4. Results, reported in Table 3, show how the performance of MSA changes with the population size. MSA has good performance with a smaller population size for F1 function, but is more robust with a larger population size for F4 function. A population declining strategy may have better trade-off between diversification and intensification [25]. In the early stage, large population can enlarge searching range, and as population declines in successive iterations, more function evaluation times may be used to search in promising areas.

Table 3 : Sensitivities to population size

P	F1			F4		
	Best	**Worst**	**Ave**	**Best**	**Worst**	**Ave**
10	0	4.4E-323	1.0E-323	0	2.95E-2	1.35E-3
20	1.2E-175	2.1E-165	9.4E-167	0	9.87E-3	3.95E-4
30	1.5E-121	6.4E-108	2.7E-109	0	1.06E-9	4.24E-11
40	8.12E-87	4.81E-80	5.63E-81	0	0	0
50	1.25E-70	8.60E-64	1.89E-64	0	0	0
60	1.37E-56	1.03E-51	1.01E-52	0	0	0
70	4.04E-48	4.68E-44	5.89E-45	0	0	0
80	2.02E-41	1.56E-37	2.08E-38	0	0	0
90	3.32E-37	6.87E-33	1.14E-33	0	0	0
100	1.51E-33	2.07E-29	3.92E-30	0	0	0

Multi-strategy MSA Algorithm

Based on the analysis of above sections, we describe MMSA algorithm as following algorithm 2. In algorithm 2, array *type* stores the type of perturbation equation used by agent. Line 16 to Line 22 are used to produce new candidate solution, whose values, stored in *PD* successive dimensions beginning from *j*, are updated using corresponding perturbation equation according to the value in *type*[*i*]. If *type*[*i*] is "CPSO", then agent *i* will use Eq. (3) to produce new velocity (see line 20), otherwise, agent *i* will use Eq. (5) to produce new velocity. Line 27 to Line 31 are opposite velocity strategy which is an application of opposition-based learning. In line 36, MMSA keep the best half agents in population after each quarter of function evaluation times has been spent.

Algorithm 2 MMSA

1. Initialize parameter w, k, c_1, c_2, t_0 and a;
2. For each agent i from 1 to M
3. Generate position $x[i]$ and velocity $v[i]$ randomly;
4. $pb[i] = x[i]$;
5. If Mod(i, 2) == 0 Then
6. $type[i]$ = "CPSO";
7. Else
8. $type[i]$ = "MPSO";
9. End if
10. End For
11. gb = findBest(x); //to find the best agent in population
12. $t = t_0$; //initial temperature
13. While (End condition is not met)
14. For each agent i from 1 to M
15. Repeat the following operation sampling times
16. Candidate solution $y = x[i]$
17. Generate a dimension d randomly
18. PD = producePD( )
19. For $j = d$ to $d + PD - 1$
20. Calculate $v[i][j]$ according to $type[i]$;
21. $y[j] = y[j] + v[i][j]$;
22. End for
23. $e = f(y) - f(x[i])$;
24. If ($e < 0$ OR random() < exp(-e/t)) Then
25. $x[i] = y$;
26. End if
27. If ($e >= 0$) Then
28. For $j = d$ to $d + PD - 1$
29. $v[i][j] = - 0.5 * v[i][j]$;
30. End for
31. End If
32. End Repeat
33. If ($x[i]$ is better than $pb[i]$) Then $pb[i] = x[i]$;
34. If ($x[i]$ is better than gb) Then $gb = x[i]$;
35. $t = at$;
36. Declining the population size if necessary
37. End for
38. End while
39. Return gb

SIMULATION RESULTS

Experimental Approach

For comparison, the experiments use ten non-linear benchmark functions which are used in many literatures. Those functions are listed as follows:

(1) Sphere Function

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2, \quad |x_i| \le 100$$

(2) Generalized Rosenbrock's Function

$$f_2(\mathbf{x}) = \sum_{i=1}^{n-1}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2), \quad |x_i| \le 30$$

(3) Ackley's Function

$$f_3(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i) + 20 + e, \quad |x_i| \le 32$$

(4) Generalized Griewank's Function

$$f_4(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n}x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1, \quad |x_i| \le 600$$

(5) Generalized Rastrigrin's Function

$$f_5(\mathbf{x}) = \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i) + 10), \quad |x_i| \le 5.12$$

(6) Generalized Schwefel's Problem 2.26

$$f_6(\mathbf{x}) = \sum_{i=1}^{n} - x_i \sin(\sqrt{|x_i|}), \quad |x_i| \le 500$$

(7) Salomon

$$f_7(\mathbf{x}) = -\cos\left(2\pi\sqrt{\sum_{i=1}^{n}x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n}x_i^2} + 1, \ |x_i| \le 100$$

(8) Whitely

$$f_8(\mathbf{x}) = \sum_{j=1}^{n}\sum_{i=1}^{n}\left(\frac{y_{i,j}^2}{4000} - \cos(y_{i,j}) + 1\right);$$

where $y_{i,j} = 100(x_j - x_i^2)^2 + (1 - x_i)^2, |x_i| \le 100$

(9) Generalized Penalized Function 1

$$f_9(\mathbf{x}) = \frac{\pi}{n}\left(10\sin^2(\pi y_i) + \sum_{i=1}^{n-1}(y_i - 1)^2(1 + 10\sin^2(\pi y_{i+1})) + (y_n - 1)^2\right) + \sum_{i=1}^{n}u(x_i, 10, 100, 4), \quad |x_i| \le 50$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$$

(10) Generalized Penalized Function 2

$$f_{10}(\mathbf{x}) = 0.1\sum_{i=1}^{n-1}(x_i - 1)^2(1+\sin^2(3\pi x_{i+1})) + (x_n - 1)(1+\sin^2(2\pi x_n))$$
$$+ 0.1\sin^2(3\pi x_i) + \sum_{i=1}^{n} u(x_i, 5, 100, 4), \quad |x_i| \le 50$$

In order to observe the performance of MMSA algorithms, we compare MMSA with the classical CPSO [18], the newly proposed MPSO [19], MSA using CPSO (MSAC), and MSA using MPSO (MSAM) on above 10 functions. Each experiment was run 25 times with random initial values of *x* and *v* in the range indicated in corresponding equation. The total function evaluation times is set to 10000**D*, where *D* is the dimension of test function.

Results for Fixed Function Evaluation Times

Table 4 is the experiment results for fixed function evaluation times. It lists the mean function error value and rank for 25 runs of the ten test functions respectively. Table 4 shows that MMSA has best performance on eight test functions. The average rank of MMSA is 1.5, which is the best among the five algorithms. The bad performance of MMSA on function F2 is caused by two factors: (1) both perturbation equations are not ideal for F2 function. This can be confirmed by the bad performance of CPSO and MPSO; (2) F2 prefers low *PD*. If the *PD* is fixed to 1, the result of MMSA is 2.94E-01, which is best among the five algorithms. It suggests that more intelligent way is needed to control the parameter *PD*.

Table 4 : Compare MMSA with CPSO, MPSO, MSAC, and MSAM

fun	CPSO		MPSO		MSAC		MSAM		MMSA	
	Mean	Rank	Mean	Rank	Mean	Rank	Mean	Rank	Mean	Rank
F1	4.04E-155	3	3.94E-207	2	7.02E-109	5	1.22E-108	4	6.77E-222	1
F2	2.03E+00	2	2.90E+01	4	1.78E+00	1	2.42E+00	3	3.22E+01	5
F3	1.79E+00	5	4.00E-15	1	7.83E-15	3	3.19E-01	4	4.00E-15	1
F4	3.27E-02	5	0	1	2.96E-04	3	1.28E-03	4	0	1
F5	4.09E+01	5	0	1	0	1	7.88E+00	4	0	1
F6	5.94E+03	4	9.33E+03	5	1.17E+03	2	3.33E+03	3	2.65E+02	1
F7	6.44E-01	3	9.99E-02	1	9.64E-01	4	1.99E+00	5	3.28E-01	2
F8	4.06E+02	4	4.13E+02	5	2.13E+02	3	3.27E+01	2	0	1
F9	2.99E-01	4	1.07E+00	5	1.57E-32	1	8.30E-09	3	1.57E-32	1
F10	4.79E-01	4	3.19E+01	5	1.35E-32	1	4.38E-07	3	1.35E-32	1
Aver Rank		3.9		3		2.4		3.5		1.5
Final rank		5		3		2		4		1

Iteration Process

In order to compare the convergence process of MMSA algorithm with other algorithms, we draw the iteration process of mean function error value of the five algorithms on the 10 problems in Fig. 5 to Fig. 14. Those figures show that, in the early stage, large population size guarantees that the convergence speed is not too aggressive, so MMSA can have good exploration ability. As showed in Fig.10 and Fig.12, the advantage of MMSA algorithm is extremely apparent on function 6 and function 8.

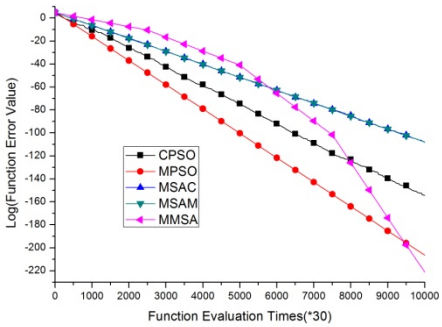

Figure 5 : Iteration Process for F1 Function

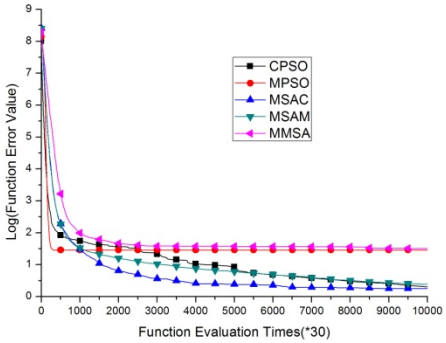

Figure 6 : Iteration Process for F2 Function

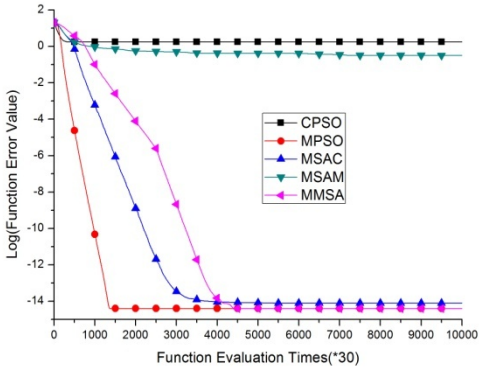

Figure 7 : Iteration Process for F3 function

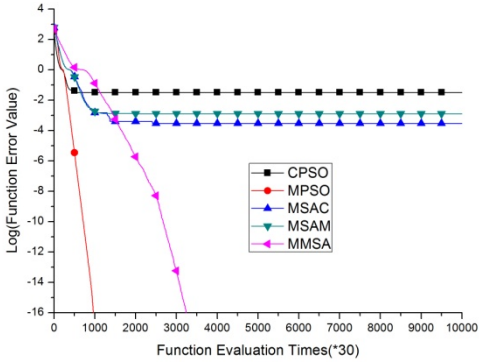

Figure 8 : Iteration Process for F4 function

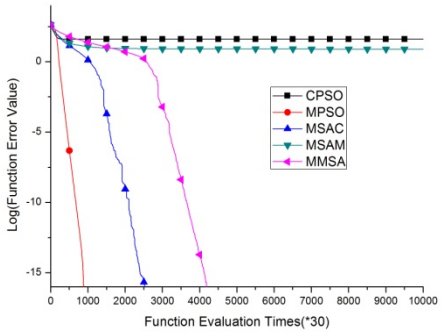

Figure 9 : Iteration Process for F5 function

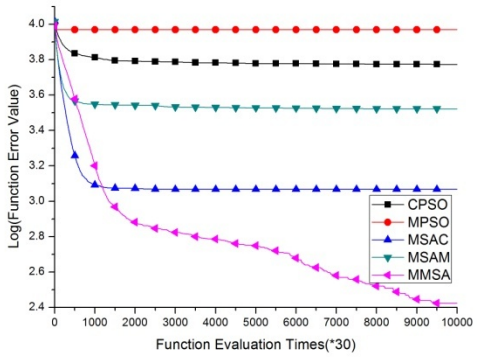

Figure 10 : Iteration Process for F6 Function

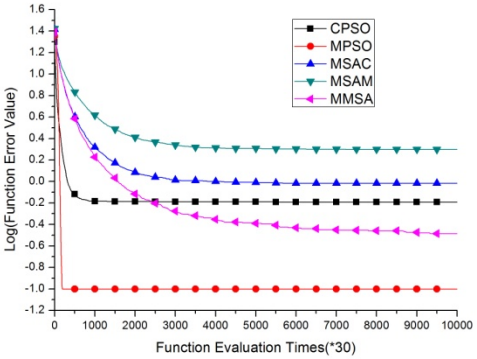

Figure 11 : Iteration Process for F7 Function

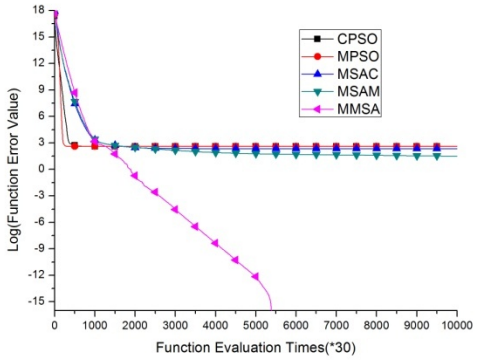

Figure 12 : Iteration Process for F8 Function

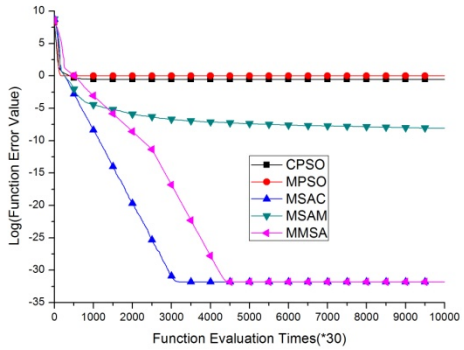

Figure 13 : Iteration Process for F9 Function

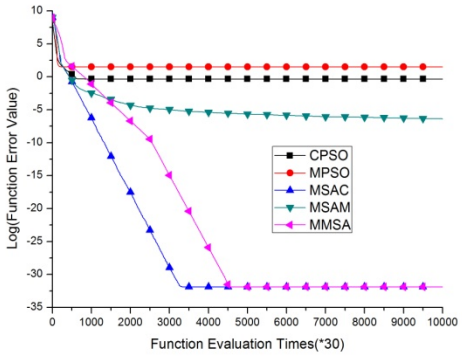

Figure 14 : Iteration Process for F10 Function

CONCLUSION AND FUTURE WORK

Motivated by the factor that different problem may require different parameters for MSA to achieve good performance, this paper proposes a Multi-strategy MSA algorithm for continuous function optimization problems. In MMSA algorithm, three parameter control strategies, multiple perturbation equations, variant number of perturbed dimensions and declining population size, are used to enhance the performance of MSA algorithm. Declining population size strategy not only can achieve better balance between exploration and exploitation, it can selects agents with suitable perturbation equation to survive also. Simulation results confirm the advantage of the MMSA algorithm.

As a first step of study, we use the equations of CPSO and MPSO algorithm only, use fixed strategy in variant number of perturbed dimensions, and use declining population size. Further study can be done in a move adaptive way, such as follows: (1) to use more perturbation equations; (2) to change number of perturbed dimensions more intelligently; (3) to control population size adaptively.

ACKNOWLEDGMENTS

This work was supported by Nature Science Foundation of Fujian Province of P. R. China (No. 2013J01216, and 2012D081) and The State Bureau of Forestry 948 project (No.2013-4-65).

REFERENCES

[1] Cerny, V., "Thermo dynamical approach to the travelling salesman problem: an efficient simulation algorithm". Journal of Optimization Theory and Application. Vol.45, No.1, pp.41-51, 1985.

[2] Kirkpatrick S., Gelatt C. D., and Vecchi M. P., "Optimization by simulated annealing", Science, Vol. 220, No. 4598, pp.671-680, 1983.

[3] Henderson D., Jacobson S. and Johnson A., "The theory and practice of simulated annealing". Handbook of metaheuristics, Vol. 57, pp.287-319, 2003.

[4] Hamma, B., Viitanen, S. and Torn, A., "Parallel continuous simulated annealing for global optimization". Optimization Methods and Software, Vol.13, No. 2, pp.95-116, 2000.

[5] Goldberg, D.E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley. Boston, MA, USA, 1989.

[6] Storn, R. and Price, K., "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". Journal of Global Optimization, Vol.11, No.4, pp. 341-359, 1997.

[7] Rajathy R., Gnanadass R., Manivannan K. and Kumar H., "Computation of Capacity Benefit Margin using Differential Evolution", Int. J. of Computing Science and Mathematics, Vol.3, No.3, pp.275 – 287, 2010.

[8] Kennedy, J. and Eberhart, R., "Particle Swarm Optimization", In: IEEE Intl Conf on Neural Networks, Perth, Australia, pp.1942-1948, 1995.

[9] Eberhart, R. and Kennedy, J., "A New Optimizer Using Particle Swarm Theory", In: Proc of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, pp.39-43, 1995.

[10] Gao Y.X., Wang Y.M. and Pei Z.L., "An improved particle swarm optimisation for solving generalised travelling salesman problem", Int. J. of Computing Science and Mathematics, Vol.3, No.4, pp.385 – 393, 2012.

[11] Yang X.S., "Firefly algorithms for multimodal optimization", in: Stochastic Algorithms: Foundations and Applications, SAGA 2009, Lecture Notes in Computer Sciences, Vol. 5792, pp. 169-178, 2009.

[12] Yang X. S. and Deb S., "Cuckoo search via Lévy flights", In: Proceedings of World Congress on Nature & Biologically Inspired Computing, IEEE Publications, USA, pp. 210-214, 2009.

[13] Yang X.S. and Deb S., "Engineering Optimization by Cuckoo Search", International Journal of Mathematical Modeling and Numerical Optimization, Vol. 1, No. 4, pp.330-343, 2010.

[14] Zhong Y.W., Ning J. and Zhang H., "Multi-agent simulated annealing algorithm based on particle swarm optimization algorithm", International Journal of Computer Applications in Technology, Vol. 43, No. 4, pp.335-342, 2012.

[15] Zhong Y.W., Wang L.J., Wang C.Y. and Zhang H., "Multi-agent simulated annealing algorithm based on differential evolution algorithm", International Journal of Bio-Inspired Computation, Vol.4, No.4, pp.217-228, 2012.

[16] Lin J., Ning J., Du Q.L., and Zhong Y.W., "Multi-agent Simulated Annealing Algorithm Based on Particle Swarm Optimization Algorithm for Protein Structure Prediction". Journal of Bionanoscience, Vol.7, No.1, pp.84-91, 2013.

[17] Shi, Y. and Eberhart, R., "Parameter selection in particle swarm optimization", In: Proceedings of the Seventh Annual Conference on Evolutionary Programming, New York, USA, pp.591-600, 1998.

[18] Clerc, M., "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization", In: Proceedings of the IEEE Congress on Evolutionary Computation, Washington, DC, USA, pp.1951 -1957, 1999.

[19] Beheshti, Z., Shamsuddin, S. M. H., & Hasan, S., "MPSO: median-oriented particle swarm optimization". Applied Mathematics and Computation, Vol.219, No.11, pp.5817-5836, 2013.

[20] Sadati, N., Amraee, T., and Ranjbar, A.M., "A global Particle Swarm-Based-Simulated Annealing Optimization technique for under-voltage load shedding problem", Applied Soft Computing. Vol.9, No. 2, pp.652-657, 2009.

[21] Hiroyasu T., Miki M. and Ogura M., "Parallel Simulated Annealing using Genetic Crossover". Science and Engineering Review of Doshisha University. Vol.41, No.2, pp.130-138, 2000.

[22] Mahfoud W. and Goldberg E., "Parallel recombinative simulated annealing: A genetic algorithm". Parallel Computing. Vo.21, No.1, pp.1-28, 1995.

[23] Ohlídal, M., Schwarz, J., "Hybrid parallel simulated annealing using genetic operations". In Proc of Mendel 2004 10th International Conference on Soft Computing. Brno, CZ, FSI VUT, pp.89-94, 2004.

[24] Zhong Y.W., Lin J., Yang H., and Zhang H., "Analysis of learning–based multi–agent simulated annealing algorithm for function optimization problems", International Journal of Computing Science and Mathematics, Vol.4, No.4, pp.382-391, 2013.

[25] Wu, Z., Zhao, N., Ren, G., & Quan, T., "Population declining ant colony optimization algorithm and its applications". Expert Systems with Applications, Vol.36, No.3, pp.6276-6281, 2009.